ADO and ADO.NET Object Model Comparisons: A Relational Perspective

Alfred J. Lendvai **Hao Shi**

School of Computer Science and Mathematics
Victoria University, Melbourne, Australia

Summary

Microsoft has, and still is, a significant contributor toward the development and innovation of software technology. An important aspect to any software technology is the creation, storage, processing and transfer of data. This paper focuses on two Microsoft data-access technologies, ADO and ADO.NET, the later being the more recent technology. ADO is an acronym for ActiveX Data Objects and though the literal 'ADO' is used in both, ADO.NET is a part of the new .NET platform, a different architecture to the one in which ADO was designed. ADO and ADO.NET are part of two different Microsoft technology infrastructure strategies.

Although .NET framework has been released by Microsoft for more than five years, migrating from ADO to ADO.NET was relative slow for the existing IT projects due to concerns associated with .NET reliability and robustness. Many IT companies started to adopt .NET as their future platform until the release of .NET framework 3.0 in November 2006.

This paper aims to shed some insights for Microsoft web developers to understand the two technologies in depth. An overview of ADO and ADO.NET from a relational perspective is presented, with particular emphasis on the ADO Recordset and the ADO.NET Dataset objects. A comparative analysis is performed providing an assessment of strengths and limitations of each technology. Conclusions drawn show that ADO.NET presents some significant advantages over its predecessor such as a superior disconnected model with greater control over the update process, improved management for multi-table processing, as well as improved scalability and object model extensibility.

Key words:
ADO, ADO .NET, Recordset, DataSet, relational database, scalability, extensibility.

1. Introduction

Access, transport, storage and manipulation of data are fundamental and underlying processes at the heart of every application/system. ADO and ADO.NET support these same processes, albeit through different supporting run-time platforms, infra-structures, classes, levels of functionality and part of different Microsoft strategic paradigms.

ADO is a data access technology that provides a high-level easy-to-use interface to access relational data as well as other types of data [1]. It is a programming and data-access model that represents a framework in the form of an object oriented programming interface. ADO is based on Microsoft's proprietary COM architecture, a building block that evolved application systems from 2-tier, to N-tier, to distributed computing frameworks and then to distributed applications that incorporated the Internet. The COM building block served as the mechanism by which distributed COM (DCOM) evolved and services such as COM+ developed [2].

ADO.NET is part of the .NET initiative introduced in June 2000 as a new strategy for the development, deployment and execution of highly distributed applications referred to as Web Services. A networking infrastructure is provided that is layered on the Internet or Intranet. This infrastructure is leveraged against technologies that are both ubiquitous and nonproprietary, and built upon industry set standards and protocols [3]. The .NET initiative provides a completely new .NET platform – a new .NET Framework development environment and a new Common Language Runtime run-time environment. ADO.NET, by default, derives many of the benefits of being part of the .NET environment such as a foundation based on the acceptance of an object-oriented approach, coupled with a new self-describing component model – the .NET assembly. Some important benefits include a rich class library that is extensible and organised by namespace, a Common Type System, managed execution of applications, language interoperability, structured exception handling and simple deployment [4].

The next two sections provide an overview of ADO and ADO.NET object models. These are then used as a basis for delivering comparisons and conclusions in Section 4 and 5, respectively.

Manuscript received January 5, 2007
Manuscript revised January 25, 2007

2. ADO

ADO is a class set providing access to relational data [5]. It provides transparency using a common interface to access various data sources using a core set of objects such as Connection, Command and Recordset. There is also Parameter, Property, Field and Error, a set of subsidiary objects that are accessed via Parameters, Properties, Fields and Errors collections respectively. All of these objects and collections have a Properties collection that holds property objects. ADO objects feature properties, methods and events to be able to manipulate the object and its contents. The ADO object model is diagrammatically represented in Fig. 1.

