

White Paper



Microsoft Data Access Technologies

SWOT Analysis

Avani Dave, Sudhanshu Hate

Abstract

While architecting .NET applications, an architect has to make a choice from several data access technologies available on Microsoft platform. This paper compares and contrasts various data access technologies available from Microsoft such as ADO.NET, LINQ, Entity Framework, and WCF Data Services for their suitability in various scenarios.

Table of Contents

Overview	4
Intent and Target Audience	4
Introduction	4
Challenges in Architecting DAL	4
Efficient Queries (Faster Read/Write)	4
Security Threat: SQL Injection Attacks	5
Writing Database-independent DAL	5
Domain/Business to Relational Mapping (Handling Impedance Mismatch)	5
Handling Transactions in Distributed Scenarios	5
Building Highly Interoperable DAL by Exposing Data as Resource	5
ADO.NET	6
Overview	6
Architecture	6
Using DataSet:	6
Using DataReader:	7
Key Characteristics	8
Familiarity with ADO	8
Support for N-tier application development model	8
XML Support	8
Framework Compatibility	8
Strengths	8
Weaknesses	9
Opportunities	9
Threats	9
SUMMARY	11
LINQ to SQL	12
Overview	12
Object Relational Modeling	12
Architecture	12
Key Features	13
Generating Object Model from Database	13
Stored Procedure and User-defined Functions Support	13
Delay Loading Support	13
Framework Compatibility	14
Strengths	14

Weaknesses	14
Opportunities	14
Threats	15
SUMMARY	15
Entity Framework	16
Overview	16
Architecture	16
Entity Data Model (EDM)	16
Object Context	17
Store-specific Provider	17
Data Access and Representation	18
EntityObject Based	19
Plain Old CLR Objects (POCO) Based	19
Self-Tracking Entities	19
Patterns to load related Objects	21
Lazy Loading (Deferred loading)	21
Eager Loading	21
Explicit Loading	21
Key Features	22
Model First approach (Top-down)	22
Generating Models from Code (Bottom-up)	22
Mapping Stored procedures	22
Change tracking	23
Identity Resolution	24
Framework Compatibility	24
Strengths	25
Weaknesses	25
Opportunities	25
Threats	25
SUMMARY	26
WCF Data Services (erstwhile Astoria or ADO.NET Data Services)	27
Overview	27
Architecture	27
Key Features	28
Visibility Control	28
Authentication	28
Interceptors	29

Service Operations	29
Summary of Latest Features	30
Framework Compatibility	30
Strengths	31
Weaknesses	31
Opportunities	31
Threats	31
Conclusion	32
Appendix A Performance Benchmarking	33
Test tools and strategy	33
Machine Configuration	33
Client Machine Configuration:	33
Database Server Configuration:	33
Test Results	34
SUMMARY	36
Appendix B Entity Framework Vs NHibernate	37
NHibernate Overview	37
Further Reading	39
Bibliography	39
About the Authors	40
ACKNOWLEDGEMENT(S)	41

Overview

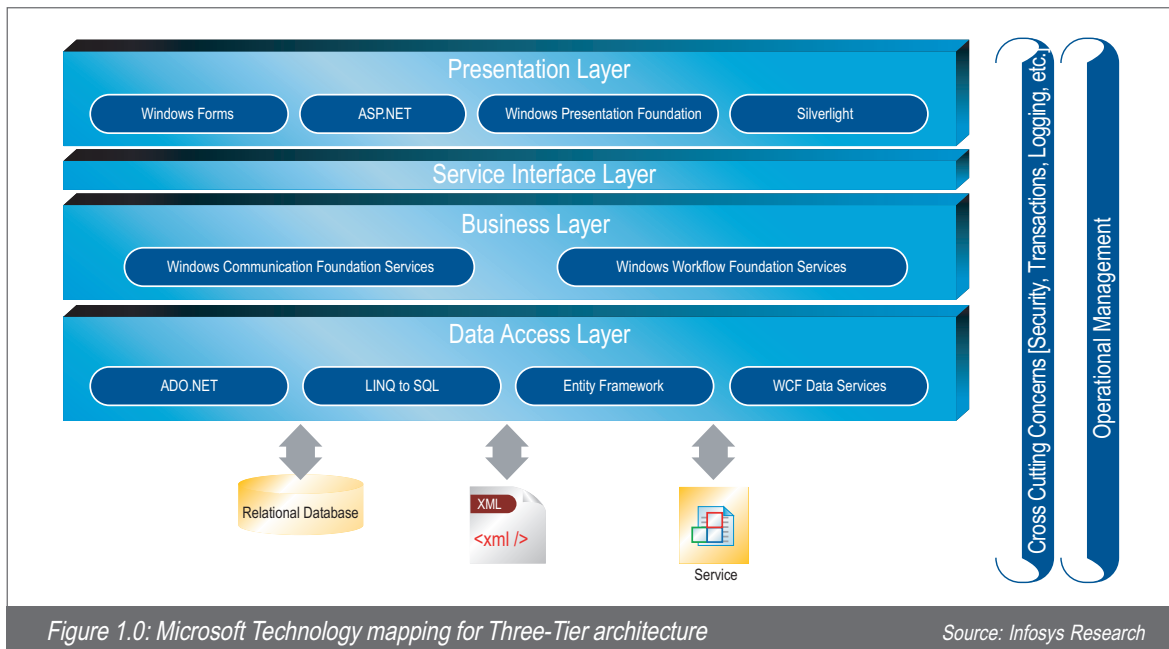
Layered application architecture provides a foundation for building applications with high performance, scalability, flexibility, reliability, and code reuse. An essential part of the classical n-tier architecture is Data Access Layer (DAL) which manages transportation of data to and from persistent storage. DAL helps in separating Data Access logic from Business and Presentation logic. It also ensures a central location for all the data access logic. This central location simplifies maintaining the application logic since, the code for specific database operation is easy to locate and required changes could be made at one place.

Intent and Target Audience

This paper is intended for architects and senior developers in Enterprise. The paper covers the Microsoft Data Access Technology evolution starting with ADO.NET, LINQ to SQL, to Entity Framework and WCF Data Services and how each of these technologies has helped to overcome the limitations from the predecessors. The paper uses SWOT Analysis to provide perspectives on each of these technologies mentioned.

Introduction

Microsoft .NET framework supports many data access technologies such as ADO.NET, LINQ, Entity Framework, and WCF Data Services which can be used in combination or isolation. Figure 1.0 below depicts the classic three-tier architecture with Microsoft technology mapping to Presentation Layer, Business Layer, and Data Access Layer.



Challenges in Architecting DAL

Following are some of the common challenges encountered in selecting appropriate technology while designing DAL for Online Transaction Processing (OLTP) applications.

Efficient Queries (Faster Read/Write)

Data Access Layer needs to be written in the most efficient and optimized way in order to minimize delay and maximize the application performance. Moreover, architects also need to ensure minimal database roundtrips and prevent any kind of blocking operations at database server.

Security Threat: SQL Injection Attacks

The embedded SQL Statements (inline queries) are prone to SQL injection attacks [18]. SQL injection attack occurs because the input provided to SQL statement is not strongly typed or incorrectly filtered string literal is provided as an input. SQL injection attack can potentially cause the Database unavailability.

Writing Database-independent DAL

With writing DALs having embedded SQL statements (inline queries), one of the key challenges is that with every underlying database, (for example, ORACLE, SQL, DB2) the query syntax is different.

For example, ORACLE SQL syntax is different from SQL SERVER SQL to DB2 SQL.

To some extent, ANSI SQL made an attempt to overcome this challenge by implementing a uniform syntax which can be ported across databases. Nevertheless it lacked comprehensiveness as each database implementation has its native functions which are used extensively while writing SQL. Hence, developers prefer to write database-specific SQL than ANSI SQL.

Domain/Business to Relational Mapping (Handling Impedance Mismatch)

In many applications, the domain/business objects are modeled based on Object Oriented (OO) concepts such as inheritance, abstraction, and polymorphism while the relational database schema is conceptualized based on various relationships (oneto-one, one-to-many) between entities, and implemented (normalized) from the standpoint of enabling faster access and ease of manipulation.

Impedance mismatch arises when you intend to map OO based objects with relational tables. Developers have to write huge amount of translation code to handle such mismatch between the OO-based types and relational database-specific types.

Handling Transactions in Distributed Scenarios

Handling transactions in applications which span across different physical locations requires a special attention as there is increased risk of duplication of data and concurrency issues. It has to be ensured that database is in consistent state irrespective of its location. (Distributed scenarios come with an advantage of distributed load).

Building Highly Interoperable DAL by Exposing Data as Resource

Service Oriented application needs data to be exposed and consumed as service. Exposing database entities as services and ability to drill down to various records and relationships is one of the common functionality needed in building Resourcecentric (REST- Representational State Transformation) architecture.

Apart from the above listed; there are several other challenges such as connection pooling, transaction management, data synchronization, execution in batch etc. However, we feel each of these are consistently handled by each of the Data Access technologies we have discussed in the paper. Hence, we would like to keep focus of our evaluation on the primary challenges described above.

ADO.NET

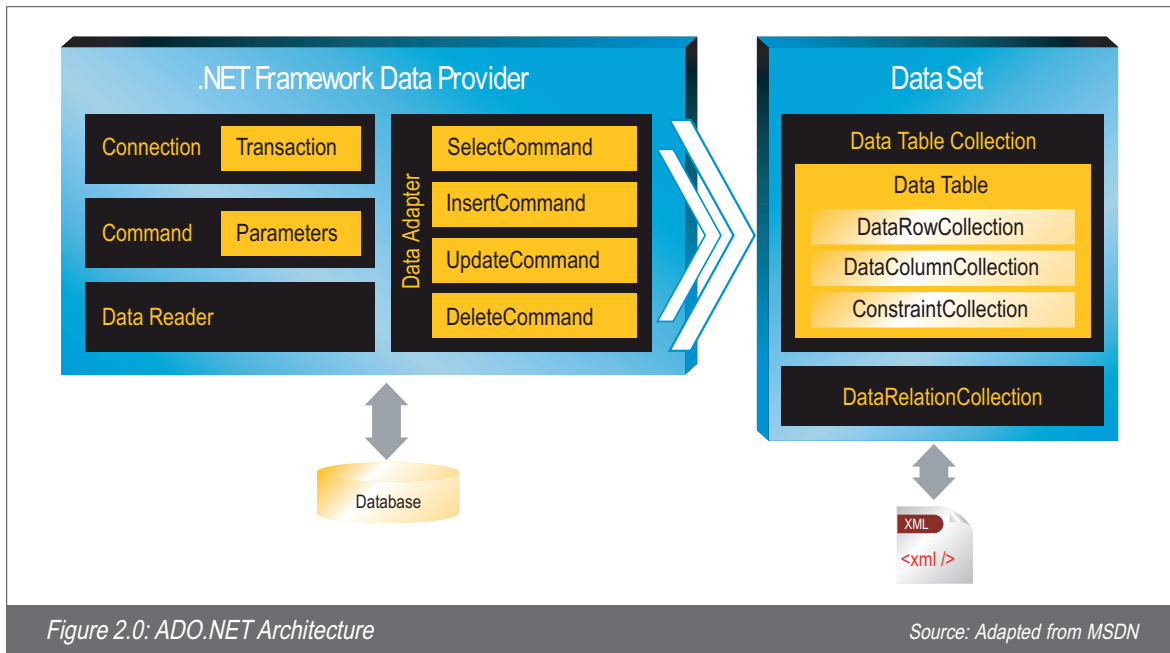
Overview

To cater to the needs of distributed architectures, applications are designed to be loosely coupled. Distributed applications use XML to pass data over network and communicate through HTTP, which implies that the state between two requests is not retained. This programming technique is largely different from the typical client-server scenario addressed by ADO, where the connection remains open throughout the life of the application.

