

Relational-Object Mapping Technique

I. O. Likhatskyi

This paper is focused on describing a technique of creating object-relational transformation components by using code generation system based on database structure. It reviews existing ORM solutions, as well as highlights their main advantages and drawbacks. We describe a technique to create relational-object transformation components by using code generation system to automatically generate persistence layer based on the database structure. Text template engine is used to generate SQL queries, business classes and APIs to access data from application code. Provided default implementations are sufficient to quickly obtain working persistence layer.

Introduction

Nowadays along with information technology rapid development and data management systems evolution one of actual important problems arising in application development is the choice of database management systems (DBMS), according to the particularities of store data applications and effective management. On one hand, the database should support the necessary degree of data abstraction, and on the other hand, it should be focused on the structural features of data organization and nature of use.

Currently widespread object-oriented development methodology of applied systems and the relational database management systems (RDBMS) dominate in the world of data storage. Therefore, appropriate solution is to use the persistence layer that provides the necessary object-oriented interfaces to access and manage the data that is stored under the control of RDBMS [1, 2]. The relevance of this approach is combined with well-known advantages of relational databases such as:

- ability to support legacy systems using traditional solutions;
- simplicity in using technology based on clear table model and mathematically rigorous theory of relational algebra;
- widespread and thorough approbation of the proposed database products on the market for many years;
- providing natural object-oriented API implemented for most popular programming languages.

However, the mentioned data models for object-oriented and relational systems differ significantly. Therefore, software developers face the need to implement conversion between object-oriented and relational representation of the same data.

Creation of persistence layer is associated with a set of design solutions that affect maintainability, performance, simplicity of use between the client application and the database server.

Sometimes such conversion is implemented manually which makes code more efficient but involves a lot of routine work that is manually transforming each database entity. In other cases, there can be used ORM [3], improving developer productivity, but often sacrificing runtime performance and flexibility.

In this paper we describe our technique to create object-relational transformation by using code generation system to automatically generate persistence layer based on database structure.

The database hiding problem

With birth of the RDBMS programmers have got the structures of logic level and SQL to access the data. Thereby software developers got rid of knowing the odds and ends of physical data storage organization. It turned out that most data can easily be described in the form of tables and relations between them. Thus, it has predetermined the success of relational databases.

The underlying data models for object-oriented and relational systems differ significantly. This means that they describe the same entity but with different parties: the relational model focuses on the structure and relationships between entities and the object model – on their properties and behavior. The relational model is used for information modeling, separating the essential attributes to save their values and subsequent retrieval, processing and analysis. The object model is largely used to simulate behavior separating the essential functions and their subsequent use [4]. In practice we have a situation where the programs are written mainly with the use of OOP, and then the data is stored in relational databases. Thus, there is a need of mapping objects in relational structure and vice versa. There are a lot of techniques that were developed in the late eighties and are reflected in number of publications [5, 6, 7, 8, 9].

The component of a software system responsible for converting data from the object into a relational form is called ORM (object-relational mapping) system. In technology of mapping objects on the RDBMS, there is an important point. Some people believe that the persistent generated by ORM layer generates SQL code, which is similar to translating high-level language into machine native code. This statement is wrong, and may also lead to the creation of hard-tracked systems with potential performance problems.

The fact that SQL – is a high-level declarative language that belongs to the fourth generation is unlike, for example, Java or C# that belongs to the third generation of imperative languages. For example, one SQL operator, performing something a little more complicated than selected by key, demand to achieve the same result much more rows in C# or Java [10].

This situation leads the ORM developers to create their own SQL-like language to manipulate objects and then to translate it to SQL code. For example, HQL – Hibernate Query Language – SQL-like query language that is used in Hibernate / NHibernate [11] or .Net Framework component – LINQ to SQL, which provides run-time infrastructure for managing relational data as objects [12]. There is also the possibility of using dynamic transformation of the SQL query into a collection of objects.

Otherwise, we would have to extract large amounts of data from the database and then process them directly in the application. Roughly the same data were

processed with no built-in SQL language. This approach is called the navigation approach to manipulate the data, and it is also a typical for network and hierarchical databases [13]. Nevertheless, getting the ORM, we get back to the navigation data processing approaches in some extent.

Thus, there is a situation in which developers are trying to hide the lack of RDBMS knowledge over an additional level of abstraction. In spite of the abstraction level that is provided by ORM and reduces development time making the application work effectively with RDBMS without basic knowledge of SQL is almost impossible.

