AUTOMATYKA • 2010 • Tom 14 • Zeszyt 3/2

Radosław Adamus*, Tomasz Marek Kowalski*,
Kamil Kuliberda*, Jacek Wiślicki*, Michał Bleja**

Tools Supporting Generation of "Data-Intensive" Applications for a Web Environment[1, 2]

1. Introduction

Dynamically changing business environments, expansion of new technologies, opening on new markets, desired business flexibility – these and many other factors make the speed of delivery one of the most important requirements for business applications' development. This especially observable for business to customers (B2C) applications offering an web interface. Development of various technologies supporting rapid building of responsive web interfaces makes this approach more and more attractive (not only in B2C solutions), despite of higher demands related to infrastructure (usually multi-tier architecture) or security.

Concerning size and complexity, business systems are not uniform. Such a system may be a small application that calculates revenues and expenditures, or a huge distributed system processing sales and warehousing logistics of a business corporation. Besides this size and functionality variety, contemporary business applications vary on non-functional scope of requirements beginning from issues of security and efficiency, ending at aspects of flexibility and scalability. For these reasons, increase in speed of development usually can be achieved only for certain types of applications which are characterised by relatively low level of complexity and are based on standard architecture (like Model-View-Controller, MVC). Undoubtedly, a large number of business applications fits this model. Applied patterns typically rely on data stored in a database and a web user interface allowing to access

* Computer Engineering Department, Technical University of Lodz, Poland
** Department of Mathematics and Computer Science, University of Lodz, Poland
[1] This research work is funded from the science finances in years 2010/2012 as a research project nr N N516 423438
[2] Author Tomasz M. Kowalski is a scholarship holder of project entitled „Innovative education ..." supported by European Social Fund.

and manipulate these data. Between these two layers there is placed middleware, whose main task is to map a data model, on which the application operates (usually an object model), onto a model in which actual data are stored (usually a relational model). The need for such mapping arises from the principle of independence of data from applications running on these data and incompatibilities of object and relational paradigms (which is referred to as the impedance mismatch) [9]. Assuming that most of actions performed within an application are related to back-end operations on data (CRUD), it is possible to automate a process translating an object model of processed data to a relational model of stored data (e.g., with application of one of various ORM solutions). Additionally, the process can be augmented with scaffolding (i.e. automated generation of a web interface for data retrieval and update).

The following article is an attempt to describe the current state of development tools whose purpose is building data intensive applications with web interface that can be (partially) generated. The article is focused on systems that rely on a relational database as a mechanism for data persistence, covered by an object-oriented application working in a client-server architecture (this should be considered rather as a functional simplification to more complex architectures) available to end users by a web interface. The leitmotif is a thesis that the fundamental element of complexity of such applications is the impedance mismatch between data models used in storage and business logics. The thesis also claims, that a commonly accepted direction of development of such solutions does not lead to any elimination of this inconsistency, but rather tends to hide it inside a mapping layer, which may, in certain cases, lead to a counterproductive effect.

The rest of the article is structured as follows. The second section describes common approaches to object-to-relational mapping. The third section presents technologies and tools allowing to generate a web interface. The task of the fourth section is to show problems arising from an object-to-relational mapping layer (supported by two examples). The fifth section presents an alternative approach to the issue of persistence. The last section provides a summary and conclusions.

2. Object-to-relational mapping as a solution for persistence

The introduction of object-to-relational mapping and the related level of complexity in the process of implementation of an information system generates a need to create specialised tools (libraries, frameworks) whose goal is to automate this task. The use of such tools effectively enforces speeding up of the building applications process. These tools are called Object-Relational Mappers (ORM) and are available for all most-popular programming languages and environments used for creation of applications operating on large amounts of data. Such applications are referred to as *data intensive applications*. Examples of popular ORM solutions are Hibernate and JPA (Java and .NET) [5, 6], ADO.NET Entity Framework (.NET) [2], Django (Python) [4], ActiveRecord (Ruby) [10], GROM (groovy) [8].