ADO.NET framework was built as an entirely new programming model for data access to leverage the existing ADO features but at the same time, overcome the limitations of ADO.

Architecture

The following figure 2.0 illustrates various components of the ADO.NET architecture:



As shown in the figure above, DataSet and the Data Provider are two central components of ADO.NET.

DataSet being independent of data source, can be used with multiple and differing data sources. DataSet consists of DataTable collection, in turn each DataTable consists of DataRow, DataColumn, and Constraint collections.

Data Provider is a set of components such as Connection, Command, DataReader, and DataAdapter. These components are explicitly designed for data manipulation and fast, forward-only, read-only access to data.

Using DataSet:

```
public DataSet getCustomer(string customerID)
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        DataSet userDataset = new DataSet();
        SqlDataAdapter myDataAdapter = new SqlDataAdapter(
```

```

        "SELECT au_lname, au_fname FROM Customers WHERE cust_id = @cust_id",
        connection);

//Add Parameters
myDataAdapter.SelectCommand.Parameters.Add("@cust_id", SqlDbType.VarChar, 50);
myDataAdapter.SelectCommand.Parameters["@cust_id"].Value = customerID.Text;

//Fill the data in DataSet
myDataAdapter.Fill(userDataset);
}
}

```

Using DataReader:

```

public void getCustomer()
{
    // Open connection to the database
    SqlConnection con = new SqlConnection(ConnectionString);
    con.Open();
    // Set up a command with the given query and associate with current
    //connection
    string CommandText = "SELECT FirstName, LastName FROM Employees";
    SqlCommand cmd = new SqlCommand(CommandText);
    cmd.Connection = con;
    // Execute the query
    SqlDataReader rdr = cmd.ExecuteReader();
    while (rdr.Read())
    {
        Console.WriteLine("\t{0}\t{1}", rdr.GetString(0), rdr.GetString(1));
    }
    con.Close();
}

```

The following table lists the functionality of each Data Provider component:

Object	Functionality
Connection	Provides connectivity to a data source.
Command	Enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.
DataReader	Provides a high-performance stream of data from the data source.
Data Adapter	Uses Command objects to execute SQL commands at the data source to load the DataSet with data, and also reconciles changes made to the data in the DataSet back to the data source.

Table 1.0: .NET framework data provider

Key Characteristics

Familiarity with ADO

Though a new programming model, ADO.NET offers a number of features of ADO, it retains similarity with ADO to the large extent. The ADO programmers do not have to learn ADO.NET from scratch.

Support for N-tier application development model

ADO.NET provides a solution for disconnected n-tier programming through DataSet. DataSet is a memory resident data representation, which provides relational programming model regardless of the source of the data.

XML Support

To provide a convenient format for transferring the contents of the DataSet to and from remote client, ADO.NET allows the creation of XML representation of a DataSet, with or without its schema. In the XML representation of DataSet, the data is written in XML and the schema is written using Schema Definition Language.

Framework Compatibility

ADO.NET was the first data access technology to be released as part of .NET framework and is supported across all versions of .NET framework and Visual Studio.

Parameter	Compatibility
.NET Framework	1.0, 1.1, 2.0, 3.0, 3.5, 4.0
Visual Studio	Microsoft Visual Studio.NET, 2001, 2002, 2003, 2005, 2008, 2010

Table 2.0: ADO.NET Framework Support

Strengths

1. ADO.NET offers best performance amongst its peers (LINQ to SQL, Entity Framework, WCF Data Services) - Refer Appendix A
2. ADO.NET objects can be readily bound to UI controls.
3. XML support ensures the interoperability in terms of transfer of content between DataSet and remote client.
4. Support for disconnected mode enables high scalability and availability.

5. Tried and Tested technology since early 2002.

Weaknesses

1. Respective to each data source, developers have to write data source-specific DAL resulting in reduced maintainability.
2. Availability of data providers defines the number of data sources application can work with.
3. Relational to Object Oriented impedance mismatch has to be handled explicitly by writing lot of custom code to translate relational data entities into Object Oriented structures. We have observed custom utilities/tools have been created to automate object relational mapping but ensuring synchronization of code with changing designs is one of the big challenges with automated utilities.
4. Lacks conceptual modeling or designer tool which can be used during object relational mapping.
5. Lacks SQL queries-specific IntelliSense support in Visual Studio to ensure correct syntax.
6. Does not support out-of-the-box COM interoperability.

Opportunities

1. ADO.NET is best suited for building lightweight thin or thick client applications.
2. ADO.NET can be used to build Service Oriented Architecture because of built-in support of XML and its lightweight characteristics.
3. ADO.NET can be used to work with multiple databases through respective database providers.

Threats

1. Lack of Out-of-Box (OoB) Object relational capability, has created opportunities for other data access frameworks, as described in the following sections.
2. Prone to SQL injection attacks due to inline queries, has created opportunities for Language Integrated Query (LINQ) capabilities.

While building .NET applications, it is important to understand the differences between DataSet and DataReader and effectively use them in appropriate scenarios.

The following table compares DataReader and DataSet and lists the recommended scenarios:

	DataReader	DataSet
1	DataReader obtains data from database through Data Provider as a connected stream. DataReaders cannot be created without connectivity to database.	DataSet provides consistent relational programming model irrespective of underlying database. It uses Data Provider specific to the database to post updates to the database. DataSets can also be loaded from XML file or created without any connection to the database.
2	DataReader is lightweight and fast	DataSet is relatively heavy as it stores the complete Table metadata such as columns, table relations, data including copy of original and changed data etc.
3	Useful only in a connected architecture. For example, to pay credit card bill through online channel	Can be used in a disconnected architecture or offline scenarios. For example, to share a checked-out basket with friend or colleague in online shopping scenario.
4	Cannot persist data	Can persist data in a disconnected cache.
5	Used to retrieve read-only and forward-only stream of data from the database	Can retrieve, update, filter, sort, search and scroll through the data
6	Cannot access data from more than one table at a time. Cannot retrieve multiple tables and associated relationships.	It is possible to access data from more than one table. Multiple tables and associated relationships can be part of DataSet.
7	OoB cannot serialize data to XML and hence not suited for Service-oriented applications. However, data can be read from DataReader and converted into XML format.	OoB integration support with XML and, hence, can be used to build applications, which needs to exchange XML in Web services scenarios.
8	Recommended when you want to retrieve large number of read-only records from database and bind it to grid or populate dropdown list. However, if the same data is required to bind to three read-only controls say, dropdowns, then DataSet is recommended over DataReader as with DataReader, the same query would hit to database three times because DataReaders are forward only. For example, list last ten or hundred transactions on particular account.	Recommended when you are interacting with data dynamically or combining data from more than one table. For example, merging checked-out baskets from two or more shopping sites to do data analysis.
9	Not recommended when DataReader needs to be passed between different tiers. However recommended practice is it to translate DataReader into Plain Objects and then move around.	Not recommended for building applications that needs to be highly performing.

Table 3.0: DataReader VS DataSet

.NET Framework 3.5 and 4 comes with data provider for ODBC, OLE DB, SQL Server, and Oracle. You can use it to connect to any data source such as ORACLE, DB2, or SQL Server.

Note

Though Oracle Provider for .NET (System.Data.OracleClient) is available as deprecated assembly in .NET 4, Microsoft is going to completely stop supporting it in future versions and recommends using providers from third parties such as ORACLE or Data Direct [1]. System.Data.OracleClient assembly is marked as deprecated in .NET framework 4, which means applications using it will compile but throw warnings [5].

ORACLE and Data Direct provides ORACLE provider for .NET which is optimized for working with ORACLE database. These providers have more features as against .NET framework ORACLE Data Provider.

SUMMARY

ADO.NET supports executing database logic through writing inline queries and database stored procedures. As mentioned in the challenges, inline queries, if not written appropriately, induce risk of SQL injection attacks.

ADO.NET interacts with database through provider mechanism and most of the times, inline queries need to be changed to adapt to respective database provider and thus is not a natural candidate for implementing database-independent DAL code.

Out-of-the box (OoB), ADO.NET does not have any solution to impedance mismatch and developers have to write code to translate business/domain objects to database-friendly entities and vice versa. However, manual approach provides benefit of highest flexibility.

Simplicity and no baggage, makes it lightweight, performant, easy to learn and adopt. It is suitable for most of the thick client (for example Windows Forms-based Payroll application), thin client (ASP.NET-based applications such as leave applications) LOB applications, or Service Oriented (B2C or B2B integration scenario such as Intra or Interbank fund transfer) in enterprise scenarios.

The weaknesses of ADO.NET such as the need to write database-specific queries; need to manually translate business objects to database objects (impedance mismatch or object to relational translation) created an opportunity for a framework like LINQ (Language Integrated Query) as explained in the following section.

LINQ to SQL

Overview

The component of LINQ family which provides query capabilities to interact with relational database is LINQ to SQL. LINQ to SQL creates a thin layer of abstraction (called object model) on top of the underlying SQL server database and allow us to write LINQ queries against LINQ to SQL classes.

Reiterating the impedance mismatch problem as this is one of the most complex but important and relevant problems in this part of the paper:

“Database system organizes data in the form of Relational Table(s), and rows whereas business applications are built using higher-level object oriented programming constructs such as classes, structures etc. Extracting data from relational table and translating it into business-friendly objects by implementing object relational mapping at DAL needs significant amount of custom code in the form of various business, data entities, and associated translations”.

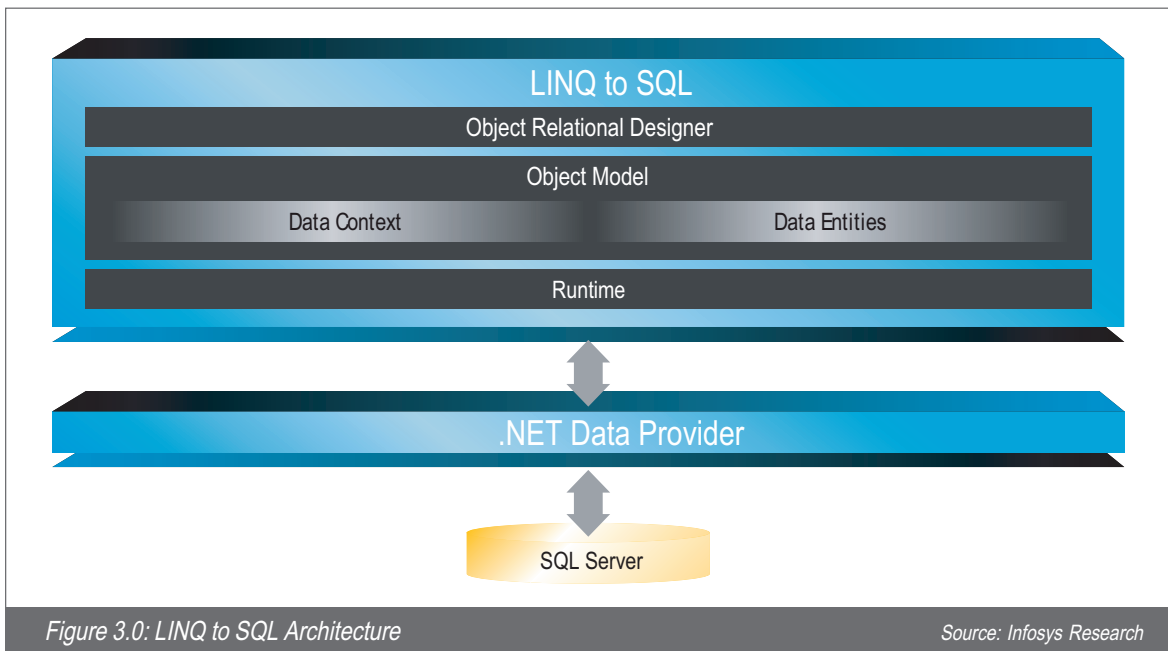
Object Relational Modeling

Object relational mapping tools connect to the database, read the database schema and establish the communication between object and data model by generating a persistence layer. Persistence layer consists of mappings between objects and database tables, single row create, read, update, delete queries, and calls to stored procedures.