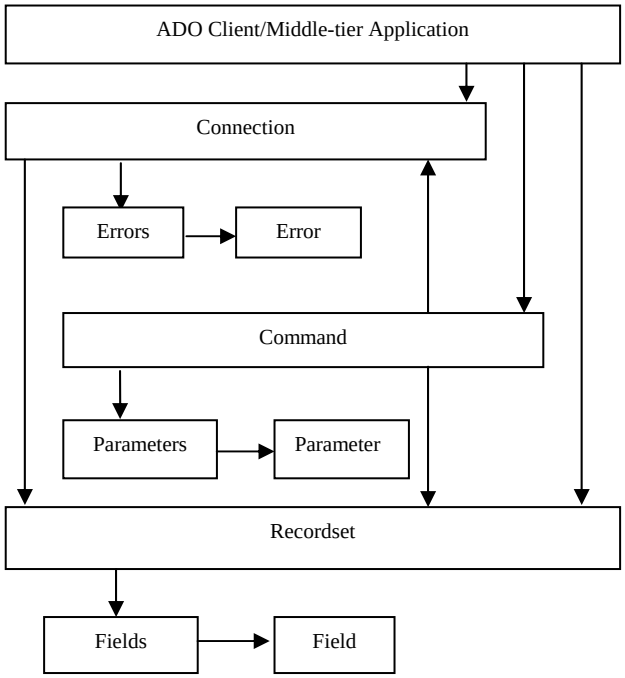

Fig. 1 The ADO object model

Using this set of objects, ADO provides flexibility in choice as to how to access and update a data source from a client application. For example, Recordset properties offer a matrix of choices such as for Cursor Location (Client, Server), Lock Type, (adLockReadOnly, adLockBatchOptimistic, adLockOptimistic, adLockReadOnly) and Cursor Type (Forward, Static, Key-Set, Dynamic). The setting of these properties need to include consideration of connected or disconnected states and cache size. Not all combinations are possible e.g. Pessimistic locking with a client-side cursor, Optimistic lock type with a Dynamic cursor, or if the lock type is set to ReadOnly, the Recordset is not updatable regardless of the cursor type setting. Still, some combinations may be appropriate but result in system resource wastage if not used correctly e.g. using a client-side cursor with adLockBatchOptimistic lock type and maintaining an open connection.

The inherent ADO flexibility creates overlapping in the use of some objects to perform the same action, or different actions performed by the same object [6] e.g. sharing common objects for different access/update approaches. There is some potential to not select the optimal combination for the desired purpose. For example, ADO allows changes to be applied directly to a data source using the following objects and methods:

- Connection object Execute method
- Command object Execute method
- Recordset object Open method

The generation and population of a Recordset may be performed in a multitude of ways as listed below, as well as by manual means.

- Recordset Open method
- Command Execute method
- Connection Execute method

Alternatively, updates may be performed using an in-memory connected or disconnected Recordset object using a server-side or client-side cursor, with read-only (adLockReadOnly), pessimistic (adLockPessimistic), batch optimistic (adLockBatchOptimistic) or optimistic (adLockOptimistic) lock types. Cursor type is always adOpenStaic if cursor location is client-side. A Recordset with Batch optimistic locking allows changes to be applied without any data source interaction, even if the Recordset holds an active connection.

An optimistic locking Recordset allows Recordset rows, individually or as batch, to be updated without holding a lock on the corresponding record at the data source. Success or failure in propagating a modified data row to the data source depends entirely on whether another user has since modified that same row. If a Recordset uses optimistic locking (adLockBatchOptimistic or adLockOptimistic), data access and updating is stateless. Choosing adLockBatchOptimistic allows Recordset updating without needing to use system resources to maintain a connection. Alternatively ADO Recordset data access and updating may be stateful when connected with a pessimistic lock.

A Recordset with a client-side or server-side adLockOptimistic cursor allows a row that is pointed to by the cursor to be updated or cancelled before proceeding to another row. If the cursor is set to adLockBatchOptimistic, changes may be made wholesale within the Recordset without data source interaction. These changes may then be

reconnected to, and updated against, the data source or cleared from the Recordset.

The default mechanism of ADO is to use a connected Recordset with server-side cursor. When performing a Recordset update, there is no provision to customise the updating logic. Sql statements are automatically generated, and in comparison to ADO.NET there is no provision to be able to choose stored-procedures to perform the updates of changes made in a Recordset when it is used in a batch mode.

3. ADO .NET

Whereas ADO provides a multi-faceted approach to data handling, the ADO.NET model provides a philosophical change in approach where its set of objects are functionally optimised for specific tasks. ADO.NET data access is based upon the Dataset and data provider that provides specific access to a data source [7].

