A Review on Different Object Relational Mapping Frameworks

[1]Vinay N R, [2]Soumya Sharma, [3]Layeeq Ahmed, [4]Soumya A, [5]Rajashree Shettar
[1, 2, 3]Undergraduate Student, [4-5]Professor of Department
[1-5]Computer Science and Engineering,
[1-5]R.V. College of Engineering, Bengaluru, India

Abstract: Object-relational mapping (ORM) has facilitated a way of establishing a connection between the object-oriented architecture and the relational database architecture. It has the benefit of querying and manipulating data that is primarily stored in databases using an object-oriented paradigm to provide a solution to the problem of impedance mismatch. This technique is widely applied, and in this paper, we highlight the importance of it. Various models of ORM exist, each with its set of unique features catering to different requirements. Few of these major models are explained in this paper. The benchmarking criteria established to judge the performance of each model so that the right model can be applied at the right time, is also explained.

Index Terms **- Persistence, Relational Mapping, Database**

1.Introduction

Object Oriented Programming has gained huge popularity among developers and architects to provide solutions to a business problem. Not only has this approach impeded the adoption of generation-old procedural programming paradigm, but has also manifested itself as the best design pattern for application development. Regardless of the criticism it faced due to emphasizing more on data-relationship modeling over computations and algorithms, it has stood out as the best alternative for a developer.

Class is the encapsulation of data and its methods in an OOP environment and objects are instances of these classes. Mapping of the real world entities with classes is the primary objective of a software architect. Leveraging on several of OOP features such as inheritance and polymorphism, an architect establishes the groundwork for developers to proceed with implementation.

In a typical enterprise application or an enterprise information system (EIS) which is distributed over the internet or the intranet, data persistence is an issue addressed by every developer. This is generally provided by database systems. Storing objects directly in the database is an option, however this is curtailed by the dependency on existing legacy systems which are relational in nature. Relational databases gained the momentum over the object-oriented databases and lead to the birth of Object Relational Mapping (ORM) frameworks.

The problem of impedance mismatch is widely discussed in the context of object-relational mapping. Objects are not stored in their native format in relational databases, hence there occurs several conceptual and technical difficulties while the data is served to an application. The relational model enforces stringent rules on data such as no by-reference attributes, and also does not allow composition of tables. This is not the case of object-oriented languages.

In this paper, the authors will address the problem of impedance mismatch, and how object relational mapping frameworks solve the problem of data transfer between the two interdependent systems. Some widely used frameworks are discussed along with their benchmarks for comparison. The authors lay special emphasis to two of the many frameworks - Hibernate and MyBatis.

2.MOTIVATION

In most cases, data is invariably stored in Relational Database Management Systems (RDBMS). However, in order to manipulate data in these tables, complex queries and repetitive Create, Read, Update and Delete (CRUD) operations have to be applied. The core idea behind implementing ORM is that it shields the coder from writing lengthy queries and at the same time use the benefit of object-oriented programming by working on objects instead. The coder can continue coding in the language chosen for coding.

Database abstraction is one of the key advantages of implementing ORMs. Due to various reasons if the database changes, it is a lot easier to adapt to the change if ORMs are used by providing a static code base for the application. It allows the developer to work on the object model solely, as the ORM tools translate it into the appropriate syntax of the RDMS in use.

Due to the availability of the host of services provided by the various ORM tools, a shift of focus from writing queries to focusing on the business logic can be made. A domain model pattern can be followed which provides the scope to deal with the project with a business logic. Also, a lot of advanced features such as transactions, streams and migration can be supported by incorporating the use of ORMs. At the same time, as entities are cached in the memory, it considerably reduces the load on the database.

Enterprise applications, as mentioned in the previous section, are large-scale applications with the purpose of solving enterprise problems. Here, multiple users use the same application concurrently. They deal with large quantities of data which requires to be persistent. They are developed in object-oriented architectures while data is stored primarily in relational databases. ORMs find a vast range of applicability here, as they bridge this gap.

Thus, with the wide range of benefits and applicability, ORMs have grown in popularity and continue to be used and applied extensively.