Using Object Relational Mapping (ORM) framework has numerous advantages over the manual coding technique. ORM framework simplifies the development process by having out-of-box support for features such as transaction, key, cache management, and navigation of related objects. The effort and time spent on development is saved as lot of code is autogenerated, which otherwise would have to be written manually.

Architecture

Key components of LINQ to SQL are the Object Model, Object Relational Designer, and a run time.



Object Relational Designer - LINQ to SQL provides O-R mapping visual designer which represents the Object Model. It also provides a toolbox using which you can drag-and-drop and create entity classes, associations, and inheritance.

Runtime - LINQ to SQL Runtime translates the query into a SQL statement and sends it to the .NET Data Provider for further execution.

Object Model is the major component of LINQ to SQL architecture. Object Model comprises Data Context and Data Entities.

- Data Entities are the partial classes generated by LINQ to SQL having one-to-one mapping with database columns. Each entity class has one or more members that map to columns in the table or view. The table relationships are depicted through associations.
- Data Context is at the heart of LINQ to SQL. It plays a key role in variety of operations. The following are the main functions of LINQ to SQL data context.

1. **Connection Management and Database mapping**

DataContext defines connection to the database. It also manages mapping of the data entities with SQL database tables and provides the ability to fill the objects into generic collection type called Table<>. Apart from mapping tables, DataContext also creates methods which map to Stored Procedures and User Defined functions. After the evaluation of mappings, DataContext translates the expression trees into SQL Server consumable structures.

2. **Object Identity management and change tracking**

Change tracking is an important function needed in multi user update scenarios. Custom building this functionality is challenging and error prone. However LINQ to SQL DataContext provides this OoB.

DataContext does this by logging each retrieved row in an identity table by its primary key. On retrieval of the same row, the DataContext checks the existence of the corresponding object in the local cache and returns the same if it already exists.

Retention of original value of an object allows the client to make changes to his/her own local set of values irrespective of the changes made by other users. Concurrency issues are handled when changes are submitted to the database.

DataContext maintains both the original as well as the changed value of the property.

The changes made in the objects are maintained in the internal cache till the time they are not submitted to the database.

In addition to changes made in the object properties, the change tracking services of DataContext also tracks the changes made in object associations. When the changes are submitted to the database, context compares original values of the object with the current values. If the values differ, the necessary queries are created and executed on the SQL server.

Key Features

Generating Object Model from Database

LINQ to SQL can connect to SQL server database and facilitates the user to create an ObjectModel by dragging and dropping tables from the server explorer.

Stored Procedure and User-defined Functions Support

LINQ to SQL supports stored procedures and user-defined functions by generating corresponding methods in DataContext. You can drag and drop the stored procedure or user-defined function from the server explorer which creates a method in the DataContext; then calling a stored procedure or user-defined function is as easy as calling a method.

Delay Loading Support

LINQ to SQL lets you specify whether to delay/lazy load the properties of an entity on accessing the entity first time. This feature gives us an option to avoid performance overhead by loading heavy columns from the database.

Framework Compatibility

LINQ to SQL was introduced first time as part of .NET framework 3.5

Parameter	Compatibility
.NET Framework	3.5, 4.0
Visual Studio	Microsoft Visual Studio 2008, 2010

Table 4.0: LINQ to SQL Compatibility

Strengths

1. With LINQ to SQL, there are no inline queries for querying the database. LINQ to SQL queries are well-integrated with the programming language. This helps in avoiding SQL injection attacks.
2. Uniform syntax for working with various data sources like XML, Object Arrays or Relational stores like SQL Server.
3. C#, VB.NET Integrated language simplifies querying Data store without needing to learn database specific query language like T-SQL for SQL Server or PL-SQL for ORACLE.
4. Improves application development productivity by providing Intelli-sense, compile time syntax checking and debugging support. The Type-checking feature of Language integration makes sure that the queried collections and entities are always strongly typed and there is no need to write extra custom code to transform database results in to C# objects as in the case of ADO.NET.
5. A visual designer shows data entities and relations amongst them helping easy comprehension of the database.
6. Auto-generated object model can be used as domain model if the domain entities directly correspond to data entities. This helps in improved productivity. A single-point access to interact with the database via context. This helps in maintenance of code.
7. Lazy loading support for related entities help in performance improvements.
8. Supports compiled queries and stored procedure to enhance the overall performance of database operations.

Weaknesses

1. LINQ to SQL works only with Microsoft SQL Server database. Understandably community and other vendors are working towards creating providers for other databases such as ORACLE, MySQL, etc. [10] but still this is nascent at this point of time.
2. The Object Model-generated entities are tightly coupled with the database columns (one-to-one mapping) so in most of the cases, data entities cannot be reused as domain entities.
3. From main object, navigation to foreign key, reference objects is not possible. The object tree as a whole cannot be modified and saved in database and each entity needs to be dealt independently.
4. Complex types can be defined as results from multiple tables in a single class.
5. No support for Model First approach. LINQ to SQL supports only bottom-up design i.e. database to object oriented and not the other way.
6. Incorporating the database changes after the creation of Data Model is not smooth.

Opportunities

Use of LINQ to SQL is best suited in the following scenarios:

1. The application is fairly simple and the domain objects are similar to data objects.
2. Rapid application development against Microsoft SQL Server and want to optimize the query performance using compiled queries.
3. If same hierarchy and associations to be maintained in the ORM solution as that of database.

4. Build quick pilots to showcase basic functionality in the application.

Threats

1. Lack of support for providers for other databases is a major concern with LINQ to Database.
2. Entity Framework described ahead provides much improved object relational capability as against LINQ to SQL.
3. Microsoft is not making significant enhancements in this technology, just enhancing to ensure support to existing customers. However, LINQ to SQL will continue to exist and being supported [3].

SUMMARY

LINQ to SQL overcame several shortcomings of ADO.NET by providing single uniform type safe query language to query relational (SQL Server) and non relational stores (Arrays, Collections, Lists etc.) in object oriented manner without the need to write inline SQL queries reducing the risk of security attacks like SQL injection.

To some extent, it also helped to overcome the issues from impedance mismatch occurring from disparate domain model and relational model; however, it could not provide necessary flexibility to adapt it in slightly complex scenarios (schema having one-to-many relationships, aggregated/complex entities).

LINQ to SQL is highly optimized for Microsoft SQL Server and gives best performance but lack providers to work with other databases such as ORACLE or DB2.

User-friendly object oriented syntax, no other baggage makes it lightweight, performant as highly optimized for Microsoft SQL Server, easy to learn and adopt makes it suitable for most of the Microsoft SQL Server based thick client (Windows Forms based Payroll application), thin client (Asp.NET based Leave application), or Service Oriented (B2C or B2B integration scenario such as Intra or Interbank fund transfer) in enterprise scenarios.

Amongst several challenges listed above two challenges which are not resolved completely are: ability to support multiple databases such as ORACLE, DB2, SQL Server and impedance mismatch (mapping domain entities to relational objects). This is where Entity Framework begins to shape up.

Entity Framework

Overview

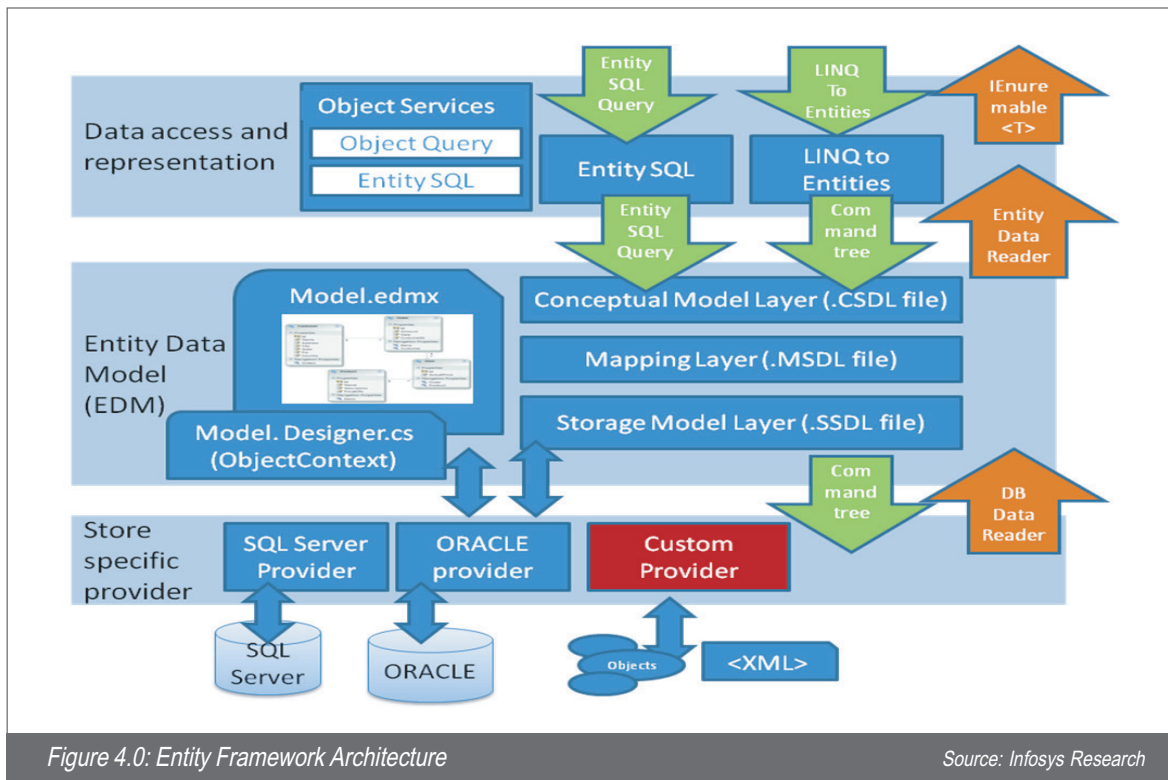
Entity Framework built upon what LINQ to SQL has to offer and worked towards incorporating aspects which were not strong, especially strengthen the Object Relational modeling and multi-database support.

