Hibernate as an object-relational mapping framework for eHealth systems

Alberto Moreno Jiménez, Elena Villalba Mora, M.T. Arredondo
Life Supporting Technologies
ETSI Telecomunicación
Universidad Politécnica de Madrid
amoreno@lst.tfo.upm.es

Abstract

In 1969 Edgar Codd invented the relational data model, a new and original software paradigm about information retrieval methodologies. This model has helped to simplify software applications for almost 30 years, finding no changes to the main concept, whatsoever. Many years have passed since then and new application programmers want tools to deal with the relational model. The object oriented programming technologies[1] are now available in all new age programming languages such as Java, C#, or ADA, and programmers prefer to design their applications in terms of objects, inheritance and polymorphism, rather than sets, properties and the relationships of a database model.

This paper presents the approach defined by Hibernate[2] for solving the matching between objects and relational database..

1. Introduction

eHealth systems need to deal with large amounts of information. When an application need to store complex data about patients, professionals, treatments and diseases, the database access becomes a conceptual bottle-neck, source of errors and performance leaks. For this reason, these applications require reliable and efficient ways to store information in databases.

Information complexity affects applications in two ways, each one concerning a different development stage. On the one hand, information can be extensive, which means that the database is large and contains a voluminous amount of registers. This situation affects an application during the system execution. It is important to avoid unnecessary accesses to the database, that is to say, the necessity of a cache system to optimize the performance. On the other hand, the information can be complex because of confusing relationships among data entities. This problem appears during the application main development. Complexity has a big impact in software reliability, thus abstract access to the information is also an important requirement.

Since the beginning of software development, data storage has been an important issue with different solutions. The first approach taken was to use small text or binary files to store the information. In this case, the storage format was proprietary. Therefore, no other application could use these files to access the information. Furthermore, this solution was easy to implement when data to store was simple. However, as soon as information relationships began to grow, the problem turned very complex to solve and the reliability of the systems decreased.

In 1969, Edward Code proposed the relational model [3] as a new paradigm to store and retrieve information in a software system. The relational model is based on the mathematical set theory, which is simple to learn and use. Research groups studied the model until, in 1979, the software company Relational Software, Inc produced Oracle, the first commercially available SQL database server. Though the main concept was a success, the relational databases are based on text queries that also become complex when the data model has a large number of relationships.

Modern applications are based on objects, while relational databases are based on sets [4]. There is a deep gap between the actual application programming languages, based on the object orientation paradigm, and the relational query language. To save this gap, and to increase the reliability in software systems, a middleware that can manage these differences and deal with the complexity of traducing from one language to the other is required. This paper proposes Hibernate framework as a bridge from the object oriented conceptualization to relational databases.

Figure 1: Hibernate Logo

Hibernate is a high performance object to relational persistence and query service for the Java language, initially developed by SourceForge and now managed from the JBoss group. Hibernate is a middleware layer, localized between the database and the business model of an application, which transforms actions performed over object oriented classes into relational database query commands. Hibernate simplifies the access to the database tables, also speeding up the access performance using cache systems to store frequent information. The user experience is enhanced both in access speed to the information, and in the reliability of the application.

2. Methodology

An application has a business model, which is formed by the set of classes that implement the concepts and entities of the designed system. This model has two kinds of classes, volatile and persistent. Volatile classes are those that create objects that are only active at a certain moment of the execution, never needing a durable storage system. Persistent classes, on the other hand, create objects that are stored and retrieved from the application into a database, or any other storage system. Hibernate works on persistent classes using two sources of information: the properties file, and the mapping XML documents. The properties file holds information about the database source, and certain performance parameters. The mapping XML documents are the class managers, maintaining the information about relational table relationships and class attributes storage.

A persistent class instance presents one of the following three states, in relation to the persistency context it is in:

- Instances that have been stored or retrieved from the database during a single session are in the *persistent state*. These objects have a primary key assigned and they are usually stored in a row of the database.
- *Transient instances* are those that have never been in the persistent state. These objects have never been in the database, so they have no Hibernate identification.
- Instances that have been persistent over a session that is already inactive, or that have been serialized to another process, are said to be in the *detached state*. These objects have an Hibernate identification but need to be reattached again to a session before they can be used again.

