ADO.NET

DEFINICION

ADO.NET es un conjunto de clases que exponen servicios de acceso a datos para el programador de .NET. ADO.NET ofrece abundancia de componentes para la creación de aplicaciones de uso compartido de datos distribuidas. Constituye una parte integral de .NET Framework y proporciona acceso a datos relacionales, XML y de aplicaciones. ADO.NET satisface diversas necesidades de desarrollo, como la creación de clientes de base de datos de aplicaciones para usuario y objetos empresariales de nivel medio que utilizan aplicaciones, herramientas, lenguajes o exploradores de Internet.

Proporciona acceso a datos relacionales, XML y de aplicaciones. ADO.NET satisface diversas necesidades de desarrollo, como la creación de clientes de base de datos de aplicaciones para usuario y objetos empresariales de nivel medio que utilizan aplicaciones, herramientas, lenguajes o exploradores de Internet.

Esta tecnología es una parte del .NET Framework 3.0 (habiendo sido parte del framework desde la versión 1.0).

ADO.NET es un subconjunto de la .NET Framework Class Library, que contiene todas las funcionalidades necesarias para conectarse e interactuar con dos tipos de repositorios permanentes de información:

1) Bases de Datos, como Microsoft SQL Server (clases del namespace System.Data, que se encuentran compiladas en System.data.dll

2) Archivos XML (clases del namespace System.XML, que se encuentran compiladas en System.Xml.dll)

Acceso a Bases de Datos Relacionales Escenario Conectado

En la actualidad se plantean dos tipos de escenarios de acceso a bases de datos relacionales. El primero de ellos es el que se conoce como “Escenario Conectado”, ya que en él se requiere una conexión física establecida con el servidor de datos durante todo momento para poder efectuar cualquier consulta o actualización sobre los datos.

Esto tiene algunas ventajas y también sus desventajas.

Algunas Ventajas:

• Al haber una única conexión a la base de datos por usuario, o incluso a veces por aplicación, establecida permanentemente, puede llegar a resultar más sencillo administrar la seguridad y el acceso al servidor de datos. Lo mismo ocurre con el control de concurrencia: en un escenario donde múltiples usuarios se estuvieran conectando y desconectando permanentemente para realizar distintas acciones, este control sería más difícil de llevar.

• Siempre la aplicación tiene acceso a los datos actualizados.

Algunas Desventajas:

• Se requiere una conexión abierta todo el tiempo con el servidor de base de datos, lo cual consume recursos innecesariamente si no se la está utilizando.

• La escalabilidad del acceso a los datos se ve limitada por la cantidad de conexiones establecidas simultáneamente contra el servidor de base de datos.

Acceso a Bases de Datos Relacionales Escenario Desconectado

El segundo escenario de acceso a bases de datos relacionales se conoce como “Escenario Desconectado”, ya que en él una parte de los datos del servidor central se copia localmente y puede luego ser consultada y actualizada sin contar con una conexión abierta. Luego si se desea puede establecerse una conexión con el servidor de base de datos para sincronizar los cambios efectuados sobre la copia local y actualizar los datos. Este tipo de funcionalidad es particularmente útil para escenarios de usuarios móviles, que salen de su oficina con una laptop, un SmartPhone o una PocketPC y desean poder continuar trabajando por más que no tengan conectividad física con el servidor de base de datos ubicado en la red interna de la empresa.

Algunas ventajas:

• La posibilidad de trabajar sobre los datos independientemente del resto de los usuarios de la aplicación

• Mayor escalabilidad en el acceso a datos y utilización más óptima de recursos del servidor, ya que se mantiene en un mínimo indispensable la cantidad y duración de conexiones abiertas.

• Mayor performance, al trabajar con una copia local de los datos.

Algunas Desventajas:

• Puede ocurrir que en un momento dado un usuario no esté accediendo a los datos más actualizados del repositorio central

• Al momento de sincronizar los cambios efectuados localmente contra el repositorio central pueden surgir conflictos, los cuales deben ser resueltos manualmente.

ADO.NET soporta el acceso a datos tanto en escenarios conectados como desconectados.

Arquitectura

La arquitectura de ADO.NET está basada en el concepto de proveedores de acceso a datos, siendo un proveedor un conjunto de clases que permiten conectarse a una base de datos, ejecutar un comando sobre ella y tener acceso a los resultados de su ejecución, tanto de forma conectada como desconectada. En las próximas diapositivas hablaremos con más detalle de estos proveedores y las clases que los componen.