For heterogeneous databases scenarios, writing and maintaining database specific data access constructs is highly challenging. Microsoft's ADO.NET Entity Framework solves this problem by implementing an Object Relational (OR) interface on top of existing Relational or non-Relational data sources. Entity Framework makes this possible by building a logical abstraction layer called Entity Data Model (EDM) which hosts mapping between objects and relational table. The mapping can be changed anytime without changing the data access code.

Entity framework supports connectivity to multiple databases such as Microsoft SQL Server, ORACLE, DB2, etc. For this, it leverages the existing ADO.NET provider model and builds on top of it.

Architecture

The key components that work together for providing an end-to-end programming environment are as follows:



Entity Data Model (EDM)

The Entity Framework provides a level of abstraction between the data store and application by using a core component called Entity Data Model (EDM). EDM defines Conceptual Model, Logical Model and a Mapping Layer between the two.

Conceptual Model Layer or the C-Space Layer defines the entities and relationships between entities. The C-Space Layer is modeled using Conceptual Schema Definition language (CSDL) and it consists of EntityContainer, EntitySets, AssociationSets, AssociationTypes, EntityType, Relationships and Functions. This enables greater flexibility both in defining objects and optimizing the logical model.

Storage layer (also called S-Space Layer) depicts the schema of underlying data store. It consists of Tables, Stored Procedures, Views and Functions. It is queried using ADO.NET Data provider and modeled using Store Schema Definition Language

(SSDL).

Based on the conceptual model, Entity Data Model consists of extensible data classes (partial classes), which can be extended by the developers to add additional members. These classes derive from base classes which provide Object Services for materializing the entities as objects and for tracking changes.

As the name suggests, the Mapping Layer is responsible for mapping the conceptual and storage layer. It maps the business objects defined in the conceptual layer with the tables and relationships defined in the storage layer. Mapping Layer also defines how conceptual and storage layers are related. The C-S Mapping layer is modeled using Mapping Schema Language (MSL).

The complete content for CSDL, MSL, and SSDL is stored as XML in one file separated in independent sections and can be seen by right-clicking Model.edmx and opening it in XML editor.

Object Context

Object Context is the heart of Entity framework and is represented using a class which derives fromObjectContext. TheObjectContext provides functions for connecting to data source using dataprovider, managing identities, relationships, entity collections, tracking, saving changes to persistence layer while managing the Entity lifecycle.

Store-specific Provider

Entity client data provider provides a layer to abstract APIs specific to relational or non-relational database provider. The Entity Client Data Provider enhances the ADO.NET provider model by accessing data in terms of conceptual entities and relationships. It executes queries that use Entity SQL. Entity SQL provides the underlying query language that enables Entity Client to communicate with the database.

The Entity Client provider transforms Canonical Command Tree, based on the EDM, into a new canonical command tree that is an equivalent operation against the data source. This provider manages connections, translates entity queries into data source-specific queries, and returns a DataReader that objects services can use to materialize entity data into objects.

Note

Canonical Command Tree is the object model representation of query and represents CRUD operations.

Data Access and Representation

Entity Framework enables the developer to access and change data, which is represented as entities and relationships. It provides the following ways to query an Entity Data Model and return objects:

	Entity Framework	Query Methods
1	LINQ to Entities	<p>LINQ to Entities support for querying entity types that are defined in a conceptual model. As depicted in figure 4.0, the EDM query returns IEnumerable<T> or IQueryable<T> e.g.</p> <pre data-bbox="565 470 1032 520">IEnumerable <Customer> = context.Customers.Where(c => c.Customer.UniqueID == uniqueCustomerId);</pre> <p>The syntax is very similar to LINQ to SQL and most of the constructs in LINQ to SQL works as is except a few ones. This article [20] provides a comprehensive list of what is supported in LINQ to Entities and what is not.</p> <p>LINQ to Entities is the simplest, easy to understand and use and hence most adopted querying technique with Entity Framework.</p>
2	Entity SQL	<p>Entity SQL uses SQL like syntax to query the Entities and relationships in Entity Framework. It works directly with entities in the conceptual model that supports EDM features such as, inheritance and relationships. Through respective Database provider It supports calling Database stored procedures, SQL statements using Command Text. However unlike SQL, they do not support Insert, Update, Delete and DDL statements. e.g.</p> <pre data-bbox="565 919 1120 1136">string query = @"SELECT VALUE Customer FROM context.Customers as Customer where Customer.UniqueID = @UId"; // The following query returns Contact object ObjectQuery<Contact> objQuery=new ObjectQuery<Contact>(query, context, MergeOption.NoTracking); objQuery.Parameters.Add(new ObjectParameter("UId", "1234"));</pre>
3	Query builder methods	<p>Enables construction of Entity SQL queries using LINQ-style query methods. e.g.</p> <pre data-bbox="565 1226 1114 1310">ObjectQuery<Contact> query= context.Customers.Where ("cust.UniqueID == @UId" , new ObjectParameter("UId", "1234"));</pre>
4	Object Services	<p>Object Services consists of .NET Objects, collection of .NET objects, types, properties and methods.</p> <p>Object Services generates a canonical command tree which represents a LINQ to Entities or Entity SQL operation against the conceptual model.</p> <p>When you query the conceptual model, Object Services work in conjunction with Entity Client Data provider and Entity Data Model to translate the query into query that a database can process. When results are returned from the database, Object Services translates them back to objects defined by the conceptual model.</p> <p>Object Services uses LINQ to Entities, or Entity SQL queries described above to query the EDM.</p>

Table 5.0: Entity Framework Query Methods

Entity Framework 4 is released with .NET framework 4. It provides following three approaches to generate DAL entities.

EntityObject Based

This is used to create persistence aware Data Access Layer and is the default code generation strategy used by Entity Framework. The generated data entity class inherits from EntityObject base class. With EntityObject based entities ability of lazy loading and change tracking are available in the most optimized way. However, these entities are strongly dependent on Entity Framework.

Note

Persistence aware means the entity classes implementation should not explicitly take care of infrastructure code to interact with database such as tracking database changes, object state management, caching queries, and will be taken care of automatically through framework. It is usually achieved through a combination of classes such asObjectContext and EntityObject from which Object Context and generated entities inherit respectively.

Plain Old CLR Objects (POCO) Based

The POCO-based code generation strategy allows creation of data entities which are persistence ignorant and loosely coupled with entity framework. The POCO data classes do not have support for lazy loading and change tracking out-of-the-box. Change tracking in pure POCO classes without proxies is done through snapshot mechanism. POCO classes with proxies implement change tracking based on notification mechanism. POCO-based classes use dynamic proxies. Dynamic proxy types are generated at runtime and they inherit from POCO types which means, the runtime type of the entities are different from the POCO type.

Self-Tracking Entities

Need for Self Tracking Entities arise when entities have to be shared across multiple tiers of an application. Self Tracking Entities are capable of tracking the information about changes made to them. The Self Tracking Entities can be moved across tiers and attached to the context in the executing process. Entity needs to be simply attached to the Object Context and then save changes on Object Context. This will save the complete changes within the object tree.

Self Tracking Entities are generated using T4 Templates. The generated entities can be POCO entities and hence Selftracking Entities are also loosely coupled with Entity Framework.

Choosing a correct Entity creation approach is important while designing the DAL using Entity Framework; hence, we decided to provide further granular comparison in terms of strengths, weaknesses, and opportunities of above three entity creation approaches.

Strengths	Weaknesses	Opportunity
Entity Object		
<ul style="list-style-type: none"> • Default code generation approach for Entity Framework and is available OoB. • Ability to track changes (Insert, Update, Delete) is available OoB. • No dynamic proxies created at runtime; hence, the entity type at runtime is exactly the same as its declaration. • Along with public properties, supports private properties of objects to be persisted into the database. 	<ul style="list-style-type: none"> • Generated entities have dependency on Entity Framework. Entities are tightly coupled to Object Context and hence passing them across tiers is not possible unless extended using partial classes. • Entities cannot be reused in other ORM as this is tightly coupled with Entity Model. • Generated code is slightly complex and bulky. • Not recommended to be used when the application architecture requires persistence ignorance (opposite to persistence aware as described above). 	<ul style="list-style-type: none"> • Recommended when persistence aware is necessity of application architecture • Recommended where domain model is not required to reuse data entities as is.
Plain Old CLR Objects		
<ul style="list-style-type: none"> • Generated Entities have no Dependency on Entity Framework and can be easily passed across tiers. The generated Entities do not inherit from any object including EntityObject and as the name suggest are Plain OLD CLR Objects. • Data Entities can be reused in domain model or with other ORM as it has no tight coupling with Entity Framework. • Generated code is very simple and easy to understand and adapt. 	<ul style="list-style-type: none"> • A separate installable add-on (free) is required on top of Visual Studio to be able to generate POCO entities. • POCO generated model may face performance issues specifically if it is required to track changes frequently because it uses snapshot mechanism to track changes made to the object graph. • All properties in the Model have to be virtual, no support for private properties. • Dynamic proxies are created at runtime to enable lazy loading and change tracking, so the entity type at runtime is different than expected. • It has issues when being used as a data source for WCF data services. • Function import does not create a method if the stored procedure is not returning anything. 	<ul style="list-style-type: none"> • Recommended to use when application architecture requires persistence ignorance • When a user wants to reuse data entities as domain entities • Not recommended when you want the model to support private properties of objects
Self-Tracking Entities		
<ul style="list-style-type: none"> • The generated Entities implement specific interfaces for change tracking, however Self-tracking entities do not have dependency on complete Entity Framework. • Generated entities have the mechanism to record changes done in the scalar, complex and navigation properties. • Optimized for serialization scenarios. 	<ul style="list-style-type: none"> • A separate add-on (free) is required on top of Visual Studio to be able to generate Self-tracking entities. • Function import does not create a method if the stored procedure is not returning anything. 	<ul style="list-style-type: none"> • Recommended to build N-Tier application using Entity Framework where sharing the same entities between client and server is possible.

Patterns to load related Objects

Entity Framework provides various patterns to load related objects (foreign key also called as navigational) namely explicit loading, lazy loading, and eager loading. This can be done by specifying in query or appropriately setting context property. The selection of pattern is important from the performance perspective. Following is the summary of each pattern:

Lazy Loading (Deferred loading)

With Lazy loading enabled, related objects (navigational) is automatically loaded from the data source when you access a navigation property. If the navigation object does not exist in the Object Context cache, each navigation property that you access, executes a separate query against the data source. When you create a new entity data model in Visual Studio 2010, the generatedObjectContext type has Lazy loading turned on by default. However, it is still possible to programmatically turn it off and do explicit or eager loading.

```
nwContext.ContextOptions.LazyLoadingEnabled = false;
```

Note

Lazy loading is preferred way, when Database roundtrip is desirable for populating each of the navigational objects. On demand loading of Objects will have relatively lighter object footprint on the server side (middle tier- app. server).

Eager Loading

By using eager loading, it is possible to extract the whole graph of related objects from the data source in a single upfront request. A query path of related objects can be defined using the Include method of ObjectQuery. This pattern of loading is recommended when you know the exact shape of graph of related objects.

Eager loading pattern can be specified as follows:

```
using (NorthwindEntities nwContext = new NorthwindEntities())
{
    var query = nwContext.Customers.
        .Include("Orders.OrderDetails")
        .Include ("Addresses")
        Where(c=>c.CustomerID == "123");
}
```

The above query will fetch all Orders and Address for customer with CustomerID == "123" in one trip to database. In Include clause one can specify any number of levels using dot operator. For example Include ("Orders.OrderDetails") will fetch all the OrderDetails for the specific Order.