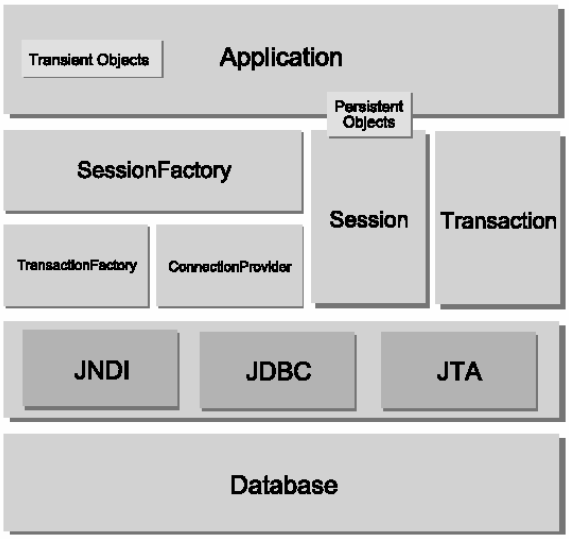

Figure 2: Hibernate architecture overview

As shown in figure 2, the Hibernate intermediate layer, located between the database and the application, is accessed through sessions and transactions. A session is a conceptualization of a database connection, wrapping the real JDBC Bridge inside of it. The session stores the information that travels back and forwards the database, in order to provide a first level cache of data. The transaction provides the implementation of a database information exchange, so a query can commit and rollback operations depending on intermediate results, among other possible events.

Sessions and transactions are created using the proper factories, provided by the Hibernate middleware. A session factory maintains references to several compiled session instances, which may provide a second level cache. This cache may be useful for multiprocessing applications, or clustered architectures, to increase the first level cache performance.

There are several tools to facilitate the use and integration of Hibernate in software applications:

- **Middlegen**: general purpose code generator, based on database structure information. Middlegen can produce classes for EJB, JDO and Hibernate, as well as interface generation for JSP and Struts. Middlegen also provide an Eclipse plug-in for better desktop integration.

- **SchemaExport**: this tool can provide a DDL database creation script based on Hibernate XML mapping files. Using SchemaExport is possible to define a class hierarchy, and generate a data model which will store all data.
- **Ant**: popular tool for Java development. It simplifies the creation, maintenance and implementation of software systems, using only portable and extensible Java classes and XML configuration files.
- **Log4Java**: debugging trace processor, which keeps track of the application steps. Hibernate uses Log4Java to export debugging information and is used to visualize queries transformations, as well as to find possible errors.

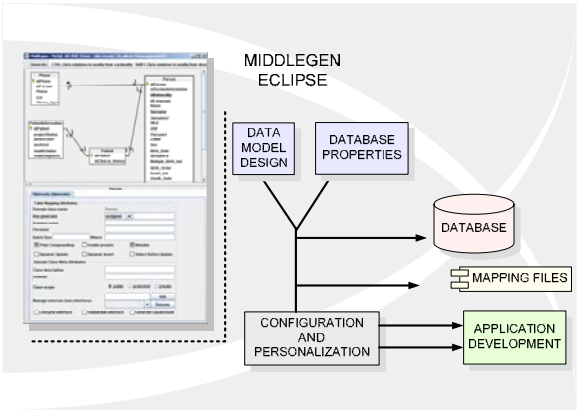

Figure 3: Eclipse Hibernate development framework,

These tools provide a flexible environment to build Hibernate based applications. A programmer may define the data model of a software application and generate, using the Middlegen Eclipse plug-in, the XML mapping files and the properties database files. Afterwards, the programmer programs the application using the Hibernate interface to the database.

3. Results

Hibernate provokes an outstanding effect on an application development. The first remarkable characteristic consists of a more natural object oriented programming model. The need for string query construction, based on variable parameters, is no longer needed. Instead, direct instance storage and retrieval is enabled through the Hibernate interface. Classes only need an XML mapping file to indicate the place to store its properties, and the relationships to other entities. All the basic relationships are covered under Hibernate: inheritance, composition and aggregation are used straight forward. For example, inheritance can be focused using different approaches: all hierarchical information in the same database table, or data distributed in different tables per class or per inheritance level. Associations can be implemented using the Java Collections Interface, and can be stored in different tables just like in any other database application. Hibernate allows a lot of flexibility in the XML mapping files to store the business model into the database: table names, relationship identifiers and event management can be specified freely.

