MapO2R: MAPPING OBJECT ORIENTED APPLICATIONS TO RELATIONAL DATABASES
A Case of Study

J. J. Astrain, A. Córdoba and J. Villadangos
Dept. Ingeniería Matemática e Informática, Universidad Pública de Navarra, Campus de Arrosadía, 31006 Pamplona, Spain

Keywords: Object mapping, Relational databases, Case study.

Abstract: This paper presents an open education tool which enables students to acquire best practices on object oriented design and programming. This tool analyzes different mapping strategies to store objects into relational databases. The tool evaluates the performances of different mapping strategies offering a measure of their costs in terms of time spent in database operations. Students can select and evaluate their object-relational mapping proposals in order to discover the best design and implementation solutions for different case studies.

1 INTRODUCTION

Object-relational mapping consists on transforming between object and relational modeling approaches and between the systems that support them. Object-relational mapping requires the understanding of both object and relational modeling, their similarities and their differences. When dealing with real systems implementing object and relational models, we often discover deficiencies and inconsistencies with the theoretical relational approaches and a lack of standardization concerning object modeling. Thus object-relational mapping becomes a complicated task.

In object-relational databases (RDBMSs) the data resides in the database and is manipulated collectively with queries in a query language, while object-oriented databases (OODBMSs) are essentially a persistent object store for software written in an object-oriented programming language, with a programming API for storing and retrieving objects. Querying support is very limited in OODBMSs, while the object retrieval from RDBMSs requires hard mapping efforts. Object-relational mapping is used to map object oriented programming to RDBMSs.

In object-relational mapping products, the ability to directly manipulate data stored in a relational database using an object programming language is called transparent persistence (Barry, 2005). Most of these products deal with the problem of translating objects to relational tables and viceversa, but they use different mappings to tackle the existing impedance between object oriented programming and relational databases. Some problems related to this impedance arrives (Objectmatter, 2005): a) while objects have state, behavior, identity and data, an RDBMS stores data only; b) while objects are traversed using direct references, RDBMS tables are related to values in foreign and primary keys; c) current RDBMS have no equivalence to object inheritance for data and behavior; and d) the goal of relational modeling is to normalize data, whereas the goal of object-oriented design is to model a business process by creating real-world objects with data and behavior.

Benefits of using object oriented applications (as encapsulation, isolation, easy business logic maintenance or component reusability) advise to continue using object oriented designs. But relational models offer an interesting way to store and manage knowledge information that must be considered. Relational theory is concerned with knowledge and object techniques are concerned with behavior. Mapping between the two models requires deciding how the two worlds can refer to each other. Furthermore, relational databases have been deficient for multiple decades in correctly implementing the core concepts of relational theory, and object modeling is not standardized. Because of these deficiencies object-relational mapping is more complicated than it needs to be. A good mapping can significantly reduce development time otherwise spent with manual data handling in SQL and JDBC, and it also can significantly reduce the time spent by the RDBMS when accessing the stored data.

This paper presents an open education tool (MapO2R) which enables students to understand the object-relational mapping. A tool which allows defining and analyzing different mappings over different RDBMSs providing a benchmark between different mapping proposals. This tool allows students to select those mappings that perform better for each problem.

The paper is organized as follows: section 2 describes the concepts that students would acquire using MapO2R; section 3 analyzes different mapping techniques (their benefits and drawbacks); section 4 describes the internals of the tool; section 5 is devoted to present and analyze a case study where different mappings are compared over relational databases; and finally, conclusions and references end the paper.

2 LEARNING CONCEPTS

The tool allows the student acquiring some essential concepts. Each object has a unique object identifier (OID), which distinguishes it univocally from all other objects. At the creation step, a unique and univocal OID is assigned to each object, whether its state happens to be equal to other objects previously created. The *state* of an object is the set of current values of the attributes associated with a given identity. Objects can have a single state through their whole life or can go through many state transitions. Due to object encapsulation, the state is an abstraction which is only visible by examining the behavior of the object. Both OID and object state are stored into the RDBMS.

