Volume 4, Issue 7, July 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Analysing the Efficiency of Tools for Object Relational Mapping

Brinda Ramanujam
Assistant Prof.Department of Computer Science
M.O.P Vaishnav college for women,
Chennai, India

Dr. Ananthi Sheshasaayee
Associate Professor and Head,
PG and ResearchDepartment of computer science
Quaid-E-Millath College for Women, Chennai, India

Abstract*— Relational databases are good to store data; object-oriented programming is well suited for building complex applications. Performing object/relational mapping, it is possible to create a layer that can easily transform objects into relational data and vice versa. The two technologies, although divergent, perform interaction both theoretically and practically. Object relational mapping (ORM) is a technology that connects relationships with object-oriented entities. This mapping eliminates duplicate layers together with maintenance costs and any errors arising from their existence. ORM is the automated persistence of objects in an application to the tables in a relational database, using metadata that describes the mapping between the objects and the database. Object relational mapping, in essence, works by transforming data from one representation to another The paper aims to automate the duplication of class and table structure and achieve mapping between object-oriented world and the relational world automatically, to study existing ORM tools, compare the existing Object relational mapping tool's performance for the Creation, Updation and Deletion of Objects in a Database.

***Keywords*— Relational databases, Object Oriented Databases, Object Relational Mapping, ORM tools**

I. INTRODUCTION

Object oriented programming has become popular today. C++, Java, and .NET platforms are some of the most recognized object oriented programming environments. The idea to build systems with interacting objects in system modeling and software development is simple and powerful. Looking at database technology, the degree of object orientation is quite small. There are various reasons, one of which is that the approach to represent data as sets of related tuples (which is called "relational data") has a long and successful history in information technology. The associations used are simple. Relational database management systems (RDMBSs) are today's most popular and widely used database systems. As requirements on applications increase, the interface area of these two worlds gains importance. In Object Oriented Programming, typically the behaviour of objects is emphasized. On the other hand, it is the data that is given importance in database technology. This fact serves as a common motive for the combination of these two paradigms. The core component of this coupling is what is called "Object-Relational Mapping" which takes care of the transitions of data and associations fromone paradigm into the other (and vice versa)

II. RELATED WORK

The author of **[13]** discusses the importance of Impedance mismatch among objects and how **Persistence Modeling Process Pattern is Important** for effectively modeling the persistence needs of the application. Data models only take into account half of the picture (data) whereas object-oriented models take into account the entire picture (data and behaviour). By using object oriented models to drive the development of data models it can be ensured that the database schema will actually support the application needs

A. Object persistence and Legacy Integration

The unique ability to not only store data, but also to decide how, where, and on what platform it is stored **[10]** is important. In most cases, this choice is independent of the application design.

B. Object Relational Access Layers

Object and Relational Access layers **[17]** are created to bridge object oriented databases with relational ones. Designing software to connect an object-oriented business system with a relational database is a tedious task. Object-orientation and the relational paradigm differ quite a bit. An application that maps between the two paradigms needs to be designed with respect to performance, maintainability and cost to name just a few requirements. Persistence Options for Object-Oriented Programs and how the Object relational mapping technique can be useful to handle Impedance Mismatch is discussed **[8]**

C. Contracted Persistent Object Programming

The author of **[3]** states that Enterprise applications, require persistence. Conventional approaches to persistence suffer from various deficiencies since they pass a considerable amount of the persistence workload on the programmer.

Programmers have to transfer data to and from storage devices and have to provide mappings from the programming data structures to the storage device data structures

III. NEED FOR THE STUDY

The paper focuses on the importance Object-relational mapping, the tools available for mapping, and on transaction handling and its problems. The major strategies used in current tools are discussed, and several well-known mapping tools are compared in terms of development effort and execution speed in order to enlighten the necessary trade-offs one has to face.