Note

Eager Loading is preferred way, when the specified Navigational Objects should be loaded along with main query and no additional database roundtrip is desired. Adopting this pattern can give user improved experience (data display, etc.) for subsequent queries. However storing the complete object tree on server can create a memory pressure, if the object tree is of large size.

Explicit Loading

Related objects are only loaded into the ObjectContext when explicitly requested. The related objects can be explicitly retrieved from the data source for an Entity Collection or Entity Reference using ObjectContext.LoadProperty() method. This ensures that a query is never executed without an explicit request for the related object.

Note

Adopting a specific Loading pattern is important design principle while designing data access tier, there is no thumb rule to choose one, as each strategy has its pros and cons and need to be adopted based on the scenario. Usually combination of strategy can be used to achieve the desired results.

Database roundtrip can be costly if the database is distributed and can cause user to wait if there is a bottleneck at Database. The current trend in memory(RAM) prices is, RAM is getting cost effective and it is not uncommon to have large sites with servers having several GB of RAM at times easily running into TB, in such scenarios Eager Loading would be preferred way of doing it and will give better user experience (page load, data retrieval performance). Several large high traffic social networking sites are adopting eager loading and caching data on middle tier [16].

Key Features

Model First approach (Top-down)

“Model First” approach lets you design conceptual model using Entity designer and then generates SQL Server database tables from that. The Model.Designer.cs file associated with model has C# code, which hosts the corresponding generated entity and entity collection. The entity and collection classes perform the role of data carrier in DAL. Once generated, models can be synchronized with Database at any point of time to reflect changes in Entity model or Database.

Note

The entity designer available with Entity Framework 4 is neither true Entity Relationships(ER) modeler nor class modeler. It lets one define entities/objects and establish relationships between them as inheritance, or one to one, one to many.

This is also called top-down approach that lets you design data access objects from logical perspective and then converts that into physical persistence layer (Database) unlike generating classes or objects based on relational tables in the database.

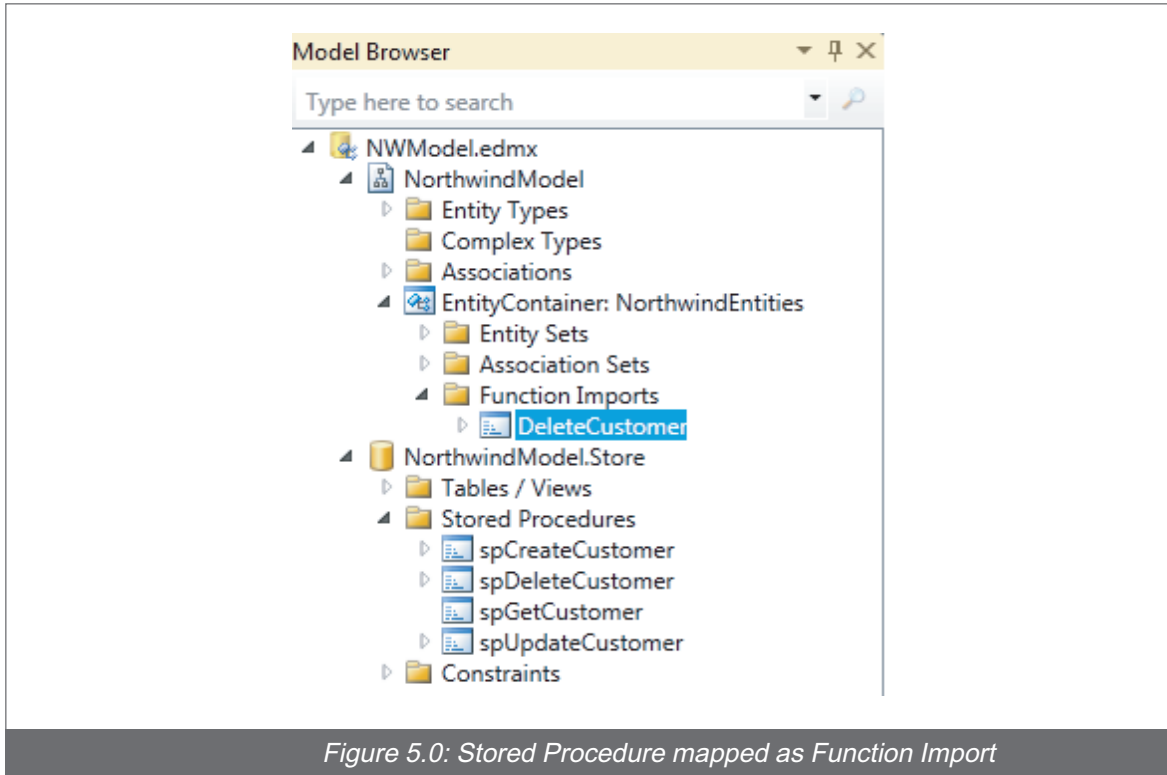
Generating Models from Code (Bottom-up)

Projects adopting Database first approach can generate Entity models from Database table. The Entity Model can connect to existing SQL Server or ORACLE Database schema, following which you can select one or more tables, based on which the corresponding classes in Model.Designer.cs get created. This approach is also called bottom-up as object oriented design is done based on Relational design.

Mapping Stored procedures

Databases having logic coded in the form of stored procedures can also benefit by using Entity Framework. The database stored procedures can be mapped to C# functions in Entities Design Model (EDM). CRUD operations implemented using stored procedures can be mapped to Entity CRUD functions, and non-CRUD stored procedures can be mapped using complex types.

The following figure 5.0 snapshot shows the function import of a stored procedure in Model Browser:



The C# code for the same would look as follows:

```
using (NorthwindEntities NWEntities = new NorthwindEntities())
{
    NWEntities.DeleteCustomer(custid);
}
```

Change tracking

The objects returned by queries and load operations are stored inside an object cache. By default, changes to the objects in the cache are not overwritten. This behavior can be changed by specifying MergeOption for queries and load operations.

Following is a brief of each option:

- AppendOnly: The default value of MergeOption, meaning the objects already existing in the cache will not be overwritten with the values from data source.
- OverwriteChanges: The objects in the cache are replaced by the latest version of the objects fetched from the data source.
- Preserve Changes: With preserve changes option, the modified values are not overwritten but others are. This lets persists only modified properties to data source when you call save changes.
- NoTracking: Change tracking and Identity Resolution are not performed on the returned objects.

The following C# code snippet shows how change-tracking option can be specified as part of the code:

```
protected void btnGetCustomers_Click(object sender, EventArgs e)
{
    var query = (nwContext.Customers.Where(c=>c.City=="London")) as
    ObjectQuery;
    //Execute the query without tracking the changes
    query.Execute(MergeOption.NoTracking);
}
```

To support change tracking in distributed application development, Entity Framework 4 provides T4 template which can be used to generate self-tracking entities. Self-tracking entities can keep track of their own change state and are recommended in a disconnected scenario.

Identity Resolution

This feature of Entity Framework helps detect and resolve concurrency conflicts. Such conflicts occur due to multiple queries returning collection of objects containing duplicates. Identity resolution is performed as the object services maintain only a single instance of an object with a specific key, called Entity Key, in the cache.

Customer Entity using Entity Key can be queried as follows:

```
protected void btnGetCustomers_Click(object sender, EventArgs e)
{
    var query = nwContext.Customers.Where(c=>c.CustomerID == "ALFKI");
}
```

Entity Framework 4 has introduced several new and enhanced existing features such as POCO, Self Tracking Entities, support for complex objects, better naming of entities (pluralization), support for Lazy Loading, support for foreign keys, etc. as lucidly explained in [19].

Framework Compatibility

Parameter	Compatibility
.NET Framework	3.5, 4.0
Visual Studio	Microsoft Visual Studio 2008 SP1, 2010

Table 6.0: Entity Framework - Framework Compatibility

With the above insights into Entity Framework, let's take a look at SWOT for the Entity Framework.

Strengths

1. Supports multiple data sources, mostly supported through respective database providers for ORACLE etc.
2. Loosely coupled with the database objects because of conceptual schema (CSDL) which allows you to manipulate the object model according to needs.
3. Supports truly object relational modeling through rich designer support, supports one to many relationships, mapping complex types, etc.
4. With the support for POCO (Plain Old CLR Objects) based entities, it is also possible to reuse the generated entities as they are plain CLR objects.
5. Provides multiple options such as Entity Object, POCO and Self Tracking based which can be used to cater to specific scenarios.
6. Support for Model First approach makes it possible to create the database according to the business needs (top-down approach).
7. Updating/refreshing the Object model from the database is possible to accommodate the schema changes done in the database.
8. Last but not least Entity Framework inherits all the advantages associated with LINQ way of interacting with Databases, some of them as listed in LINQ to SQL Strengths above.

Weaknesses

1. Single Entity Data Model cannot be created from heterogeneous databases such as Microsoft SQL Server, ORACLE, SYBASE, DB2, etc. A single EDM cannot connect to multiple databases. Recommended way is to create independent database-specific EDMs.
2. Entity Framework is relatively new as compared with widely used ORM frameworks such as NHibernate.
3. Huge number of lines of generated code decreases readability/adaptability of the code.
4. Slightly less performant(Entity Object based approach) as compared to ADO.NET and LINQ to SQL due to bulky mapping layer - Refer Appendix A

Opportunities

Entity Framework is best suited to be used in the following scenarios:

1. Building enterprise-scale and enterprise-wide DAL needing connectivity to multiple heterogeneous databases. As recommended above, create independent EDMs to deal with heterogeneous databases and have sizable (maximum sixty to ninety) entities in each EDM to be easily manageable.
2. Building database-independent applications/products so that DAL can easily plugged in and out from one database to another.
3. Applications having very strong domain model and needing object relational mapping.
4. Build quickly persistence aware thick, thin client applications.

Threats

1. Primary threat from the open source NHibernate framework as the later has more proven track record.

Note

Refer to AppendixB for a brief overview of NHibernate and how Entity Framework compares with it.

SUMMARY

Entity Framework first got released alongside .NET framework 3.5 and is now available with .NET framework 4. It has shown a great promise to solve issues in object relational mapping world and ability to support multiple databases. It helps to leverage database logic by mapping stored procedures. The new Model-based approach helps in taking top-down approach (conceptual/domain to relational model) apart from earlier provided bottom-up (relational to conceptual/domain - code first approach). Entity Framework helps build the DAL for thick, thin, service oriented applications, through variety of options such as Entity Object, POCO, Self tracking entities.

Within Microsoft stack, Entity Framework is a sole candidate which qualifies to build Enterprise-wide DAL. However, it is in early days, the framework API is evolving fast and there is further scope to add lot of APIs and functionality which can ease the life of developer, also adoption of Entity Framework is still at nascent stage.

WCF Data Services (erstwhile Astoria or ADO.NET Data Services)

Overview

In the era of Service Oriented Architectures (SoA), where more and more functionality is exposed and consumed as services, database data/entities are no exception. More and more applications demand data to be exposed as Internet-accessible URLs and further slicing and dicing of data is possible. For example, sharing online inventory status with partners/vendors, sharing shopping basket with friends in online shopping scenarios etc.

However, if you have to implement above kind of scenario, combination of technologies like WCF services and any data access technology like Entity Framework or ADO.NET, LINQ needs to come together. Even after that, through internet URL, slicing and dicing (drill downs, drill through) of data would not be easily achievable. Ensuring secured data access through such web based services would be additional challenge.