Objects provide an abstraction that clients can interact with. The *behavior* of an object is the collection of provided interactions (called methods or operations and, collectively, an interface) and the response to these method calls (or messages). Interactions with an object must be through its interface and all knowledge about an object is from its behavior (returned values or side effects) to the interface interaction.

Encapsulation provides an abstraction that makes possible the user to use this abstraction without knowing the implementation of the object. The user can known the state and behavior of an object, but not their implementation. That ensures an easy way to develop complex applications, using modular techniques and the power of object oriented design. Other concepts concerning relations among objects that are commonly used are inheritance, association and aggregation relationships. *Inheritance* applies to types (the specification of an interface that objects will support) or to classes (that define the implementation for multiple objects). When applied to types, inheritance specifies that an object can be used just like an object of both types. All objects that have a certain type, have also the new inherited type. When applied to classes, inheritance specifies that a class uses the implementation of another class with possible overriding modification implying type inheritance or not. *Association* is related to objects that establish a relationship (1-N, M-N) between them. And *aggregation* implies that an object is part of another object.

In the relational side, concepts as attribute, relationship, domain and tuple must be also introduced. A *relationship* is a truth predicate that defines what attributes are involved in the predicate and what the meaning of the predicate is. An *attribute* identifies a name that participates in the relationship and specifies the domain from which values of the attribute must come. A *domain* is simply a data type and a *tuple* is a truth statement in the context of a relation. A tuple has attribute values which match the required attributes in the relation and that state the condition that is known to be true. The state of an object is directly related to the attribute values of a certain tuple or a set of them. The object type determines the domains of all the attributes of a tuple. Concepts as aggregation, association and inheritance concern the relationships.

3 OBJECT-RELATIONAL MAPPING

Performance is limited by other secondary goals as flexibility and maintenance of the relational storage, as redundancy and fault tolerance, as space consumption and many others. The number, frequency and distribution of the read/write operations has also an important contribution in the final performance result obtained for a certain mapping schema. The application style determines the read/write operations that are performed over the persistent storage system implemented over an RDBMS. MapO2R considers different approaches and strategies that can be applied when mapping object-oriented applications to relational schemas as Keller presented in (Keller, 1997).

Aggregation can be mapped using two different strategies:

1. **Single Table Aggregation:** it is the optimal solution in terms of performance since only one table is accessed to retrieve an aggregating object with all its aggregated objects. As drawback, it can introduce data redundance.
2. **Foreign Key Aggregation:** it increases the number of database operations since it needs a join operation or at least two database accesses. This schema follows the normalization theory, but it can provide poor results when access-

ing/retrieving aggregated objects is performed together with the aggregating object. This strategy is more flexible and scalable, since it allows an easy way to support new aggregated objects. As drawback, aggregated objects are not automatically deleted on deletion of the aggregating objects, the mapping must to deal with this task.

Inheritance can be mapped using three different strategies:

1. **Single Table Inheritance:** as happened in the single table aggregation case, it is the optimal solution in terms of performance, but it introduces data redundance, poor scalability, contradicts the normalization theory and can introduce data inconsistency when performing schema evolutions.
2. **Inheritance Tree:** it minimizes or eliminates data redundance. This schema provides a high scalability degree. The main drawback is the number of database operations needed to insert, delete or update objects. So, performances decrease notably.
3. **Path Table:** it makes easy to perform path navigation across the objects against data redundancy and performances. Parent attributes are duplicated in each inheritance class.

An important factor is the inheritance hierarchy depth. Some solutions that work acceptable with flat inheritance hierarchies become inefficient and complicated with very deep inheritance hierarchies. Polymorphic read operations support and space storing reduction obtained when using multiple tables inheritance introduce low or very low write/update performances. Mapping solutions that clutter a single object's data across several tables might be fast for polymorphic reading, although they are very hard to maintain in case new object attributes are added or existing object attributes are deleted.