A. Need for Object Relational Mapping tools

The principle of object-relational mapping is to delegate to tools the management of persistence, and to work at code-level with objects representing a domain model, and not with data structures in the same format as the relational database. Object-relational mapping tools establish a bidirectional link with data in a relational database and objects in code, based on a configuration and by executing SQL queries. They all have their pros and cons, just as it's the case for mapping tools themselves of course. The perfect tool for all situations does not exist.

B. Criteria specific to object-relational mapping tools

The tools should be able to use inheritance, create hierarchies between entities, and use polymorphism. It should support grouping , handle any type of relations (1-1, 1-n, n-n) Support Aggregates (equivalent to SQL's SUM, AVG, MIN, MAX, COUNT) .

C. Customization of queries.

We often need to go beyond what is possible with the provided query language. In these cases, we need to be able to provide custom SQL queries. SQL, which is a strong point of Hibernate/NHibernate, allows for this. The tool should support any type of SQL joins (inner join, outer join) .Concurrency management (support for optimistic and pessimistic approaches). Support for the data types specific to the database management system (identity columns, sequences, GUIDs, autoincrements). The tools used must further be able to map a single object to data coming from multiple tables (joins, views). Most of the tools handle a direct mapping of a class to one table. We often need more. The tool should be able to dispatch the data from a single table to multiple objects. GUI to set up the mapping. Such a graphical tool presents the relational data model and lets the objects to be created or at least the links between the objects and the tables to be created.

D. Generation of the classes.

This can speed up the development, even if in a lot of cases we prefer to map the database to hand-coded classes or to lasses generated from UML for example.

E. Generation of the database schema.

Some tools work only with a database they generated. This can be a big constraint, especially if you have to work with a legacy database of course. Otherwise, it all depends on whether you are an expert in database modeling, or if you prefer not to have to deal with the database schema. If you have a DBA who takes care of your databases, or if you prefer to design them by yourself, be sure to select a mapping tool that doesn't require its own data model.

F. Global performance

Lazy loading (the loading of some data is deferred until it's needed) for the data through relations for some columns. When we want to display just a list of names, we don't need all the columns of a table to be loaded. We may need the fields only at certain point, under certain conditions, and so it's better to load them only at that time. Cache dynamically generated queries, so that they don't get rebuilt at each call. Cache some data to avoid too many calls to the data source.

G. Evolution And Compatibility

The tool used should support maintainability. There must also be a possibility to move to a new mapping tool if required. Serialization can be used to persist data outside of the database. Serialization can be done into a binary format, or more important, in XML Distributed objects (remoting, web services; requires support for serialization)

Learning to configure and use an ORM tool takes a lot of effort, but it does not pay off to reinvent the wheel and to write one's own mapping layer when the project size exceeds a certain limit. Especially for business-size projects using an established solution is economically reasonable. So, if you have a relational database and you want to use C++ or Java, by all means use an object-relational product. Writing a mapping layer is much harder than you might expect.

There are certain risks and perspectives of either developing or using a pre-built mapping layer. When deciding upon that point, we must consider implementing a test scenario and try out common mapping tools to see whether it works well for the particular case. This paper gives an impression about some tools.

IV. TOOLS AVAILABLE FOR OR MAPPING

Hibernate is a powerful, high performance object/relational persistence and query service. Hibernate lets you develop persistent classes following object-oriented idiom - including association, inheritance, polymorphism, composition, and

collections. Hibernate allows you to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with an object-oriented Criteria and Example API.

Unlike many other persistence solutions, Hibernate does not hide the power of SQL from you and guarantees that your investment in relational technology and knowledge is as valid as always.

NHibernate is a port of Hibernate Core for Java to the .NET Framework. It handles persisting plain .NET objects to and from an underlying relational database. Given an XML description of your entities and relationships, NHibernate automatically generates SQL for loading and storing the objects.

Gentle.NET is a free, database independent object persistence framework. It features automatic SQL generation and object construction, a SQL factory for creating custom queries, DataView construction helpers, and more. Gentle.NET supports providers for Oracle, PostgreSQL, MySql, Firebird, SQL Server, MSDE, Access, SQLite, and Sybase.