Proveedores de Acceso a Datos

Los proveedores de acceso a datos ADO.NET (conocidos como “Managed Data Providers”) representan conjuntos específicos de clases que permiten conectarse e interactuar con una base de datos, cada uno utilizando un protocolo particular.

El .NET Framework incluye cuatro proveedores de acceso a datos, que en conjunto le permiten conectarse e interactuar virtualmente con cualquier base de datos existente en la actualidad:

- **Data Provider For SQL Server:** es el proveedor de acceso nativo a servidores de bases de datos Microsoft SQL Server 7.0 o superior, y Microsoft Access. Al conectarse vía protocolos nativos de bajo nivel, provee la alternativa más performante para conexiones contra estos motores de bases de datos. Sus clases se encuentran en el namespace System.Data.SqlClient.

- **Data Provider For OLE DB:** es el proveedor de acceso a datos que permite interactuar vía el protocolo estándar OLE DB con cualquier repositorio de datos que lo soporte. Sus clases se encuentran en el namespace System.Data.OleDb.

- **Data Provider For ODBC:** es el proveedor de acceso a datos que permite interactuar vía el protocolo estándar ODBC con cualquier repositorio de datos que lo soporte. Sus clases se encuentran en el namespace System.Data.Odbc.

- **Data Provider For Oracle:** es el proveedor de acceso nativo a bases de datos Oracle, desarrollado por Microsoft utilizando las herramientas de conectividad de Oracle. De esta forma puede lograrse un acceso más performante a bases de datos Oracle desde aplicaciones .NET que utilizando ODBC u OLE DB.

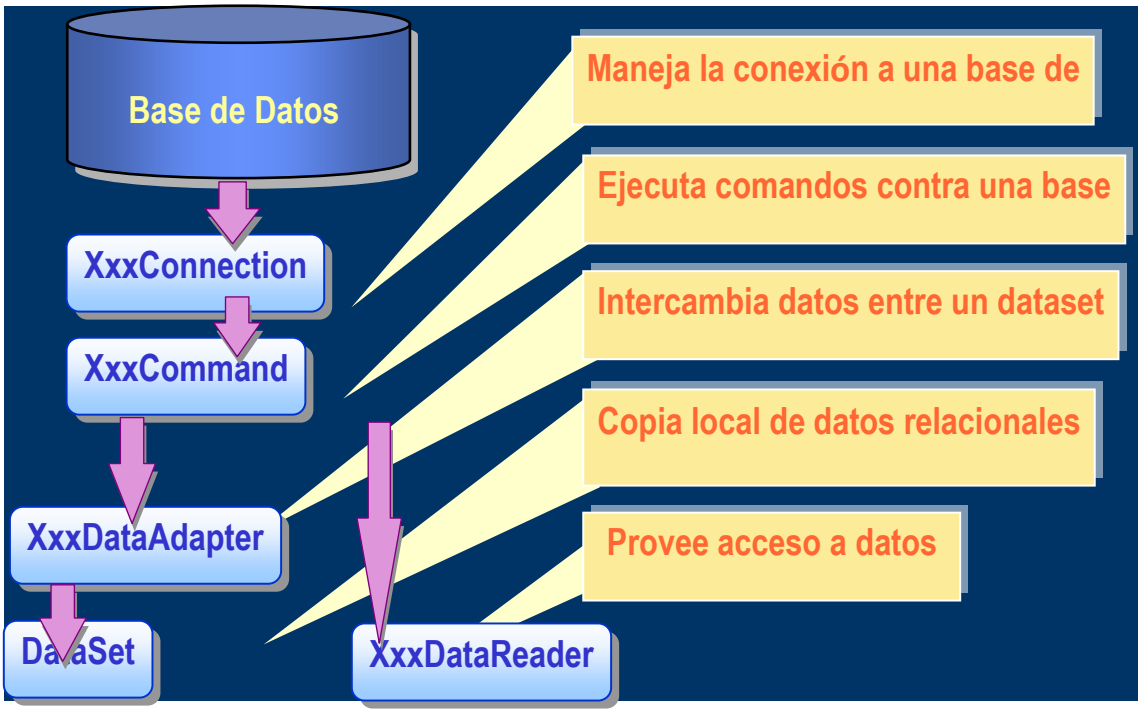
  Sus clases se encuentran en el namespace System.Data.OracleClient, y están compiladas en un assembly diferente al resto: System.Data.OracleClient.dll.

