Alternativas de acceso a datos

Memoria

Autor : José Manuel Ponce Honorato

Consultor: Xavier Navarro Esteve

11 de Abril de 2012

Dedicado a mi familia

INDICE

1 – Descripción del Proyecto

1.1 – Resumen

Este proyecto tiene como objetivo principal el aprendizaje de las tecnologías de acceso a datos .NET de Microsoft, se trata de hacer un análisis de estas diferentes técnicas para poder crear una documentación precisa que nos permita comparar estas tres tecnologías, y para ello se detallarán las principales ventajas y desventajas de las mismas, comparándolas entre sí.

Para el aprendizaje y comparativa de estas tecnologías se ha desarrollado una web cuya funcionalidad se limita a la implementación de los diferentes tipos de acceso en diferentes situaciones de acceso a datos.

Las tecnologías de acceso a datos a estudiar son:

1. ADO.NET: Es un conjunto de componentes del software que pueden ser usados por los programadores para acceder a datos y a servicios de datos. Es una parte de la biblioteca de clases base que están incluidas en el Microsoft .NET Framework. Es comúnmente usado por los programadores para acceder y para modificar los datos almacenados en un Sistema Gestor de Bases de Datos Relacionales, aunque también puede ser usado para acceder a datos en fuentes no relacionales. ADO.NET.

2. ENTITY FRAMEWORK: Permite a los desarrolladores crear aplicaciones de acceso a datos programando con un modelo de aplicaciones conceptuales en lugar de programar directamente con un esquema de almacenamiento relacional. El objetivo es reducir la cantidad de código y el mantenimiento necesarios para las aplicaciones orientadas a datos.

3. oDATA: es un protocolo Web para consultar y actualizar los datos, se basa en tecnologías Web como HTTP, Atom Publishing Protocol (AtomPub) y JSON proveyendo acceso a la información desde una variedad de aplicaciones, servicios y almacenamientos.

El objetivo de este proyecto no es el de hacer un aplicativo utilizando cada una de estas tecnologías, consiste en hacer un estudio del funcionamiento de cada tecnología una dependiendo del tipo de acceso a datos que necesite hacer en cada momento y que se pueda elaborar una comparativa entre ellas, proporcionando una información que nos permita estudiar las ventajas y desventajas de cada método dependiendo de la necesidad en la que nos encontremos.

1.2- Definición del alcance

1.2.1 - Objetivos del proyecto

Con el desarrollo de este proyecto se pretende evaluar los diferentes sistemas de acceso a datos, oData, ADO.NET Entity Framework y acceso de datos tradicional para poder evaluar la funcionalidad de estos y obtener una guía de cuál es el método más apropiado para cada aplicación, además de lo anteriormente expuesto se pretende:

- Hacer un estudio de las diferentes situaciones en el que uno de los sistemas resulte ventajoso con respecto a los otros.
- Evaluar el estado actual de la tecnología de acceso a datos en el entrono .NET.

Este resultado lo conseguiremos mediante el estudio de los tres sistemas de acceso a datos y con un aplicativo que testeara cada uno de los sistemas propuestos en diferentes condiciones.

1.2.3 - Requisitos y características del proyecto

Disponer de Visual Studio 2010, ya que también incluye un aprendizaje de la tecnología WebForms incluida en la plataforma NET para el desarrollo del aplicativo que se encargara de poner en práctica los conocimientos adquiridos.

Disponer de SQL Server 2008, esto es debido que la tecnología que utilizamos va íntimamente ligada a un sistema de persistencia, en este caso utilizaremos todos los componentes del mismo fabricante para minimizar cualquier problema de compatibilidad entre productos.

. Generar un conjunto de pruebas test que puedan garantizar una correcta evaluación ya que este proyecto pretende hacer un estudio entre los diferentes sistemas de los que disponemos actualmente usando la tecnología NET, al término del cual nos permitirá evaluar las ventajas y desventajas entre los diferentes sistemas

1.2.4 - Enfoque y método aplicado

El método que se sigue para conseguir el objetivo principal del proyecte, que es el de analizar las diferencias entre las diferentes tecnologías de acceso a datos y la utilidad real de disponer de diferentes métodos, es el de hacer un estudio detallado de los diferentes sistemas y desarrollar una aplicación que pruebe todas estas metodologías desde diferentes puntos de vista. De esta forma se podrá analizar, en cada caso las ventajas e inconvenientes de estar utilizar un sistema u otro.

1.2.5 - Tecnologías empleadas

Para el desarrollo de este proyecto utilizamos toda la arquitectura .NET desarrollada por Microsoft, más concretamente Visual Studio 2010 y SQL Server 2008.

Microsoft.NET es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con los objetivos de:

- Mejorar sus sistemas operativos.
- Mejorar su modelo de componentes COM+.
- Obtener un entorno específicamente diseñado para el desarrollo y ejecución del software en forma de servicios que puedan ser tanto publicados como accedidos a través de Internet de forma independiente del lenguaje de programación, modelo de objetos, sistema operativo y hardware utilizados tanto para desarrollarlos como para publicarlos.

2 – Planificación del Proyecto

2.1 – Estructura de distribución del trabajo

Código: 1.1	Nombre: Elección PFC
Descripción	Selección del trabajo PFC propuesto por el tutor
Recursos	Enunciados proporcionados por el tutor
Duración	2 días
Resultados	Elección del proyecto a realizar
Restricciones	
Dependencias	
Criterios aceptación	Aprobación por el tutor

Código: 1.2	Nombre: Documentación
Descripción	Buscar información preliminar de los diferentes métodos de acceso
Recursos	Internet y biblioteca
Duración	5 días
Resultados	Visión general del alcance del proyecto
Restricciones	
Dependencias	
Criterios aceptación	

Código: 1.3	Nombre: Reunión presencial
Descripción	Reunión presencial con el consultor para ampliar detalles de cómo realizar el proyecto y las entregas
Recursos	
Duración	1 días
Resultados	Visión general del alcance del proyecto
Restricciones	
Dependencias	
Criterios aceptación	

Código: 1.4	Nombre: Elaboración plan de trabajo
Descripción	Documento que contiene todo el plan de trabajo a realizar
Recursos	Ejemplos años anteriores
Duración	3 días
Resultados	Documento con la planificación del proyecto
Restricciones	
Dependencias	Fechas de entregas preestablecidas para el proyecto
Criterios aceptación	Que sea un reflejo fiel del trabajo a realizar en el proyecto

Código: 2.1	Nombre: Documentación
Descripción	Buscar información exhaustiva de los diferentes métodos de acceso para poder hacer una documentación lo más completa posible
Recursos	Internet y biblioteca
Duración	2 días
Resultados	Documentación y bibliografía para el proyecto
Restricciones	
Dependencias	
Criterios aceptación	Bibliografía recogida en el proyecto

Código: 2.2	Nombre: ADO NET
Descripción	Documentación del funcionamiento y especificaciones técnicas del acceso a datos del tipo Ado NET
Recursos	Documentación y visual Studio 2010
Duración	3 días
Resultados	Análisis de funcionamiento de este tipo de acceso a datos
Restricciones	
Dependencias	Tiempo invertido en la investigación del resto de sistemas de acceso a datos.
Criterios aceptación	Documentación completa de esta tecnología

Código: 2.3	Nombre: Entity Framework
Descripción	Documentación del funcionamiento y especificaciones técnicas del acceso a datos del tipo Entity Framework
Recursos	Documentación y visual Studio 2010
Duración	3 días
Resultados	Análisis de funcionamiento de este tipo de acceso a datos
Restricciones	
Dependencias	Tiempo invertido en la investigación del resto de sistemas de acceso a datos.
Criterios aceptación	Documentación completa de esta tecnología

Código: 2.4	Nombre: oDATA
Descripción	Documentación del funcionamiento y especificaciones técnicas del acceso a datos del tipo oDATA
Recursos	Documentación y visual Studio 2010
Duración	3 días
Resultados	Análisis de funcionamiento de este tipo de acceso a datos
Restricciones	
Dependencias	Tiempo invertido en la investigación del resto de sistemas de acceso a datos.
Criterios aceptación	Documentación completa de esta tecnología

Código: 2.5	Nombre: LINQ
Descripción	Documentación y especificaciones del lenguaje LINQ
Recursos	Documentación y visual Studio 2010
Duración	3 días
Resultados	Análisis de funcionamiento del tipo de tratamiento de los datos
Restricciones	
Dependencias	Tiempo invertido en la investigación del resto de sistemas de acceso a datos.
Criterios aceptación	Documentación completa de esta tecnología

Código: 2.6	Nombre: Redacción documentación
Descripción	Documentación que recoge el análisis y las comparativas entre los diferentes accesos a datos
Recursos	Documentación
Duración	5 días
Resultados	Documento de entrega PAC 2
Restricciones	
Dependencias	
Criterios aceptación	Que cumpla con los requisitos de la PAC 2

Código: 3.1	Nombre: Implementación aplicación
Descripción	Elaboración del programa que sirve para probar todas las tecnologías investigadas e implementarlas en el estudio.
Recursos	Visual Studio 2010
Duración	20 días
Resultados	Programa para evaluación de las diferentes tecnologías
Restricciones	
Dependencias	
Criterios aceptación	Que contenga ejemplos de la implementación de las diferentes tecnologías

Código: 3.2	Nombre: Pruebas
Descripción	Pruebas funcionales del aplicativo elaborado para entrega.
Recursos	Visual Studio 2010
Duración	2 días
Resultados	Verificación del perfecto funcionamiento del aplicativo
Restricciones	
Dependencias	
Criterios aceptación	Un correcto funcionamiento.

Código: 3.3	Nombre: Redacción documentación
Descripción	Documentación que recoge el análisis y la estructura del aplicativo
Recursos	
Duración	8 días
Resultados	Documento de entrega PAC 3
Restricciones	
Dependencias	
Criterios aceptación	Que cumpla con los requisitos de la PAC 3

Código: 4.1	Nombre: Memoria
Descripción	Documentación que recoge todo el trabajo realizado en las diferentes entregas realizadas durante el proyecto
Recursos	PAC 1, PAC 2, PAC 3
Duración	10 días
Resultados	Documento de entrega Memoria
Restricciones	
Dependencias	
Criterios aceptación	Que recoja todo el trabajo realizado en las diferentes PACs

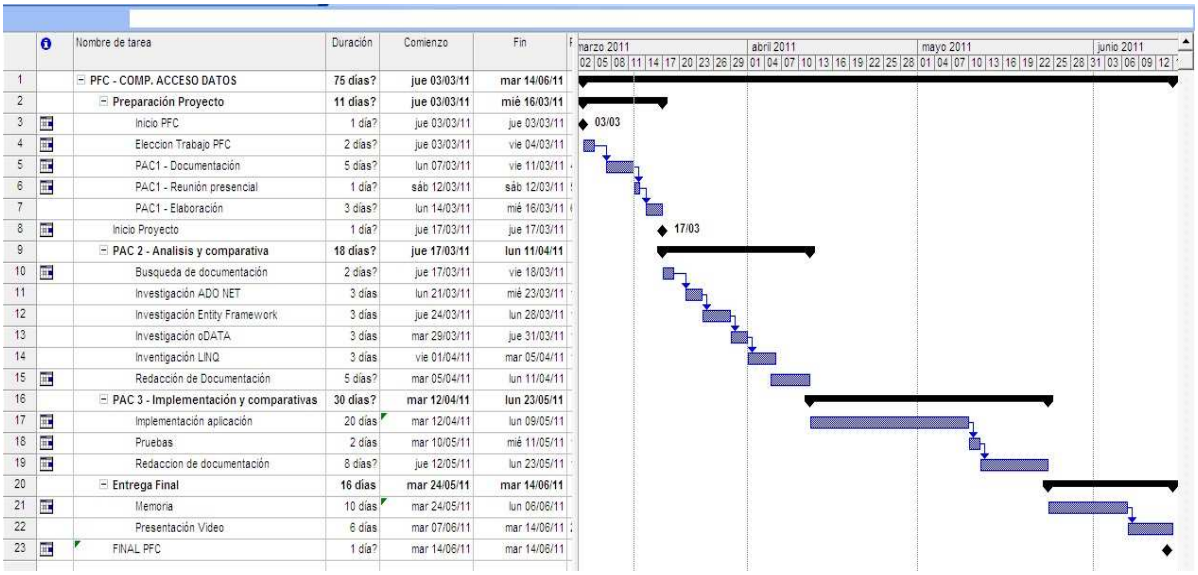
Código: 4.2	Nombre: Memoria
Descripción	Presentación en video del proyecto
Recursos	Memoria
Duración	6 días
Resultados	Documento de entrega Memoria
Restricciones	
Dependencias	
Criterios aceptación	Que explique de una forma clara una síntesis del proyecto

2.2 – Calendario de trabajo

El calendario para realizar el proyecto se proyecto para trabajar 2 horas diarias.