ORM development tools strive, on the one hand, towards higher levels of mapping process automation (like automated generation of a database schema based on an object model of data available in application, automated generation of queries, etc.), and, on the other hand, in a direction of even closer integration of a query language with an object model and a programming language. The purpose of this integration is to avoid the problem of direct embedding of SQL queries within an application code – in a form of strings that do not have ant semantics (from an application programming language point of view) and consequently are not a subject of static typological control and verification. Increasing the automation level is often supported by use of a declarative programming model (such as attributes in C#, annotations in Java, or specialised descriptors based on XML) which allows defining a detailed mapping specification. Better integration of a query language with an object model and a programming language can be achieved by sharing (an intermediate) query language operating on object data (e.g., HQL, JPQL). In the next step of evolution, an integration of the query language with a programming language can be done (like in LINQ). Such an approach allows complete hiding of a mapping layer. This solution combined with an automated model building process for stored data (based on the object model), leads to a situation in which the relational model is treated as a „machine code" of the data warehouse. This, in turn, often leads to an undesired loss of control over the mapping layer.

3. Generating the Web applications

Frameworks being a bridge between a database and an application minimise efforts involved in building a layer of a business application area. On the other hand, the process of building an application layer responsible for a dynamically generated web interface also has many implementations reflected in various tools. They assist a programmer by providing a standard architecture which is mostly based on the MVC pattern, or its variants. Having to use such a skeleton of an application, including a technology allowing dynamic generation of HTML pages (interpreted by a web browser at the client side), a developer builds system functions which, basing on a browser requests, operate on data using the O/R mappers' mechanisms. Frameworks permitting to build a dynamic web interface (processed at the server side) additionally provide many features which facilitate implementation of non-functional system requirements (like security, multi-language support). Moreover, they can be equipped with facilities simplifying testing of such an application easier (this requires a web application server, in the simplest case it could be the web server equipped with an application service module). Such frameworks could also have extensions providing auto-generation of scripts interpreted in the browser (JavaScript) or building of quickly reacting web interfaces (e.g., with AJAX technology).

More complex applications may require additional services that require remote calls of external services in relation to the built application. This introduces an additional level of

complexity associated with non-functional aspects of distributed programming (integration of systems, legacy systems, distributed transactions, security, reliability, etc.) and technologies supporting the same type of model for building applications. The result is that very often an additional layer is introduced inside such systems' architecture (called a service layer). Such a layer by, sharing a specific interface to the service (or group of services), makes a web interface layer independent from the complexity of the problem itself. A selection of an appropriate distributed technology is restricted by many factors, from which the most important is a platform for implementation of an application. This limitation is currently defeated by the increasingly popular Service Oriented Architecture (SOA) based on so called the Web-Services technology (independent from a platform technology).

Building of web applications is a so important part of Software Engineering, that more and stronger accents are put on use of application generators which could allow automatic (or partially automated) building of this kind of applications. This type of frameworks often consist of O/R mappers with a web interface framework based on the MVC pattern. They provide a set of tools allowing automated generation of all application elements, particularly a database structure and corresponding objects at the application side. It is done in such way that objects and a database are connected together with mapper mechanisms and user interface mechanisms responsible for processing (CRUD) of these data. This technique is called *scaffolding* and first appeared in context of the MVC pattern. Scaffolding allows automatic generation of components: a *model* – structure of a database and mapping, a *view* – a view of data providing CRUD operations, a *controller* – a web page controller constituting a base for combining the view with the model.

Depending of a technology and an advance level of a tool, it is also possible to automatically generate system's components responsible for realisation of non-functional system requirements (e.g., security) or horizontal services (like postal services, queuing of messages, system integrations). However, there is lack of possibility for generation of more complex scenarios for applications. Usually, development of generators for this kind of features is possible through propagation of advanced programming techniques (e.g., generic or aspect-oriented). Examples of tools equipped with automated generation of an application are: Rails (Ruby) [10], Django (Python) [4], Grails (Groovy) [8], Spring Roo (Java) [3].

4. Criticism of O/R mapping

Development of tools for O/R mapping seems to be a reasonable compromise. On the database side, such tools enable using of mature and trusted DBMS-s (characterised by high reliability, credibility, a possibility of choice system's vendor) and use of a standard query language (SQL). Simultaneously, from the application level, there can be used any popular object-oriented programming language having the tool support, a wide range of p directly prepared components, a ready runtime environment (in context of web applications). An application programmer works directly on both sides of objects – at a query and its result.

Queries in SQL are automatically generated independently from a particular vendor of the database (although the standardisation of a specific SQL database system has its own dialect of the language which differs slightly from the standard). Additionally, the issues addressed to the management of transactions in database are largely transferred into the mapping layer level.

