Analysis of Object Persistence Frameworks in the .NET Framework

Rok Žontar, Uroš Goljat
Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova 17, SI - 2000 Maribor, Slovenia
{rok.zontar, uros.goljat}@uni-mb.si

Abstract. *Today, data is an important part of almost every modern application. However programming languages normally don't offer the same support for data manipulation as for program logic. It is obvious that there is a big gap between the two worlds; namely the object-oriented world and relational world. The consequences of the resultant gap are reflected in the fact that developers spend a lot of time transferring data from one form (object) to another (relational) and vice versa. In this paper, we will present the impedance mismatch between object-oriented programming languages and relational databases. Object-relational mapping will be presented as a technology that helps developers overcome the impedance mismatch. Thus, the primary focus will be on object-relational patterns and persistence ignorance. We will describe the different object-relational mapping frameworks based on the .NET framework. On this basis, comparison criteria for evaluating applied O-R mapping technologies will be defined. These criteria will then be used in comparison and analysis of the presented O-R frameworks.*

Keywords. Object-relational mapping, persistence ignorance, domain driven design, .NET framework

1 Introduction

With the introduction of two Microsoft's frameworks in the .NET framework 3.5 and 3.5 SP1, began the ORM hype in the .NET world. This was the impetus behind our research in this field. We decided to put the new frameworks to the test, comparing them with the already established NHibernate.

Our goal was to provide classification criteria, upon which an analysis could be done. In this paper, we will present some criteria that best resemble the qualities of a good ORM framework. Based on these criteria, we will conduct a comparison and analyze its results.

The paper is organized as follows: Section 2 discusses object-relational mapping with an emphasis on patterns and persistence ignorance. Section 3 presents three persistence frameworks that work on the .NET framework. Section 4 introduces comparison criteria, which are used in section 5, where we compare and analyze the results. Section 6 concludes the paper.

2 Object-relational mapping

Object-relational mapping is defined as 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 . ORM, in essence, works by transforming data from one representation to another. The main reason to use an ORM is to bridge the gap between the objects and database tables, called an impedance mismatch.

2.1 Object-relational impedance mismatch

When developing object-oriented information solutions based on relational databases, we quickly noted the differences between those two systems. This is known as the Object-relational impedance mismatch.

The first things we noticed were the different data types used in the databases. Although there are some standardized types, each database management system uses some unique types. For example, a `System.String` type in .NET has a few equal types in a database, like `char`, `nchar`, `nvarchar` or `text`.

The other obvious difference can be found in the nullable data types. As we know, database columns

can be set to accept null values. This is a problem in the object-oriented world, because not all languages support nullable primitive data types. This is not the case in the .NET framework, because it has offered support for the use of nullable primitive types since version 2.0. These types must be declared with a question mark (?).

Another difference is how both systems handle navigation. Relational databases use a combination of keys to determine the relationship between two rows. On the other hand, objects use references to navigate from one to another.

Finally, we have to mention inheritance as a fundamental object-oriented concept. It presents itself as a significant problem when trying to persist an inheritance hierarchy to a relational database.

In the next chapter, we will present how O-R mapping helps overcome these problems.

2.2 O-R mapping patterns

Patterns are a very important aspect of object-oriented programming. In this section we will present some of the patterns that are most commonly used in object-relational mapping. The patterns presented in this section are further explained in and .

Unit of Work - This pattern is important in the aspect of tracking changes in order to write them back to the database. Certainly, you could write back every change made to an object, but this would lead to lots of very small database calls. To avoid those unnecessary calls to the database, the Unit of work pattern is used. It keeps track of all the changes made to an object and then applies those changes when we are done.

Identity map - The Identity map pattern results from the previously mentioned O-R impedance mismatch. An identity map pattern keeps a record of all objects, to insure that each object is loaded only once. This ensures that you can compare two objects that represent the same database entry, even though they were loaded separately.

Lazy loading - When loading data from the database into memory, it is useful to load only the data needed. The lazy loading pattern is able to load related objects at runtime. This means that we keep only the minimum number of objects in memory and load further objects only when necessary. There are various implementations of this pattern. The most commonly used has a marker, which signals if an object or collection has been loaded or not.

Mapping associations - We already described the different handling of associations in a previous chapter. The association mapping pattern deals with this problem. It uses metadata to determine how to map one to many and many to many relationships between database tables. This pattern also implies the use of bidirectional associations within objects.