ADO.NET provee una arquitectura extensible, posibilitando que terceras partes creen sus propios proveedores de acceso nativo para aplicaciones .NET. Algunos ejemplos de esto son:

- Data Provider For DB2, desarrollado por IBM
- Oracle Data Provider For .NET, desarrollado por Oracle
- Providers de acceso nativo a bases de datos OpenSource, como MySQL y PostgreSQL

Es importante volver a destacar que utilizando estos proveedores de acceso a datos cualquier aplicación .NET puede utilizar casi cualquier base de datos relacional existente en la actualidad como repositorio de información persistente (esto incluye, además de MS SQL Server, a IBM DB2, Oracle, Sybase, Informix, TeraData, MySQL y PostgreSQL, entre otras).

Clases más comunes



En la figura se pueden apreciar las clases más comunes que componen a todos los proveedores de acceso a datos de ADO.NET. Nótese que algunos nombres empiezan con las letras “Xxx”: esto se debe a que los nombres de esas clases varían según el proveedor específico que se esté utilizando. Por ejemplo, la clase que representa una conexión con la base de datos usando el Data Provider For Sql Server es “SqlConnection”, mientras que si usamos el Data Provider For Oracle podemos obtener la misma funcionalidad de la clase “OracleConnection”. Mas allá del ejemplo, pasemos a describir cada una de estas clases y su funcionalidad:

ADO.NET consiste en dos partes primarias:

- **Data provider**

  Estas clases proporcionan el acceso a una fuente de datos, como Microsoft SQL Server y Oracle. Cada fuente de datos tiene su propio conjunto de objetos del proveedor, pero cada uno tiene un conjunto común de clases de utilidad:

  - ✓ **Connection:** Para conectar con una base de datos y administrar las transacciones en una base de datos. Proporciona una conexión usada para comunicarse con la fuente de datos. También actúa como Abstract Factory para los objetos command.
  - ✓ **Command:** Para emitir comandos SQL a una base de datos.

    Usado para realizar alguna acción en la fuente de datos, como lectura, actualización, o borrado de datos relacionales.

  - ✓ **Parameter:** Describe un simple parámetro para un command. Un ejemplo común es un parámetro para ser usado en un procedimiento almacenado.
  - ✓ **DataAdapter:** Para insertar datos en un objeto DataSet y reconciliar datos de la base de datos.

    "Puente" utilizado para transferir data entre una fuente de datos y un objeto DataSet

  - ✓ **DataReader:** Proporcionan una forma de leer una secuencia de registros de datos sólo hacia delante desde un origen de datos SQL Server.

Es una clase usada para procesar eficientemente una lista grande de resultados, un registro a la vez.

- **DataSets**

Para almacenar datos sin formato, datos XML y datos relacionales, así como para configurar el acceso remoto y programar sobre datos de este tipo.

Los objetos DataSets, un grupo de clases que describen una simple base de datos relacional en memoria, fueron la estrella del show en el lanzamiento inicial (1.0) del Microsoft .NET Framework. Las clases forman una jerarquía de contención:

- **Un objeto DataSet** representa un esquema (o una base de datos entera o un subconjunto de una). Puede contener las tablas y las relaciones entre esas tablas.
  - ✓ **Un objeto DataTable** representa una sola tabla en la base de datos. Tiene un nombre, filas, y columnas.
  - ✓ **Un objeto DataView** "se sienta sobre" un DataTable y ordena los datos (como una cláusula "order by" de SQL) y, si se activa un filtro, filtra los registros (como una cláusula "where" del SQL). Para facilitar estas operaciones se usa un índice en memoria. Todas las DataTables tienen un filtro por defecto, mientras que pueden ser definidos cualquier número de DataViews adicionales, reduciendo la interacción con la base de datos subyacente y mejorando así el desempeño.
    - **Un DataColumn** representa una columna de la tabla, incluyendo su nombre y tipo.
    - **Un objeto DataRow** representa una sola fila en la tabla, y permite leer y actualizar los valores en esa fila, así como la recuperación de cualquier fila que esté relacionada con ella a través de una relación de clave primaria - clave extranjera.
    - **Un DataRowView** representa una sola fila de un DataView, la diferencia entre un DataRow y el DataRowView es importante cuando se está interactuando sobre un resultset.