ORM drawbacks

Using ORM tool to provide interaction between the applications and database server, developers often faced with a number of problems. Once the developers have implemented CRUD-logic using SQL directly is difficult. This concerns data mapping strategies and application portability problems between databases. In fact, every SQL query to the database is a kind of projection of the result set to a specific class. In this regard, developers often have to use ORM query language (if supports). Frequently such languages are not standard and do not have the tools for debugging and profiling. For example, in the .NET since version 3.5, it is possible to use LINQ, which can detect some errors at compile time.

The result is that the ORM query language generates not the most optimal SQL code. To solve these problems developers often turn to partial processing data within the application: selecting a collection of objects in cycles, or filtered, using the same LINQ queries over the processed arrays, generating new queries. The number of such queries to the database in such processing may number in the thousands.

Relational-object representation as an alternative to ORM

In this section we describe the relational-object representation technique that can be used as a solution for converting data between incompatible type systems in object-oriented programming languages.

The main task of relational-object design is to achieve maximum performance in the interaction between the application and database server, as well as achieving a high level of automation through the use of code generation.

The base principle of relational-object design is the concept of information systems based on the model that starts from database. There was proposed a concept based on the construction of high-performance database structure that describes developed information system model. Providing developer with full access to the database management (index building, writing complex SQL queries, using views, etc.), we solve one major issues related to performance. Thus, the use of specialized high-level declarative language SQL will optimize the process of retrieving the data and the performance of the system in general, compared to the processing large data amounts in the application.

Naturally on one hand system performance will depend on qualifications of the

developer who is in charge of design of the database structure and query writing but on the other hand we have complete control over this process. In that view it can be concluded that the development of high-performance applications without knowledge of SQL cannot be done, and implementation of complex structures, such as complicated SQL queries, or building the right indexes, can be hardly automated.

However, there is one process that can be automated; it is the process of writing CRUD stored procedures. CRUD stored procedures generation process performs each database object (table, view). This kind of stored procedures has a clear structure that is why the process of their generation can be easily automated [14].

To set the correspondence between the relational and object data representation we use the methodology of “three projections” [15]. This methodology describes the mapping rules which help to transform the data from relational to object-oriented model and vice versa. A clear allocation of relational objects and their properties allows us to present them in the form of the object.

Thus the code generation process of the data access layer is fully automated. The developer receives a full set of API to access and manipulate the data stored under the control of RDBMS.

Code generation system C-Gen

In this section we describe the C-Gen – our code generation system that can be used as a solution of the object-relational paradigm mismatch.

The C-Gen system generates a persistence layer of the application using a database structure as an input. The database describes the entities in a subject domain and therefore can be used to create business objects. Currently the system is unidirectional: we can generate business objects from the database structure but not vice versa.

The C-Gen uses a code generation approach, i.e. the automatic generation of source code from the given input data. As a code generation tool we use a Text Template Transformation Toolkit (T4 [16]). Templates are used to generate a program source code based on the model (database structure). The generated file can use an arbitrary text format, in particular, it can be a program source code in any language.

Code generation process

The generation process starts from existing database that is designed manually. A domain model is represented as a set of tables and views with relations. The database developer takes all responsibility for creating database. This approach supports maximum performance and flexibility of the created application. The developer has a full control over the process of designing and creating the database. As a result the developer can create a high-quality and efficient SQL code.

When the domain model is implemented in the database, we use the C-Gen system to generate a persistence layer. It consists of stored procedures inside the

database, as well as business objects and access methods in a source code. The C-Gen supports two types of stored procedures:

- *simple stored procedures* support CRUD (create, retrieve, update, delete) operations and are generated automatically;
- *custom stored procedures* are specific to a definite situation, and their design is fully controlled by the developer.

Thus, by generating simple stored procedures automatically the developer is relieved from writing them by hand. On the other hand we reserve the possibility of implementing performance-critical stored procedures for the developer.

The next step after generating stored procedures is creating business objects. For each entity in the database the C-Gen system generates a corresponding class. Each database field is represented as a strongly typed property. For each custom stored procedure the C-Gen system also generates a corresponding class based on the procedure name and return fields. The final step is the generation of access methods for simple and custom stored procedures; their parameters and return type depend on stored procedure signature.