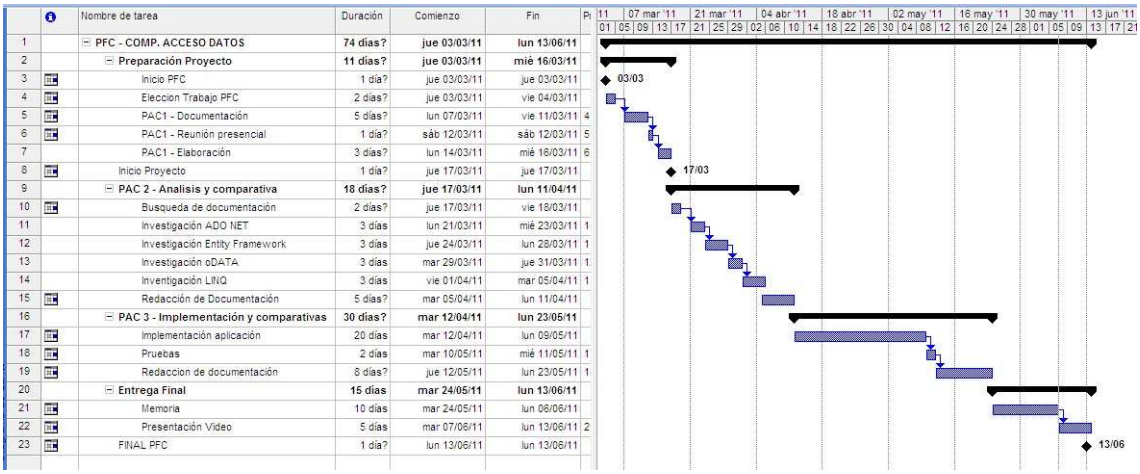
La fecha límite para la entrega final es el 13 de junio de 2011.

Planificación inicial:



	Nombre de tarea	Duración	Comienzo	Fin
1	PFC - COMP. ACCESO DATOS	75 días?	jue 03/03/11	mar 14/06/11
2	Preparación Proyecto	11 días?	jue 03/03/11	mié 16/03/11
3	Inicio PFC	1 día?	jue 03/03/11	jue 03/03/11
4	Eleccion Trabajo PFC	2 días?	jue 03/03/11	vie 04/03/11
5	PAC1 - Documentación	5 días?	lun 07/03/11	vie 11/03/11
6	PAC1 - Reunión presencial	1 día?	sáb 12/03/11	sáb 12/03/11
7	PAC1 - Elaboración	3 días?	lun 14/03/11	mié 16/03/11
8	Inicio Proyecto	1 día?	jue 17/03/11	jue 17/03/11
9	PAC 2 - Analisis y comparativa	18 días?	jue 17/03/11	lun 11/04/11
10	Busqueda de documentación	2 días?	jue 17/03/11	vie 18/03/11
11	Investigación ADO NET	3 días	lun 21/03/11	mié 23/03/11
12	Investigación Entity Framework	3 días	jue 24/03/11	lun 28/03/11
13	Investigación oDATA	3 días	mar 29/03/11	jue 31/03/11
14	Inventigación LINQ	3 días	vie 01/04/11	mar 05/04/11
15	Redacción de Documentación	5 días?	mar 05/04/11	lun 11/04/11
16	PAC 3 - Implementación y comparativas	30 días?	mar 12/04/11	lun 23/05/11
17	Implementación aplicación	20 días	mar 12/04/11	lun 09/05/11
18	Pruebas	2 días	mar 10/05/11	mié 11/05/11
19	Redaccion de documentación	8 días?	jue 12/05/11	lun 23/05/11
20	Entrega Final	16 días	mar 24/05/11	mar 14/06/11
21	Memoria	10 días	mar 24/05/11	lun 06/06/11
22	Presentación Video	6 días	mar 07/06/11	mar 14/06/11
23	FINAL PFC	1 día?	mar 14/06/11	mar 14/06/11

Planificación final:



	Nombre de tarea	Duración	Comienzo	Fin
1	PFC - COMP. ACCESO DATOS	74 días?	jue 03/03/11	lun 13/06/11
2	Preparación Proyecto	11 días?	jue 03/03/11	mié 16/03/11
3	Inicio PFC	1 día?	jue 03/03/11	jue 03/03/11
4	Eleccion Trabajo PFC	2 días?	jue 03/03/11	vie 04/03/11
5	PAC1 - Documentación	5 días?	lun 07/03/11	vie 11/03/11
6	PAC1 - Reunión presencial	1 día?	sáb 12/03/11	sáb 12/03/11
7	PAC1 - Elaboración	3 días?	lun 14/03/11	mié 16/03/11
8	Inicio Proyecto	1 día?	jue 17/03/11	jue 17/03/11
9	PAC 2 - Analisis y comparativa	18 días?	jue 17/03/11	lun 11/04/11
10	Busqueda de documentación	2 días?	jue 17/03/11	vie 18/03/11
11	Investigación ADO NET	3 días	lun 21/03/11	mié 23/03/11
12	Investigación Entity Framework	3 días	jue 24/03/11	lun 28/03/11
13	Investigación oDATA	3 días	mar 29/03/11	jue 31/03/11
14	Inventigación LINQ	3 días	vie 01/04/11	mar 05/04/11
15	Redacción de Documentación	5 días?	mar 05/04/11	lun 11/04/11
16	PAC 3 - Implementación y comparativas	30 días?	mar 12/04/11	lun 23/05/11
17	Implementación aplicación	20 días	mar 12/04/11	lun 09/05/11
18	Pruebas	2 días	mar 10/05/11	mié 11/05/11
19	Redaccion de documentación	8 días?	jue 12/05/11	lun 23/05/11
20	Entrega Final	15 días	mar 24/05/11	lun 13/06/11
21	Memoria	10 días	mar 24/05/11	lun 06/06/11
22	Presentación Video	5 días	mar 07/06/11	lun 13/06/11
23	FINAL PFC	1 día?	lun 13/06/11	lun 13/06/11

La planificación final a quedado muy similar a la proyectada inicialmente, solo existen variaciones en el Pac 3 y en el Pac 4, el mayor problema se centro en el Pac3 la implementación ya que se selecciono el lenguaje ASP MVC para la realización del proyecto y no se ha podido hacer la implementación en Ado .net tradicional ya que uno de los problemas del modelo desconectado DataSet es que no soportaba el patrón Modelo Vista Controlador o la programación en tres capas con lo cual aunque ha variado algo la planificación también se ha invertido tiempo intentado una solución al problema, pero en líneas generales se ha cumplido con el objetivo planificado.

2.3 – Hitos principales

Los hitos principales coinciden con el inicio y las entregas de cada Pac, ya que se ha podido entregar puntualmente el material se considera que la planificación inicial era bastante exacta.

FECHA	DESCRIPCION
03/03/2011	Preparación Proyecto
16/03/2011	PAC 1, plan de trabajo
17/03/2011	Inicio Investigación
11/04/2011	PAC 2, Investigación y diseño de la aplicación
12/04/2011	Inicio implementación
23/05/2011	PAC 3, Implementación
24/05/2011	Inicio memoria y presentación
13/06/2011	PAC 4, Entrega final

2.4 – Productos entregados

Los productos que se han realizado durante el PFC han sido:

Documento de plan de trabajo.

Documento de requisitos.

Investigación de los diferentes modelos de acceso a datos.

- Documentación Ado .NET.
- Documentación oData.
- Documentación Entity Framework.

Documento de análisis preliminar:

- Casos de uso.
- Modelo Conceptual.
- Diagrama de clases.
- Diagramas de secuencias.

Implementación incluyendo tres proyectos.

- PfcEntity – Proyecto con acceso a datos con Entity Framework.
- PfcServicioOData – Proyecto que publica un servicio de datos oData.
- PfcClienteOData – Proyecto que consume el servicio de datos oData.
- Manual de usuario del PFC.
- Manual de instalación del PFC.

Memoria.

Presentación virtual del PFC.

2.5 – Resultados Obtenidos

Una vez completado el proyecto se ha conseguido:

A nivel de formación académica: disponer de los conocimientos técnicos necesarios para la realización del PFC, también me ha permitido profundizar en las nuevas técnicas de acceso a datos, principalmente con oData que será de aplicación inmediata en la empresa donde trabajo, concretamente gracias a la investigación

realizada se está proyectando aplicarlo en unos terminales PDA móviles y para la captura de datos se accederá directamente al origen de datos mediante oData, con esto se conseguirá una aplicación a tiempo real fácilmente implementable.

2.6 – Trabajos futuros

Este proyecto ha abierto una puerta que es toda la tecnología LINQ (Languaje Integrated Query) de Microsoft, aunque en este documento se ha hecho referencia a dos de ellas la tecnología LINQ va mucho más lejos, en concreto este proyecto incluye:

Para objetos

- Linq to Objects

Para base de datos relacionales

- Linq to DataSets
- Linq to SQL
- Linq to Entities

Para XML

- Linq to XML

Creo que la simpleza de cómo se pueden introducir consultas los hace un complemento ideal para el lenguaje de programación ya que el objetivo de crear LINQ es permitir que todo el código hecho en Visual Studio (incluidas las llamadas a bases de datos, datasets, XMLs) sean también orientados a objetos. Antes de LINQ, la manipulación de datos externos tenía un concepto más estructurado que orientado a objetos. Además LINQ trata de facilitar y estandarizar el acceso a dichos objetos, su simplicidad radica en que LINQ define operadores *de* consulta estándar que permiten a lenguajes habilitados con LINQ filtrar, enumerar y crear proyecciones de varios tipos de colecciones usando la misma sintaxis.

3 – Arquitectura del proyecto, investigación y diseño

3.1 – Introducción

Históricamente, han existido diferentes modos de acceso a las bases de datos. Las primeras bases de datos permitían, además de manejar el almacenamiento de los datos, escribir el código necesario para interactuar con el usuario final, también empezaron a aparecer lenguajes de programación en donde la finalidad principal era ofrecer una interfaz gráfica rica, pero con sentencias de acceso a datos limitadas. Con el lanzamiento de Visual Basic que ganó popularidad rápidamente, los programadores tuvieron que acostumbrarse a la forma de acceso de los datos que este lenguaje ofrecía, lo cual lo hacía a través de objetos que encapsulaban su tratamiento. Así fue entonces como se sucedieron DAO (*Data Access Objects*), RDO (*Remote Data Objects*) y ADO (*ActiveX Data Objects*).

Desde el punto de vista de la arquitectura, las aplicaciones pasaron de ser monolíticas a cliente-servidor y utilizaban archivos de datos independientes o un sistema de gestión de bases de datos relacionales (RDBMS). Internet se popularizo y las grandes empresas detectaron su potencial para desarrollar aplicaciones en lugar de ser solamente sitios institucionales. Microsoft, por su lado, llevó su tecnología a un estándar propietario, llamado COM (*Component Object Model*), en el cual los objetos utilizados por ciertas aplicaciones podían ser reutilizados fácilmente por otras. De esta forma es como ADO se consolida, ya que un programador de aplicaciones Win32 o un programador ASP aplicaban los mismos objetos para acceder a datos. Cuando ADO estaba siendo utilizado por millones de programadores en todo el mundo, Microsoft se encontraba trabajando en su nueva generación, que en un principio la llamaban ADO+ y que luego se convertiría en ADO.NET.

Podríamos definir ADO.NET como un conjunto de interfaces, clases y estructuras que permiten el acceso a datos desde la plataforma .NET de Microsoft. Permite un modo de acceso a datos desconectado, esto quiere decir que, a través de ADO.NET, sólo estaremos conectados al servidor el tiempo estrictamente necesario para realizar la carga de los datos en el DataSet. Acceso Desconectado. Mediante el acceso desconectado que proporciona la plataforma .NET a través de ADO.NET, reduciremos el número de conexiones aumentando la capacidad de carga de trabajo de nuestro servidor ya que se soportarán una mayor cantidad de usuarios por unidad de tiempo.

3.2 – Metodología de investigación empleada

El proyecto de investigación que presentamos a continuación analiza las diferentes tecnologías que ofrece el entorno .NET, primero desde una primera visión documental enumerando sus características y después haciendo una comparativa de sus funcionalidades con el objetivo común del acceso a datos.

El primer problema con en el nos encontramos es comunicar un programa o aplicación con una base de datos y más que comunicar se pretende que el programa o aplicación realice una serie de procesos u operaciones con la base de datos o mejor aun con el conjunto de tablas que contiene una base de datos, pero esta tarea se realiza de forma diferente dependiendo del método empleado para el acceso a datos.

La primera nota a recordar es que una base de datos puede estar físicamente en el servidor y en alguna carpeta o directorio del disco duro de dicha maquina. Otra cosa que debemos recordar es que así como existen servidores de páginas (Web Server), servidores de correo (Mail Server), servidores de ftp (ftp Server), también existen servidores de bases de datos (DataBase Server), los más comunes son el SqlServer de Microsoft, Oracle, MySql, y muchos más, en nuestro caso nos centraremos en la tecnología Microsoft y utilizamos SqlServer, estos servidores también pueden crear, administrar y procesar una base de datos por si mismos ya que todos disponen de un programa manager para tal fin.

A continuación hacemos una descripción de las tecnologías empleadas en el estudio enumerando su estructura, características y formas particulares de funcionamiento.

3.3 – ADO .NET

Es una evolución del modelo de acceso a datos de ADO que controla directamente los requisitos del usuario para programar aplicaciones escalables. Se diseñó específicamente para el Web, teniendo en cuenta la escalabilidad, la independencia y el estándar XML.

ADO.NET utiliza algunos objetos ADO, como Connection y Command, y también agrega objetos nuevos. Algunos de los nuevos objetos clave de ADO.NET son DataSet, DataReader y DataAdapter.

La diferencia más importante entre esta fase evolucionada de ADO.NET y las arquitecturas de datos anteriores es que existe un objeto, DataSet, que es independiente y diferente de los almacenes de datos. Por ello, DataSet funciona como una entidad independiente. Se puede considerar el objeto DataSet como un conjunto de registros que siempre está desconectado y que no sabe nada sobre el origen y el destino de los datos que contiene. Dentro de un objeto DataSet, de la misma manera que dentro de una base de datos, hay tablas, columnas, relaciones, restricciones, vistas, etc.