3.BENCHMARKS FOR ORM COMPARISON

Java is one of the most popular languages for application development today mainly because of its cross platform compatibility feature. The Java ecosystem offers several ORM frameworks and choosing the right one will be dependent on the application’s technology constraints. Multiple factors such as database supportability, integration with an IOC container such as Spring come into effect while deciding the framework to adopt. In this section, the authors attempt to establish a benchmark for analyzing different ORM frameworks and allude on the comparison criteria used. It is imperative that developers evaluate frameworks additionally on the features available, documentation, speed and community support.

3.1 Architecture

This section deals with the underlying architectural features of the framework. Every framework provides a set of APIs to aid the developer during implementation, often eliminating the need of boilerplate code. The authors shall streamline their investigation on how structures from Domain model can be mapped to the schemas of the relational databases. A number of techniques are present such as hierarchical mapping, each class mapped to a table, or only a concrete class mapped to a table of a database. However hierarchical mapping summons redundancy in data and is often a sparse schema. Achieving the third normal form is necessary to gain the best performance and hence only certain mapping techniques are ubiquitous in the industry today.

The association between the domain model and the mapper can be realized using two methods- Reflection and Code Generation. Reflection suffers from performance and also poses many challenges to developers while debugging. Code Generation is more flexible from a developer’s perspective and provides a configurational file mechanism to create the mappings. With this approach, the domain model is independent of the relational model.

3.2 Benchmarking for ORM Frameworks

In this section, the authors discuss the different criteria to compare and contrast the ORM Frameworks. Although this section provides a holistic view on the frameworks and its performance evaluation, many of its implementation details are disregarded to keep the overview succinct. The authors provide the following benchmarking criteria to evaluate the ORM frameworks:

3.2.1 Ease of Development

Most business applications must be delivered to the clients as early as possible. The market today is highly competitive and it is fairly common for an application to be disregarded due to its failure to meet deadlines. Hence ORM frameworks must enable a developer to deal with the persistence logic quite effortlessly. The inclusion of boilerplate codes only makes it abstruse from the perspective of the developer.

3.2.2 Documentation

Documentation is a very crucial criteria to evaluate a software package. Documentation must be rich and coherent as this expedites the process of software delivery. Seeking a solution to a common problem must not be cumbersome and should be easily available on the internet. An open source software has an added benefit that it is widely supported from all professionals throughout the world.

3.2.3 Cross-Platform Compatibility and Dependencies

If development is one side of the coin, deployment is another side. Much emphasis should be given to making the application run everywhere. One of the reasons for employing Java is because it is converted to a byte-code format which is supported on all operating systems.
ORM frameworks must also support a plethora of database servers and must integrate with any Inversion of Control (IOC) framework such as Spring. Hibernate and MyBatis are two open sourced frameworks that stands out in these aspects.

3.2.4 Usability

Many frameworks provide a rich set of APIs to make the life of a developer easy. However, only a subset of the entire functionality of the package is utilized. Therefore, it is crucial that the framework is usable by any medium-skilled developer and does not take much time for development. There exists a tradeoff between the feature set provided by the framework and the comfort level of developers in employing the framework for development.

4. OVERVIEW ON DIFFERENT ORMs

4.1 Hibernate

Hibernate is a popular Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence. Hibernate is free software that is distributed under the GNU Lesser General Public License 2.1 and is well documented. It is widely-spread among the community and thus many tutorials and articles about various aspects of using of this framework exist.