WCF Data Services tries to solve above-mentioned challenges of exposing database model, entities and logic as REST services.

RESTful way of identifying and sharing information/resource is considered easy and lightweight for making seamless information exchange. Implementing REST architecture style needs a way to expose functionality as REST services. In the REST architecture style, everything is treated as resource and accessed using URIs over http.

Note

To understand REST Architecture, refer to “How I explained REST to SOAP pro” in the References [4] section.

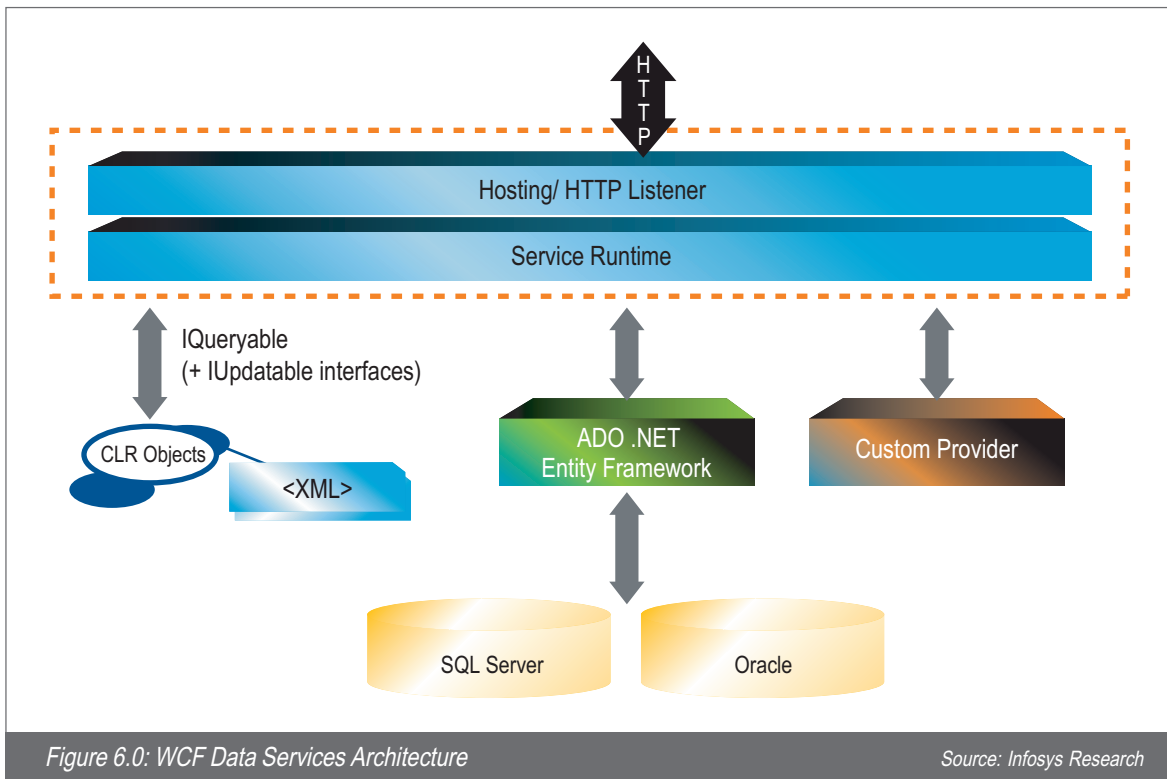
WCF Data Services was originally released with .NET framework 3.5 SP1 as ADO.NET Data Services technology. The exposed database entities are treated as resources and the associations between database entities are treated as links. Resources are accessed as URIs using well-known data formats such as JSON, ATOM. The database data can be fetched over HTTP and operated upon by using standard HTTP verbs such as GET, POST, PUT, and DELETE. The protocol which helps doing this process is referred to as Open Data Protocol (ODP) by Microsoft [12].

WCF Data Services leverages Entity Data model (EDM) to establish semantics and relationships between various relational and non-relation entities through provider model. Relational entities can be SQL Server, or ORACLE database tables and relations. Non-relational entities can be anything from unstructured data to plain old CLR objects (POCO). It also provides special client libraries for consuming data services in .NET Forms and Silverlight applications.

Architecture

To expose the data that resides in relational database, WCF Data Services can use Entity Data Model, which allows you to view data in terms of conceptual model. For scenarios other than relational database, WCF Data Services framework supports any LINQ-enabled data source to be exposed through HTTP interface. Thus, the data is exposed regardless of the data store. WCF Data Services defines a generic hosting interface IDataServiceHost that abstracts its implementation from a specific host. This allows WCF Data Services to run in a range of host environments, from custom HTTP server-side implementations such as WCF, ASP.NET, and IIS.

It provides IQueryable and IUpdateable interfaces to query the datastore. IQueryable interfaces that produce query trees, can be used for querying the data exposed in the application. IUpdateable interface that defines the semantics of HTTP methods can be used for performing read-write operations.



Key Features

Visibility Control

Visibility Control feature helps the data service owner to hide sensitive data (for example, login and password details) from the unauthorized users. Using visibility control, it is possible to specify entity sets that are visible and within each visible entity set, specify the valid operations.

Authentication

WCF data service will always have a host and a runtime in which it is executed. What it means is WCF data service does not invent its own authentication mechanism. It integrates with the authentication mechanisms of the host that it runs inside. For example, you can use forms authentication or basic or NTLM or WCF authentication schemes.

The following code explains how to control access to Entity Set as well as Operations:

```
// Set Visibility per entity set
config.SetEntitySetAccessRule("Products", EntitySetRights.All);

//Set Operation Access per entity set
config.SetServiceOperationAccessRule("Products",
ServiceOperationRights.ReadMultiple);

//Enable error description
config.UseVerboseErrors = true;

//Limit number of records per request for entity set
config.SetResourceSetPageSize("Products", 5);
```

Interceptors

Query interceptors customize how the service behaves and what data goes through the service and more importantly what data does not. For example, using query interceptor, you can create a service-side mechanism which does not allow the logged-in user to see the details of other users. This gets pushed down along with query into the data source. It is a general mechanism for instance level or row-level security implemented at the server side. Just like query interceptors, update interceptors allow enforcement of row-level security while performing Update on the database. Interceptors do not look like operations, they preserve the REST interface.

Code snippet for Intercepting “Customers” entity set is

```
//Specify which entity set you are going to intercept
[QueryInterceptor("Customers")]
public Expression<Func<Customers, bool>> QueryCustomers()
{
    return c => c.CustomerID == "ALFKI";
}
```

Service Operations

Using Service Operations, it is possible to wrap the business logic in a method and expose it on the server. A service operation can contain functionalities such as role-based security, validation logic, enhanced queries, stored procedures functions etc. It is a service-side facility for adding pre-built operations (“Service Operations”), which can run arbitrary logic that can be parameterized using simple serialized types from the URI.

The following is an example of how a Service Operation to fetch Fixed Deposit Account for a particular customer:

```
[HttpGet]
public IQueryable<ACCOUNT> FetchFDAccount(int Customer_Id, string
Account_Type_Code)
{
    IQueryable<ACCOUNT> query = from ac in this.CurrentDataSource.ACCOUNTs
                                where ac.CUSTOMER.Customer_Id == Customer_Id &&
                                ac.ACCOUNT_TYPE.Account_Type_Code == "FD5YS"
                                select ac;

    return query;
}
```

Summary of Latest Features

To leverage WCF Data Services 4 completely, it is important to be aware of the latest features available with version 4.

	Feature	Description
1	Data Binding	It is possible to create client types which notify the client context for changes made to the entities. On subsequent call of SaveChange on the context, the changes automatically reflect on the data store. This feature relieves the developer from manually detecting the changes and saving them to the data source.
2	Row Count	Applications exposing large amount of data from the server, may not want to download all the data at once. Row count is an addressing scheme to allow a client to obtain information about the total number of entities in a set without having to download all the entities and, thus, reducing the load on the server. The following URI shows how the total number of records existing on the server can be fetched: http://localhost/DataService.svc/Customers?\$inlinecount=allpages
3	Feed Customization	This feature helps customize the mapping between the Data Service runtime and data entity properties and thus eliminate the unwanted properties. The service author has the declarative control to specify the customization attribute; WCF data service customizes the returned feeds accordingly. This feature is useful when we do not want the feeds as is from the data source but want to modify them according to our requirement.
4	Server Driven Paging (SDP)	Allows a service author to set per collection limits on the total number of entities returned for each request. This feature is extremely useful in defining paging for large-size data objects like Documents which can be fetched and displayed in chunks with link to next set of records.
5	Streaming of Binary Resources	The binary content of large media resources (for example, image, audio, video) is separated from entity. Loading media resource as binary content includes performance overhead which can be avoided by streaming them separately.
6	New "Data Service Provider" Interface for Custom Provider Writers	You can write custom providers for cases when the Entity data model and arbitrary .NET classes do not meet the needs.
7	Projections	Data Services URI format has been extended to work with the subsets of the properties of an entity. Projection feature extends the URI format of WCF Data service and allows the client to explicitly specify the properties to be returned. Projections help in filtering result sets in vertical fashion (on column names). Following URI shows how selected columns from Customers entity can be browsed using \$select query option. The following URL will fetch only CustomerID and ContactName: http://localhost/DataService.svc/Customers?\$select=CustomerID,ContactName

Table 7.0: WCF Data Services Features

Framework Compatibility

Parameter	Compatibility
.NET Framework	3.5, 4.0
Visual Studio	Microsoft Visual Studio 2008 SP1, 2010

Table 6.0: Entity Framework - Framework Compatibility

Strengths

1. Data centric applications can focus more on writing business logic than building plumbing related to service endpoints, etc. as this is available OoB with WCF Data Services.
2. The applications are able to retrieve and manipulate data using simple HTTP verbs.
3. Use of HTTP enables uniformity as each piece of data is available in the form of a URI. Moreover use of HTTP also allows you to reuse available mechanisms such as proxies, caching, authentication etc.
4. Supports simple payload formats such as ATOM and JSON.
5. Support for server-side paging mechanism prevents bottleneck on the server.

Weaknesses

1. Slowest performance as compared with other data access mechanisms - Refer Appendix A

Opportunities

WCF Data services are best suited to be used in following scenarios:

1. For exposing database entities as RESTful services which can be consumed by different clients.
2. For building medium size B2B or B2C Web applications needing resource/ URI based drill down access to database entities.
For Example, sharing real time Bill of Material (BoM), inventory status in partner integration, supply chain scenarios via internet URLs.
3. Very useful for exposing database data to clients consuming java scripts.
4. For building applications which need to expose RSS or ATOM feeds.
5. For building ASPNET Ajax, Forms, WPF and Silverlight based highly interoperable RESTful applications
6. WCF Data Services cannot be used for proxy generation and hence cannot be consumed by clients relying on proxybased access.
7. Useful in accessing data from Microsoft SQL Azure cloud storage.

Threats

1. When used with Silverlight client, WCF RIA Services [2] can be considered as better solution as it provides out-of-thebox bindings and templates to work with Silverlight.

Conclusion

Number of options in data access technology from Microsoft brings the challenge of making a judicious choice. With evolving architecture styles, each style needs its own specialized and optimized way to interact with the database.