El modo de comunicación entre nuestra aplicación y la base de datos implica que ambos manejen un lenguaje de programación común, es decir no se puede mandar una instrucción en C# .net, o en visual Basic .net o en cualquier otro lenguaje, a la base de datos y además esperar que esta última la entienda. Para resolver este problema de comunicación se usa un lenguaje común de bases de datos que tanto los lenguajes de programación existentes como las bases de datos entienden, este lenguaje común de bases de datos es el SQL (Structured Query Languaje) o lenguaje estructurado de consultas.

Para mandar las instrucciones SQL a la base de datos se utilizan los OBJETOS ADO.NET, las cuales proporcionan acceso coherente a orígenes de datos como Microsoft SQL Server, así como a orígenes de datos expuestos mediante OLE DB y XML.

En la actualidad ADO.NET ya es parte del .NET Framework, esto quiere decir que es, de alguna manera, parte del sistema operativo y no un complemento de 4 ó 5 MB que se necesita instala independientemente al cliente o incluido en el instalador de una aplicación. Esto significa que nosotros, como desarrolladores, estaremos enfocados más al acceso a datos y a la lógica para manipular estos datos, y no tendremos porqué preocuparnos si tenemos instalado un complemento u otro.

A continuación se detallan los conceptos más importantes sobre ADO.NET.

3.3.1 - Acceso a datos con 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.
- También podemos decir que ADO.NET es un conjunto de clases que exponen servicios de acceso a datos al programador de .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.
- 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.
- 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.
- Es un conjunto de clases que se pueden utilizar 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 esta tecnología:
    - System.Data.
    - System.Data.SqlClient.
    - 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

3.3.2 - Espacios de nombres para datos en el .NET Framework

Entre los espacios de nombres de .NET Framework relativos a datos y XML se incluyen:

Nombre de la clase	Descripción
System.Data	Espacio de nombres que integra la gran mayoría de clases que habilitan el acceso a los datos de la arquitectura .NET
System.Data.Common	Contiene las clases compartidas para los ".NET Providers"5 (proveedores .NET). Proporcionan la colección de clases necesarias para acceder a una fuente de datos (como por ejemplo una Base de Datos).
System.Data.SqlClient	Espacio de nombres que permite el acceso a proveedores SQL Server en su versión 7.0 y superior. Este espacio de nombres ha sido ampliado para soportar las nuevas características de SQL Server 2005.
System.Data.Sql	Espacio de nombres con multitud de herramientas para interactuar con el nuevo motor de SQL Server 2005: enumeración de servidores, gestión del servidor,
System.Data.OleDB	Espacio de nombres que permite acceder a proveedores .NET que trabajan directamente contra controladores basados en los ActiveX de Microsoft
System.Data.Odbc	Espacio de nombres que contiene las clases que actúan de pasarela con el modelo de drivers ODBC de Windows. Emplean InterOp para acceder a los drivers nativos ODBC, pero proporcionan un marco de compatibilidad hacia atrás muy importante para muchos fabricantes de SW.
System.Data.Oracle	Espacio de nombres, desarrollado por Microsoft, que posibilita el acceso a recursos de sistemas gestores de Oracle. Dependen del cliente nativo de Oracle instalado en la máquina. Es recomendable que accedáis a la web del fabricante para acceder a versiones más actualizadas y desarrolladas por el propio fabricante
System.Data.Internal	Integra el conjunto de clases internas de las que se componen los proveedores de datos.
System.Data.SqlTypes	Proporciona la encapsulación en clases de todos los tipos de datos nativos de SQL Server y sus funciones de manejo de errores, ajuste y conversión de tipos, etc.

3.3.3 - Arquitectura y funcionalidad de ADO.NET

El proveedor de datos .NET provee del enlace entre el Origen de Datos y el DataSet.

Un proveedor de datos de .NET Framework sirve para conectarse a una base de datos, ejecutar comandos y recuperar resultados. Esos resultados se procesan directamente o se colocan en un DataSet de ADO.NET con el fin de exponerlos al usuario para un propósito específico, junto con datos de varios orígenes, o de utilizarlos de forma remota entre niveles. El diseño del proveedor de datos de .NET Framework hace que sea ligero, de manera que cree un nivel mínimo entre el origen de datos y su código, con lo que aumenta el rendimiento sin sacrificar la funcionalidad.

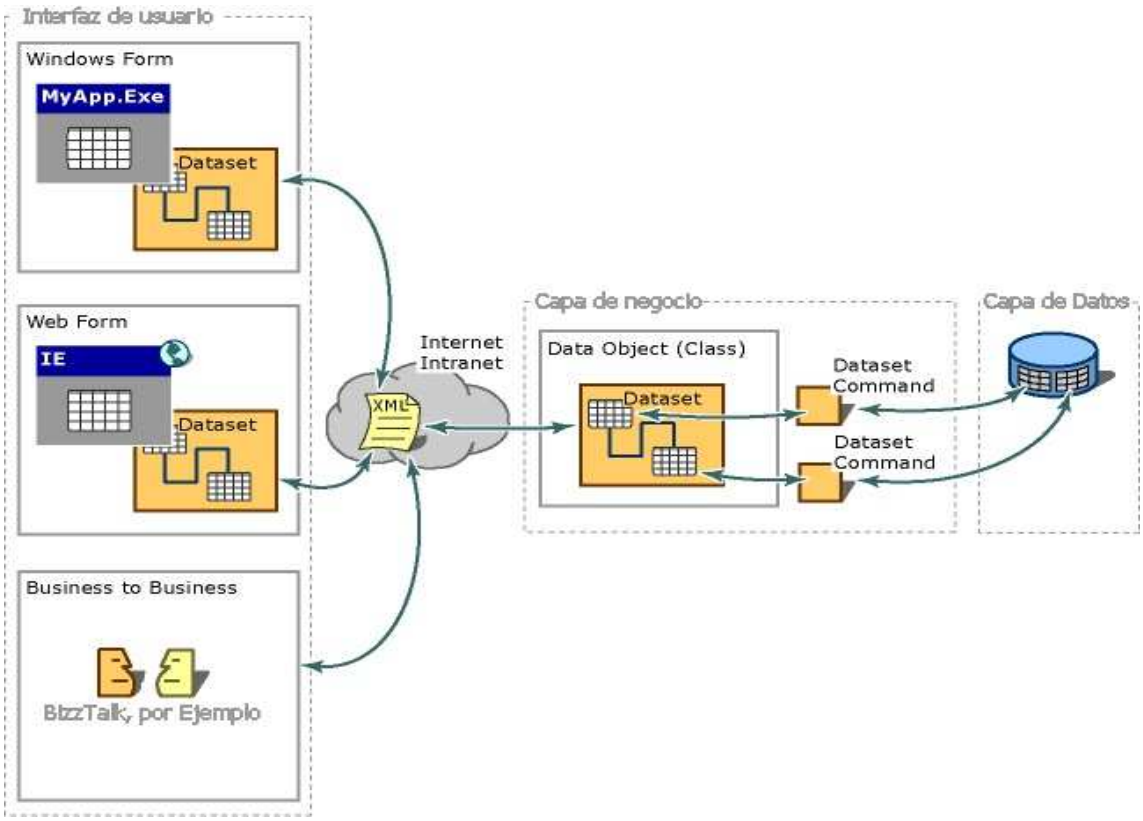
Proveedor de datos de .NET Framework	Descripción
Proveedor de datos de .NET para SQL Server	Para Microsoft SQL Server versión 7.0 o posteriores.
Proveedor de datos de .NET para OLE DB	Para orígenes de datos que se exponen mediante OLE DB.
Proveedor de datos de .NET para ODBC	Para orígenes de datos que se exponen mediante ODBC.
Proveedor de datos de .NET para Oracle	Para orígenes de datos de Oracle. El proveedor de datos de .NET Framework para Oracle es compatible con la versión 8.1.7 y posteriores del software del cliente de Oracle.

3.3.4 - Objetos provistos por distintos proveedores de datos .NET

Los objetos Connection, Command, DataReader y DataAdapter son los elementos fundamentales del modelo de proveedor de datos de .NET Framework. En la tabla siguiente se describen estos objetos.

Objeto	Descripción	Objeto SQL Server	Objeto para un origen OLEDB
Connection	Establece una conexión a un origen de datos determinado.	SqlConnection	OleDBConnection
Command	Ejecuta un comando en un origen de datos.	SqlCommand	OleDBCommand
DataReader	Lee una secuencia de datos de sólo avance y sólo lectura desde un origen de datos.	OleDBDataReader	OleDBConnection
DataAdapter	Llena un DataSet y realiza las actualizaciones necesarias en el origen de datos.	SqlDataAdapter	OleDBDataAdapter

El siguiente grafico muestra los componentes principales de una aplicación ADO.NET.



Los componentes presentados en el gráfico de la esta plataforma de datos son:

Componente	Descripción
Dataset	Modelo jerárquico de representación de una base de datos con arquitectura desconectada.
DataTable	Representa la estructura de una tabla: colección de columnas y filas
Dataset command	Conjunto de procedimientos ADO.NET de ejecución de comandos SQL

3.3.5 - Los Datos se almacenan en DataSets pero también en DataTables

Frecuentemente, las aplicaciones que desarrollamos no necesitan una única fila, sino un conjunto de ellas. Además, también frecuentemente, ese conjunto de filas procede no de una tabla sino de una unión de múltiples tablas (join de tablas). Una vez que estos datos son cargados, la aplicación los trata como un bloque compacto. En un modelo desconectado, es inviable el tener que conectar con la base de datos cada vez que avanzamos un registro para recoger la información asociada a ese registro son las condiciones del join. Para solucionarlo, lo que se realiza es almacenar temporalmente toda la información necesaria donde sea necesario y trabajar con ella. Esto es lo que representa un Dataset en el modelo ADO.NET.

Un DataSet es una caché de registros recuperados de una base de datos que actúa como un sistema de almacenamiento virtual, y que contiene una o más tablas basadas en las tablas reales de la base de datos. Y que, además, almacena las relaciones y reglas de integridad existentes entre ellas para garantizar la estabilidad e integridad de la información de la base de datos. Es importante tener en cuenta que los Datasets son almacenes pasivos de datos, lo que significa que no se ven alterados ante cambios de la base de datos. Es necesario recargarlos (FillDataSet) siempre que queramos estar actualizados en cuanto a datos se refiere.

Una de las mayores ventajas de esta implementación, es que una vez recogido el Dataset, éste puede ser enviado en forma de flujo XML entre distintos componentes de la capa de negocio como si de una variable más se tratase, ahorrando así comunicaciones a través de la base de datos.

Pero la clase DataSet es una clase que consume muchos recursos. En su modelo de clases es posible subdividir su complejidad en clases más pequeñas. Por lo tanto al igual que una base de datos se compone de tablas, un Dataset se compone de DataTables. Pero la novedad no está ahí, sino en proporcionar nuevas funcionalidades que permiten evolucionar DataTable y colocarlo al mismo nivel funcional que DataSet. Esto significa que un DataTable se puede serializar y puede ser utilizado de forma autónoma, sin requerir el uso completo de la clase Dataset contenedora.

3.3.6 - Los Datos persisten en forma de documento XML

En un sistema de trabajo Off-Line como el que plantea ADO.NET, la persistencia es un mecanismo fundamental. Podemos cerrar la aplicación y mantener persistentes todos los DataSets necesarios, de manera que al reiniciarla, nos encontramos los DataSets tal y como los dejamos. Ahorrando el tiempo que hubiera sido necesario para recuperar de nuevo toda esa información del servidor. Optimizando todavía más el rendimiento del sistema distribuido.

El formato que emplea ADO.NET para almacenar su estado es XML. Puesto que ya es un estándar de la industria, esta persistencia nos ofrece:

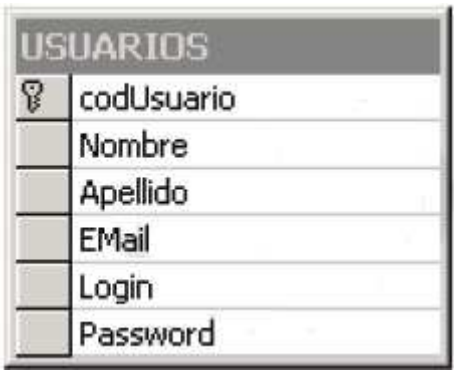
- La información podría estar accesible para cualquier componente del sistema que entienda XML.
- Es un formato de texto plano, no binario. Que lo hace compatible con cualquier componente de cualquier plataforma y recuperable en cualquier caso.

Muchas veces necesitamos conocer el cómo se estructura un Dataset para poder averiguar qué columnas tenemos disponibles, con qué tipo de datos, tamaño, etc. A esta información que define el cómo se estructura la información de le denomina Metadatos (Datos que definen datos).

En el caso de los documentos XML, el que determina la estructura que éstos tienen son los Esquemas. No son necesarios para la codificación del documento XML, pero refuerzan su estructura y establece una manera común de introducir nuevos datos respetando un juego de reglas básico.

En ADO.NET la generación de los esquemas así como de los documentos XML asociados a una base de datos son automáticos y transparentes al usuario. No se necesitará acceder a ellos a bajo nivel, a menos que sea requisito del diseño. Dichos esquemas se actualizarán cuando se modifique la base de datos o las consultas empleadas para acceder a los datos.