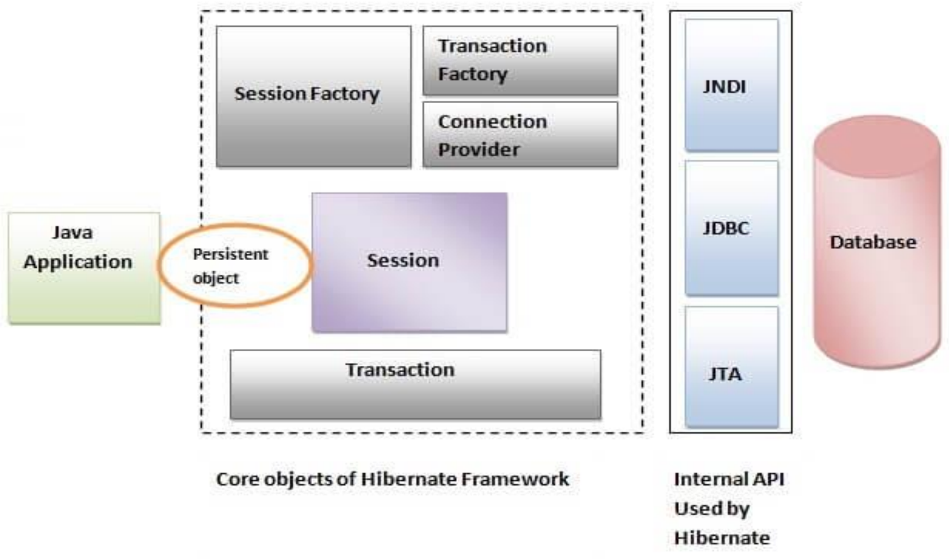

Figure 1: Architecture of Hibernate

The Hibernate architecture is categorized in four layers, which are Java application layer, Hibernate framework layer, Backhand API layer, Database layer and is illustrated in Figure 1.. Hibernate has a layered architecture which helps the user to operate without having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application. It includes many objects such as persistent object, session factory, transaction factory, connection factory, session and transaction.

Hibernate uses XML configuration files for storing information about mapping. Mapping can be optionally configured through a Java. properties file. Usually, a rule of one configuration file for one class is followed. This makes it easier to maintain and is easier to find errors in configuration when using this approach. The class configurations include <hibernate-mapping> parent tag, <class> tag to map Java class to a table , <id> element maps the unique ID attribute in class to the primary key of the database table, <generator> element within the id element is used to generate the primary key values automatically, <property> to map the various properties of the class to table columns.

Hibernate support by Java annotations has removed XML mapping for classes. Some of the basic annotations are discussed. *@Entity* annotation marks this class as an entity. *@Table* annotation specifies the table name where data of this entity is to be persisted. If you don't use *@Table* annotation, hibernate will use the class name as the table name by default. *@Id* annotation marks the identifier for this entity. *@Column* annotation specifies the details of the column for this property or field. This makes the mapping integrated with the code, makes it simple to understand, debug or perform changes to it.

Hibernate also supports lots of variants of association mapping. It supports one-to-many, many-to-one, as well as many-to-many (association table) associations. Hibernate has very rich support for lazy processing. It is easy to configure in XML mapping file which attribute or which association has to be processed as lazy. Usually XML element corresponding to class of association has boolean attribute named lazy.

Some advantages of Hibernate framework are listed below:

1) Open Source and Lightweight - Hibernate framework is open source under the LGPL license and lightweight.

2) Fast Performance -The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

3) Database Independent Query - HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So, you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

4) Automatic Table Creation - Hibernate framework provides the facility to create the tables of the database automatically. So, there is no need to create tables in the database manually.

5) Simplifies Complex Join - Fetching data from multiple tables is easy in hibernate framework.

6) Provides Query Statistics and Database Status - Hibernate supports Query cache and provide statistics about query and database status.

4.2 MyBatis

MyBatis is an open source, lightweight, persistence framework that couples objects with stored procedures or SQL statements using an XML descriptor or annotations. MyBatis is a free software that is distributed under the Apache License 2.0. It automates the mapping between SQL databases and objects in Java, .NET, and Ruby on Rails. The mappings are decoupled from the application logic by having the SQL statements in XML configuration files.

It abstracts almost all of the JDBC code, and reduces the burden of setting of parameters manually and retrieving the results. It provides a simple API to interact with the database. It also provides support for custom SQL, stored procedures and advanced mappings.

A significant difference between MyBatis and other persistence frameworks is that MyBatis emphasizes the use of SQL, while other frameworks such as Hibernate typically uses a custom query language i.e. the Hibernate Query Language (HQL) or Enterprise JavaBeans Query Language (EJB QL).

Mapper XML is an important file in MyBatis, which contains a set of statements to configure various SQL statements such as select, insert, update, and delete. These statements are known as Mapped Statements or Mapped SQL Statements. All the statements have unique id. To execute any of these statements you just need to pass the appropriate id to the methods in the Java Application.