Schema evolution becomes a nightmare when multiple inheritance is allowed. When considering unique inheritance, the use of a unique table in a schema evolution implies the modification of the number and/or types of the fields of a table. This fact force to realize a big effort to ensure the consistency of the table. The system must ensure that previous and new data is consistently stored. When using multiple tables, scalability is easier to implement although the hierarchy depth can become a problem. Path navigation is complex and expensive in terms of database operations. The use of a inheritance tree strategy minimizes the data redundancy existent in the single table inheritance and in the path table strategies, but is very difficult to perform path navigation across object attributes since a great number of complex queries must be performed. Note that multiple inheritance is not considered.

Association can be mapped using different strategies:

1. **Foreign Key Association:** using two tables for a 1:N association, where the N is mandatory.
2. **Association Table:** using a new table to map N:M associations.

Association presents the same drawbacks and advantages previously described for the aggregation scenario, but involving two o three tables respectively.

4 MapO2R INTERNALS

MapO2R is a JAVA application that interacts with some RDBMSs located in different sites. Both computers (client and server) are interconnected by a switched Gigabit Ethernet LAN. MapO2R supports the following RDBMSs: MS Access 2007, Oracle 9i, SQL Server 2008, MySQL 5.1 and PostgreSQL 8.4.1. Due to its modularity and scalability, MapO2R can support other RDBMSs by adding the corresponding plugin. MapO2R follows a client-server architecture.

MapO2R acts on two stages: the generation and the benchmarking (queries) stages. In the generation stage, the student selects an RDBMS and the number of registers to consider (in the domain [500, 2,000,000] registers). Those registers are generated automatically in order to obtain the benchmarking of all the mappings described in Section 2. The benchmarking stage provides the results obtained including the average, the harmonic average, the median and the standard deviation. Furthermore, the scale can be either linear either logarithmic. Results can be provide in terms of figures or tables. The time is always measured in milliseconds. In order to obtain reliable benchmarking results, each query (INSERT, SELECT, DELETE and UPDATE) is performed following the number of iterations selected by the student (ten iterations by default).

MapO2R does not maintain a JDBC connection pool. Database session is closed and reopened for each iteration. If a student selects an UPDATE query over a set of 1,000 registers, with 25 iterations, the session between the the application and the RDBMS server is closed an reopened 25 times. The results obtained from each set of queries are stored in a filesystem in order to allow the comparison among the performances provided for both different mapping strategies and RDBMSs. MapO2R allows the visualization of the results and also allows exporting the benchmark results to CSV, plain text or MS Excel formats.

5 CASE STUDY

MapO2R is used on the *Analysis and Design Software* course of the *Computer Science* degree (Software Engineering) at the State University of Navarra. It has been validated during the courses 07/08 and 08/09 by 54 students. Students have answered a questionnaire concerning MapO2R and its relevance in the student's learning process at the end of each course, . In this section we include a practical session example, and the evaluation of the tool by students.

5.1 Practical Session

This practical session focus on the analysis and evaluation of the mapping schemas proposed by students. First of all, students analyze a given case study and with the aid of MapO2R obtains the different mapping schemas. Then, students select the databases where they want to apply the mappings. Results obtained allow the validation of the students mapping proposals.

Figure 1 shows the different GUIs that allow the configuration of the benchmark: automatic or manually execution of the benchmark, automatic benchmark settings, number of registers to be evaluated and database to be used, respectively. As it can be appreciated in Figure 1 (upper right corner), the student can select the mapping related to each association, aggregation or inheritance operation, and even select the kind of SQL query (select, update, insert and delete).