The Dataset is akin to a mini-relational database, but stored in memory. It is composed of a core set of objects that include its namesake, as well as the DataTable, DataColumn, DataRow, DataView, DataRelation and Constraint objects. The Dataset is equated with the disconnected aspect of ADO.NET. The DataSet is a standalone object existing independently of any data source.

The data provider provides the means to communicate with any data source. A .NET data provider provides a set of objects that enable the connection to, data retrieval from, data update to, and disconnection from, a data source. Each data provider is for a specific data source. Each data provider provides its own set of classes such as for SQL Server .NET, OLE DB .NET or Oracle .NET. As an example SQL Server .NET uses SqlConnection, SqlCommand, SqlParameter, SqlDataReader, SqlDataAdapter and SqlTransaction objects (located in System.Data.SqlClient namespace) and OLE DB .NET data providers use OleDbConnection, OleDbCommand, OleDbParameter, OleDbDataReader, OleDbDataAdapter and OleDbTransaction (located in System.Data.OleDb namespace).

The ADO.NET model introduces a level of abstraction where there is a separation between data and the process by which data is retrieved and updated as shown in Fig. 2. The centerpiece of this model is the data adapter. Each data adapter has its own specific data provider. It is the bridge between the DataSet and the data source. It is the coordination layer between the in-memory DataSet and the permanent sources of data access through the data provider.

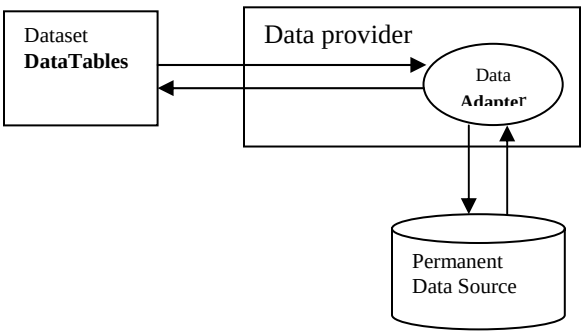

Fig. 2 ADO.NET abstractions

The data adapter uses the data reader of the same data provider to populate a Dataset using the Fill method. The data adapter uses four instances of the Command object: SelectCommnand, InsertCommand, UpdateCommand and DeleteCommand, to propagate, via the Update method, the changes of the same type in a Dataset to a data source e.g. an inserted row in the Dataset is used with the InsertCommand. It does not automatically include this logic. The filling of a single DataSet may be performed using multiple data adapters. The population of a Dataset is not limited only to data providers but may be sourced from user input, programmatically or XML input. The data adapter does not interact directly with the connection object but indirectly through its set of four command objects. The data adapter always leaves the state of a connection as it was, that is opened or closed, after executing. This is a default process that supports the disconnected paradigm of the ADO.NET model. [8] describes the various overloaded forms of the data adapter Update method. The Connection object provides access to a data source either to retrieve data, or to provide the pipeline for applying data changes to a data source.

.NET provides data providers. These are also referred to as managed providers such as OLE DB .NET and SQL Server .NET. The OLE DB .NET managed provider provides access to various data sources requiring COM interoperability services of the .NET Framework to access OLE DB functionality. A managed provider such as SQL Server .NET provides a more efficient, direct and streamlined access to the API of the database, in comparison to the OLE DB provider. For example, the SQL Server .NET managed provider uses SqlDataReader. which uses its own Tabular Data Stream protocol. This protocol is managed by the CLR to access data where SQL Server exposes its data types to conform to the .NET Framework types as well as being able to expose its data in native format. That is, when retrieving results from a managed

DataReader such as SqlDataReader, columns are retrieved in their native data type form without requiring expensive conversions.
Managed providers as part of the object-oriented framework provide opportunity for extensibility [9].

SQL Server .NET data provider excels in performance, but there are benefits in using the OLE DB route to access data. OLE DB for Sql Server is slower, due to the requirement of COM communication, but provides greater functionality - SQL Server .NET provider uses only a subset of the functionality. There are other circumstances for using OLE DB such as the reliance of SQL server on OLE DB to manage joins across heterogeneous data sources [10].