Si la tabla de SQL Server, es del estilo:



El esquema generado por el Dataset para esta tabla seria:

```
<xsd:schema id="USUARIOS" targetNamespace="http://www.tempuri.org/USUARIOS.xsd"
    xmlns="http://www.tempuri.org/USUARIOS.xsd"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xsd:element name="USUARIOS">
    <xsd:complexType content="elementOnly">
      <xsd:all>
        <xsd:element name="codUsuario" msdata:ReadOnly="True"
             msdata:AutoIncrement="True" type="xsd:int"/>
        <xsd:element name="Nombre" minOccurs="0" type="xsd:string"/>
        <xsd:element name="Apellido" minOccurs="0" type="xsd:string"/>
        <xsd:element name="EMail" minOccurs="0" type="xsd:string"/>
        <xsd:element name="Login" minOccurs="0" type="xsd:string"/>
        <xsd:element name="Password" minOccurs="0" type="xsd:string"/>
      </xsd:all>
    </xsd:complexType>
    <xsd:key name="Constraint1" msdata:PrimaryKey="True">
      <xsd:selector>.</xsd:selector>
      <xsd:field>codUsuario</xsd:field>
    </xsd:key>
  </xsd:element>
  <xsd:element name="USUARIOS" msdata:IsDataSet="True">
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element ref="USUARIOS"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

3.3.7 - Arquitectura orientada a conexión

Otra posibilidad consiste en estar conectados permanentemente a los datos, esto ocurre cuando una aplicación tiene que estar conectada con el sistema gestor para acceder en “tiempo real” a los datos que almacena. En ADO.NET 2.0 este modelo no ha cambiado mucho respecto a la versión anterior. Es que el que menos ha cambiado, pero incorpora mejoras en la funcionalidad que eran necesarias.

3.3.8 - Los proveedores gestionados de datos “.Net Providers”

Los proveedores gestionados de datos hacen referencia a los mecanismos por los cuales un programa puede acceder a los recursos de un servidor de bases de datos. Son el conjunto de componentes que interactúan como mediadores para independizar el programa de la base de datos. En términos ya conocidos, son el equivalente a la arquitectura OLE-DB de Microsoft. Un conjunto de componentes que encapsulan el acceso, manejo de cursores y comunicaciones a un servidor de datos. En el caso de ADO.NET los proveedores gestionados están encapsulados en un conjunto de clases que, precisamente, hacen transparente al programador el acceso a los recursos de los drivers de acceso a un servidor de datos. Son el modelo de clases del API de programación de un origen de datos.

Por este motivo nos encontremos con clases específicas de algunos fabricantes como es el case de *System.Data.SqlClient*, que encapsula toda la potencia y flexibilidad de la API de SQL Server. De esta manera el acceso a los datos es más directo y no se requiere de multitud de componentes intermedios para realizar una acción, ni perder funciones específicas por compatibilidad con otras plataformas.

En la plataforma .NET se ofrecen con el SDK, los proveedores gestionados:

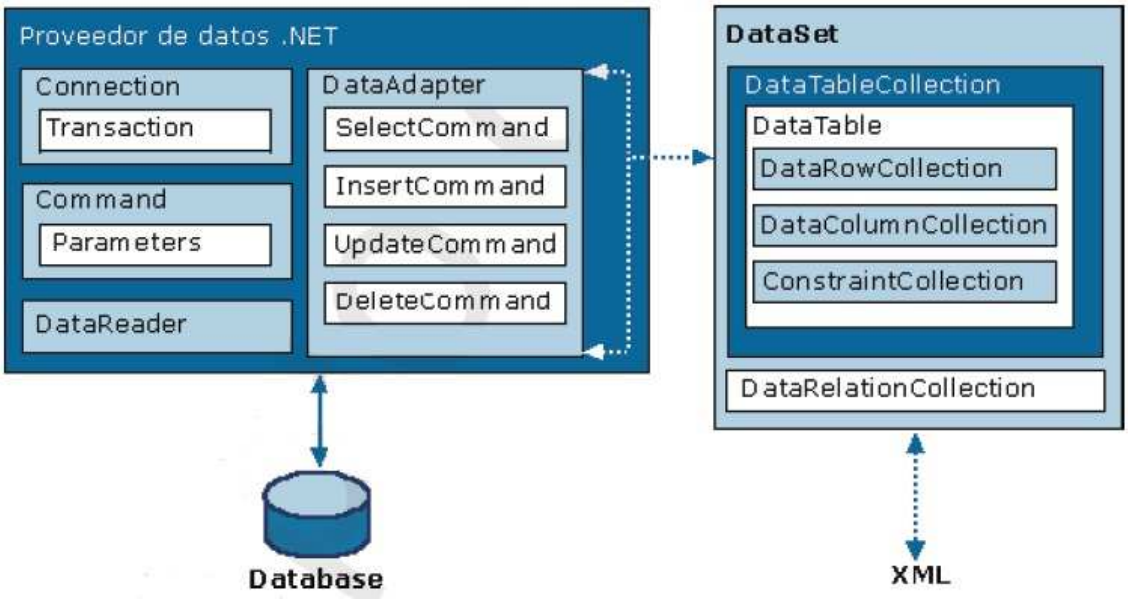
- **SQL Managed Provider**. Ofrece el conjunto de clases necesarias para comunicarse con los comandos de SQL Server en su versión 7.0 ó superior. Este espacio de nombres ha sido evolucionado para soportar las nuevas tecnologías de SQL Server 2005.
- **OleDB Managed Provider**. Ofrece el conjunto de clases necesarias para acceder fuentes de datos accesible a través de drivers OLEDB/ODBC. En las mejoras de ADO.NET 2.0 el haber ampliado el soporte que el CLR da a estos tipos de proveedores, como por ejemplo la integración mejorada de la seguridad, el pool de conexiones, etc.
- **ODBC Managed provider.** Conexiones con el modelo de drivers de la pasarela ODBC nativa de Windows. No se garantiza en todos los casos el mismo soporte por parte del CLR, pero se ha extendido la funcionalidad para soportar más y mejores características de los más potentes y recientes.
- **ORACLE Managed Provider**. Portado de la versión ADO.NET 1.1. Extendido para garantizar la compatibilidad con el cliente de Oracle.

3.3.9 - La arquitectura de los .NET Provider

Toda clase que sea del tipo .NET Provider, debe cumplir que en su espacio de nombres implemente las clases derivadas de:

- **Connection**. Será la clase encargada de encapsular toda la funcionalidad de conexión a las fuentes de datos.
- **Command**. Será la que encapsula el envío de comandos SQL al servidor.
- **DataReader**. Contiene todo el subsistema de lectura de los cursores resultados de una Query
- **DataAdapter**. Es el componente encargado de **adaptar** el conjunto de operaciones realizadas desde un origen genérico (por ejemplo, un dataSet) hacia un destino (un servidor de bases de datos relacionales concreto, por ejemplo SQL Server) y viceversa.

En el siguiente grafico podemos ver el modelo de datos de ADO .NET



Los .NET Providers tienen la capacidad de enviar/recibir y almacenar persistentemente de manera off-line todo el flujo de información en formato XML. Suelen implementar los métodos **ReadXML** y **WriteXML** para poder implementar esta funcionalidad.

Nombre del proveedor OLEDB	**Descripción del proveedor**
SQLOLEDB	Proveedor SQL Server OLEDB
MSDAORA	Proveedor ORACLE OLEDB
JOLT	JET OLEDB Provider (proveedor de bases de datos JET)
MSDASQL/SQLServer ODBC	Acceso vía Direct-ODBC a sistemas SQL Server (conexiones ODBC a través de drivers OLEDB)
MSDASQL/Jet ODBC	Acceso vía Direct-ODBC a sistemas JET (conexiones ODBC a través de drivers OLEDB)

En la actualidad el soporte de terceros a la plataforma .NET ha crecido hasta prácticamente ofrecer proveedores para todo sistema gestor que se puede encontrar en el mercado: DB2, MySQL, Informix, etc.

3.4 – OPEN DATA PROTOCOL (Odata)

OData permite a la creación de los servicios HTTP basados en datos, que pone a nuestra disposición recursos identificados usando los identificadores uniformes de recurso (URIs) y definidos en un modelo abstracto de los datos, pueden ser publicados y ser corregidos por los clientes a través de la Web con simples mensajes HTTP. Este protocolo se utiliza para exponer y tener acceso a la información de multitud de fuentes de datos incluyendo bases de datos, sistemas de ficheros o sitios web tradicionales.

La base de OData son los feeds, Son colecciones de entradas. Cada entrada representa una estructura con una clave que contiene una lista de las propiedades de los tipos primitivos o complejos que contiene. Las entradas pueden ser parte de una jerarquía y pueden relacionarse con otras entradas mediante enlaces.

Algunas entradas son especiales ya que describen un elemento multimedia (BLOB, binary large object), convirtiéndose en dos recursos relacionados: los medios ligan la entrada que contiene los datos estructurados que describen el blob y el recurso de los medios que es el blob en sí mismo.

Los servicios simples de OData pueden consistir en apenas una entrada pero los servicios más sofisticados pueden múltiples entradas, en este es útil exponer un documento de los servicios que enumere todas las entradas a nivel superior así los clientes pueden consultarlas y descubrir las direcciones de cada uno de ellas.

Además de las consultas y de las entradas, los servicios de OData pueden exponer sus operaciones, que son las funciones simples específicas del servicio que aceptan parámetros de entrada y retorna valores de vuelta de la entrada y tipos complejos o primitivos.

Para ayudar a clientes a descubrir de qué forma un servicio de OData tiene estructurados sus recursos, los enlaces entre los recursos, y las operaciones del servicio expuestas, un servicio de OData pueden también exponer un documento de los metadatos del servicio. Los documentos de los metadatos de OData describen el modelo de los datos de la entidad (EDM) para un servicio concreto, que es el modelo abstracto usado por los servicios de OData para formalizar la descripción de los recursos que expone.

Los servicios de OData pueden proporcionar dos tipos de documentos de metadatos para describirse. El primero expone un documento de los servicios que enumeran todas las entradas a nivel superior así los clientes pueden consultarlas y descubrir las direcciones de cada uno de ellas. El documento del servicio está típicamente disponible en el URI de la raíz del servicio y se puede ajustar a formato en ATOM o JSON. El otro documento contiene los metadatos del servicio que describen el modelo de los datos, es decir la estructura y organización de todos los recursos.

3.4.1 - Modelo abstracto de datos

Aunque usemos el EDM (Entity Data Model) como modelo para los datos vemos que en oData no es una implementación para la persistencia de los datos utilizado un servicio de OData, ya que el único requisito para ser un servicio de OData es que el interfaz de HTTP expuesto por el servicio sea igual a las especificaciones del protocolo para oData.

Un documento con los metadatos del servicio (Service Metadata Document) de OData, describe sus datos en términos de EDM usando lenguaje XML para describir modelos (Conceptual Schema Definition Languaje, CSDL).

3.4.2 - Entity Data Model (EDM)

El punto principal en el EDM son las entidades y asociaciones. Las entidades son instancias de los tipos de la entidad (por ejemplo, cliente, empleado, etc.) que son registros estructurados que consisten en un nombre, un tipo y una clave. Los tipos complejos se estructuran en una lista de propiedades pero sin clave, y solo pueden existir como propiedad de la entidad que la contiene o como un valor temporal. Una clave de la entidad se forma de un subconjunto de las propiedades del tipo de la entidad. La clave de la entidad (por ejemplo, PedidoId) es un concepto fundamental para identificar las instancias de los tipos de la entidad y permitir que las instancias de la entidad participen en las relaciones. Las entidades se agrupan en grupos de la entidad (por ejemplo, los clientes son un grupo de los tipos de la entidad del cliente).

Las asociaciones definen la relación entre dos o más tipos de la entidad (por ejemplo, departamento de TrabajaPara del empleado). Las instancias de las asociaciones se agrupan en grupos de asociación. Las propiedades de navegación son unas propiedades especiales de los tipos de la entidad los cuales están unidos a una asociación específica y se pueden utilizar para hacer referencia a las asociaciones de una entidad.

Todos los contenedores de instancias (grupo de entidad y grupos de asociación) se agrupan en un cotenedor de la entidad. En los términos de OData, las entradas expuestas por un servicio de OData son representadas por grupos de entidad (Entity Sets) o una propiedad de navegación en un tipo de la entidad que identifique una colección de entidades. Cada entrada de OData es descrita en el EDM por un tipo de la entidad y cada enlace entre las entradas esta descrito por una propiedad de navegación.

Los recursos de OData se describen a continuación:

Servicio OData	Descrito en EDM como:
Collection	Grupo de entidades. Propiedad de navegación como un tipo que define una colección de entidades.
Entry	Un tipo de la entidad. Un tipo de entidad puede formar parte de la jerarquía de un tipo.
Property of an entry	Propiedad de un tipo de entidad primitiva o compleja.
Link	Propiedad de navegación definida por el tipo de entidad
Service Operation	Función Importar

3.4.3 - Service Metadata Document

Este documento describe el modelo de los datos (es decir. estructura y organización de todos los recursos) expuestos en el servicio HTTP. Un documento de los metadatos del servicio describe sus datos en términos de EDM usando el lenguaje de XML para describir los modelos Conceptual Schema Definition Languaje (CSDL) como ya se comento anteriormente en el apartado modelo abstracto de datos. Cuando es un ofrece servicio de OData como un documento de metadatos, se empaqueta el documento de CSDL usando el formato EDMX (Entity Data Model for Data Service Packaging Format).

A continuación se muestran los metadatos de un servicio que describen tres tipos de la entidad (categorías, productos y surtidores), las relaciones entre ellos y uno "operación del servicio de ProductsByRating"

```
- <EntityContainer Name="DemoService" m:IsDefaultEntityContainer="true">
    <EntitySet Name="Products" EntityType="ODataDemo.Product" />
    <EntitySet Name="Categories" EntityType="ODataDemo.Category" />
    <EntitySet Name="Suppliers" EntityType="ODataDemo.Supplier" />
- <AssociationSet Name="Products_Category_Categories"
    Association="ODataDemo.Product_Category_Category_Products">
    <End Role="Product_Category" EntitySet="Products" />
    <End Role="Category_Products" EntitySet="Categories" />
</AssociationSet>
- <AssociationSet Name="Products_Supplier_Suppliers"
    Association="ODataDemo.Product_Supplier_Supplier_Products">
    <End Role="Product_Supplier" EntitySet="Products" />
```

```
        <End Role="Supplier_Products" EntitySet="Suppliers" />
</AssociationSet>
- <FunctionImport Name="GetProductsByRating" EntitySet="Products"
        ReturnType="Collection(ODataDemo.Product)" m:HttpMethod="GET">
          <Parameter Name="rating" Type="Edm.Int32" Mode="In" />
</FunctionImport>
 </EntityContainer>
```

3.4.4 - Formatos de presentación

OData soporta dos tipos de presentar recursos, el formato XML- basado del Atom y el formato de JSON. Según lo descrito en la especificación del HTTP [RFC2616], los clientes pueden indicar su preferencia en la representación del recurso incluyendo en el request header una aceptación “accept” de los tipos del MIME que puede manejar.

Un cliente que desea solamente respuestas de JSON pondría en el accept del header “application/json”.