Mapping inheritance - Inheritance is a complex problem when using a relational database. Because managing inheritance is not something that a relational database was designed to handle. To date, three solutions have evolved on how to store an inheritance hierarchy in a relational database. The first solution is called the table per class hierarchy. It uses only one table to store the entire hierarchy. A discriminator column is used to determine, which class each entry refers to. Another solution is the table per concrete class. This means that we have to create a table for each non-abstract class. This solution is far more scalable than the first, but has some problems if the abstract class has a lot of relations. And finally there is the table per class hierarchy solution, which is the most scalable, but needs complex querying to retrieve entries.

2.3 Persistence Ignorance

Persistence Ignorance (PI) is a term that describes the coupling between the domain model and persistence framework. This term was introduced by Jimmy Nilsson in , when he argued with Martin Fowler about the term POCO. POCO is an acronym for Plain Old CLR Object and as such describes the ability to use regular classes in an O-R framework. They decided that POCO does not accurately represent the topic and therefore came up with the term Persistence Ignorance. PI consists of seven characteristics, which an ORM framework should avoid:

- Inherit form a certain base class.
- Instantiate only via a provider factory.
- Use specially provided data types, such as for collections.
- Implement a specific interface.
- Provide specific constructors.
- Provide mandatory specific fields.
- Avoid certain constructs and usage of certain constructs.

Only when a model is decoupled from its ORM framework can we create quality solutions that are scalable, reusable and easily maintainable.

3 O-R mapping frameworks for the .NET Framework

In this chapter, we will introduce the most frequently used frameworks for O-R mapping based on the .NET framework. For each framework, we will provide a short description, highlighting its special features while also explaining how it manages mapping, and what kind of query language it provides.

3.1 LINQ to SQL

LINQ to SQL (LTS) is a dialect of LINQ which stand for “Language Integrated Query” and is a new innovative way to create queries in .NET programming languages.

LINQ is a set of APIs and language enhancements that allow developers to create strongly typed queries in programming languages. The new features introduced in C# 3.0 are: implicitly typed local variables, object and collection initializers, lambda expressions, extension methods and anonymous types . Standard query operators are introduced to allow more familiar queries to be created, compared to relational queries. As we can see in Fig. 1, LTS is only one of several dialects that allow developers to query different data sources.

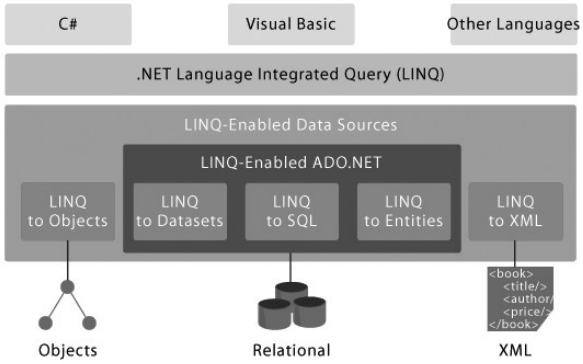

Figure 1. LINQ architecture

LTS manages the mapping between objects and tables with the use of an XML file and mapping attributes. These attributes are used to map classes and properties to the relational database, described in the XML file.

Including own custom classes in LTS has proven quite difficult. First of all, one needs to implement some specific interfaces that manage change tracking. Furthermore, mapping attributes and a starting point for queries must be provided by developers.

3.2 NHibernate

NHibernate is an open-source framework, which was ported from a Java framework called Hibernate to the .NET framework. Hibernate has been the most popular and widespread framework for O-R mapping in Java for years. The development of the .NET version of the framework began in 2004 and the first final version was released a year later . NHibernate is not purely the result of a code migration, because it adds specifics of the .NET framework.

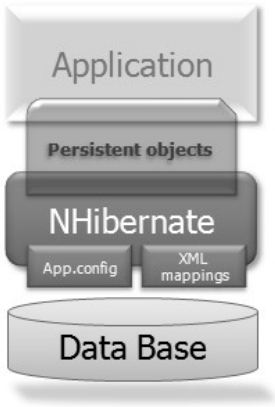

Figure 2. NHibernate architecture

Fig. 2 demonstrates a quick overview of the architecture. We can see that the application doesn't depend on the persistence framework. Furthermore, we can see that the framework relies only on a configuration file like `App.confing` and XML mappings. These mappings provide information on how to map classes and properties to tables and columns. These configuration files are also used to create SQL queries and commands.