N.Persist Framework is a free open source .NET O/R Mapping persistence framework for version 1.1 (or later) of the .NET Framework. Although it is written in C#, it supports persisting objects written in any .NET language. Mapping information for NPersist is stored in an XML file.

ObjectBroker is an O/R Mapping framework for the .NET platform. Its main features consist of transparent persistance and transaction management. Mapping configuration settings are stored in an XML file. The transaction management features of ObjectBroker are noteworthy. Support for ObjectBroker is provided by the public forums at the products SourceForge home page and via e-mail.

Genome is an O/R Mapping tool that provides automated persistence for .NET classes. OQL is supported as the language with which to specify object criteria. Another useful feature of Genome is its ability to generate DDL scripts for any given business layer-to-database mapping. Genome's mapping information is stored in XML files that can be edited by hand. Genome also supports features like result paging, partial object loading, and greedy loading hints in order to help with performance.

Grove is an O/R Mapping component library for .NET requiring that each object needing persistence contains an attribute that lists the table name from the data source. These attributes also let you specify information such as join types for multi-table objects and primary and foreign key information for class members. Grove supports basic transactions through methods like Begin Transaction, Commit, and Rollback, and supports SQL Server, OLEDB, and ODBC Data Sources.

iBATIS DataMapper is a Java/.NET framework meant to help design and implement persistence layers for Java/.NET applications. iBATIS couples objects with stored procedures or SQL statements using an XML mapping file. iBATIS Data Mapper supports providers for Access, SQL Server, Oracle, MySql, PostgreSQL, DB2, and generic ODBC and OLEDB providers. Using the XML mapping file, the iBATIS framework allows you to query for the objects you want, and returns them to you using the List interface.

Although a number of tools exist, the study will be refrained to only a few of them namely, ADO.Net, Gentle.Net, N.Hibernate. The tools described here are all useful for the Dot net Platform. Property Comparison of the above tools is performed and the results are depicted. Test cases were run for Framework initialization, Simple object insertion, Object Updates, and Query performance. The observations will be depicted graphical

V. METHODOLOGY

A. Mapping persistent objects without Object-Relational Mapping

Without ORM, as in figure, the mapping between the persistent objects and a relational database is performed by calling some methods. The application code issues the SQL statements to the database through a specific database API like Java Database Connectivity (JDBC). The SQL statements might have been written by hand and embedded in the code, or they might have been generated dynamically at runtime, and the application retrieves data from the result set. This implementation is tedious and error prone. The figure depicts an application without using the Object Relational mapping tool. Any client is able to run the application, manage data by using stored Procedures.

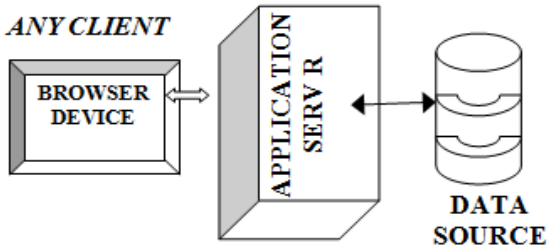

Figure1. An Application architecture without ORM

B. Methodology for mapping persistent objects with Object-Relational Mapping

In figure the Browser device is able to run the application with the help of Object Relational mapping tools. It is less time consuming, and easier to use this tool for the application. When ORM is introduced into the application, the ORM component can automatically translate operations on an object model into operations on a relational model and vice versa. This is accomplished through the use of mapping metadata that relates persistent attributes and associations in the object model with tables and columns in the relational model. Developers need not worry about implementing large number ofSQL statements. The fewer lines of code they write, the fewer programming bugs in the application, and the more time they have to focus on the application.