```
GET /OData/OData.svc/Products HTTP/1.1
host: services.odata.org
accept: application/json
```

Para el formato Atom se implica más de un tipo MIME, pero cuando se direccionan links o propiedades de un elemento el recurso devuelto es XML. AtomPub también entrega documentos del servicio, con un tipo de contenido “application/atomsvc+xml”. Para poder elegir el tipo de recurso en el accept del header se tiene que indicar “application/atom+xml, application/atomsvc+xml, applicationo/xml”.

```
GET /OData/OData.svc/Products HTTP/1.1
host: services.odata.org
accept: application/atom+xml,application/atomsvc+xml,application/xml
```

3.4.5 - Operaciones

El interfaz del servicio de OData tiene un número fijo de las operaciones para acceder a todos sus recursos. Estas operaciones son GET, POST, PUT/MERGE y los métodos DELETE HTTP. Indicando cuando actúa cada uno de ellos usando un URI. Además de las operaciones del interfaz, OData permite que los servidores publicar las operaciones especiales (conocidas como operaciones de servicio) que se pueden invocar con GET o POST.

3.4.6 - Creando un servicio oData en ADO.NET

El primer paso para crear un servicio de datos con ADO.NET es determinar la fuente de datos que debe ser publicada como puntos de acceso, seleccionar la capa de acceso. Para los datos relacionados almacenados en el servidor de Microsoft SQL u otras bases de datos de terceros, los servicios de datos de ADO.NET permiten publicar fácilmente un modelo conceptual usando ADO.NET Entity Framework (EF). Para el resto de las fuentes de datos (documento de XML, Web service, capa lógica de la aplicación, etc.) o utilizar las tecnologías adicionales del acceso de base de datos (LINQ to SQL), un mecanismo que permite cualquier fuente de datos, según el modelo descrito, para ser publicado como servicio de ADO.NET.

Ahora veremos un ejemplo de creación de un servicio Odata para ASP.

El primer paso tenemos que crear un proyecto web con Visual Studio

A continuación creamos el modelo de datos esta operación la podemos realizar de tres formas diferentes

1- Usando ADO.Net Entity Model
2- Usando clases Linq to SQL
3- Usando un modelo de datos personalizado

Para este ejemplo optamos por usar ADO.Net Entity Data Model primera opción por la sencillez ya que en este ejemplo no se pretende hacer un tutorial de creación de servicios oData sino documentar la sencillez de su creación.

Con el Entity Data Model seleccionamos la base de datos que vamos a modelar y elegimos la tabla que utilizaremos, en nuestro caso clientes

Al utiliza el asistente de Entity Framework automáticamente nos crea la entidad

Creamos el servicio de datos WCF (WCF Data Service) simplemente agregándolo en Visual Studio, esto creara un fichero con extensión .svc que tendrá nuestro servicio de datos.

La única modificación que tendremos que realizar en el código es asignar nuestro modelo de datos al servicio.