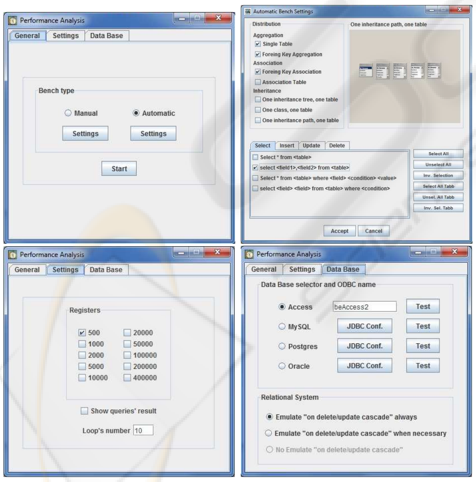

Figure 1: Benchmark configuration.

We present the results obtained for a basic case study, where aggregation, inheritance and association relationships are evaluated over a MySQL database. The practical sessions take place in a laboratory equipped with 25 PCs with Windows XP (SP2), where 20 of them are used by students (with MapO2R) and the rest host the different databases.

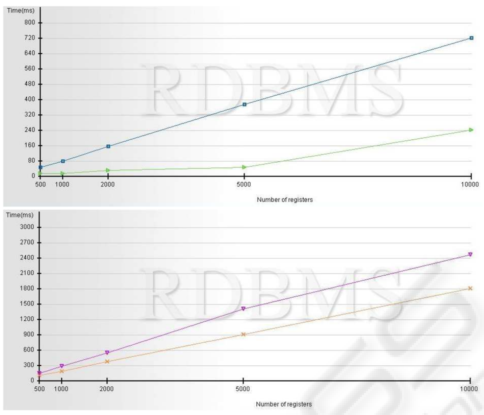

Figure 2: Select queries for the aggregation schema, following a certain criterion (up) or not (down).

Aggregation: Figure 2 presents the time spent when performing select queries following a certain criterion or not. Select queries performed following a certain criterion have a lower cost than recovering all the objects (all the rows of the table), and the use of the foreign key table strategy reduces notably the time spent when looking for a certain field only stored in one of the tables. When the query is performed over fields contained in both tables, the use of a single table performs better than de foreign key strategy. The queries involved in the comparison are:

*SELECT * FROM Clients SELECT Clients.ClientID, Clients.Surname, Clients.Name, Clients.TotalAmmount, Adress.Street, Adress.ZIPCode, Adress.City FROM Address INNER JOIN Clients ON Address.AdressID = Clients.BillAdress*

The cost of inserting new objects following a certain criterion is higher than the simple insertion (without criteria), as it can be seen in Figure 3. The queries involved in the comparison are:

INSERT INTO Clients VALUES ('123456789','Manuel','Ruiz Lopez', 1500, 'Main street','Madrid','28115','Oak','Guadalajara','19030')

INSERT INTO Address VALUES ('123456789','Main street', 'Madrid','28115')

INSERT INTO Address VALUES ('987654321','Oak', 'Guadalajara', '19030')

INSERT INTO Clients VALUES ('123456789','Manuel', 'Ruiz Lopez', 1500, '123456789', '987654321')

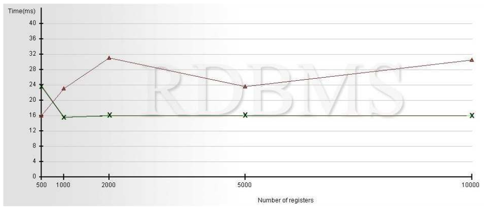

Figure 3: Insert queries for the aggregation schema.

Figure 4 shows that the deletion cost is lower when using the foreign key aggregation. It is interesting to remark that the deletion of certain rows following a certain criterion has a higher cost than the deletion of all the table.

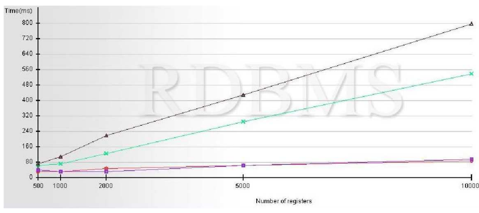

Figure 4: Delete queries for the aggregation schema.