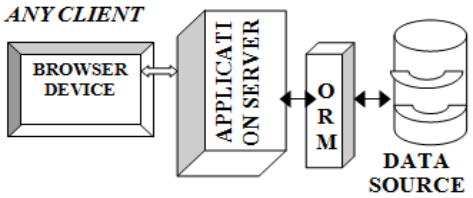

Figure 2.An Application architecture with ORM

Object Relational Mapping tool is an open source. It can be integrated into projects without concern for licensing costs. Moreover, great care has been taken to keep the source code concise, transparent and documented so that it can be easily understood. It is a natural choice for most Object/Relational mapping requirements.

C. Impedance Mismatch

Impedance mismatch is a term naming the gap between the two paradigms OOPS vs RDBMS. It is therefore the main reason for mapping and trade-offs one has to encounter, since impedance mismatch creates the need for transforming data which consumes development time and runtime. Impedance mismatch costs performance especially when treating complex data, while object database management systems (ODBMSs) due to their nature don't face the impedance mismatch problem at all.

Impedance mismatch – the inherent disconnect between object and relational data models – cannot be avoided, but it can be significantly mitigated by the proper choice of database technology. For new application development, it makes sense to store data in a multidimensional database such as cache, which, through its Unified Data Architecture, allows data to be concurrently projected as both objects and tables. For ongoing application development, when existing data is already stored in, say, a relational database, developers should consider converting that data into a multidimensional form. By making a onetime effort to convert their data, they avoid the performance penalties and data model evolution headaches that are endemic to object-relational mapping.

A complete solution to the problem of impedance mismatch must provide both a clean programming model and high performance. While issues of mapping data between databases and programming languages have largely been resolved, significant issues remain. The interface should leverage the best capabilities of both databases and programming languages to for optimization, static typing, and modular development. Each of these aspects has a solution by itself. The problem of impedance mismatch is meeting all the goals simultaneously.

VI. IMPLEMENTATION

Most business database applications use relational databases. Need to map the objects in the application to tables in the database is most essential. Sometimes be a simple matter of mapping individual classes to separate database tables. However, if the class structure is more complex, then the mapping must be carefully considered to allow data to be represented and accessed as efficiently as possible.

Guidelines for OR Mapping

- Use Vertical mapping when there is significant overlap between types – changing types is common
- Use Horizontal mapping when there is little overlap between types changing types is uncommon
- Use Filtered mapping for simple or shallow hierarchies with little overlap between types

OR Mapping Frameworks

Most databases are relational. Much effort has been put in recently to making OR mapping more convenient. When using a framework, the programmer can work only with objects, there are no SQL statements in the code, the selected objects are initially marked as being persistent –thereafter, changes in those objects are transparently changed in the database, and the programmer does not have to write code specifically to update the database. The framework handles the mapping of the objects to relational database tables where they are actually stored – mapping of objects to database tables is usually defined in XML descriptor files.

Products tested

The products tested here are open source and freely available solutions useful for Dot net applications. Choice of the products is that they are simply available. Secondly, there are products being more flexible in terms of the choice of the underlying database than proprietary tools (like e.g. Oracle®'s TopLinkTM mapping tool). Of course, the chosen products can only be a small selection out of a broad spectrum of applications, but they are well-known ones.

N-Hibernate

Product Name and Version	**N-Hibernate 1.2**
Home Page of the Product	www.hibernate.org
Special Features	• Automatic generation of SQL for object loading and storing • Supports a variety of mappings

Gentle.Net

Product Name and Version	**Gentle.NET 1.2.7**
Home Page of the Product	www.sourceforge.net
Special Features	• SQL generation, • Object construction • Caching • Uniqing and validation.