4. ADO and ADO.NET Comparisons

Both ADO and ADO.NET provide similarities with their command objects allowing retrieval from, and update of, a data source. However as noted above, ADO provides multiple ways to update a data source directly, whereas using ADO.NET, the only way to update a data source directly is by using the command object of a data provider. ADO.NET provides alignment of Command object methods to the type of result returned. The methods offered include ExecuteReader, ExecuteNonQuery, ExecuteScalar and ExecuteXmlReader. This compares to the ADO Command object where only one method, the Execute method, is used, and where the CommandType property determines the type of result expected. The CommandType property may be assigned an enumerated value representing values such as a stored procedure, table name or Sql query.

ADO.NET command types may be text i.e. an SQL string, a parameterised stored procedure with input or output parameters or a table name(s). Additionally ADO provides access to data from persisted files or data in semi-structured form.

Furthermore, the Connection object is only associated with the ADO.NET Command object whereas with ADO the Connection object is associated with the ActiveConnection property that is available to both the Recordset and Command objects.

In ADO.NET, the Command object ExecuteReader method may be used to return a result into the data providers data reader object. In comparison, and as described earlier, ADO may return a Recordset result set from a database using the Recordset, Command or Connection objects.
The ADO.NET data reader object provides a fast-forward read-only data stream that may be read to access row data consecutively using the Read method. It cannot be instantiated – it is always provided through the command object interface. However it is limiting if random access is required. It may instead be stored in an array or populated into a Dataset using the data adapter. DataReader objects are essentially the equivalent of the forward-only read-only Recordset cursor of ADO.

.NET managed data providers use separate libraries and classes for different data stores. ADO uses the same Connection, Command and Recordset set of objects for different data sources. This is in comparison to ADO.NET, which provides a different group of objects for each data provider. This means that using more than one data provider requires more than a simple connection string substitution the exception being OLE DB .NET data provider.

ADO represents a single unified object model attempting to be all encompassing for all data sources - ADO.NET does not.

4.1 Recordset and Dataset Comparisons

Compared to the ADO.NET Dataset object, an intimate understanding of the Recordset is required to obtain maximum benefits when choosing between a server or client located cursor, the degree of locking required and the level of visibility desired to changes applied by other users to the data source. The overlapping in usage of the Recordset does not provide a clear delineation between using disconnected and connected models. This provides the potential to not attain optimal benefits when designing access to, and update of, a data source.
Although the Recordset and Dataset provide similarities, there are also some compelling differences [11]. The DataSet may be identified with a Recordset that has a client-side CursorLocation, a CursorType that is open static and a LockType that is optimistic. However, the DataSet extends the capabilities of the Recordset for managing application data.
Both an ADO Recordset and an ADO.NET Dataset may be created manually, implicitly or explicitly, from the schematics of a database. Both in-memory models enable the storage of tabular/relational type data and which may be populated in various ways. ADO.NET provides the DataView object which is based on a DataTable object. Using the DataView object, data may be modified, sorted and filtered with the capability to clear changes or to apply changes permanently to a data source e.g. database. There is no equivalent of this object in ADO. Both facilitate the creation of a smaller Recordset or Dataset that is an extraction containing only changes. This provides a lower transmission load when transmitting changes across the network to update a data source.

In comparison to the Dataset, the Recordset is limited to a single tabular result. The Dataset resembles a relational database system i.e. it may have multiple tables, relationships between tables through column associations, primary and foreign key referencial integrity, update and delete cascading control. Of course a DataTable may hold a result set derived through a JOIN SELECT statement. This may be appropriate at times when a non-hierarchical view of data is sufficient, e.g. through a gridcontrol. A Recordset that contains a SELECT JOIN result set is difficult to synchronise [12].

The Dataset is completely data source agnostic, whereas the Recordset is not. Once a Dataset is populated e.g. from a data provider, it has no knowedge of where the data came from - it has no direct interface with the Connection object. This is in contrast to the Recordset object that has a 'connection' to a data source through the ActiveConnection property. When using the DataSet, the connection is only open when retrieving data from a source, or propagating changes to a data source.

A Dataset may be typed or untyped. A typed Dataset provides several advantages over the Recordset such as providing compile-time checking, Intellisense, as well as code that provides easier programming – it is more readable and concise.

4.2 Table Relationships

The Recordset may be populated with a result set from an SQL data source representing a single table (or subset of). If only a single table is retrieved, the Recordset and Dataset have much in common.