The advantages of O/R mapping tools mentioned above can not be achieved without costs and can not obscure the fact that the impedance mismatch has never been and will not be fully covered by the O/R mapping layer. The reasons (the costs) are:

- The optimisation problem. The popularity of relational systems, and their thirty years of development meant that many of the techniques for optimising queries relating specifically to the most expensive operations (join) has been discovered and implemented. Moreover, DBMS-s are equipped with advanced tools to optimise costs, whose besides relying on statistics can also be supported by administrators. Use of mapping layer mechanisms which automatically generate SQL queries heavily restricts and in many cases effectively prevents such optimisation (lack of control over a query form).
- Semantic traps (so-called semantic reefs). There are many examples showing what semantic problems occur at the level of non-orthogonal SQL constructions [12]. An additional mapping layer, which automatically, basing on a query written in an object--oriented language, generates an SQL query may introduce additional traps due to bugs at the semantics level of the translator (or inability to make an unambiguous translation). These problems will recur with increasing levels of ORM tools' semantics complexity, or adding new elements allowing mapping of more complex object structures.
- The problem of building a relational model. To take advantage of ORM tools, a developer must adapt a data model (database structure) to the needs of the tool. For new applications, mapping tools have function of automated data model generation, even when object model has relatively small complexity, its automatically generated relational equivalent can be very complex. As a result, there is no control over the data model. This may increase costs of software changes' introduction (it especially concerns data model changes).
- The problem with changes. Managing changes in software are an integral part of Software Engineering. Minimisation of cost of software maintenance, especially for systems developed for a specific customer, is more important than minimisation of cost of manufacturing the same software. Implementation of a mapping layer can minimise time (as well as cost) of a developed system. Notice, that restrictions related to process automation and complexity of the mapping layer (which itself increases the level of the system complexity) may result in increasing of software maintenance costs. This particularly applies to systems whose data model requirements are not stable.

As a result, pragmatic use of a tool causes that a developer limits an object model not to complicate mapping and instead of this he or she does manual writing of SQL queries. It is

done wherever he or she encounters a problem with semantics or lack of optimisation of a query execution plan. Use of this type of bypasses requires that a relational data model has to be controlled by a programmer. In most cases, except the simplest ones, this excludes any use of automated relational model generation.

A separate problem concerns optimisation of data reading. A result of a query invoked in database must be materialised in a form of objects understood and available within the application. The process of object creation is time consuming, thus a mapper must also implement optimisation mechanisms which are associated with this process itself. There are some optimisation methods supporting such a process. They are mainly caching query results and lazy initialisation. However, their use generates new problems associated with validity of results, boundaries of a database session, transactions, etc.

The development of mappers and query languages based on the mappers heads to the ever closer integration with programming language. This is an obvious direction of evolution aimed to minimise the problem of duality of environments (application vs. database) and the inability of use of typological controlling at the compile level of an application code. The most advanced language in this context is LINQ which is directly integrated with the .NET platform environment and its languages (C#, VB.NET). It does not change the fact that all the time we deal with a language which implicitly operates on a relational model, which, in turn, implies limits on any use of power of the object model. These limits stem from O/R mapper mechanisms whose level of complexity increases with growth of a mapped object-oriented mechanisms' set.

Integration of a database query language with an object-oriented programming language is based on introduction into the language syntax level a syntactic sugar in a form of keywords like *select-from-where* together with semantics of operators represented by the keywords. The semantics provides support for collections and standardisation of types (which gives possibility of the static typological control). When a developer builds a query, he or she operates on the object model, while the object-to-relational mapping mechanism for the model and queries is automatic. Notice, that such integration does not eliminate the impedance mismatch problem, but simply hides it deeper, at the platform level implementation. The combination of an object-oriented programming language with a query language based indirectly on the relational model whose queries are directed to the object model and then transformed to the corresponding SQL queries will not change the paradigm of data storage. An application code will be still divided into structures concerning the persistence of data and structures concerning the volatile data.