ADO.NET

Product Name and Version	**ADO.NET**
Home Page of the Product	www.wikipedia.org/wiki/ADO.NET
Special Features	• Couples objects with stored procedures or SQL statements using a XML descriptor. • Simplicity is the biggest advantage of the Data Mapper over object relational mapping tools

Experimental Tests Performed

The study's goal is to provide comparative results corresponding to real-world use of some of the object relational mapping tools. All relevant database operations have to be tested and taken into consideration:
The following were performed on the database
• Adding data: INSERT
• Retrieving data: SELECT
• Modifying data: UPDATE

Since creating tables and structural units of a database is generally an installation or deployment task, it is not included in our study. All tests have been done at least 5 times in a row to gain statistical relevance. It turned out that the times remained within a ±10% margin. However, the major significance of these benchmarks lies in the relative performance. So the numbers can only be understood as coarse guidance and primarily in comparison to each other.
The tools tested here are chosen from a number of available tools. They are all chosen for developing applications that use the .Net platform. Mapping Objects to relational databases using these tools can be done easily. Each tool has its own advantage and disadvantages. Using the study, we would be able to choose the right tool for the application to be used.

Framework Initialization

This benchmark provides an imagination of the overall complexity of the frameworks. A broad scope of functionality tends to need longer initialization times, e.g. initializing cache providers, mapping layers, and so on.
Actions counted in this benchmark are primarily the initialization of the database connection and the configuration of the framework parameters – reading XML files, setting up the mapping itself up to the point where objects can be retrieved from or stored in the database.

Table 1. Framework Initialisation Time

S.NO	TOOL USED	TIME IN MILLISECONDS
1	N-Hibernate	303
2	Gentle.Net	0
3	ADO.NET	78

The tabulated values (Table 1) shows the time taken in milliseconds for Framework Initialisation using the tools under study.
In the figure below, graphical representation of the time taken for framework initialization is shown. From the figure, it is clear that Gentle.Net requires the least time for framework initialization. This property of framework initialization is important only for small applications.

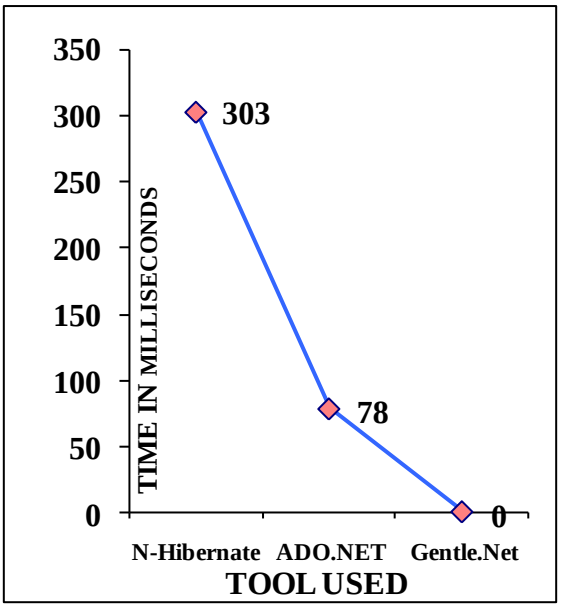

Figure 3. Framework Initialisation time

Creating Data-INSERT Statements

A major aspect of interest is the throughput that a mapping tool can achieve. In this benchmark, the test tool creates a number of objects as fast as possible. It would be even faster by using aggregated INSERT statements directly on the database (by a factor of 10 and more). The following numbers show the achieved object count per second for simple (unrelated) data creation in a single transaction. There are two different problem sizes: in the first benchmark 1000 objects were created, while in the second there are 10000. From the difference, it is possible to deviate the influence of transaction overhead and possibly internal caching effects.

Table2. Insertion of Objects

No. of Objects	N-Hibernate	Ado.Net	Gentle.Net
1000	0.03	0.36	0.46
10000	0.27	3.54	3.89

The tabulated values shows the time in milliseconds for 1000 to 10000 inserts of simple object insertions that are created using each tool