When there is a requirement to access data from two or more tables, differences arise between the use of a Dataset and a Recordset. If a Recordset receives a result set from a join of two or more tables, two points of interest arise:

- The presence of duplicated data
- Ascertaining the source of the data is not straightforward

Although reading and updating is straightforward, extra programming effort is required to control the insertion and deletion of Recordset rows against root tables. Base and unique tables need to be established, otherwise undesirable results may occur [13].

If multiple Recordsets are retrieved, one table per Recordset, no data duplication is present. This may be performed efficiently by using a stored procedure to return multiple result sets in one round trip from a data source. However for multiple Recordsets, programming overheads are incurred in establishing relationships for navigational purposes.

Extra care is required when updating a Dataset to a database. The enforcement of constraints and cascading rules supports the integrity of a Dataset, however diligent management control is required in controlling and ensuring relationally connected Datatable changes are applied correctly to the data source. There is a possibility that the order in which changes are applied to a Dataset with multiple tables and relationships may not be applied correctly, i.e. in the same sequence when updating a database using a data adapter [13]. That is, the update may produce incorrect results compromising the integrity of the database. ADO.NET enables Datatable changes to be isolated by type of change. Consideration of change type and correct order of DataTable updates determined by parent/child relationships, enables database updates to be performed in the correct order.

The ADO.NET CommandBuilder provides an alternative, but has limited control. There are no options to configure the update process. It is useful for single table access and updates [14].

To enable improved processing of Recordsets with relationships, Recordset Data Shaping (MSDataShape provider) based on the Recordset may be used. It is related to the organisation, access, manipulation and display of Recordsets. It provides the ability to embed Recordsets in Recordsets, known as hierarchical Recordsets. It uses a special query syntax that uses the SHAPE clause with SELECT statements. It provides several alternative representations of hierarchies including Relation, Parameterised, Grouping and Hierarchy. These may be combined with each other in various ways to form a structure that facilitates drill-down search capability. A Recordset is updateable at any level in the hierarchy and is based on the use of optimistic or batch optimistic lock types. The shaped Recordset provides certain advantages such as, returning data that is less than from joined queries and applying single table updates, however submitting pending changes against multiple tables can be problematic. Data shaping has further limitations: the query syntax is not easy to follow, queries apply only to a single data source and filtering is difficult [12].

The DataSet may contain one or more DataTable objects that optionally have relationships between rows of the tables, may have primary or foreign key constraints, as well as cascading rules that operate against update and delete operations. The Dataset provides management support of its data through the use of collections e.g. DataRelation, DataTableCollection, DataRowCollection, DataRelationsCollection and DataColumnCollection. Data relationships may be created, deleted and modified dynamically. Data redundancy may be eliminated using individual tables instead of SELECT JOINs. Hierarchically related tables may be sourced from different data providers. ADO.NET does not require an additional data provider to

support relationships between tables - the DataRelation object is implicitly provided.
The rows of data in a DataTable may be accessed, searched and modified and they may be the source of customised views using DataView objects. The Dataset may be filtered using the Select method to create an array of rows. DataView objects may be sorted, filtered, searched and bound to Web and Windows controls e.g. the DataView object may be bound to ASP.NET Web Server controls such as DataGrid, Repeater and DataList.
Compared to ADO, ADO.NET is more useful for representing hierarchical or network relationships between tables.

4.3 DataSet and Recordset Navigation

A difference between ADO and ADO.NET is navigational access of Rows within a Recordset and a DataSet.
Navigation within a Recordset is sequentially based - rows are accessed through the use of a cursor. The Recordset provides a client-side cursor or a server-side cursor. Using client-side cursors, updates are performed optimistically against a data source. Using server-side cursors, updates are performed either pessimistically or optimistically. The use of a server-side pessimistic locking cursor provides greater control and visibility to data changes applied by other users at the data source.
There is no concept of an updateable server-side cursor in a DataSet. The Dataset is completely independent of any data source. The ADO.NET DataReader however does maintain state on the server when reading the database data as a stream.
Navigation within a DataTable occurs through the use of indexing where DataRows may be accessed randomly as well as sequentially. Navigation between DataTables is supported with the Relations property of the Dataset, ChildRelations and ParentRelations properties of the DataTable, and GetChildRows and GetParentRows methods of the DataRow. Each of these relation properties contains a collection of DataRelations. Taken together, all these relational representations between tables of a Dataset provide for hierarchical and easy manipulation of parent/child relationships.