Mapper XML file prevents the burden of writing SQL statements repeatedly in the application. In comparison to JDBC, almost 95% of the code is reduced using Mapper XML file in MyBatis. All these Mapped SQL statements are resided within the element named <mapper>. This element contains an attribute called ‘namespace’. The element <resultmap> is used to define a standard mapping for SQL select queries and allows multiple select queries to utilise a single result map, especially when it is for the same object.

MYBATIS offers the following advantages:

1) Supports stored procedures − MyBatis encapsulates SQL in the form of stored procedures so that business logic can be kept out of the database, and the application is more portable and easier to deploy and test.
2) Supports inline SQL − No pre-compiler is needed, and you can have the full access to all of the features of SQL.
3) Supports dynamic SQL − MyBatis provides features for dynamic building SQL queries based on parameters.
4) ORM features − MyBatis supports many features such as lazy loading, join fetching, caching, runtime code generation, and inheritance.

4.3 Java Data Objects (JDO)

The Java Data Objects (JDO) specification provides for interface-based definitions of data stores and transactions; and selection and transformation of persistent storage data into native Java programming language objects. The two primary goals of the specification are to provide an API for transparent data access and to allow implementations of the specification to be plugged into application servers. JDO is standard that enables objects to persist not just to relational databases, but to XML or file system as well. Many third-party vendors implement this specification and sell it, such as SolarMetric Kodo JDO and Signsoft intelliBO. TJDO is an open source implementation of JDO standard.

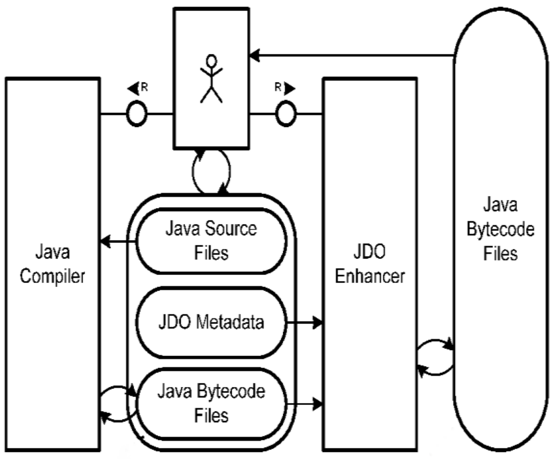

Figure 2: Architecture of JDO

In order to have persistent classes, it has to implement an interface called PersistenceCapable. It is interface defined by JDO standard. This interface contains methods which helps JDO implementation to manage fields of persistent object. Methods of this interface can be very complex and are not expected to be implemented by the developer. Usually, a bytecode enhancer tool is used to adjust the byte code to implement PersistenceCapable interface. Figure 2 describes this process. The byte code enhancer is provided by JDO implementation vendor. During enhancement of byte code it uses some metadata supplied by the developer. Usually line numbers of source code and similar details are preserved, so there is no problem to debug enhanced code. Requirement of JDO specification is that code enhanced by different JDO implementation vendor have to be binary compatible, making it portable and independent of one concrete implementation.

JDO Standard defines set of states that class can be in as well as possible life cycles of an object. States defined by standard are Transient, Persistent-new, Persistent-dirty, Hollow, Persistent-clean, Persistent-deleted and Persistent-new-deleted. Note that some JDO implementations augment this set of states by new states.

JDO bytecode enhancer approach signifies that JDO approach is closer to (byte) code generation than to reflection. It can be seen that any JDO implementation is similar to frameworks like Hibernate when supported scenarios are considered. There is Domain model, database model and configuration file. It is a choice of concrete vendor and corresponding JDO implementation whether tools for generation of database schema from Domain model, generation of Domain model from database schema are provided.