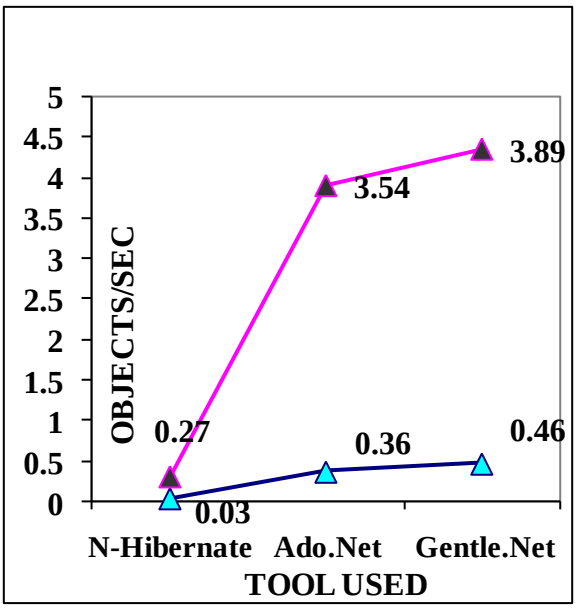

Figure 4. Insertion of objects

Selecting Data- SELECT Statement

Selection of the object from the database is another benchmark. Each tool takes different time for selecting the objects from the database.

Table 3. Selection for 10000 Objects

N-Hibernate	ADO.NET	Gentle.Net
0.06	0.02	0.09

The tabulated values shows the time in milliseconds for selection of 10000 objects using each tool

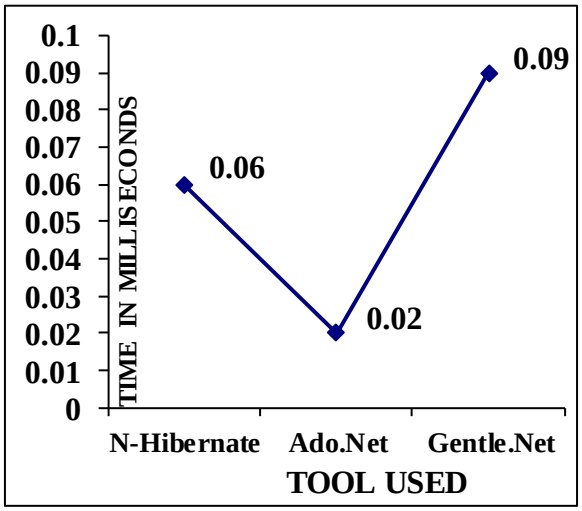

Figure 5. Selection for 10000 objects

From the figure above, it is clear that Gentle.Net requires the maximum time for selection of 10000 objects

Changing data – UPDATE statements

The next benchmark measures the speed of modifications on existing data (and objects). While the time to modify properties of an object locally is negligibly small, making changes persistent does have an impact in terms of CPU time. Here the number of objects considered in 10000.

Table 4. Updation for 10000 objects

N-Hibernate	Ado.Net	Gentle.Net
0.19	0.26	4.26

The update throughput shows the time taken in milliseconds by N-Hibernate, Ado.Net and Gentle.Net for the Updation of 10000 objects in a database. N-Hibernate is found to take the least number of milliseconds for the updation, followed by Ado.Net and Gentle.Net. The tool that takes the least time should be chosen for the application

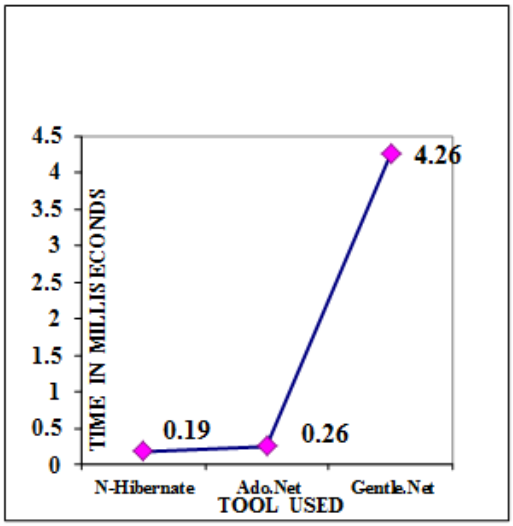

Figure.7. Updation of 10000 objects

VII. CONCLUSION

Performing object relational mappping is complex. Understanding the complexity of mapping, and why it can be very difficult to implement a flexible self-built approach, one has to consider many several relevant for the application to be used. Performance and functionality not only count, but also the overall development costs and the re-usability of components. It is not desirable to implement a hard-coded mapping scheme. Apart from modularity requirements, there are many functional requirements a mapping layer has to fulfil. The risks and perspectives of either developing or using a pre-built mapping layer should be taken care of. When deciding upon that point, a test scenario should be considered and tried out.