- ✓ **Un DataRelation** es una relación entre las tablas, tales como una relación de clave primaria - clave ajena. Esto es útil para permitir la funcionalidad del DataRow de recuperar filas relacionadas.

- ✓ **Un Constraint** describe una propiedad de la base de datos que se debe cumplir, como que los valores en una columna de clave primaria deben ser únicos. A medida que los datos son modificados cualquier violación que se presente causará excepciones.

Un DataSet es llenado desde una base de datos por un DataAdapter cuyas propiedades Connection y Command que han sido iniciados. Sin embargo, un DataSet puede guardar su contenido a XML (opcionalmente con un esquema XSD), o llenarse a sí mismo desde un XML, haciendo esto excepcionalmente útil para los servicios web, computación distribuida, y aplicaciones ocasionalmente conectadas.

CONEXIONES

Para establecer la comunicación con bases de datos, se utilizan las conexiones y se representan mediante clases específicas de proveedor, como SQLConnection. Los comandos viajan por las conexiones y devuelven conjuntos de resultados en forma de secuencias que puede leer un objeto DataReader o que se pueden insertar en un objeto DataSet.

En el ejemplo siguiente se muestra la forma de crear un objeto de conexión. Las conexiones se pueden abrir explícitamente mediante llamadas al método Open de la conexión; también se pueden abrir implícitamente al utilizar un objeto DataAdapter.

ADO .NET puede usar:

_ **El proveedor SQL Server:** es la forma más aconsejable de conectarse a una Base de Datos

SQL Server (versión 7.0 o superior). Usa un protocolo propietario para conectarse a la Base de

Datos y se basa en el namespace System.Data.Sqlclient.

_ **El proveedor OLEDB:** se usa con Bases de Datos que soportan interfaces. ADO.NET entre los que se encuentran SQLOLEDB (Microsoft OLE DB Provider for SQL Server)

, MSDAORA (Microsoft OLE DB Provider for Oracle), Microsoft.Jet.OLEDB.4.0 (OLE DB

Provider for Microsoft Jet) y VFPOLEDB (Provider para Visual FoxPro), entre otros. Se basan en el namespace System.Data.Oledb.

_ **El proveedor ODBC:** se usa con Bases de Datos que soportan accesos a datos mediante ODBC.

Aquí nos fijaremos en los proveedores OLEDB y ODBC.

COMO ACCEDER A DATOS UTILIZANDO ADO.NET.

ADO.NET es una tecnología de acceso a datos que se basa en los objetos ADO (Objetos de Datos ActiveX) anteriores.

Es una manera nueva de acceder a los datos construida sobre ADO. ADO.NET puede coexistir con ADO.

ADO.NET también es un conjunto de clases que exponen servicios de acceso a datos al programador de .NET.

ADO.NET proporciona un conjunto variado de componentes para crear aplicaciones distribuidas de uso compartido de datos. Forma parte integral de .NET Framework, y proporciona acceso a datos relacionales, datos XML y datos de aplicaciones.

ADO.NET es compatible con diversas necesidades de programación, incluida la creación de clientes de bases de datos clientes y objetos empresariales de nivel medio utilizados por aplicaciones, herramientas, lenguajes o exploradores de Internet.

ADO.NET utiliza un modelo de acceso pensado para entornos desconectados. Esto quiere decir que la aplicación se conecta al origen de datos, hace lo que tiene que hacer,

por ejemplo seleccionar registros, los carga en memoria y se desconecta del origen de datos.

ADO.NET es un conjunto de clases que usted utiliza para acceder y manipular orígenes de datos como por ejemplo, una base de datos en SQL Server o una planilla Excel.

ADO.NET utiliza XML como el formato para transmitir datos desde y hacia su base de datos y su aplicación Web.

Hay 3 espacios de nombres que se importará en un formulario Web o formulario Windows si está usando ADO.NET:

o System.Data.

o System.Data.SqlClient.

o System.Data.OleDb.

El modelo de objetos ADO.NET provee una estructura de acceso a distintos orígenes de datos.

Tiene 2 componentes principales: El Dataset y el proveedor de Datos .NET

CARACTERISTICAS:

- ✓ Trabaja desconectado del origen de datos que se utilice.
- ✓ Tiene una fuerte integración con XML Y ASP.NET.
- ✓ El uso de ADO.NET es independiente del lenguaje de programación que se utilice.