Inheritance: Figure 5 presents the median time spent in select queries for the three inheritance mappings proposed. Symbols *x* and ▲ (lowest one) represent the values obtained for a unique table inheritance mapping, ♢ and + for the inheritance tree mapping, and ○ and ▲ for the inheritance path mapping. Higher costs are related to the path strategy, followed by the tree strategy. The use of a unique table has the lowest cost.

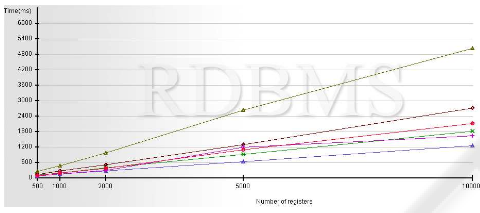

Figure 5: Select queries for the inheritance schema.

As it can be seen in figure 6, the time spent in insert operations does not depend on the number of registers (objects), and is almost constant. On this figure, the lowest curve represents the cost of the insertion when using a unique table strategy. So the path and tree strategies have a higher cost in terms of time than the unique table strategy.

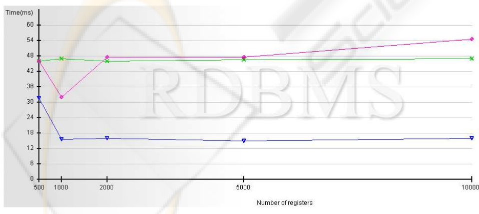

Figure 6: Insert queries for the inheritance schema.

Figure 7 presents the median time spent in delete queries for different inheritance mappings. On this figure, symbols ● and □ represent the values obtained for a unique table inheritance mapping, ▲ and *x* for the inheritance tree mapping, and ▼ and ♢ for the inheritance path mapping. For each case, the upper value corresponds to the deletion of all the registers of a table (all the objects in the object oriented domain), and the lower one corresponds to the deletion of certain registers-objects that verifies a certain criterion. As it can be observed, deletion costs (measured in time terms) increase linearly with the number of registers (objects) managed, although the tangents are very different.

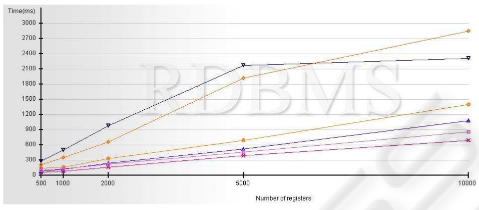

Figure 7: Delete queries for the tree inheritance mappings.

Association: Figure 8 (up) shows the linear cost of select queries, where the foreign key strategy performs better than the association table strategy. The deletion of a certain object has a higher cost when we follow a certain criterion for the table association strategy when the field to compare is located in the association table. The use of the foreign key strategy is faster than the use of an association table. Figure 8 (down) shows that differences between using both strategies is not relevant when performing insertions.

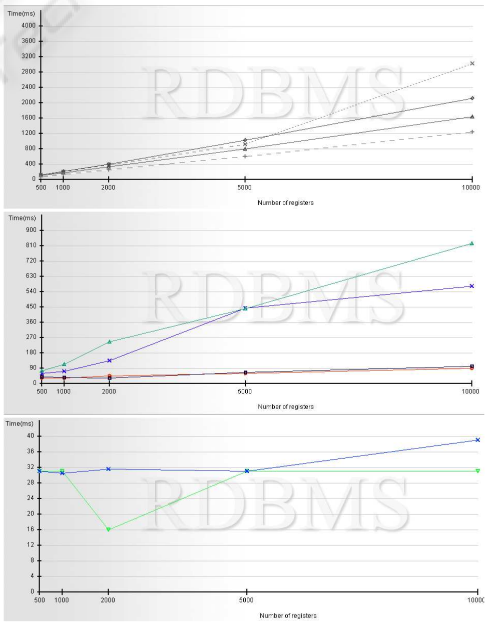

Figure 8: Select, delete and insert queries (respectively) for the association schema.