4.1. Examples of ORM tools' problems

Objects provided by the ORM layer are populated with result data coming from relational queries arising from the description of mapping (e.g., by application code annotations or external documents such as XML). A developer cannot influence a form of generated queries. He or she loses possibility of controlling performance of execution of a query, so optimisation issues are completely not used then. Moreover, semantics of a performed SQL

query which was used to generate a result may not correspond to semantics of an original „object" query (e.g., HQL). Examples listed below describe kinds of traps about which we have written above. Such traps will concern HQL – because of the popularity of Hibernate and using of its parts in the latest JPA standard. Due to limitations of this paper we show only two examples.

Consider a simple object model in which an object of a Parent class has two subobjects of Child1 and Child2 classes. But from the business logic concludes that only one of them can appear (the other one takes the *null* value then). In a relational model this relationship will be realised by foreign keys to tables Child1 and Child2 which are located inside the Parent table (or vice versa – Child1 and Child2 will have a foreign key to the Parent table). Considered an example query aiming to retrieve objects Parent where Child1 or Child2 have a specific value of a selected attribute: *select p from Parent p where p.Child1.Attribute = 'value' or p.Child2.Attribute = 'value'*. A programmer will expect that if one of the „children" does not appear the „*or*" logical expression will be evaluated for second „child". However, the SQL query generated by Hibernate contains joins of the Parent table with the Child1 and Child2 tables. Their join conditions are connected with the „*and*" operator (e.g. *Parent.Child1_ID = Child1.ID and Parent.Child2_ID = Child2.ID*). Of course, evaluation of logical expression „*and*" always returns *false*, because, due to *null* values occurring in any foreign key field, one of the operands will be always *false*. As a result, the query will always return an empty result. The workaround is evaluation of two separate queries and then make an union of their results. But this still does not change the fact, that in absence of knowledge about such a trap, a developer may have serious trouble.

Another, equally trivial as previous, example concerns calculation of collections of child objects. An HQL query is converted into an SQL query containing all joins resulting from the given mapping. A relational result is just a table whose records are interpreted as objects and their data. Consider the same objects presented in previous example where objects of class Parent contain collections of objects child1 and child2 coming from classes Child1 and Child2, respectively. In a relational model there are appropriate foreign keys within tables Child1 and Child2. An „object" query aims to retrieve data of an appropriate „parent" with its „children" (including both types): *select p from Parent p where p.id = 1* (we assume that *p.id = 1* uniquely identifies the parent). When „children" collection loading is defined as *eager* (which means instant initialisation of collection references, one relational query is executed), the result table will contain all columns of tables Parent, Child1 and Child2 – due to a specific joins generated within an SQL query (the Cartesian product of tables). But the result table will also contain „empty" columns corresponding to non-existent „children" (which are a result of the outer join). Processing such a result table will primarily lead to retuning by the mapper (Hibernate) many objects with type Parent, despite the fact that the programmer expects one. This problem can be eliminated by forcing only one root object (provided that this phenomenon is known to a programmer). Collections child1 and child2 will contain *null* elements arising from the relational outer join. This will result as incorrect multiplicity of collections and unpleasant surprises at the runtime.

Here can be shown more examples of the traps lurking for developers using mappers. These examples can be deeply analysed and some workarounds presented to prevent the threats. However, our goal was only to draw attention to the problems that may arise when working with ORM solutions.

5. Generating web applications based on an object-oriented database environment

One of the factors affecting the speed of building applications while minimising its cost of maintenance is to reduce their complexity. In context of sustainability, it can be achieved by eliminating the problem of the impedance mismatch. The way, which cannot be dismissed, is to build a tool (hosting the application model) which is able to provide a mechanism of orthogonal persistence based on composition of objects together with a programming/query language equipped with powerful typological controlling. Additionally, the mechanism of orthogonal persistence should be able to store persistent data with any level of complexity in terms of their heterogeneity and distribution. Also, it should provide a mechanism for processing and updating through integrated language constructs orthogonal to the durability features. Such a framework, besides eliminating the O/R mapping layer, should provide a set of components capable for automatic generation of all elements of an application for handling and processing data in a database from a web interface level.

The experiences that authors of this article have gained during the project devoted to building a prototype platform ODRA (Object Database for Rapid Application Development) [11] have shown that development of this environment type is possible. ODRA is an object-based prototype database management system based on the Stack Based Architecture (SBA) [12, 5]. The main objective of the ODRA project [1] was to develop a new paradigm for building database applications. This paradigm assumes raising of the abstraction level at which developers work. ODRA introduces a new universal programming language and distributed object-oriented runtime environment. With these elements, ODRA simply provides functionalities which in others the most popular environments can be achieved, but require combination of multiple tools (object-relational databases, various types of middleware, a programming language for general use including its runtime environment).