NHibernate offers two ways to create queries . The first is the Hibernate Query Language. Although it is very similar to SQL, it does not query the relational, but the entity model. The other method to create queries is the Criteria Query API. This method is based on a more object-oriented method, because it uses objects and methods to create queries.

3.3 ADO.NET Entity Framework

The last, and also newest, framework we would like to present is the ADO.NET Entity Framework. Although it was first presented at a conference in 2006, its final version wasn't released until the .NET framework 3.5 SP1 in August 2008. The ADO.NET Entity Framework introduces some new features to the world of O-R mapping. First of all, it uses an Entity Data Model (EDM) that describes all of the entities and relationships in the domain . Next it builds on the `EntityClient` provider, which is similar to an ADO.NET data provider. The difference is that this provider works on EDM rather than the relational database model. Another very important component of this framework includes Object services. They enable the use of strongly typed objects as query results . All these components are presented in Fig. 3.

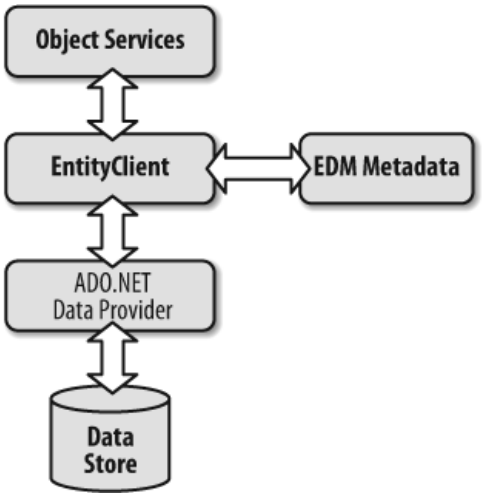

Figure 3. ADO.NET Entity Framework architecture

Besides the previously mentioned EDM, which is only one of the XML mapping files, the framework used two other XML files. One describes the relational data store and the other stores mapping information between the other two.

The Entity Framework offers two methods for querying data . The primary focus of its developers was on Entity SQL. This is a query language very similar to SQL. With the arrival of LINQ, the developers had to make a shift and adopt a new way of querying. A new dialect called LINQ to Entities was developed to offer the ability to query the EDM using LINQ.

4 Classification of comparison criteria

It is very important to choose the right criteria for a comparison. In our case, we did not include the more obvious criteria, like the ability to use stored procedures or transactions, because we think that those are standard characteristics for every ORM framework. In this section, we will present some of the more viable criteria with a special focus on persistence ignorance.

Graphical and supporting tools - For the first criteria, we will show what kind of support the framework offers when it comes to graphical or command line tools. In the time of modern IDEs, we have grown accustomed to having a graphical representation of the domain model. This is very important in the case of O-R mapping, because all three frameworks use XML as metadata and it is important to have a visual representation of that data.

Inheritance mapping - As we mentioned in a previous section, it has proven difficult to map an inheritance hierarchy to a relational database. For this criterion, we will research the ways the selected frameworks support inheritance mapping. We will also research which inheritance mapping patterns are supported by each framework.

Support for multiple RDBMS - Today there are a multitude of relational database management systems on the market. Because of this, an ORM framework must offer support to integrate these systems. The purpose of this criterion is not to evaluate the number of supported database systems, but rather to research how databases can be incorporated into the framework.

Caching - Caching is well known in computer hardware, where processors use a cache to store the last accessed memory blocks. In our case, a cache would hold objects that represent database entries in memory to enable better performance of queries.

Locking - When accessing data from a database we encounter the problem of data consistency. The problem is that we do not know if an entry has been changed after it has been acquired. To counter this problem, two alternatives have been developed. These are called optimistic and pessimistic locking. The first does not physically lock an entry, but deals with an inconsistency when saving data back to the database. The other method uses a physical lock to prevent changes to an entry while it is in use . We will research what kind of locking mechanism is being used by the compared frameworks.

Persistence ignorance - We already presented persistence ignorance explicitly in this paper. Because we find that the seven criteria describe the decoupling of the domain model from the ORM framework well, we have chosen to include these criteria in our comparison. The result will give us a finding on how much effort is needed to join a custom domain model with an ORM framework.

Efficiency - The final criterion we have chosen is framework efficiency. In this criterion, we will research the performance and memory efficiency of each framework. For the purpose of this criterion, we will build a simple test application, which will measure the time needed to accomplish queries and CRUD operations.