Slightly old fashioned [ADO.NET](#) can be considered for writing plain vanilla multi-tier Windows Forms or Web applications. ADO.NET does not support OoB O-R mapping but provides flexibility to write O-R transformation code. However, manual code in the long run raises maintainability concerns. Microsoft will continue to support ADO.NET in future and hence existing applications can be enhanced or new applications can be planned using ADO.NET.

[LINQ to SQL](#) makes it possible to write data access code without needing to learn PL/SQL for ORACLE or T-SQL for SQL Server or ANSI SQL or any other database specific constructs. To some extent, it supports O-R and simplifies accessing and working with different Relational Tables in Object oriented manner. LINQ to SQL supports only Microsoft SQL Server and has constraint to work with other Relational Databases, supports primarily one-to-one mapping. LINQ to SQL is not the recommended solution to build DAL for any new applications.

[Entity Framework](#) provides complete support to O-R mapping. LINQ based queries eliminates possibilities of SQL injection attacks. Entity Framework is a perfect candidate to build enterprise-wide DAL because of the support for host of data providers. It provides high flexibility to adapt to specific scenarios. For building any new medium-to-large applications, it is the candidate to be considered for DAL.

[WCF Data Services](#), a new kid on the block extends the advantages provided by Entity Data Model (EDM) and helps in exposing database entities as JSON, AtomPub Feeds. For truly realizing RESTful architectures, it can be used to build lightweight REST services over http protocol.

It is important that the Architecture and Development teams are aware of the SWOT of each of the above technologies and judiciously decide one over other or a combination of the above technologies for solving specific enterprise problems.

Design Consideration	ADO.NET	LINQ to SQ	Entity Framework	WCF Data Services
Efficient Queries (Faster read/write) - Performance	High	Medium	Medium	Low
Security threat: SQL Injection attacks	High	Low	Low	Medium
Writing Database independent DAL	Medium	Low	High	Medium
Domain/Business to Relational mapping (Handling impedance)	Low	Medium	High	Medium
Build highly interoperable DALs by exposing data as resource	Low	Medium	Medium	High
Maintainability	Medium	Medium	High	Medium
Flexibility	High	Low	High	Medium
Forward Looking (Microsoft Supportability)	Medium	Low	High	High

Appendix A Performance Benchmarking

Performance is an important dimension of data access technology evaluation and selection, hence, we benchmarked ADO.NET, LINQ to SQL, ADO.NET Entity Framework and WCF Data Services on common business operations such as create loan request for customer, update loan request, fetch loan request and delete request.

The following methods were used to measure performance of above listed data access techniques. Each of the methods was used to perform Insert, Read, Update and Delete 1000 records from SQL Server 2008 database.

AddLoanRequest	The AddLoanRequest method accepts Loan Request data and inserts a row into AccountRequest table.
UpdateLoanRequestStatus	The UpdateLoanRequestStatus method accepts old as well as new request status; it searches for Account Request rows having an old status and updates them with new status.
FetchAccountRequest	The FetchAccountRequest method accepts AccountRequestID and fetches a single AccountRequest record.
DeleteAccountRequest	The DeleteAccountRequest method accepts AccountRequestID and deletes the corresponding AccountRequest record from the table.

Test tools and strategy

For benchmarking the performance of various technologies, we wrote above listed methods using ADO.NET, LINQ to SQL, ADO.NET Entity Framework (Entity Object based) and WCF Data Services and ran them via unit tests. Each method returns a collection of TimeSpan which represents time taken for database trips.

We used parameterized queries; at the end of each method execution cleaned database cache as well as Visual Studio cache.

Machine Configuration

Client Machine Configuration:

#of Clients	#of CPUs	Processor	Disk	Installed Memory	Software
1	1	Athlon Dual core 5600+ Processor 2.80 GHz	160 GB	2 GB (1.75 GB Usable)	Microsoft Windows 7 Visual Studio Ultimate 2010

Table 9.0: Performance Benchmarking - Client Machine Configuration

Database Server Configuration:

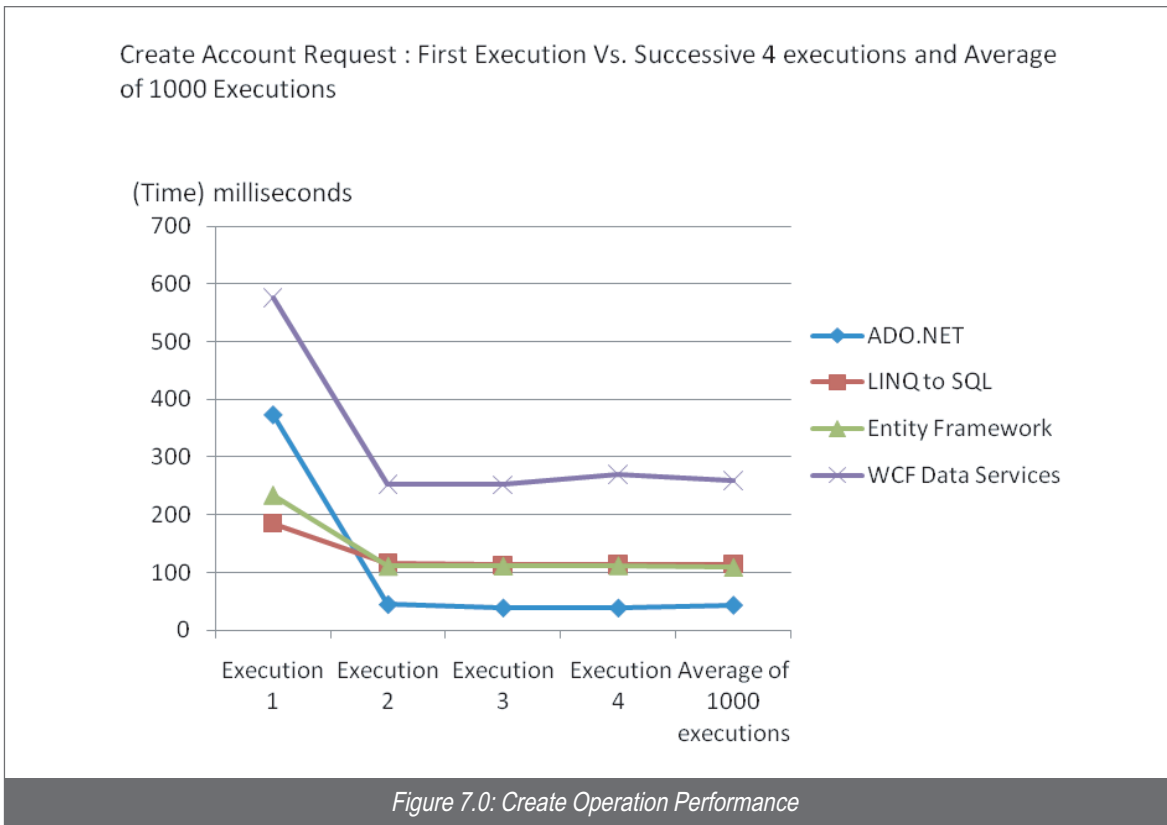
#of Clients	#of CPUs	Processor	Disk	Installed Memory	Software
1	1	AMD Athlon™ 62 X2 Dual core Processor 5200+ 2.60 GHz	160 GB	4 GB	Microsoft Windows server 2008, SQL server 2008

Table 10.0: Performance Benchmarking - Database Server Configuration

Test Results

Note

1. Lazy loading option is turned off for LINQ to SQL as well as ADO.NET entity Framework
 2. WCF Data Services has the same Entity Data Model as a data source as created by ADO.NET Entity Framework.
 3. ADO.NET query is performed using parameterized query option and ExecuteNonQuery.
 4. The Y - axis on the graph represents time taken to execute query in milliseconds and X-axis represents four consecutive executions and average of 1000 executions for the same query.
- Performance results are captured and plotted as charts for each of the Data Access Techniques we have discussed in the paper.



Fetch Account Request : First Execution Vs. Successive 4 executions and Average of 1000 Executions

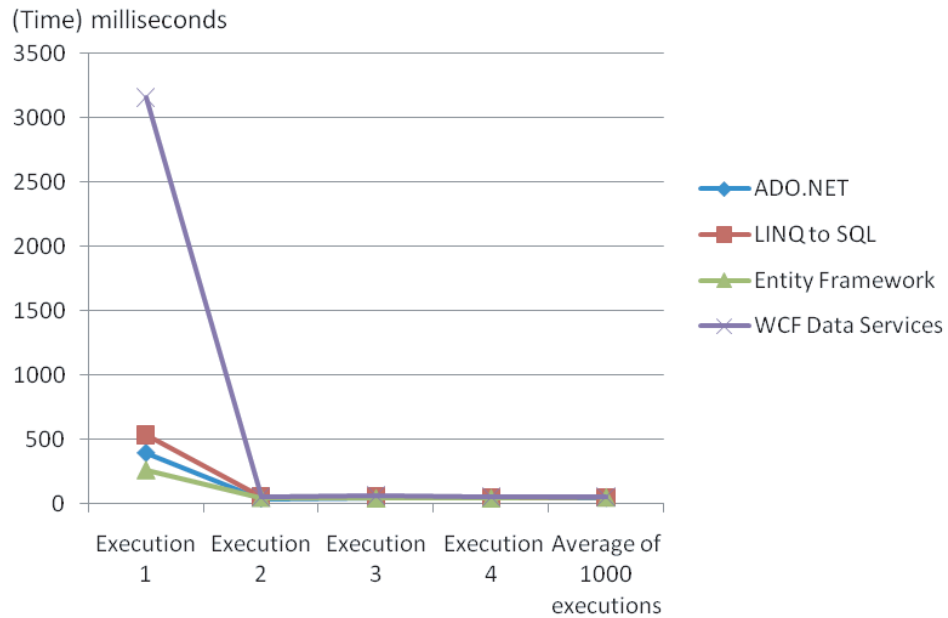


Figure 8.0: Read Operation Performance

Update Account Request Status : First Execution Vs. Successive 4 executions and Average of 1000 Executions

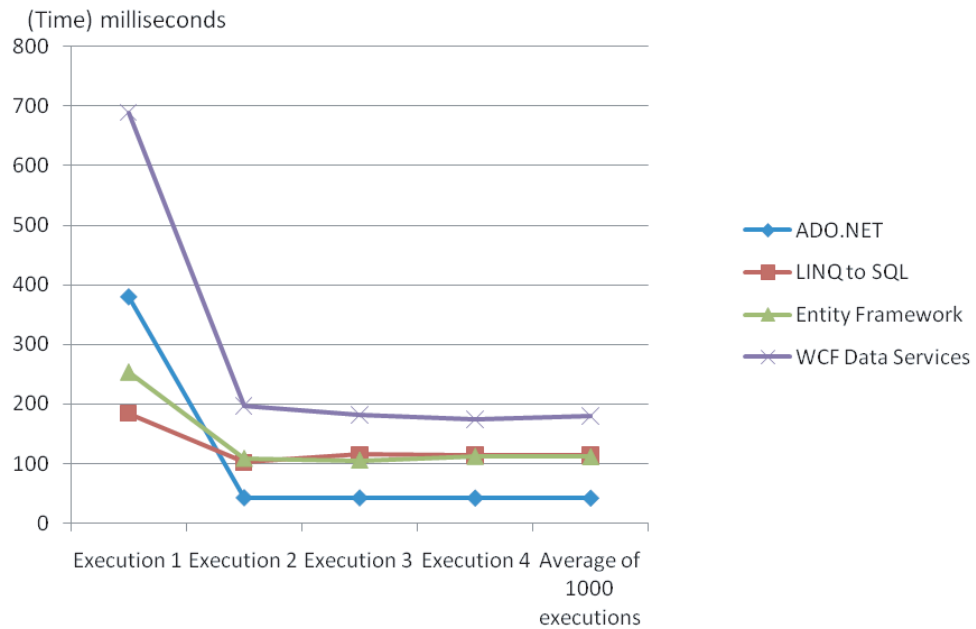


Figure 9.0: Update Operation Performance

Delete Account Request Status : First Execution Vs. Successive 4 executions and Average of 1000 Executions