REFERENCES

[1] Adya, Atul, Philip A. Bernstein, and Sergey Melnik. "Generation of query and update views for object relational mapping." U.S. Patent No. 7,647,298. 12 Jan. 2010.

[2] Awang, Mohd Khalid, Nur Lian Labadu, and Gong Badak Campus. "Transforming Object Oriented Data Model to Relational Data Model." International Journal of New Computer Architectures and their Applications (IJNCAA) 2.3 (2012): 402-409.

[3] Balzer, Stephanie. "Contracted Persistent Object Programming." (2008).

[4] Briggs, Mario DS, and Ganesh K. Choudhary. "Use-case based configuration of an object-relational mapping framework." U.S. Patent No. 8,489,647. 16 Jul. 2013.

[5] Burzańska, Marta, et al. "Recursive Queries Using Object Relational Mapping."Future Generation Information Technology (2010): 42-50.

[6] Chen, Tse-Hsun, et al. "Detecting performance anti-patterns for applications developed using object-relational mapping." Proceedings of the 36th International Conference on Software Engineering, ICSE. 2014.

[7] Dearle, Alan, Graham Kirby, and Ron Morrison. "Orthogonal persistence revisited." Object Databases (2010): 1-22.

[8] Evgeny, Grigoriev. "Impedance mismatch is not an." arXiv preprint arXiv:1208.3307 (2012).

[9] Gruca, Aleksandra, and Przemysław Podsiadło. "Performance Analysis of. NET Based Object–Relational Mpping Frameworks." Beyond Databases, Architectures, and Structures. Springer International Publishing, 2014. 40-49.

[10] Keller, Wolfgang. "Persistence Options for Object-Oriented Programs."Proceedings of OOP. 2004.

[11] Pereira, Oscar M., Rui L. Aguiar, and Maribel Yasmina Santos. "CRUD-DOM: A Model for Bridging the Gap between the Object-Oriented and the Relational Paradigms." Software Engineering Advances (ICSEA), 2010 Fifth International Conference on. IEEE, 2010.

[12] SanFrancisco Schema Mapping:Object Persistence and Legacy Integration" IBM July 1999

[13] Schink, Hagen, et al. "Hurdles in multi-language refactoring of hibernate applications." International Conference on Software And Data Technologies (ICSOFT). 2011.

[14] Scott. W. Ambler – President –Ronin International, White paper "Mapping Objects to Relational Databases" October 2000.

[15] Sharma, Er Prof Vikrant, et al. "Usage & Performance of Object Databases compared with ORM tools in Java Environment."

[16] Simitci, Aysun. "Object-Relational Mapping in Database Design."

[17] Somma, Ryan. "The O/R Problem: Mapping Between Relational and Object-Oriented Methodologies."

[18] Svirskas, Adomas, and Jurgita Sakalauskaite. "An approach for solving Java object persistence issues using RDBMS and other data sources." V East-European Conf. ADBIS'2001, Professional Communications and Reports, September 25–28, 2001, Vilnius, Lithuania. 2001.

[19] Briggs, Mario DS, and Ganesh K. Choudhary. "Use-case based configuration of an object-relational mapping framework." U.S. Patent No. 8,489,647. 16 Jul. 2013.

[20] Vickers, Arthur John Cerdic, et al. "Object-relational mapped database initialization." U.S. Patent No. 8,600,925. 3 Dec. 2013.