5.2 MapO2R Evaluation

We have performed some questionnaires to students in order to validate the accuracy of MapO2R. Students are asked for the goodness and usefulness degree of the tool MapO2R in their learning process.

The questionnaire consists on 20 questions concerning many aspects of the tool and the mapping learning. Students are ask about their opinion about the use of the MapO2R tool (interactivity, ease of use, user-friendly...) and about the mapping learning using MapO2R (does MapO2R help to understand the different mapping strategies?). Each question is rated in the interval [0,20], where 0 represents a total disagreement and 20 a total agreement.

Figure 9 shows the overall assessment of the students to the MapO2R tool. The 87% of the students consider MapO2R a useful tool to learn the object-relational mapping.

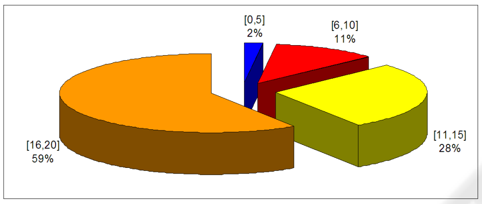

Figure 9: Overall assessment to MapO2r.

As Figure 10 shows, a 93% of the students consider that MapO2R is a good tool for the learning of the object-relational mapping. The 91% (49/5) of the students declare they would use again the tool.

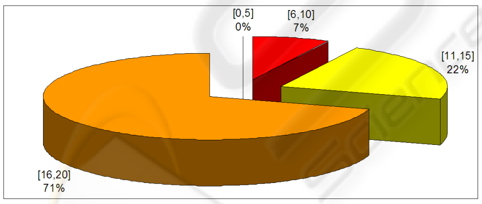

Figure 10: Learning degree for the object-relational mapping using MapO2R.

Figure 11 shows that the inheritance is more difficult to understand instead the aggregation or the association. Students consider the inheritance mappings more complex and hard to implement than association and aggregation. The 33% of the students consider the inheritance very complex and 46% consider it complex. So, the 79% of the students find the inheritance difficult and complex, while only 81% and 64% of the students find the association and the aggregation (respectively) easy and simple.

The analysis of the questionnaires shows that MapO2R is suitable for learning object-relational mappings. Due to its ease of use, suitable graphical interface, the ability to play practical scenarios and the ability to import and export the results, students consider MapO2R a valid learning tool.

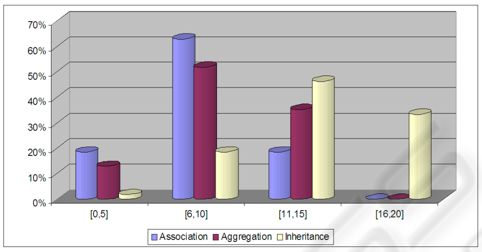

Figure 11: Learning complexity degree.

6 CONCLUSIONS

This paper presents an open education tool, MapO2R, which enables students to acquire best practices on object-relational mapping. Students can evaluate experimentally different mapping strategies, identifying the most adequate for each proposed scenario. MapO2R implements the mapping language defined by Keller (Keller, 1997). MapO2R has been evaluated favorably by the students that have used it.

ACKNOWLEDGEMENTS

Research partially supported by the Spanish Research Council under research grant TIN2008-03687.

REFERENCES

Barry, D. K. (2005). Transparent persistence in object-relational mapping. http://www.service-architecture.com/object-relational-mapping/articles/transparent_persistence.html.

Keller, W. (1997). Mapping objects to tables - a pattern language. In *Proc. Of European Conference on Pattern Languages of Programming Conference (EuroPLOP)97*.

Objectmatter (2005). Object relational mapping strategies, http://www.objectmatter.com/vbsf/docs/maptool/ormapping.html.