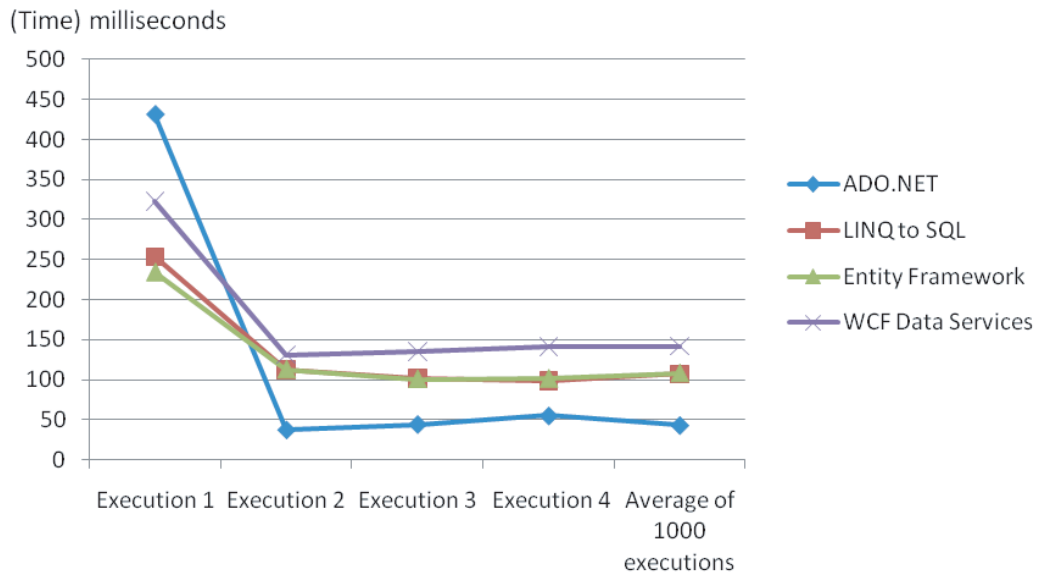


Figure 10.0: Delete Operation Performance

SUMMARY

ADO.NET gives the best performance in terms of time taken to execute a query. However, it takes a significantly higher amount of time for first execution of Create, Delete, and Update operations. LINQ to SQL proves to be second best; though it has marginal difference when compared with ADO.NET Entity Framework. WCF data services being REST (based on http as against others using TCP or native protocol) enabled Data Access Technologies, takes the maximum amount of time to perform database operations.

Appendix B Entity Framework Vs NHibernate

NHibernate Overview

NHibernate[17] is widely used, open source, Object Relational Mapping solution for .NET framework. It provides a framework where developer can define mapping between Object Oriented Model and Relational Database objects and facilitates querying and retrieval of data from the database. NHibernate is highly extensible and has very effective batching and caching capabilities. It is possible to integrate NHibernate with Visual studio using add-ins like Active writer.

We have summarized the comparison [14] between Entity Framework with NHibernate in the following table.

Parameter	Entity Framework	NHibernate
Goal	Apart from solving impedance mismatch problem, Entity Framework aims at spreading EDM awareness to a wide spectrum of Microsoft Products. Having an EDM into application should facilitate creating REST-based Web service, Reporting service, build entity - aware workflow etc. It should allow you to form atomic entities from multiple tables of the data source.	Aims at solving impedance mismatch problem by providing a bridge between application domain and database.
Maturity	Entity Framework is relatively new and is considered to be less mature than NHibernate.	Established and widely used ORM.
Extensibility	Less extensible as compared to NHibernate. Provides extensibility in terms of code generation scenarios as described in main section.	Being an Open Source ORM, NHibernate is more extensible. It has extension projects such as NHibernate Validator, NHibernate Search, NHibernate Shards etc.
Mapping between Entities	Mapping between the entities is auto-generated and slightly complicated. Unmapped fields throw a compile-time error.	Mapping is relatively easy to read. NHibernate allows you to write the entities first and then the mapping between them separately.
Visual Designer	Out-of-the-box designer available to create and manipulate simple as well as complex type entities. However, it has stability issues while dealing with large number of entities.	No out-of-the-box designer available (Active Writer and Fluent NHibernate are add-ins)
Documentation	Sufficient Documentation, code samples, forum support to adapt Entity Framework.	NHibernate lacks in Documentation as compared to Entity Framework.

Parameter	Entity Framework	NHibernate
Batching and caching capabilities	Batching several queries in a single database call is possible through Include clause. Entity Framework does not have support for second-level caching; but it has properties like INotifyPropertyChanged which do not require snapshot-based caching.	Has extensive support for batching with options for example write batching, read batching, multi queries and futures. Caching support in NHibernate includes second-level caching. INotifyPropertyChanged is not available with NHibernate.
Support for POCO	With Entity Framework 4, POCO support is available. In Entity Framework version 1, POCO entities were not supported.	Creates only POCO-based entities.
Support for LINQ	Support for LINQ is available out-of-the-box.	NHibernate does not have out-of-the-box support for LINQ. LINQ to NHibernate is available with version 3.0 of NHibernate.
Integration with WCF data services	Entity Framework integrates well with WCF data services.	Integrates with WCF data services
Learning Curve	Easy to learn	NHibernate has steep learning curve. It requires deep understanding of internal working and mapping.
Usability Scenario	Recommended to use when the database design is medium complex and less than 100 number of entities. For larger number of entities multiple EDMXs can be used to manage complexity.	Recommended to use when the database schema is having complex mappings, constraints and relatively more number of entities are involved.

Table 11.0: Entity Framework vs. NHibernate

Further Reading

1. Microsoft .NET 4 and Visual Studio 2010 for Enterprise
<http://www.infosys.com/microsoft/resource-center/Documents/net-framework-4-vs2010.pdf>
2. VS2010 Architecture Modeling
<http://www.infosys.com/microsoft/resource-center/Documents/architecture-modelingvisual-studio.pdf>
3. Dynamic Language runtime
<http://www.infosys.com/microsoft/resource-center/Documents/dynamic-languageruntime.pdf>
4. Functional Programming on the .NET Platform
<http://www.infosys.com/microsoft/resource-center/Documents/functional-programmingplatform.pdf>
5. Infosys .NET 4 blogs
http://www.infosysblogs.com/microsoft/net_40/

Bibliography

1. ORACLE Provider for .NET from Data Direct
<http://web.datadirect.com/products/net/net-for-oracle/index.html>
2. WCF RIA Services
<http://www.silverlight.net/getstarted/riaservices/>
3. Roadmap for LINQ to SQL
<http://msdn.microsoft.com/en-us/data/bb525059.aspx#Q3>
<http://blogs.msdn.com/b/adonet/archive/2008/10/29/update-on-linq-to-sql-and-linq-toentities-roadmap.aspx>
4. How I explained REST to a SOAP pro.
http://www.infosysblogs.com/microsoft/2009/08/how_i_explained_rest_to_a_soap.html
5. .NET Oracle provider deprecated
http://www.infosysblogs.com/microsoft/2009/07/time_to_bid_adieu_to_net_oracl.html
6. ADO.NET
<http://blogs.msdn.com/adonet/>
[http://msdn.microsoft.com/en-us/library/e80y5yhx\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/e80y5yhx(VS.80).aspx)
7. LINQ
<http://language-integrated-query.com/>
<http://hookedonlinq.com/>
<http://msdn.microsoft.com/en-us/netframework/aa904594.aspx>
8. Entity Framework
[http://msdn.microsoft.com/en-us/library/aa697427\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/aa697427(VS.80).aspx)
[http://msdn.microsoft.com/hi-in/magazine/cc700331\(en-us\).aspx](http://msdn.microsoft.com/hi-in/magazine/cc700331(en-us).aspx)
9. WCF Data Services
<http://blogs.msdn.com/astoriateam/>
<http://msdn.microsoft.com/en-us/data/bb931106.aspx>
10. LINQ Providers for ORACLE, My SQL, etc.
http://code2code.net/DB_Linq/

11. Microsoft's Open Data Protocol
<http://msdn.microsoft.com/en-us/data/ee844254.aspx>
12. Positives of NHibernate
<http://ayende.com/blog/archive/2010/01/05/nhibernate-vs.-entity-framework-4.0.aspx>
13. NHibernate vs Entity Framework
<http://stackoverflow.com/questions/1639043/entity-framework-4-vs-nhibernate>
14. Entity Framework vs NHibernate performance test
<http://gregdoesit.com/2009/08/nhibernate-vs-entity-framework-a-performance-test/>
15. Architecture Scalability
<http://highscalability.com/blog/2009/10/13/why-are-facebook-digg-and-twitter-so-hard-to-scale.html>
16. Nhibernate home page
<http://www.nhibernate.com/>
17. SQL Injection attack
http://en.wikipedia.org/wiki/SQL_injection
18. Entity Framework 4 New features
<http://www.devproconnections.com/article/microsoft-net-framework/Renovations-to-NET-4-0-s-Entity-Framework.aspx>
19. Supported and Un-Supported LINQ methods
<http://msdn.microsoft.com/en-us/library/bb738550.aspx>

About the Authors

Avani Dave (Avani_Dave@infosys.com) is Systems Engineer with Microsoft Technology Center (MTC) in Infosys. She has over 2 years of industry experience in Microsoft .NET. Earlier, she has worked with .NET 3.0 and 3.5 frameworks. She has been working on understanding .NET 4 stack and building a Reference Implementation in .NET 4 since last one year. Currently, she is consulting one of the Fortune 500 customers on .NET implementation.

Sudhanshu Hate (Sudhanshu_Hate@infosys.com) is Senior Architect with Microsoft Technology Center (MTC) in Infosys. He has over 12 years of industry experience with last seven years on Microsoft .NET. For last several years, Sudhanshu has consulted on Microsoft technology solutions to Fortune 500 customers in EMEA and US. Sudhanshu has published papers and presented in external forums such as Microsoft Virtual TechDays. He also blogs at <http://www.infosysblogs.com/microsoft/>.

ACKNOWLEDGEMENT(S)

Authors would like to acknowledge

Naveen Kumar

Principal Architect

nkumar@infosys.com

Atul Gupta

Principal Architect

AtulG@infosys.com

Mohammed Farrukh Nizami

Senior Architect

Farrukh_Nizami@infosys.com

and

Sidharth Ghag

Senior Architect

Sidharth_Ghag@infosys.com

for their help with review of this paper.



For more information, contact askus@infosys.com

About Infosys

Many of the world's most successful organizations rely on Infosys to deliver measurable business value. Infosys provides business consulting, technology, engineering and outsourcing services to help clients in over 30 countries build tomorrow's enterprise.

For more information about Infosys (NASDAQ:INFY), visit www.infosys.com.