Hibernate provides several access points to the information in the database. These access points implement different ways of understanding the nature of data retrieval. The following table sketches these approaches:

HQL	
Properties	The Hibernate Query Language (HQL) provides a query a-like access to data. The main difference with SQL resides in the referred entities: SQL work with tables, HQL work with objects. Thus, programming with HQL is a fast and natural way to work with Hibernate, using the concepts learnt from SQL, but with the ease and power provided by the business model.
QBC	
Properties	An application can access the database avoiding any kind of string based query using Query By Criteria language. It allows the programmer to choose the needed class, and set certain properties to specify the search conditions. This approach is more pragmatic than HQL, yet it is less powerful and intuitive for a SQL expert programmer.
QBE	
Properties	Query By Example (QBE) is similar in concept to QBC. The purpose is to supply Hibernate with a similar object to the one the programmer is looking for. Hibernate will return all related objects that it finds in the database.

Table 1: Hibernate SELECT approaches

Translating Hibernate high level commands to relational SQL queries is a complex operation. Hibernate takes into account the database structure to reduce the complexity of relational table joining, thus diminishing the use of system resources. Even though it is possible for a person to write more efficient

queries to the database, Hibernate performs this action conveniently, even with high complexity level requests.
Hibernate based applications are faster than those programmed directly using JDBC. The reason for this is the use of data caches, which saves unnecessary accesses to previously stored information. The use of caches can be the source of synchronization problems with other systems. This may force the application programmer to choose between high Hibernate performance, or interoperability with other non-Hibernate applications. If multiple Hibernate applications are running over the same database, a second level session cache can be used to increase speed, maintaining interoperability.

4. Conclusions

Hibernate provides the application programmer with a fast access to the information stored in a database. Moreover, the programming model that it suggests is more natural and intuitive than the relational SQL. The programmers no longer need to deal with the complex issue of database abstraction, and obtain a simple and easy object oriented access to the information. The application obtained is more reliable and it accesses to the stored data in a faster way.
Furthermore, Hibernate provides several tools that facilitate the integration with current systems. These tools can generate the mayor configuration needs of Hibernate, so the programmer only personalizes the application.
These characteristics have promoted the use of Hibernate in the Java environment. Hibernate is the main data access engine of the Open JBoss Application server, and its model has been closely followed during the definition of the Enterprise Java Beans (EJB) standard. Hibernate has gone beyond the Java Platform with the creation of Nhibernate, which is a persistence and query service for the Microsoft .NET Framework.

5. Acknowledgment

Hibernate is a professional product provided by the JBoss group to build expert enterprise applications. It has been used in the Heart Failure Management System, part of the MyHeart project, “Fighting Cardiovascular Diseases by prevention and early diagnosis” (IST-2002-507816). MyHeart is a 6th Framework Project of the IST Programme, partly funded by the European Commission.

6. Bibliography

[1] “Design patterns : elements of reusable object-oriented software". Erich Gamma, Richard Helm, Ralph Johnson, John Vissides . Publisher: Addison Wesley. ISBN: 0201633612.
[2] “Hibernate Reference Documentation”. Version 3.1.2.
[3] "Transaction Processing: Concepts and Techniques ". Jim Gray, Andreas Reuter. Publisher: Morgan Kaufmann Publishers Inc,US . ISBN: 1558601902
[4] "Advanced Database Systems". Carlo Zaniolo, Stefano Ceri, Christos Faloutsos, Richard T. Snodgrass, V.S. Subrahmanian, Roberto Zicari. Publisher: Morgan Kaufmann Publishers Inc,US . ISBN: 155860443X

Address for correspondence

Alberto Moreno Jiménez
Life Supporting Technologies
ETSI Telecomunicación. D-204
Ciudad Universitaria s/n.
Madrid 28040.
e-mail: amoreno@lst.tfo.upm.es
Tel. +34915495700 ext 3407 / Fax: +34913366828