```
Public Class WcfDataService
    ' TODO: reemplace [[class name]] con el nombre de la clase de datos
    Inherits DataService(Of ClienteDBModel.ClientesDBEntities)
```

Para probar si funciona nuestro servicio ponemos en marcha el proyecto y ponemos en el explorer: http://localhost:3356/Servicio_oData/WcfDataService.svc, y obtenemos un resultado en formato Atom.

3.5 – ENTITY FRAMEWORK

Entity Framework es un conjunto de tecnologías de ADO.NET que permiten el desarrollo de aplicaciones de software orientadas a datos. Los programadores de aplicaciones orientadas a datos se han enfrentado a la necesidad de lograr dos objetivos muy diferentes. Deben modelar las entidades, las relaciones y la lógica de los problemas empresariales que resuelven, y también deben trabajar con los motores de datos que se usan para almacenar y recuperar los datos. Los datos pueden abarcar varios sistemas de almacenamiento, cada uno con sus propios protocolos; incluso las aplicaciones que funcionan con un único sistema de almacenamiento deben equilibrar los requisitos del sistema de almacenamiento con respecto a los requisitos de escribir un código de aplicación eficaz y fácil de mantener.

Entity Framework permite a los programadores trabajar con datos en forma de objetos y propiedades específicos del dominio, por ejemplo, con clientes y direcciones, sin tener que pensar en las tablas de las bases de datos subyacentes y en las columnas en las que se almacenan estos datos. Para ello, se eleva el nivel de abstracción en la que los programadores pueden trabajar al tratar con datos y se reduce el código requerido para crear y mantener las aplicaciones orientadas a datos.

3.5.1 - Modelo de datos conceptual

La creación de un modelo relacional extendido, llamado Entity Data Model (EDM), que engloba las entidades y las relaciones como conceptos de primera clase, se maneja con un lenguaje de consultas para EDM, un motor de mapeado completo que traduce del nivel conceptual al lógico (relacional), y un conjunto de herramientas guiadas por modelos que ayudan a crear los transformadores entidad-objeto, objeto-xml y entidad-xml.

El primer paso es definir un modelo conceptual propio. EDM representa una expresión formal, de diseño y ejecución de un modelo. EDM nos permite describir el modelo en términos de las entidades y las relaciones. Podemos definir el modelo explícitamente de forma manual escribiendo el XML o a través de una herramienta gráfica de diseño.

Lo que hace ADO.NET Entity FrameWork es mapear los objetos de Negocio a tablas relacionales atacando así directamente a los datos entre un modelo entidad relación y un modelo orientado a objetos. ADO.NET Entity FrameWork almacena metadatos EF (CSDL,SSDL,MSL content) , en un archivo *.edmx estos están separados en los siguientes elementos XML :

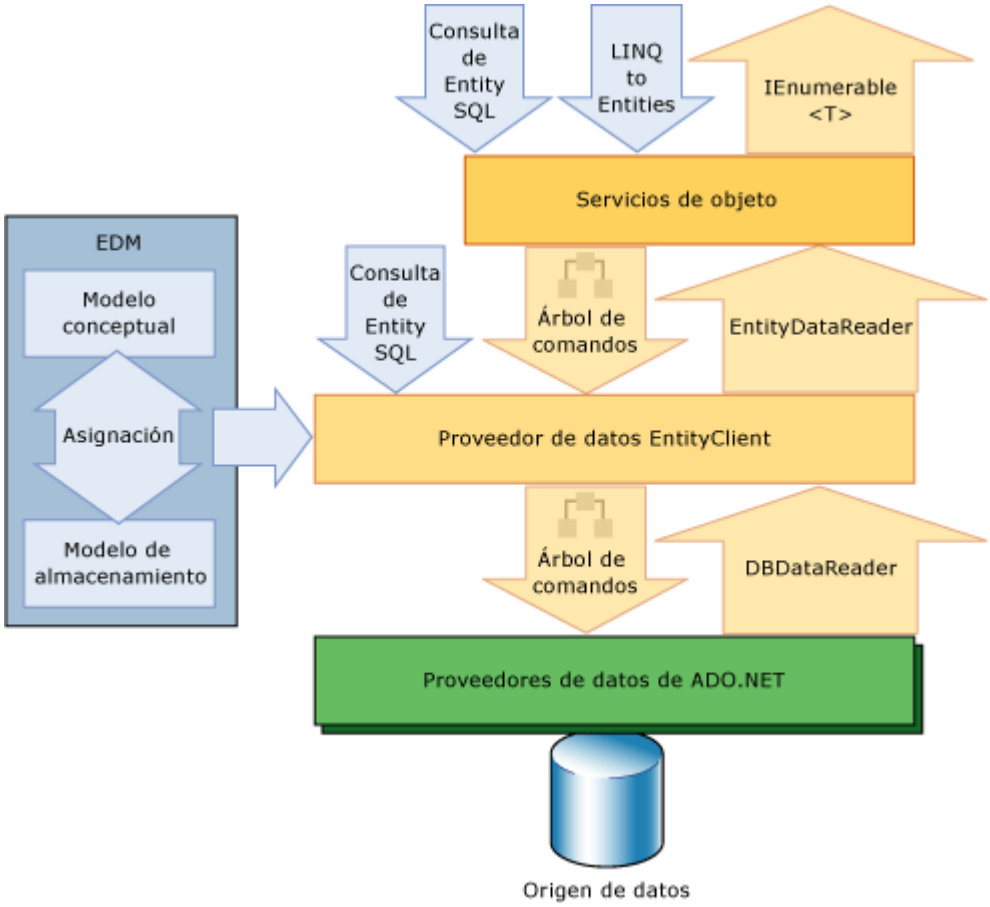
- Archivo de lenguaje de definición de esquemas conceptuales (.csdl): define el modelo conceptual.
- Archivo de lenguaje de definición de esquemas de almacenamiento (.ssdl): define el modelo de almacenamiento, que también se denomina modelo lógico.
- Archivo de lenguaje de especificación de asignaciones (.msl): define la asignación entre los modelos conceptuales y de almacenamiento.

Como algo más que otra solución de asignación objeto-relacional, Entity Framework trata fundamentalmente de permitir que las aplicaciones obtengan acceso y cambien los datos que están representados como entidades y relaciones en el modelo conceptual. Servicios de objeto usa el EDM para traducir las consultas de objeto con los tipos de entidad que se representan en el modelo conceptual en consultas específicas del origen de datos. Los resultados de la consulta se materializan en los objetos que Servicios de objeto administran. Entity Framework proporciona las maneras siguientes de consultar un EDM y devolver objetos:

- LINQ to Entities: proporciona compatibilidad con Language-Integrated Query (LINQ) para consultar los tipos de entidad que se definen en un modelo conceptual. Para obtener más información..
- Entity SQL: dialecto independiente del almacenamiento de SQL que funciona directamente con las entidades del modelo conceptual y que admite características del EDM como la herencia y las relaciones. Entity SQL se utiliza con las consultas de objeto y con las consultas que se ejecutan con el proveedor de EntityClient.
- Métodos del generador de consultas: permite construir consultas de Entity SQL utilizando los métodos de consulta del estilo de LINQ.

El Entity Framework incluye el proveedor de datos de EntityClient. Este proveedor administra las conexiones, traduce las consultas de entidad en consultas específicas del origen de datos y devuelve un lector de datos que Servicios de objeto usa para materializar los datos de la entidad en los objetos. Cuando no se requiere la materialización de los objetos, el proveedor de EntityClient también se puede utilizar como un proveedor de datos ADO.NET estándar habilitando las aplicaciones para ejecutar las consultas de Entity SQL y usar el lector de datos de solo lectura devuelto.

El diagrama siguiente muestra la arquitectura de Entity Framework para el acceso a datos:

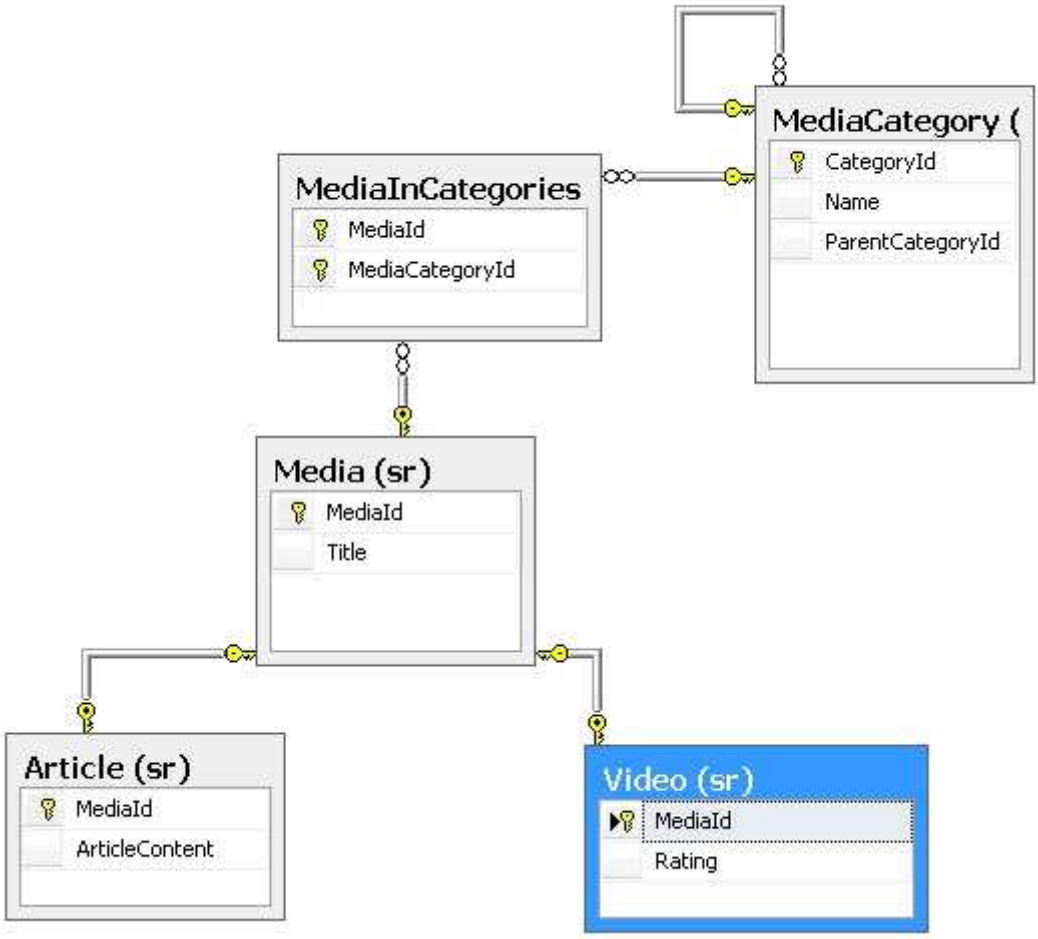


El Entity Framework genera una clase derivada de Object Context que representa el contenedor de entidades definido en el modelo conceptual. Este contexto del objeto proporciona los medios para realizar el seguimiento de los cambios y administrar las identidades, la simultaneidad y las relaciones. Esta clase también expone un método SaveChanges que escribe las inserciones, actualizaciones y eliminaciones en el origen de datos. Al igual que las consultas, estas modificaciones son realizadas bien por los comandos que el sistema genera automáticamente o bien por los procedimientos almacenados que el programador especifica.

3.5.2 - Modelando entidades

Tablas con autoreferencias

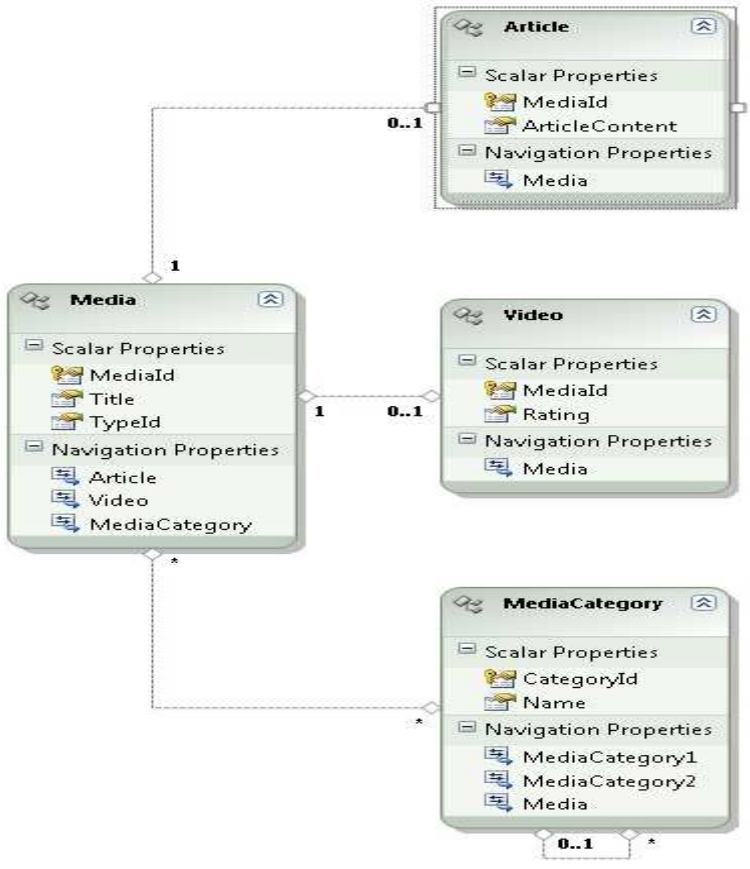
En la figura siguiente se muestra un diagrama de base de datos con diferentes tipos de artículos con diferentes categorías.



En la figura tenemos la tabla de MediaCategories que contiene categorías para diferentes tipos de artículos incluyendo Videos y el resto de artículos. Cada categoría puede tener subcategorías que sean identificadas por la columna de ParentCategoryId en la tabla de MediaCategory. Esto hace MediaCategory se referencie a si misma donde las categorías y las subcategorías se almacenan en la misma tabla y para descubrir la categoría padre de una categoría, nosotros tiene que mirar el valor de ParentCategoryId. Cuando el ParentCategoryId es nulo, estamos en la categoría raíz. La tabla Media contiene campos comunes para los artículos y los Videos. Los campos específicos al vídeo y a los artículos se almacenan cada uno en su tabla. Vamos a importar esta estructura usando la autoreferencia de entidad que permitiría que consiguiéramos las subcategorías para una categoría dada.

Cuando una tabla tiene una relación con si misma se importa en el EDM, Entity Framework crea automáticamente una asociación así misma a la entidad. Para importar la estructura anterior se utilice a diseñador del Entity Framework. El asistente creará una asociación a MediaCategory consigo misma. Además, una relación M a M entre MediaCategory y la tabla Media, el Entity Framework no hará caso de esta tabla. Puesto que la tabla Media contendrá dos tipos de Media (Article y Video), extenderá la clase article y video para heredar de entidad Media.