Advantages of JDO for Application Programming are listed below:

1) Ease of use: Application programmers can focus on their domain object model and leave the details of persistence (field-by-field storage of objects) to the JDO implementation.
2) Portability: Applications written with the JDO API can be run on multiple implementations without recompiling or changing source code. Metadata, which describes persistence behaviour external to the Java source code including most commonly used features of O/R mapping, is highly portable.
3) Database independence: Applications written with the JDO API are independent of the underlying database. JDO implementations support many different kinds of transactional data stores, including relational and object databases, XML, flat files, and others.
4) High performance: Application programmers delegate the details of persistence to the JDO implementation, which can optimize data access patterns for optimal performance.
5) Integration with EJB: Applications can take advantage of EJB features such as remote message processing, automatic distributed transaction coordination, and security, using the same domain object models throughout the enterprise.

5. CONCLUSION

In this paper, the need for ORMs and their importance was discussed in detail. The various advantages it imposes over the existing systems were illustrated. Reduction in code, database abstraction, rich query ability, concurrency support, cache management and transaction management were the main advantages of implementing ORM.

The main underlying architecture of an ORM framework was discussed and how it achieves the task of bridging the gap between the object-oriented environment and relational database environment. Following this, a set of benchmarking criteria was outlined, to establish the criteria to judge the performance of a given ORM model.

The first type of framework is where the database schema is generated from the domain model (DataObjects.NET). There are models where the domain model is given by the database schema (LLBLGen Pro). Flexible frameworks like Hibernate which allow both the above properties, was discussed in the previous section. Frameworks manipulate business objects in mainly three ways- runtime reflection (DataObjects.NET, Hibernate), code generation (LLBLGen Pro) or byte code enhancement (Hibernate, JDO). Through attributes in the source code of the domain model and configuration files, mappings can be configured most of the times. However, some models like the LLBLGen Pro, require mappings to be configured only through the use of specialized IDE.

In table 1 is a comparison study of the three frameworks dealt with against the benchmarking criteria detailed in the paper. This gives an opportunity to decide which framework to use at any time according to the requirements of the project.

	Hibernate	MyBatis	Java Data Objects
Ease of Development	High	High	High
Documentation	High	Medium	High
Cross-Platform Compatibility and Dependencies	High	Medium	High
Usability	High	High	Medium

Table 1: A comparison of ORM models

Whether a robust framework offering multiple features, which might be difficult to use is required or a simple framework, but with a high adaptability is required, can be decided based on the purpose of the project being developed. Different models cater to different needs. Thus, ORMs have found a wide range of applicability and usage in today's world.

REFERENCES

[1] Pieter Van Zyl, Derrick G Kourie and Andrew Boake, "Comparing the Performance of Object Databases and ORM Tools", Department of Computer Science, University of Pretoria, Proceedings of SAICSIT, 2016.

[2] Jaroslav Orsag, "Object Relational Mapping", A Diploma Thesis, Department of Computer Science, Comenius University, Bratislava, 2016.

[3] Derek Corelly, Clare Stanier, Md Asaduzzaman, "The Impact of Object-Relational Mapping Frameworks on Relational Query Performance", ISBN:978-1-5386-4904, IEEE 2018

[4] J.N. Cappella, "Vectors into the future of mass and interpersonal communication research: Big data, social media, and computational social science". Human Communication Research, vol. 43, issue 4, pp.545-558, 2017.

[5] M. Stonebraker, G. Held, E. Wong and P. Kreps, "The design and implementation of INGRES". ACM Transactions on Database Systems (ToDS), vol. 1, issue 3, pp.189-222, 1976.

[6] S. Balaji and M.S. Murugaiyan, "Waterfall vs. V-Model vs. Agile: A comparative study on SDLC". International Journal of Information Technology and Business Management, vol. 2, issue 1, pp. 26-30, 2012.

[7] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries and J. Kern, "Manifesto for agile software development.". Available online at http://agilemanifesto.org (Accessed May 2018), 2001.

[8] A. Miller, “A hundred days of continuous integration”. AGILE '08 Conference, pp. 289-293, Aug. 2008.

[9] S.W. Ambler, Mapping objects to relational databases: What you need to know and why. Ronin International, 2000.

[10] A. Cheung, S. Maddenand A. Solar-Lezama, “Sloth: Being lazy is a virtue (when issuing database queries)”. ACM Transactions on Database Systems (ToDS), vol. 41, issue 2, p.8, 2016.