LOS ESPACIOS DE NOMBRES DE ADO.NET

ADO.NET se encuentra en la biblioteca System.Data.dll, y ofrece clases en cinco espacios de nombres bien diferenciados que explico brevemente a continuación:

- **System.Data:** es el espacio de nombres primario. Dentro de este espacio de nombres tenemos un conjunto de clases que representan, digamos, una base de datos virtual, tablas, filas, columnas, relaciones, etc. Sin embargo, ninguna de estas clases ofrece conexión alguna con un origen de datos, sino que simplemente representan los datos en sí mismos.

- **System.Data.Common:** ofrece clases comunes entre distintos orígenes de datos. Para lo que vamos a tratar en este artículo, podemos decir que estas clases sirven de clase base para las que están contenidas en los dos espacios de nombres que vienen a continuación.

- **System.Data.OleDb:** contiene una serie de clases que nos permiten conectarnos con cualquier origen de datos e interactuar con él al tiempo que sirven de "intermediarios" entre el origen de datos y las clases del espacio de nombres System.Data que, según decíamos, no tienen conexión alguna con dicho origen de datos. Las clases de System.Data.OleDb usan OLEDB como tecnología subyacente.

- **System.Data.SqlClient:** contiene clases que permiten interactuar con orígenes de datos SQL Server de un modo mucho más directo que OLEDB, mejorando el rendimiento para este tipo de origen de datos. Por lo tanto, solamente se pueden utilizar para acceder a bases de datos de SQL Server. El uso de sus clases es prácticamente equivalente al de las que se encuentran en System.Data.OleDb.

- **System.Data.SqlTypes:** este espacio de nombres ofrece los tipos primitivos que usa SQL Server. Obviamente, aunque se pueden usar los tipos equivalentes del CTS, los que se incluyen en este espacio de nombres están optimizados para trabajar con SQL Server.

LAS CLASES BÁSICAS DE ADO.NET

Las he llamado básicas porque, realmente, serán el pilar fundamental para casi cualquier aplicación que tenga que acceder a un origen de datos con unos mínimos requisitos de eficiencia y escalabilidad. Las clases más importantes del espacio de nombres System.Data son:

- **DataSet:** aun a riesgo de equivocarme, diré que es la piedra angular del modelo de objetos: **Esta clase permite tener en memoria una auténtica "base de datos virtual"**, con sus tablas, relaciones, etc. Un hecho importante es que esta base de datos virtual está total y absolutamente desconectada de cualquier origen de datos físico y, en consecuencia, **siempre** se aloja toda entera en la memoria. En otras palabras, puede contener uno o varios conjuntos de filas distintos, que pueden estar o no estar relacionados entre sí, pero siempre en la memoria y siempre desconectados del origen de datos. La clase DataSet está dentro del espacio de nombres System.Data.

- **DataTable:** un DataTable representa un conjunto de filas y columnas también en memoria y desconectado del origen de datos, como el DataSet. Pertenece al espacio de nombres System.Data. La propiedad Tables de un objeto DataSet contiene una colección de objetos DataTable, y dicha colección es de la clase DataTableCollection. Por otra parte, cada objeto DataTable representa sus filas en la propiedad Rows (de la clase DataRowCollection), siendo cada fila, a su vez, un objeto de la clase DataRow, y sus columnas en la propiedad Columns (de la clase DataColumnCollection), siendo cada columna un objeto de la clase DataColumn.

- **DataRelation**: representa una relación entre dos objetos DataTable. También pertenece al espacio de nombres System.Data. Todas las relaciones que haya en un DataSet se encuentran en la colección Relations, de la clase DataRelationCollection.

Como podrás suponer, hay más, pero no serán necesarias para el desarrollo de este artículo. Ahora, vamos a pasar a ver las clases más importantes del espacio de

nombres System.Data.SqlClient, y sus equivalentes en el espacio de nombres System.Data.OleDb:

- **SqlConnection:** su equivalente en OleDb es **OleDbConnection.** Son más o menos equivalentes a la clase Connection del antiguo ADO, en tanto en cuanto que proporcionan la conexión con el origen de datos y mantiene algunas de sus antiguas propiedades y métodos, como son ConnectionString, ConnectionTimeOut, Open y Close. Sin embargo, al igual que las otras clases que también tienen un equivalente (más o menos) en la tecnología antigua, ten presente que no se manejan exactamente igual, como tendremos ocasión de ver más adelante.