A continuación se muestra el modelo que ha creado el asistente a partir del modelo de datos.



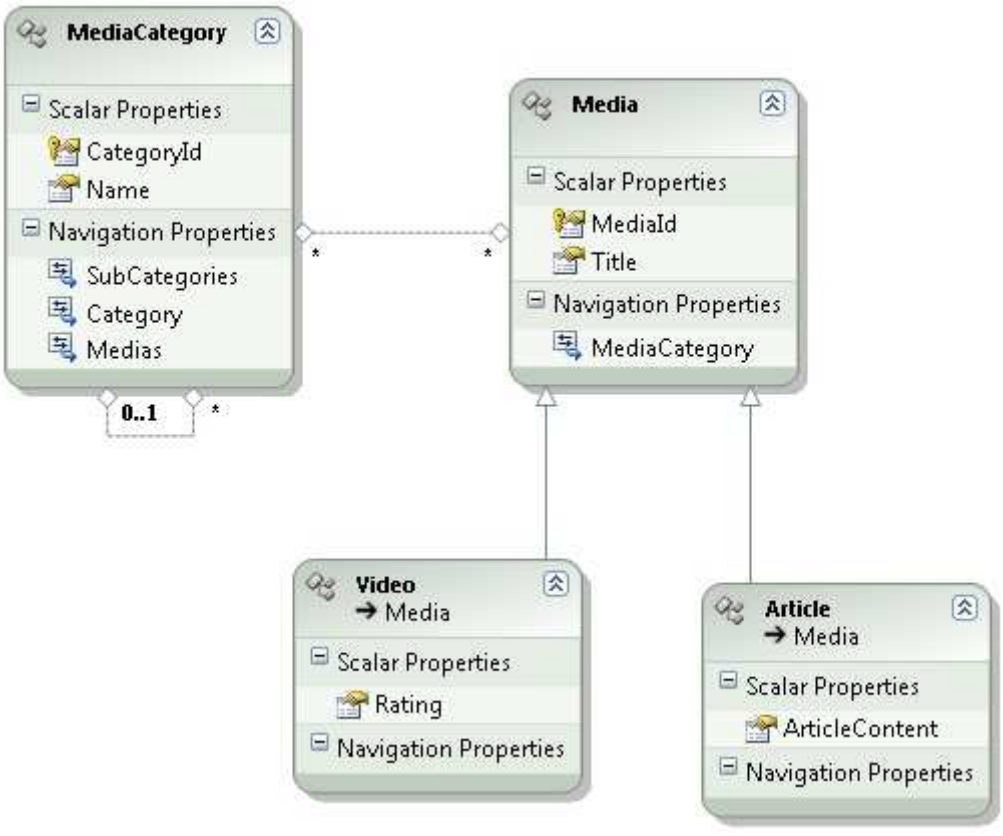
El diseñador creó una relación consigo mismo con la entidad Media y creó una relación M a M Media omitiendo la tabla de enlace MediaInCategory, los siguientes cambios se hacen para optimizar estas relaciones:

Cambiamos la propiedad de navegación MediaCategory1 a MediaSubCategories.

Quitamos la asociación de Article y video de Media.

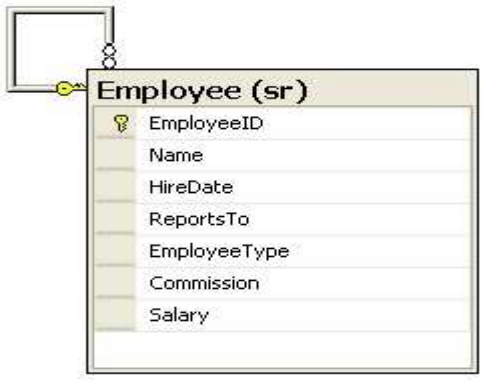
Hacemos que Article y Video heredan de Media, quitamos MediaId de Article porque utilizaremos MediaId heredado de Media, hacemos lo mismo con la entidad Video donde la columna de MediaId también hereda de Media.

El modelo terminado se muestra a continuación.



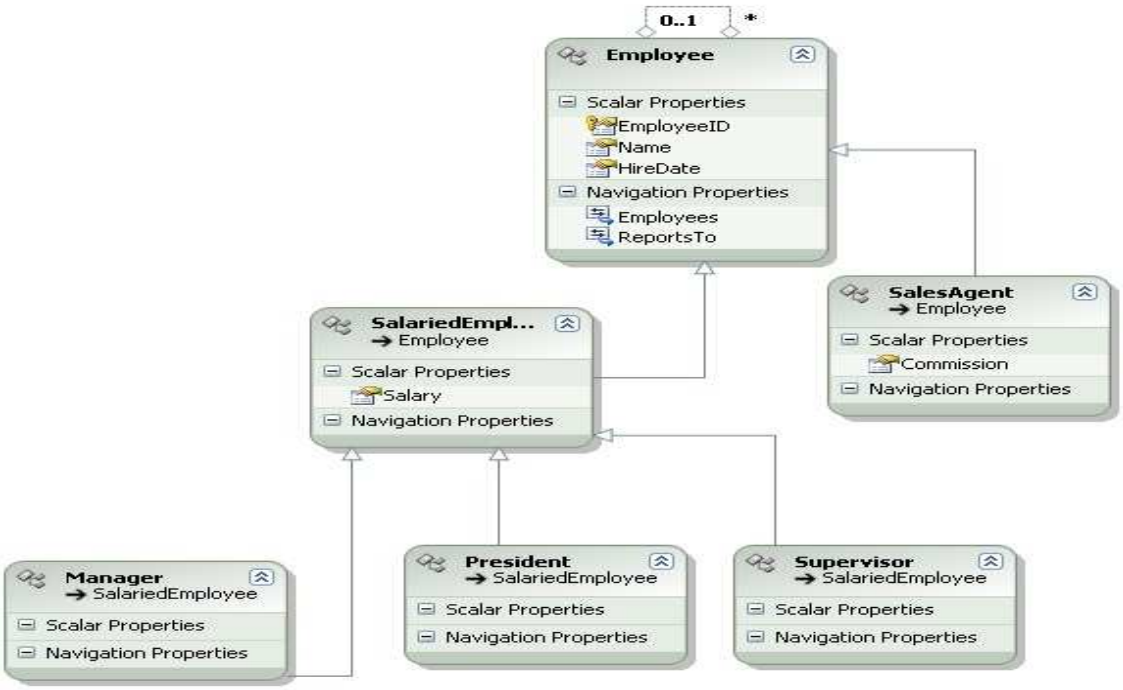
Entidad autoreferenciada con tablas con herencias

La figura que viene a continuación muestra un diagrama de base de datos para una tabla de empleados de diferentes categorías.

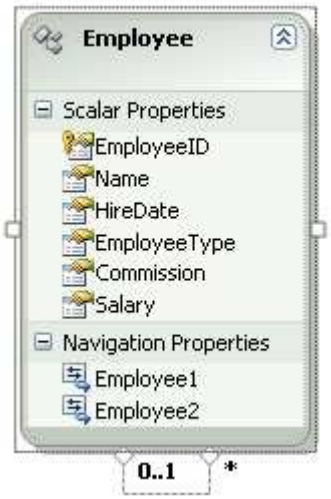


La tabla Employee contiene empleados de diversos tipos. Un empleado podría ser presidente, encargado, supervisor o un comercial identificado por la columna EmployeeType. Cada empleado está a cargo de otro empleado identificado por ReportTo que contiene un EmployeeId, por ejemplo un comercial notifica a un supervisor y un supervisor notifica al encargado y el encargado notificaría al presidente. Importamos el esquema de la tabla usando tablas por herencia y cada empleado debe tener una propiedad de navegación ReportsTo.

El modelo resultante es el siguiente.

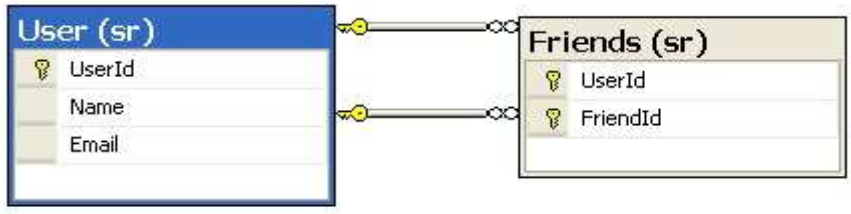


Para hacer la correcta importación de la tabla empleado como una entidad que es recursiva y por herencia el asistente de EDM nos genera la siguiente entidad.

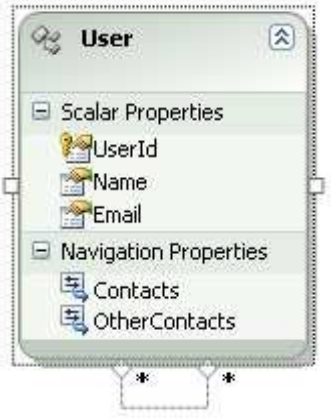


M a M asociación con entidades recursivas

A continuación se muestra el diagrama de base de datos que muestra un usuario que tiene muchos amigos y cada amigo puede conocer a muchos usuarios. Esta relación M a M en la tabla de usuario usando la tabla amigo como enlace.



Cuando importamos el modelo usando el asistente, Entity Framework crea una entidad User y también crea una relación M a M entre User y Friends.



Con el modelo conceptual generado, si lo que se pretende es encontrar todos los contactos de un usuario se necesita utilizar la propiedad de navegación Contacts. La propiedad de navegación Contacts retorna todos los contactos que tiene el usuario. Pero si el usuario es el contacto para algún otro este tercero también pertenecerá a los contactos del usuario. Para tener acceso a estos contactos, puedo utilizar la propiedad de navegación OtherContacts para encontrar a los usuarios que lo incluyeron en su contacto.

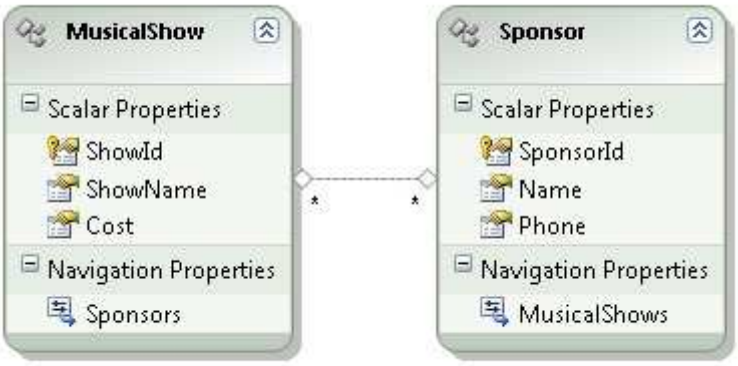
Relación M a M

Se ha definido 3 tablas en la base de datos llamadas MusicalShow, Sponsor y Show_Sponsor. Un Sponsor puede contribuir a muchos MusicalShow y una MusicalShow puede tener muchos Sponsors. Vamos a utilizar el Entity Framework para tratar la tabla M a M de la relación. También se necesita saber cómo crear o eliminar entidades cuando tienen una relación M a M.

Para montar la tabla enlazada de M a M la tabla que enlaza las dos entidades no puede tener campos utilices. Si existe esta información adicional en la tabla de enlace Entity Framework tiene que crear la entidad para que la relación pueda contener estos campos adicionales y poder actualizar esto datos. En nuestro ejemplo, la tabla de Show_Sponsor no contiene campos adicionales solo las claves primarias de Sponsor y de MusicalShow. Por lo tanto cuando importamos el modelo usando el asistente de la actualización, Entity framework que la tabla que hace de enlace no contiene ningún campo útil y la elimina automáticamente.

En la figura siguiente se muestra la relación entre las tablas.

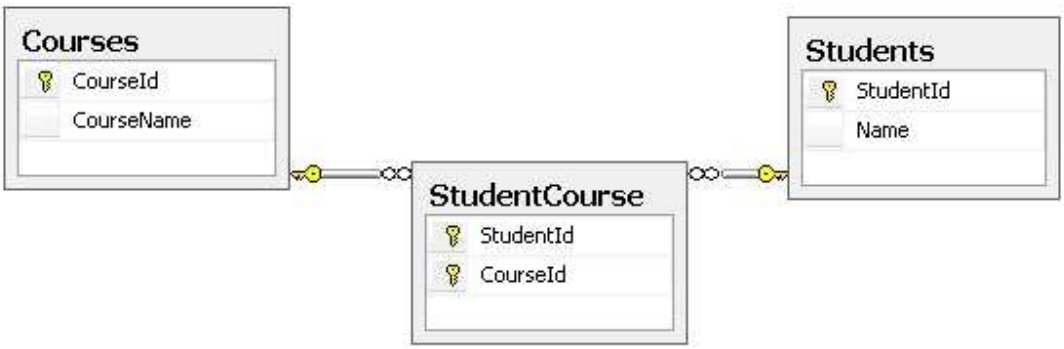
En el diagrama de la base de datos, Show_Sponsor actúa como tabla de enlace entre MusicalShow y Sponsor, esto se hace para comprobar que un MusicalShow no pueda tener el mismo Sponsor repetido dos veces definiéndose ShowId y SponsorId para ser la clave primaria de la tabla de enlace. Después de el modelado que hace Entity Framework elimina esta tabla y deja MusicalShow y Sponsor con una relación M a M.



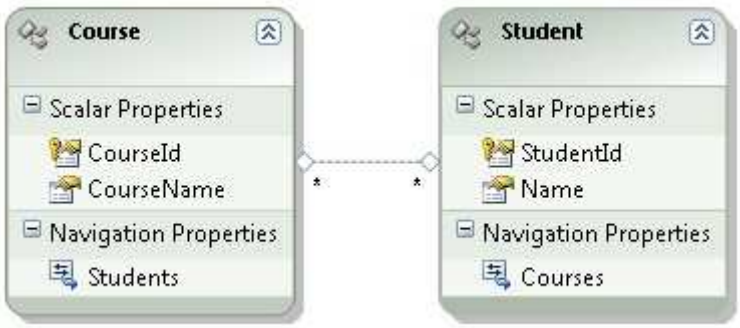
Relación 1 a M a 1

Se ha definido una tabla de enlace StudentCourse entre la tabla Student y Course en la base de datos. Un estudiante puede estar en muchos cursos y un curso puede tener muchos estudiantes. La tabla de la relación no tiene ningún campo adicional pero se tiene previsto una ampliación con lo cual si que contendrá campos adicionales y no se quiere cambiar la estructura cuando esto ocurra. Cuando se importa la tabla en EDMX, la entidad Course y Student tiene una relación M a M. Necesitamos cambiar la relación a 1 a M a 1 para poder ampliar la tabla StudentCourse en el futuro

Por defecto cuando se dos tablas unidas por una tabla del enlace sin campos adicionales Entity Framework la elimina. Si se desea mantener la tabla de enlace se puede corregir el modelo generado por el diseñador e introducir la tabla de enlace.



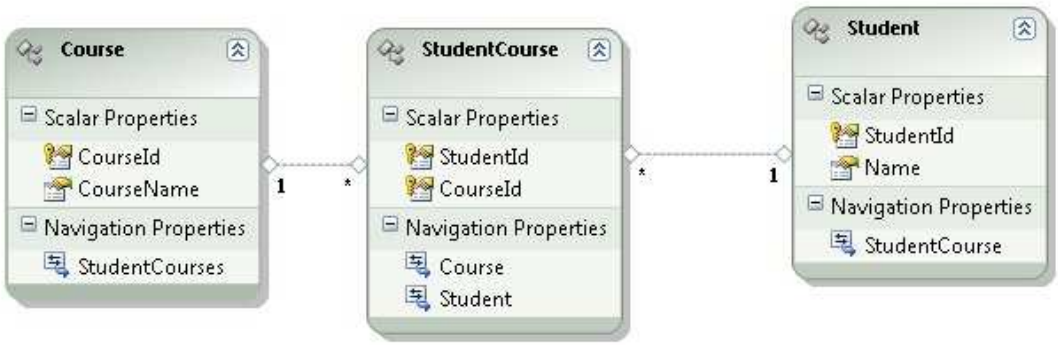
Después de crear las tablas podemos importar las tablas en EDM. Cuando importamos el modelo, Entity Framework elimina la tabla StudentCourse y crea una asociación M a M como se muestra en la figura siguiente.



Para cambiar el modelo, primero suprimiremos la relación M a M y crearemos una nueva entidad StudentCourse con dos propiedades CourseId y StudentId definidas como claves primarias.

A continuación se muestra como seria esta nueva entidad.

El siguiente paso es crear asociaciones entre el Course y StudentCourse, después entre Student y StudentCourse, finalmente el modelo queda.

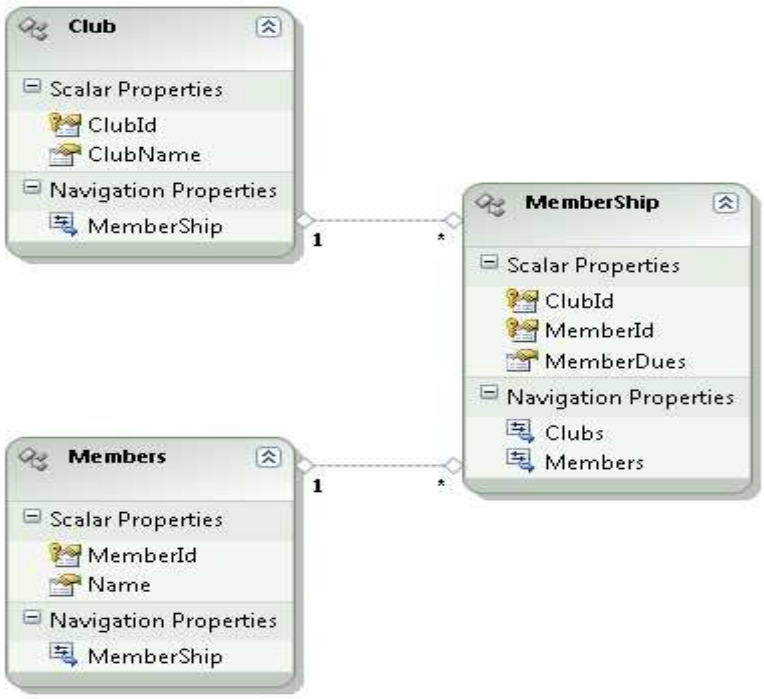


Modelar una relación 1 a M a 1 como M a M

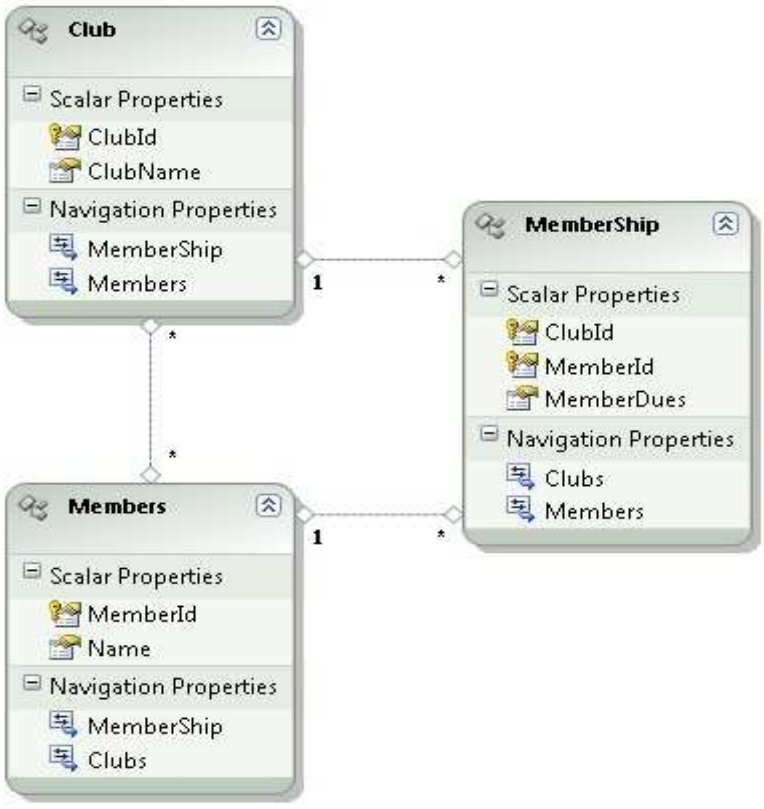
Se definen 3 tablas en la base de datos, Clubs, MemberShip y Members. Un club puede tener muchos miembros y un miembro puede pertenecer a muchos clubs. Esta relación se almacena en tabla MemberShip. La tabla MemberShip lleva una columna MemberDues que almacena las deudas de un miembro con el club. Como la tabla MemberShip contiene columnas adicionales aparte de las claves primarias de la tabla del Clubs y de members, Entity Framework modela la relación como 1 a M a. Lo que se pretende es que el modelo tuviera una relación M a M.

Cuando importamos la tabla Clubs y Membres, Entitty Framework no puede eliminar la tabla MemberShip porque tiene columnas adicionales. Para conseguir nuestro propósito podemos crear una asociación adicional que tenga una relación M a M entre Clubs y Members. Para lograrlo tenemos que modificar manualmente el archivo de SSDL ya que esto no está contemplado en el diseñador. En el modelo del ssdl, tenemos que crear un EntityType que contenga solamente las claves primarias de las entidades Clubs y Members. Además tenemos que definir un EntitySet que utilice un DefiningQuery. Después de crear EntityType y EntitySet, podemos entrar en el diseñador y crear una sociación M a M muchos a muchos asociación entre los Clubs y Members