4.4 Update Control

4.4.1 Batch Recordset and Dataset

This section performs comparisons in the update of disconnected ADO.NET Datasets and batch optimistic Recordsets (adLockBatchOptimistic) to a data source e.g. database.
The update of an optimistic locking Recordset row, using Update or BatchUpdate methods, implicitly generates an SQL UPDATE, DELETE and INSERT statement. There is no provision to control the generation of updating logic for applying Recordset changes to a database.
This is in contrast to the capabilities of ADO.NET. ADO.NET offers two new features:

- The ability to control the update process of a data adapter object by assigning custom commands to the action-based command properties of the data adapter.
- The ability to perform a stored procedure call to update Dataset changes to a database. The restriction in the use of Sql action statements is lifted.

The first point above presents a new level of control when performing data adapter updates of DataSets and DataTables. The second point provides accessibility to stored procedures providing performance advantages over the use of Sql statements.

4.4.2 Manual Recordset and Dataset

A data adapter requires the explicit creation of three action-based commands for its three command properties i.e. InsertCommand, DeleteCommand and UpdateCommand. These three commands are used to perform data base updates against each inserted, deleted or modified DataTable row. The commands are customised and parameterised and having access to row columns, row state (Added, Modified, Deleted, Unchanged) and row version (Original, Current). The update of the database is initiated with a single call to the data adapter Update method. The update may be performed for a Dataset, a DataTable(s), or user selected DataRows.
Alternatively, Dataset changes may be updated to a data source without using a DataAdapter object. This requires direct access to rows of a DataTable from a Dataset and the use of Command objects. This approach requires direct and iterative access to a row set e.g. a DataTable. As above, each row provides access to row state, row version and column values when executing the parameterised Command objects. More programming effort is required to perform updates manually compared to using the data adapter Update method [11]
It is possible to perform a database update from a Recordset without using the Recordset Update method. This provides fine-grained control at the expense of a greater programming effort. As per the Dataset, the Recordset may be iterated through providing access to Field names, Recordset EditMode property values (adEditDelete, adEditAdd, adEditInProgress) and Field values (OriginalValue, UnderlyingValue). Each command object may use any number of parameters that may be specified dynamically at run time. This may be straightforward for a

single table Recordset but becomes more complex when multiple tables are involved

5. Conclusions and Recommendations

This paper presented a comparative analysis of two Microsoft data access technology models: ADO and ADO.NET. The comparative analysis focused on relational aspects with particular attention to the ADO Recordset and the ADO.NET Dataset objects.

Providing a simplistic feature-versus-feature comparison is not totally realistic, however Table 1 (next page) attempts to reflect in a summarised form the differences by grouping similar features together.

Based on the table, it may be concluded that ADO is beneficial in instances where:

- scalability, that is demand on resources, is not an issue
- controlled access using persistent connections to the database is a priority
- controlled visibility to changes made to a database (using persistent connections) by other users is a priority

ADO would not be beneficial in instances where comprehensive support and ease of use is sought in processing hierarchically related Recordsets.

ADO.NET provides benefits where

- there is a requirement for comprehensive management and control in batch updates
- ease of use is a priority in the update of disconnected data to a data source
- scalability is an issue
- the use of stored procedures in batch updating is important.