- **SqlCommand:** su equivalente en OleDb es **OleDbCommand**. También son parecidos a los antiguos objetos Command de ADO y, como estos, representan procedimientos almacenados o instrucciones SQL que se ejecutan en el origen de datos.

- **SqlParameter:** su equivalente en OleDb es **OleDbParameter**. Del mismo modo que los dos anteriores, se parecen a los objetos Paremeter del antiguo ADO, y representan un parámetro dentro de la colección Parameters del objeto SqlCommand u OleDbCommand, según el caso. La colección Parameters de un objeto SqlCommand es de la clase SqlParameterCollection (OleDbParameterCollection en OleDb).

- **SqlDataAdapter:** su equivalente en OleDb es **OleDbDataAdapter**. Esta clase no tiene nada parecido (ni de lejos) en el antiguo ADO. Contiene un conjunto de objetos SqlCommand (ú OleDbCommand, según proceda) en sus propiedades SelectCommand, InsertCommand, UpdateCommand y DeleteCommand. Cuando se invoca el método Fill, el DataAdapter rellena un objeto DataSet o DataTable con el conjunto de filas resultante de ejecutar el comando establecido en su propiedad SelectCommand. Cuando se invoca el método Update, el DataAdapter ejecuta el comando establecido en su propiedad InsertCommand para añadir al origen de

datos las filas nuevas añadidas a un DataTable, el comando UpdateCommand para modificar las filas que hayan sido modificadas en el DataTable y el comando DeleteCommand para eliminar las filas que hayan sido eliminadas en el DataTable. Por lo tanto, el DataAdapter es el nexo que une los objetos DataSet y DataTable, totalmente desconectados, con el origen de datos físico. Si la conexión estaba cerrada antes de ejecutar los métodos Fill o Update, el DataAdapter se ocupa de abrir dicha conexión para efectuar la operación requerida, cerrando de nuevo la conexión una vez que ha terminado. Si la conexión estaba abierta, el DataAdapter deja que dicha conexión siga abierta después de haber terminado.

- **SqlCommandBuilder:** su equivalente en OleDb es **OleDbCommandBuilder**. Tampoco había nada en el antiguo ADO que hiciera lo que hace esta clase. Sencillamente se ocupa de generar los objetos command necesarios para un determinado DataAdapter, gracias a los métodos GetInsertCommand, GetUpdateCommand y GetDeleteCommand.

OBJETIVOS DE DISEÑO PARA ADO.NET

A medida que la programación de aplicaciones ha evolucionado, las nuevas aplicaciones se han convertido en aplicaciones de correspondencia imprecisa basadas en el modelo de aplicación Web. Las aplicaciones de hoy en día utilizan cada vez más XML para codificar datos que se van a pasar a través de conexiones de red. Las aplicaciones Web utilizan HTTP para las comunicaciones entre niveles y, por tanto, deben controlar expresamente el mantenimiento del estado de una solicitud a otra. Este nuevo modelo es muy diferente del estilo de programación con conexión y de correspondencia precisa que caracterizaba la época cliente-servidor, en la que una conexión permanecía abierta durante toda la vida del programa y no hacía falta controlar el estado.

A la hora de diseñar herramientas y tecnologías para satisfacer las necesidades del programador de hoy en día, Microsoft se dio cuenta de que hacía falta un modelo de programación totalmente nuevo para el acceso a datos, un modelo basado en .NET Framework. Tomar .NET Framework como base garantizaba que la tecnología de acceso a datos sería uniforme: los componentes compartirían un sistema de tipos, unos modelos de diseño y unas convenciones de nomenclatura.

ADO.NET se diseñó para cumplir con los objetivos de este nuevo modelo de programación: arquitectura de datos sin mantener una conexión abierta, estrecha integración con XML, representación común de datos con la posibilidad de combinar datos procedentes de múltiples y variados orígenes, y servicios optimizados para interactuar con una base de datos, todo ello nativo de .NET Framework.