5 Analysis and comparison of O-R mapping frameworks

We conducted our analysis and comparison based on LINQ to SQL, released with the .NET framework 3.5, NHibernate 2.0.1. GA and ADO.NET Entity Framework released with the .NET framework 3.5 SP1. As a common data source for all three frameworks we chose Microsoft's Adventureworks database. This is a sample database for Microsoft's SQL Server, which represents a bike manufacturing and retailing company.

Graphical and supporting tools - While analyzing the frameworks we discovered that only the two Microsoft frameworks offer a graphical representation. This is a huge disadvantage for NHibernate. Despite scouring the internet, we were unable to find a tool to visualize NHibernate's

mapping files. The most what we can expect from this framework is IntelliSense support inside the Visual Studio development environment for writing and validating xml mapping files.

The use of a tool is unavoidable with the Entity Framework. The mapping file, which consists of three schemas, is not manageable by hand. On the other hand, the simplicity of NHibernates mapping files has its advantages.

While providing a high quality graphical interface included in Visual Studio 2008, both Microsoft frameworks also feature command line tools for generating mapping files form a database or classes from a mapping files.

Inheritance mapping - We discovered that all compared frameworks offer some kind of inheritance mapping support. From the described scenarios on how to map inheritance, only NHibernate and ADO.NET Entity Framework support all three. LINQ to SQL supports only the table per class hierarchy solution.

Support for multiple RDBMS - Support for multiple RDBMS is dependent on the frameworks architectures. NHibernate and Entity Framework build on ADO.NET data providers to connect to a data source. While NHibernate uses standard data providers, the ADO.NET Entity Framework needs those to be Entity Framework enabled . This means that the current providers need to be updated for this framework. At the time of writing, many data providers have been updated to support the new framework.

LINQ to SQL does not offer support for databases other than the SQL Server. The reason for this is that it has been developed on Microsoft's SQL Server data provider, known as SqlClient.

Caching - Caching is a weak point in Microsoft's frameworks, because neither of them supports it. Only NHibernate offers the ability to cache objects. To fully understand how NHibernate uses caching we have to look at its architecture. This framework offers two levels of cache. The first level cache is used by the session and identity map to store references of all objects. The second level cache is what we are interested in. This cache does not store entire objects, but rather uses a hash table to store objects properties.

When querying, the framework first checks if the object is located in the cache. If a result is found, an object is materialized from the hash table and returned. Otherwise the query is sent to the database and the results are then written to the cache.

Locking - When analyzing the frameworks, we learned that all of them use optimistic locking by default. This means that the frameworks handle concurrency exceptions when updating entries in the database. Pessimistic locking is available only in NHibernate. This allows developers to not only fully lock, but also to restrict access to a database entry.

Persistence ignorance - For evaluating the persistence ignorance criteria we decided to use three values. The first, which is represented by a tick (✓), means that the framework fulfills the condition. On the other hand, a cross (✗) means that a framework does not fulfill the criterion. We decided to also employ a third value (-), which is in the middle. It means that a framework only partially fulfills a condition.

The first framework to be evaluated using the persistence ignorance criteria was LINQ to SQL. This ORM framework performed well in the evaluation. It fails only in two criteria. The first is that it uses special classes for collections. Also, the persisted classes need to implement some interfaces in order to support change tracking. Because this is not mandatory we decided to assign it a middle value. The other weak point of this framework is that it uses mapping attributes and this is why it fails the last criterion in Tab. 1. Overall we can say that this framework provides a good mix between functionality and a weak dependency of the model.

NHibernate achieved the highest level of persistence ignorance in our comparison. This is not a surprise, if we consider that this framework is ported from Hibernate, which is known to provide a high level of PI. The reason for this is certainly the use of XML-files to describe the mapping. The framework only relies on these files to provide the O-R mapping. This leaves the domain model free of any unwanted constructs. Developers are free to choose any kind of collections they want. The only limitation of this framework, as far as PI is concerned, is the use of a default constructor and the mandatory keyword `virtual` on all properties. Furthermore, the use of the reserved keyword `readonly` is prohibited.

As previously concluded, LINQ to SQL achieved a moderate level of PI. Now we will discuss how its advanced version, the ADO.NET Entity Framework, tackled the PI criteria.