References

1. Otey Michael, Conte Paul (2001)*: SQL Server 2000 Developers Guide*, Osborne/McGraw-Hill.
2. Blexrud, C et al. (2000) Professional Windows DNA: Building Distributed Web Applications with VB, COM+, MSMQ, SOAP, and ASP, Wrox.
3. Bustos J. and Karli, W. (2002) Beginning .Net Web Services with VB.NET, Worx.
4. Anderson Richard, Francis Brian, Homer Alex, Howard Rob, Sussman Dave, Watson Karli (2002): *Professional ASP.NET 1.0*, Wrox Press Ltd.
5. *The Role of ADO in Universal Data Access, ADO Programmer's Guide*, http://msdn.microsoft.com/library/en-us/ado270/htm/mdconroleofadoinuda.asp, Viewed on 08 Dec 2004
6. MacDonald Rob (2000): *Serious ADO: Universal Data Access with Visual Basic*, Apress.
7. .NET Framework Developers Guide – Accessing Data with ADO.NET, http://msdn.microsoft.com/library/en-us/cpguide/html/cpconaccessingdatawithadonet.asp, last viewed on 18 Dec 2006.
8. .NET Framework Class Library –Class Library, http://msdn.microsoft.com/library/en-us/cpref/html/cpref_start.asp, last viewed on 18 Dec 2006.
9. McManus Jeffrey P., Goldstein Jackie, Price Kevin T. (2003): *Database Access with Visual Basic .NET (Third Edition)*, Addison-Wesley.
10. Petroutsos Evangelos, Bilgin Asli (2002): *Mastering Visual.Basic .NET Database Programming*, Sybex Inc.
11. Lendvai A J (2004): A Comparative Analysis of ADO and ADO.NET, Minor Thesis, School of Computer Science and Mathematics, Victoria University.
12. Sceppa David (2002): *Microsoft ADO.NET*, Microsoft Press.
13. MacDonald Matthew (2002): *The Complete Reference ASP.NET*, McGraw-Hill/Osborne.
14. McManus Jeffrey P., Kinsman Chris (2002): *Visual Basic .NET Developer's Guide to ASP.NET*, XML and ADO.NET, Addison-Wesly.

Table 1: ADO and ADO.NET Comparisons

ADO	*ADO.NET*
ADO Recordset is an all-purpose object. There is an overlap in data access approaches.	*ADO.NET provides greater alignment of objects and methods with their intended purpose.*
Direct updates may be performed with several different objects.	*Direct update of source data may only be performed with Command object.*
Recordset may be populated from a database using several different objects.	*Dataset is populated from a data provider only through a single DataAdapter object.*
Provides a functionally rich connected data access model (persistent connection and pessimistic concurrency) Provides a limited disconnected data access model (optimistic concurrency updating) that is less comprehensive.	*Provides a functionally rich disconnected data access model (optimistic concurrency updating). There is no concept of an updateable server-side cursor in a DataSet.*
The Recordset is directly linked to a data source (via the Source property).	*Once disconnected, the Dataset is completely data-source agnostic.*
The updating of a (default) Connected server-side cursor Recordset absorbs resources in maintaining a persistent connection – scalability is hampered.	*The disconnected model places less demand on resources only requiring connections to update to, or retrieve data from, a data source. The default data model does not use a 'live' persistent connection. This improves scalability. Connection pooling is used.*
Recordset access to a variety of data stores communicates through the OLE DB interface (layer). COM (marshalling) interoperability is required.	*.NET Data Providers such as System.Data.SqlClient provides great performance in communicating with the database directly using its native TDS protocol.*
Batch (optimistic) Recordset updating implicitly generates an Sql action-query against each type of modified Recordset row. There is no provision to control the update process.	*The DataAdapter provides considerable flexibility to control and manage the update process. There is full access to DataTable row data, type of change and state information, and to database row data. Stored procedures may be used.*
The Recordset may be a container for either a single database table or an SQL JOIN result set.	*The Dataset provides a familiar data model based on relational database functionality using tables, relationships, constraints and cascading rules.*
Processing an SQL JOIN Recordset (updating) is not simple. Data Shaping (optimistic or batch optimistic lock types) provides a solution for processing hierarchical (single table) Recordsets, however updating can be problematic.	*Manual control is required to ensure the sequence of Dataset changes is maintained when updating the data source. Loss of integrity is possible.*

Alfred J. Lendvai received a M.S. degree in Computer Science from Victoria University in 2004. During 2003-2004, he completed his Minor Thesis in the area of Microsoft web technologies. He has worked in the IT industry for many years providing IT development/support, management and consulting services to organizations in manufacturing, education, government and IT service sectors.

His current role is with the Environment Protection Authority providing specialist IT services to support and upgrade their business systems, as well as IT development practices.

Dr. Hao Shi obtained her BE in Electronics Engineering from Shanghai Jiao Tong University, China in 1986. She joined the then Department of Electrical and Electronic Engineering, Victoria University as a Lecturer after completion of her PhD at University of Wollongong in 1992 and was promoted to a Senior Lecturer in 2001. She joined School of Computer Science and Mathematics in March 2003. She has been actively engaged in R&D and external consultancy activities. Her research interests include p2p Network, Location-Based Services, Web Services, Computer/Robotics Vision, Visual Communications, Internet and Multimedia Technologies.