A la hora de crear ADO.NET, Microsoft se propuso los siguientes objetivos de diseño:

- Aprovechar la tecnología de objetos ADO (ActiveX Data Objects) actuales.
- Admitir el modelo de programación n-tier
- Integrar la compatibilidad con XML

VENTAJAS DE ADO.NET

ADO.NET ofrece varias ventajas sobre las anteriores versiones de ADO y sobre otros componentes de acceso a datos. Estas ventajas se incluyen en las siguientes categorías:

- **Interoperabilidad**

Las aplicaciones ADO.NET pueden aprovechar la flexibilidad y la amplia aceptación de XML. Dado que XML es el formato de transmisión de conjuntos de datos a través de la red, cualquier componente que pueda leer el formato XML podrá procesar los datos. En realidad, no es necesario en absoluto que el componente receptor sea un componente ADO.NET: el componente transmisor puede transmitir simplemente el conjunto de datos a su destino, independientemente de cómo esté implementado el componente receptor. El componente de destino podría ser una aplicación de Visual Studio o cualquier otra aplicación implementada con cualquier herramienta. El único requisito es que el componente receptor pueda leer XML. Como estándar del sector, XML se diseñó precisamente con el propósito de alcanzar este tipo de interoperabilidad.

- **Mantenibilidad**

A lo largo de la vida de un sistema implementado es posible hacer cambios modestos, pero raramente se intenta hacer cambios importantes, estructurales, debido a su dificultad. Es de lamentar que sea así puesto que, en el curso natural de los acontecimientos, tales cambios importantes pueden hacerse necesarios. Por ejemplo, a

medida que una aplicación implementada se hace más popular entre los usuarios, es posible que el aumento de la carga de rendimiento haga necesarios cambios estructurales. A medida que crece la carga de rendimiento en un servidor de la aplicación implementada, los recursos del sistema pueden escasear y el tiempo de respuesta y el rendimiento pueden resentirse. Ante este problema, los arquitectos de software pueden tomar la decisión de dividir en el servidor el procesamiento de la lógica de empresa y el procesamiento de la interfaz de usuario en niveles diferentes y en equipos separados. En la práctica, el nivel del servidor de aplicación se reemplaza con dos niveles, lo que alivia la escasez de recursos del sistema.

El problema no consiste en diseñar una aplicación de tres niveles. Por el contrario, se trata de aumentar el número de niveles después de implementar una aplicación. Si la aplicación original se implementó en ADO.NET mediante conjuntos de datos, esta transformación resulta más sencilla. Cuando reemplace un solo nivel por dos niveles, no olvide organizar estos dos niveles para que intercambien información. Dado que los niveles pueden transmitir datos por medio de conjuntos de datos con formato XML, la comunicación es relativamente fácil.

- **Programabilidad**

En Visual Studio, los componentes de datos ADO.NET encapsulan funcionalidad de acceso a datos de diversas formas que ayudan a programar de modo más rápido y con menos errores. Por ejemplo, los comandos de datos condensan la tarea de generar y ejecutar instrucciones SQL o procedimientos almacenados.

De igual forma, las clases de datos ADO.NET generadas por las herramientas del diseñador tienen como resultado conjuntos de datos con tipo. Esto, a su vez, permite el acceso a los datos mediante programación con tipo.

- **Rendimiento**

Para las aplicaciones desconectadas, los conjuntos de datos ADO.NET ofrecen ventajas de rendimiento frente a los conjuntos de registros ADO desconectados. Cuando se utiliza el cálculo de referencias de COM para transmitir un conjunto de registros desconectado entre niveles, la conversión de los valores del conjunto de registros a tipos de datos

reconocibles por COM puede suponer un costo de procesamiento significativo. En ADO.NET, tal conversión de tipos de datos no es necesaria.

- **Escalabilidad**

Dado que el Web puede incrementar en gran medida la demanda de datos, la escalabilidad adquiere una importancia crucial. Las aplicaciones para Internet tienen un suministro ilimitado de usuarios potenciales. Aunque una aplicación pueda dar un buen servicio a una docena de usuarios, eso no significa que pueda dar un servicio igualmente bueno a cientos o cientos de miles de ellos. Una aplicación que consuma recursos tales como bloqueos y conexiones de base de datos no dará un buen servicio a un gran número de usuarios, porque la demanda de recursos limitados por parte de los usuarios acabará por superar su disponibilidad.

Para facilitar la escalabilidad, ADO.NET anima a los programadores a ahorrar recursos limitados. Las aplicaciones ADO.NET utilizan un acceso desconectado a los datos, por lo que no retienen bloqueos ni conexiones activas con bases de datos durante largos períodos de tiempo.