All generation steps are performed using T4 run-time templates that allow creating a program source code and SQL queries. An example of template for creating simple stored procedure (insert) is shown on Figure 1.

```
AS
SET NOCOUNT ON

insert into <#= string.Format("[{0}].[{1}]", SchemaName, Tbl.Name) #>
(
<#
    PushIndent("\t");
    isFirst = true;
    foreach (string str in cols)
    {
        if (isFirst)
        {
            isFirst = false;
            Write(" ");
        }
        else
            Write(",");

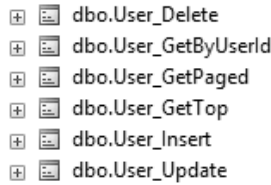
        WriteLine(String.Format("[{0}]", str));
    }
    PopIndent();
#>
)
values
(
<#
    PushIndent("\t");
```

Figure 1. A piece of T4 template for Insert stored procedure.

As can be seen from the code fragment, the template consists of literal frag-

ments that are included into resulting file without processing, and control logic fragments, marked by special symbols such as `< #`. During code generation, the control logic fragments are executed and their results are included into generated text file. Control logic has access to model metadata, represented as typed variables. Notice that templates can use all control structures of C# language, such as branching and loops. Therefore code generation can include quite complex custom logic.

As an example of generation process consider a simple database with single table `User` containing information about users of some system. As a first step of C-Gen generation process, 6 simple stored procedures are generated (Figure 2).



```
+ [icon] dbo.User_Delete
+ [icon] dbo.User_GetByUserId
+ [icon] dbo.User_GetPaged
+ [icon] dbo.User_GetTop
+ [icon] dbo.User_Insert
+ [icon] dbo.User_Update
```

Figure 2. Simple stored procedures.

On the next step C-Gen generates `User` class with properties corresponding to the fields of the `User` table in the database. On the final step, 6 methods are generated, one for each simple stored procedure.

After the C-Gen system completed its work, the application developer can work with business classes as required in application. Working with database is completely hidden behind generated methods. If some changes are made to the database structure the persistence layer can be regenerated. Therefore, the C-Gen combines the automation of ORM systems with flexibility and performance of hand-coded persistent layer.

Performance evaluation

To evaluate advantages of our approach, we have compared the performance of the persistence layer generated by the C-Gen system with a code generated by `NetTiers` [17] and `.Net Entity Framework 4.0`. For an 18Mb sample database (representing an Internet shop) we have generated all relevant code using these systems. Then we measured the performance of `Select` and `Insert` operations for different query loads. Measurements were performed on a server with `Core i5-2500k 3.3 GHz` CPU and `8Gb RAM`, running `Windows 7 x64 SP1` and `Microsoft SQL Server 2008 R2 x64 Express`. The results of performance measurements are shown on Figure 3.

As it can be seen from measurement results the C-Gen system generates more efficient code: for `Select` operations it is about 3 times faster compared to `NetTiers` and about 5 times compared to `Entity Framework`; for `Insert` operations – 1.66 times faster compared to `NetTiers` and about 3.4 times compared to `Entity Framework`.

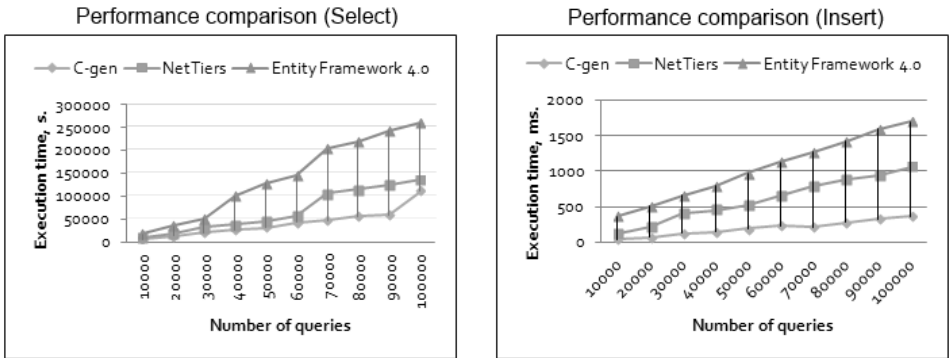


Figure 3. Performance comparison of C-Gen and NetTiers systems.

Conclusions

This paper determines the relevance of systems usage that provides integration between relational and object data representation. There was concluded one of the most common approaches of such interaction implementation: the use of ORM systems. Also there were analyzed the advantages and drawbacks of the ORM-like systems; relational-object mapping techniques were proposed as well.