The first thing we noticed when researching the Entity Framework was that all entity classes derive from an `EntityObject` class. This is the default way the framework's generator creates classes from the entity data model. The other way to achieve persistence is the use of interfaces -- what Microsoft developers call IPOCO . That means POCO with the use of interfaces. Although this is certainly a way to provide persistence, it is not the way most developers will use it. Because of this, we decided to evaluate the first criterion with a middle score. The other deficits are the need to store an `EntityKey` object and lots of code to provide change tracking.

Table 1. Criteria persistence ignorance

Criterion	LTS	NH	EF
Inherit form a certain base class	✓	✓	-
Instantiate only via a provider factory	✓	✓	✓
Use specially provided data types, such as for collections	✗	✓	✗

Implement a specific interface	-	✓	✗
Provide specific constructors	-	-	✓
Provide mandatory specific fields	✓	-	-
Avoid certain constructs and usage of certain constructs	✗	✓	✗

LTS – LINQ to SQL, NH – NHibernate, EF – ADO.NET Entity Framework

Overall, we can say that NHibernate is the standard in persistence ignorance in the field of ORM frameworks. Neither LINQ to SQL or Entity Framework can reach it.

Efficiency - The efficiency of a framework is what most users will probably be interested in. For this criterion, we measured the time needed to load (TTL), perform a simple query (SQ), a complex query (CQ), a query with eager loading (EL) and create, update and delete operations (CUD). All the benchmarks were performed on a computer with 3.0 GHz, 3 GB RAM, Windows Vista SP1 and SQL Server 2008. As mentioned before, we used the Adventureworks database, from which we imported all of the tables. The results of these benchmarks are presented in Tab. 2. In all benchmarks, we used only the standard components of each framework with no caching or anything that could compromise our results. We also performed a reference measurement with the standard components of ADO.NET and SQL.

First, we performed a simple query (SQ) on one table that returned approximately 30.000 entries. To achieve realistic results we ran the query 100 times. At this point we have to mention the time to load (TTL). This is the time between the benchmark's start and the execution of the first query. This time has to be considered when only a few queries will be made. As we can see from Tab. 2, NHibernate has a very high TTL. The reason for this is the creation of the session object. Because this process takes up a lot of time it is recommended to do this only once in a lifetime of an application.

To return to the results of the simple query, we can see that the fastest framework for this task is LINQ to SQL, followed by NHibernate and ADO.NET Entity Framework. When comparing these results with our reference measurement, we see that LINT to SQL is approximately 20% faster than SQL. This can be brought back to the use of patterns, especially the identity map.

In the second run, we used a complex query (CQ), which included projections, joins, grouping and sorting. The results were similar to the previous test. LINQ to SQL is still in the lead, followed by NHibernate and the Entity Framework. Only the criteria query of NHibernate performed poorly in this test. When comparing these results with our reference, we discovered that now two frameworks had performed this task quicker than SQL. These are LINQ to SQL and NHibernate with HQL.

To demonstrate eager loading, a third test was devised. We queried the employee table and told the frameworks to eager load the manager, which is a relation to itself. Because LINQ to SQL threw an exception saying that we cannot eager load a relation to the same table, we measured only the other two frameworks. We were not surprised to see that NHibernate completes this task faster than the Entity Framework.

The last benchmark we ran was to determine the performance of execution on basic operations like insert, update and delete (CUD). The test created and filled an object with data, inserted it, updated some properties and updated those in the database. Finally we deleted the entry from the database. We repeated this procedure 10,000 times to achieve a representative average value. The results show a surprisingly poor performance of LINQ to SQL. The framework that dominated so far, suddenly finished last. The fastest framework, NHibernate, was almost 6 times faster and even the second placed Entity Framework was 2.5 times faster.

Table 2. Framework efficiency

	LTS	NHibernate		Entity Framework			UM
		HQL	CQ	ESQL	LTE	EC	
TTL	140	2700	2700	230	230	-	[ms]
SQ	27,4	49,5	102,4	104,0	104,0	-	[s]
CQ	28,5	30,6	73,0	44,6	35,0	33,8	[s]
EL	-	50,2	49,9	100,0	102,0	-	[s]
CUD	19,5	3,3		7,5			[ms]
CPU	65,0	93,3	94,2	98,6	98,5	-	[%]
RAM	31,2	60,3	60,8	61,9	64,8	-	[MB]

LTS – LINQ to SQL, HQL – Hibernate Query Language, CQ – Criteria Query, ESQL – Entity SQL, LTE – LINQ to Entities, EC – Entity Client

Finally, we would like to present some data that we gathered during our benchmarks. We measured the average CPU usage time and memory usage, which are shown in Tab. 2. We can see that the best performing framework, which is LINQ to SQL, also uses the minimal amount of system resources. This is impressive, because it shows that although it is already the fastest framework it still has not reached its maximum potential. The other two frameworks are at the same level with a high CPU usage and a moderate memory acquisition.