ODRA consists of the following, strongly connected components:

- An Object-Oriented Database Management System to enable storage and processing of data in form of persistent objects.
- A Compiler and an Interpreter for an object-oriented query and programming language called SBQL (to unify the level of application programming and data processing).

- Distributed communication mechanisms based on a distributed technology to distributed databases.
- Wrappers and filters providing seamless integration and import of data from heterogeneous sources (relational databases, RDF repositories, XML documents).

6. Summary

Issues described in this paper focus, in context of web applications, on the problem of information systems durability. Observed development of durability mechanisms lies in evolution of tools intended to be used as object-to-relational mappers. This evolution makes on the one hand, increasing opportunities in supporting of automated mapping services and gradual integration of a query language with a programming language of an application. On the other hand, together with extension of mapping mechanisms, increases complexity of ORM tools, in our opinion, disproportionate to their potentiality. In case of simple applications, from the point of view of a data model, such an increase of a tool complexity level is unnecessary and leads to excessive complexity of application structure. When applications have a complex (not infrequently distributed) data model, use of mapping tools may not produce any expected results (especially in the field of optimisation). As a result, this applies different workarounds with manual construction of a data model which conflicts with speed requirement of system delivery. Therefore, we believe that the accepted direction of durability tools' development by a building of a bridge between the application platform and database leads to excessive complexity, which may finally lead to simplification of an application data model where use of ORM would be efficient. For this reason, we believe in importance of searching for alternative solutions based on a good semantic foundations and implementing durability mechanisms in the form of object-oriented database with object-oriented query/programming language. It should be a task for many new projects realised by scientists and business.

References

[1] Adamus R., Daczkowski M., Habela P., Kaczmarski K., Kowalski T., Lentner M., Pieciukiewicz T., Stencel K., Subieta K., Trzaska M., Wardziak T., Wiślicki J., *Overview of the Project ODRA*. Proc. of the ICOODB 2008 Conference Berlin, ISBN 078-7399-412-9, 2008, 179–197.

[2] ADO.NET Entity Framework 4 Reference, http://msdn.microsoft.com/en-us/library/bb399572(VS.100).aspx (03.2010).

[3] Ben Alex, *Introducing Spring Roo 1.0.0*, Spring Source: http://www.slideshare.net/benalexau/introduction-to-spring-roo-100-2805183 (01.2010).

[4] Django project documentation, http://docs.djangoproject.com (03.2010).

[5] King G., Bauer Ch., Andersen M.R., Bernard E., Ebersole S., *Hibernate Reference Documentation 3.3.2.GA*. http://docs.jboss.org/hibernate/stable/core/reference/en/pdf/hibernate_reference.pdf (02.2010).

[6] Java™ Persistence 2.0 Expert Group: *JSR 317 Java™ Persistence 2.0* (*Final Release*), http://jcp.org/aboutJava/communityprocess/final/jsr317/index.html (01.2010).
[7] Kozankiewicz H., *Updateable Object Views*. PhD Thesis, Warszawa, 2005, http://sbql.pl/phds/PhD%20Hanna%20Kozankiewicz.pdf.
[8] Lee Chuk Munn, *Agile Web Development with Groovy and Grails*, Sun Tech Days 2009, http://developers.sun.com/events/techdays/presentations/locations-2009/hyderabad/td_hyd_groovy_lee.pdf (09.2009).
[9] Neward T., *The Vietnam of Computer Science*. http://www.odbms.org/download/031.01%20Neward%20The%20Vietnam%20of%20Computer%20Science%20June%202006.PDF 2006.
[10] Marshall K., Pytel Ch., Yurek J., *Pro Active Record: Databases with Ruby and Rails*. Apress, 2009.
[11] *ODRA*(*Object Database for Rapid Application development*) *Description and Programmer Manual*, http://sbql.pl/various/ODRA/ODRA_manual.html, (01.2010).
[12] Subieta K., *Teoria i konstrukcja obiektowych języków zapytań*. Wydawnictwo PJWSTK, Warszawa, 2004.