In this paper we have described our approach to creating a persistence layer to connect object-oriented and relational data. This approach is based on two concepts: best performance and high degree of automation. The relational-object mapping technique is based on principles of building the informational systems based on the database model. This allows solving two main problems:

- *performance*. The queries written by SQL developer are much more effective than the code that was generated by ORM system.
- *automation*. Code generation process based on the methodology of “three projections” allows establishing a correspondence between the object and relational representation of the data and provides API for accessing the data.

To this end we use a text template system to generate SQL queries, business classes and data access methods. We have implemented the C-Gen code generation system that creates easy to use and efficient persistence layer based on the database structure. Code generation mechanisms based on T4 text templates allow generating the source code in any programming language. Performance evaluation demonstrates high efficiency of the code generated by the C-Gen system compared to the similar systems. Detailed information and performance test of this system are described in the articles [14, 15, 18].

One of the areas of use of this technique may be the sphere of support and modernization of inherited (legacy) systems. For example, it is difficult to find a

corporation with more than 25 years of age which would not use information subsystems based on earlier hardware and software platforms from IBM. Such database subsystems contain huge amounts of valuable information and the corporations have to support it.

On the other hand, this approach can be used in the systems for which runtime is the critical option, or there is a limit of hardware resources that can be used by the system. For example, a non-optimal SQL queries or additional processing of data in an application can lead to increased consumption of system resources required to maintain the information system. For small systems, this fact may seem insignificant, but when it comes to industrial scale, and the principle of cloud hosting is built on the billing of CPU time and RAM consumed by the application, the savings can be substantial.

References

- [1] W. Keller, "Object/relational access layers – a roadmap, missing links and more patterns," in *Proceedings of the "3-rd European Conference on Pattern Languages of Programming and Computing (EuroPLoP)"*, 1998.
- [2] V. P. Ivannikov, S. S. Gaisaryan, K. V. Antipin, and V. V. Roubanov, "Object-oriented environment that provides access to relational databases," in *Proceedings of the "Institute for System Programming of Russian Academy of Sciences"*, pp. 89–114, 2001.
- [3] C. Russell, *Bridging the Object-Relational Divide*, vol. 6, ch. Object-Relational Mapping, pp. 18–28. Sun Microsystems, 2008.
- [4] "Overview of object-relational mapping (orm) for the .net." <http://arbinada.com/main/node/33>.
- [5] J. Eggers, "Implementing express in sql. document iso tc184/sc4/wg1/n292." <http://www.iso.org/iso/home.htm>.
- [6] M. Mead and D. Thomas, "Proposed mapping from express to sql. technical report," tech. rep., Rutherford Appleton Laboratory, 1989.
- [7] K. C. Morris, "Translating express to sql: A user's guide. technical report nistir 4341," tech. rep., National Institute of Standards and Technology, Gaithersburg, Maryland, 1990.
- [8] D. Sanderson and D. Spooner, "Mapping between express and traditional dbms models," in *Proceedings of the "EUG'93 - The Third EXPRESS Users Group Conference"*, 1993.
- [9] L. Klein, A. Stonis, and D. Jancauskas, "Express/sql white paper. document iso tc184/sc4/wg11/n144." <http://www.iso.org/iso/home.htm>.
- [10] "Orm or object-relational projector." <http://habrahabr.ru/company/piter/blog/165327/>.
- [11] "Hql examples." <http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html#queryhql-examples>.
- [12] "Linq to sql [linq to sql]." <http://msdn.microsoft.com/ru-ru/library/bb386976.aspx>.

- [13] “Navigational approach to data manipulation and personal databases.” http://www.mstu.edu.ru/study/materials/zelenkov/ch_4_8.html.
- [14] I. O. Lihatskiy, “Code generation tools for interacting with the database through the objects,” *Problems of programming*, no. 2-3, pp. 384–385, 2012.
- [15] I. O. Lihatskiy, “About one method of forming an object view of relational data,” *Problems of programming*, no. 3, pp. 81–82, 2013.
- [16] “Code generation and t4 text templattess.” <http://msdn.microsoft.com/en-US/library/bb126445.aspx>.
- [17] “nettiers application framework.” <http://nettiers.com>.
- [18] I. Lihatsky, A. Doroshenko, and K. Zhreb, *Information Systems: Methods, Models, and Applications*, vol. 137, ch. A Template-Based Method to Create Efficient and Customizable Object-Relational Transformation Components, pp. 178–184. Springer, 2013.

Authors

Igor Oleksandrovych Likhatskyi — the Junior Researcher, Institute of Software Systems of National Academy of Sciences of Ukraine, Kyiv, Ukraine; E-mail: igor-md@ukr.net