To sum up, we discovered that LINQ to SQL proved itself to be the fastest framework. The only weak point was its poor result in the CUD test. It is closely followed by NHibernate, which scored good results in all of the performed benchmarks. The ADO.NET Entity Framework was a disappointment. These benchmarks have proven that the framework is fresh on the market and therefore not mature enough for a serious business application.

6 Conclusion

The choice of a persistence framework is vital for assuring the quality of an information system. This

paper introduced object-relational mapping as a suitable solution to overcome the impedance mismatch. We covered three main topics: object-relational impedance mismatch, object-relational patterns and persistence ignorance. We also described the three most frequently used ORM frameworks on the .NET framework. Based on this, we defined comparison criteria and evaluated all three frameworks.

Based on this comparison, we can conclude that NHibernate has proven itself to be the best. It proved itself with a high level of PI and consistently good performance in our benchmarks. Although it uses simple xml files to describe the mapping, it doesn't include a graphical interface, which is a major weak point of this framework. LINQ to SQL has presented itself as surprisingly good. Its solid points are a graphical interface, a moderate level of PI and good benchmark results. On the other hand, we have the ADO.NET Entity Framework, which is the more advanced ORM framework. Based on our comparison we can say that this framework is not mature enough for business applications. Although it comes with an advanced mapping system and powerful GUI, these factors alone cannot outweigh its weaknesses.

References

[1] J. Nilsson: **Applying Domain-Driven Design and Patterns**, Addison-Wesley, 2006

[2] C. Bauer, G. King: **Java Persistence with Hibernate**, Manning, 2007

[3] E. Evans: **Domain-Driven Design: Tackling Complexity in the Heart of Software**, Addison Wesley, 2003

[4] M. Fowler: **Patterns of Enterprise Application Architecture**, Addison Wesley, 2002

[5] V. P. Mehta: **Pro LINQ Object Relational Mapping with C# 2008**, Apress, 2008

[6] C# 3.0, The Evolution Of LINQ And Its Impact On The Design Of C#, available at `http://msdn.microsoft.com/sl-si/magazine/cc163400(en-us).aspx`, Accessed: January 2009

[7] F. Marguerie, S. Eichert, J. Wooley: **LINQ in Action**, Manning, 2008

[8] LINQ to SQL, Code Generation in LINQ to SQL, available at `http://msdn.microsoft.com/en-us/library/bb399400.aspx`, Accessed: January 2009

[9] Hibernate, Relational Persistence for Java and .NET, available at `http://www.hibernate.org/`, Accessed: December 2008

[10] NHibernate Forge, NHibernate 2.0 Architecture, available at `http://nhforge.org/wikis/reference2-0en/architecture.aspx`, Accessed: February 2009

[11] P. H. Kuate, T. Harris, C. Bauer, G. King: **NHibernate in Action**, Manning, 2009

[12] J. Lerman: **Programming Entity Framework**, Manning, 2009

[13] A. Adya, J. A. Blakeley, S. Melnik, S. Muralidhar: **Anatomy of the ADO.NET Entity Framework**, International Conference on Management of Data, 2007, pp. 877 – 888

[14] J. Blakeley, D. Campbell, S. Muralidhar, A. Nori: **The ADO.NET Entity Framework: Making the Conceptual Level Real**, SIGMOD Record, 2006, Vol. 35, No. 4, pp. 31-38

[15] MSDN, EDM Generator, available at `http://msdn.microsoft.com/en-us/library/bb387165.aspx`, Accessed: February 2009

[16] MSDN, Understanding Concurrency Control, available at `http://msdn.microsoft.com/en-us/library/ms378709(SQL.90).aspx`, Accessed: February 2009

[17] ADO.NET team blog, Entity Framework-Enabled Providers Lists, available at `http://blogs.msdn.com/adonet/archive/2009/01/26/entity-framework-enabled-providers-lists.aspx`, Accessed: February 2009