Después de crear las tablas, importaremos el modelo en EDM usando el modelo de la base de datos. A continuación se muestra el modelo de los datos que ha creado Entity Framework cuando importa las tablas.

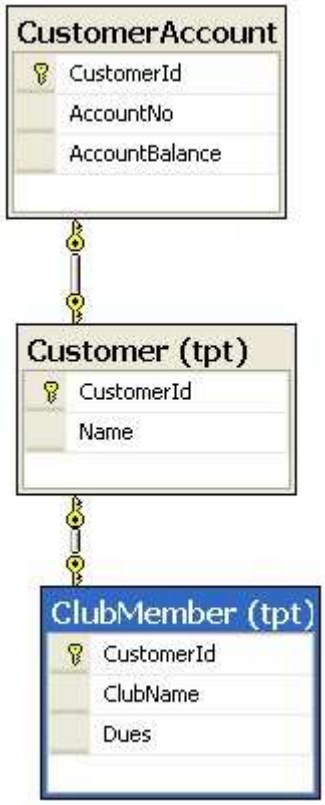


El siguiente paso será crear la relación entre Club y Members.



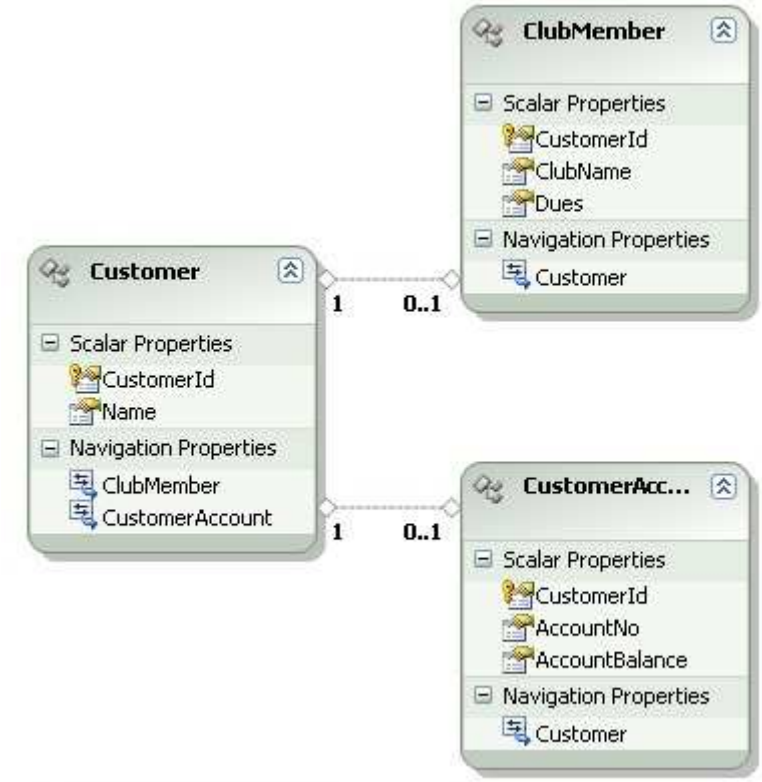
Dividiendo entidades

A continuación se muestra el diagrama de base de datos para los Cliente y las tablas relacionadas con información adicional.

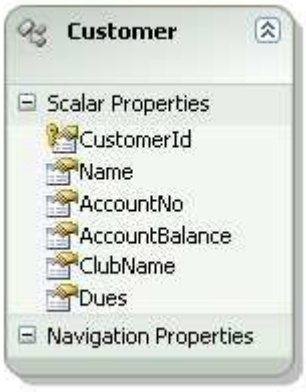


En esta relación la tabla Customer tiene datos adiciones en la tabla CustomerAccount que contiene datos de la cuenta del Cliente. Además si el cliente es miembro de un club en la tabla ClubMenber guardamos el nombre del club y las deudas que pueda tener el Cliente con el Club. Como la interfaz de Usuario requiere que todos estos campos estén siempre disponibles, se pretende mostrar las tres tablas como una sola entidad.

Para implementar esta estructura en una sola entidad Poner en práctica la susodicha mesa estructuran como una entidad sola, Todas las tablas que participan en la división de entidad deben compartir una clave común primaria, esto quiere decir que una tabla generaría la clave primaria y otras tablas la usarían como su clave primaria, a continuación podemos ver el modelo transformado por el Entity Framework a partir del modelado de la base de datos anterior.



Como lo que necesitamos es tener solamente una sola entidad que englobe a las 3 tablas, se mueven ClubName, Dues, AccountNo y AccountBalance a la entidad Customer y eliminamos la entidad ClubMember y a la entidad de CustomerAccount creadas por el diseñador. El siguiente grafico muestra el modelo actualizado.



3.6 – LINQ TO SQL

Es una implementación de O/RM(object relational mapping, mapeador de objetos relacionales) que viene con .NET Framework, y nos permite modelar bases de datos relacionales con clases de .NET. Podemos consultar bases de datos con LINQ, así como actualizar/añadir/borrar datos de ellas.

Tradicionalmente, las consultas con datos se expresan como cadenas sencillas, sin comprobación de tipos en tiempo de compilación ni compatibilidad con IntelliSense. Además, es necesario aprender un lenguaje de consultas diferente para cada tipo de origen de datos: bases de datos SQL, documentos XML, servicios Web diversos, etc. LINQ convierte una *consulta* en una construcción de lenguaje de primera clase en C# y Visual Basic. Las consultas se escriben para colecciones de objetos, utilizando palabras clave del lenguaje y operadores con los que se está familiarizado.

LINQ permite consultar datos desde una base de datos de SQL Server, XML, matrices y colecciones en memoria, conjuntos de datos ADO.NET o cualquier otro origen de datos remoto o local que admita LINQ. Los objetos con los que hacemos la consulta admiten IntelliSense; por tanto, se puede escribir código más rápidamente y detectar los errores de las consultas en tiempo de compilación en vez de en tiempo de ejecución. Las consultas LINQ se pueden usar como el origen de consultas adicionales para refinar los resultados. También se pueden enlazar a los controles para que los usuarios puedan ver y modificar con facilidad los resultados de la consulta.

3.6.1 - Proveedores Link

Un *proveedor LINQ* asigna las consultas LINQ al origen de datos que se consulta. Al escribir una consulta LINQ, el proveedor toma la consulta y la traduce a los comandos que podrá ejecutar el origen de datos. Además, convierte los datos del origen en los objetos que constituyen el resultado de la consulta. Finalmente, convierte los objetos en datos cuando envíe actualizaciones al origen de datos.

Por ejemplo Visual Basic incluye los siguientes proveedores LINQ.

Proveedor	**Descripción**
LINQ to Objects	El proveedor LINQ to Objects permite consultar colecciones y matrices en memoria. Si un objeto admite las interfaces IEnumerable o IEnumerable(OF T), el proveedor LINQ to Objects permite consultarlo.
LINQ to SQL	El proveedor LINQ to SQL permite consultar y modificar los datos de una base de datos de SQL Server. De esta forma, es fácil asignar el modelo de objetos de una aplicación a las tablas y los objetos de una base de datos.

LINQ to XML	El proveedor LINQ to XML permite consultar y modificar XML. Puede modificar XML en memoria o puede cargarlo desde un archivo y guardarlo en él.
LINQ to DataSet	El proveedor LINQ to DataSet permite consultar y actualizar los datos de un conjunto de datos ADO.NET. Puede agregar la eficacia de LINQ a las aplicaciones que usen conjuntos de datos para simplificar y ampliar las funcionalidades de consulta, agregación y actualización de los datos del conjunto.

3.6.2 - Conexión a una base de datos con LINQ

Cuando se dispone de una conexión válida a una base de datos de SQL Server, se puede agregar una plantilla de elementos Clases de LINQ to SQL al proyecto. De esta forma, se mostrará el Object Relational Designer. Este diseñador permite arrastrar los elementos a los que desee obtener acceso en el código desde el Explorador de servidores o el Explorador de base de datos hasta la superficie del diseñador. El archivo LINQ to SQL agrega un objeto Data Context al proyecto. Este objeto incluye propiedades y colecciones para las tablas y vistas a las que desee tener acceso, así como métodos para los procedimientos almacenados a los que desee llamar. Después de haber guardado los cambios en el archivo LINQ to SQL (.dbml), puede tener acceso a estos objetos en el código haciendo referencia al objeto DataContext que define el Object Relational Designer. Al objeto DataContext del proyecto se le asigna un nombre basándose en el nombre del archivo LINQ to SQL. Por ejemplo, un archivo LINQ to SQL que se denomina MiPrueba.dbml creará un objeto DataContext denominado MiPruebaDataContext.

3.6.3 - Ejecución de una consulta aplazada o inmediata

La ejecución de una consulta no tiene nada que ver con su creación. Después de crear una consulta, un mecanismo independiente desencadena su ejecución. Se puede ejecutar un consulta en cuanto esté definida (*ejecución inmediata*), o se puede guardar la definición y ejecutar la consulta más tarde (*ejecución aplazada*).

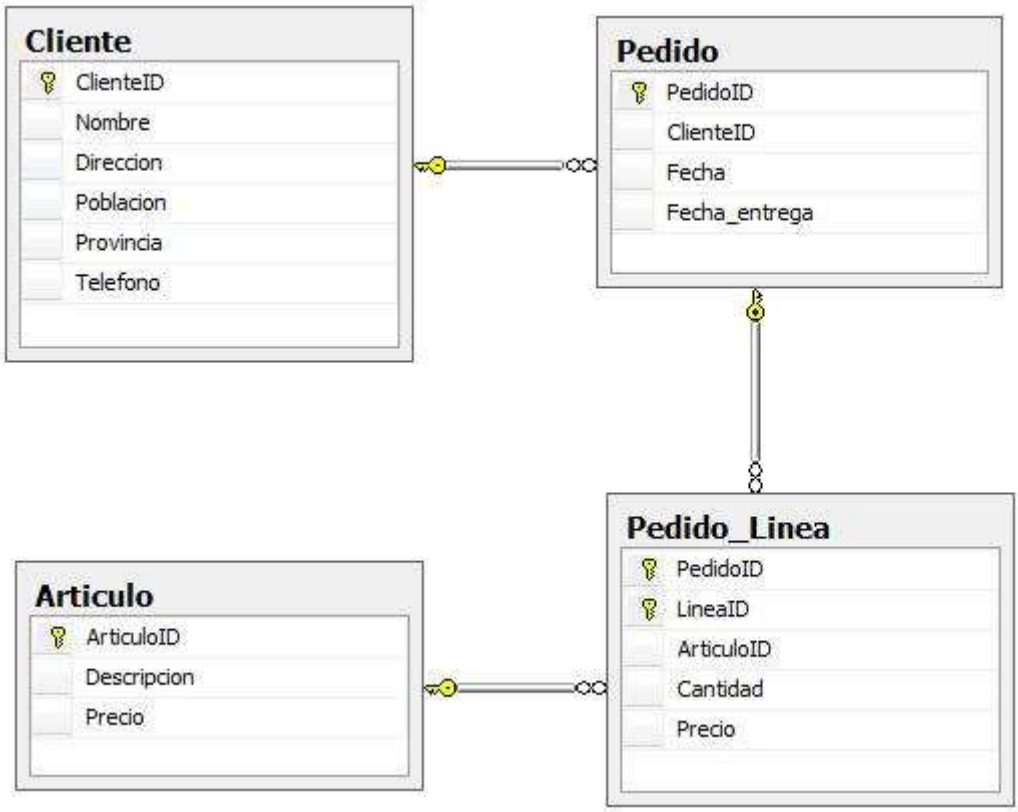
De forma predeterminada, al crear una consulta, no se ejecuta inmediatamente. En su lugar, se almacena la definición de la consulta en la variable que se usa para hacer referencia a su resultado. Más tarde cuando se tiene acceso a la consulta en código, como en un bucle For…Next, se ejecuta. Este proceso se denomina *ejecución aplazada*.

Las consultas también se pueden ejecutar cuando están definidas, a lo que se denomina *ejecución inmediata*. Puede desencadenar una ejecución inmediata aplicando un método que requiere acceso a elementos individuales del resultado de la consulta. Este puede ser el resultado de incluir una función de agregado, como Count, Sum, Average, Min o Max. Al usar los métodos ToList o ToArray, también se forzará la ejecución inmediata. Esto puede ser útil si desea ejecutar la consulta inmediatamente y almacenar en caché los resultados.

3.6.4 - Ejemplo de uso

Ahora veremos un pequeño ejemplo de uso de esta tecnología aunque el propósito de este trabajo no es hacer un tutorial de funcionamiento si no una evaluación de esta tecnología de manejo de datos para evaluar la utilidad del mismo.

Para hacer esto cogeremos una base de datos de ejemplo, en nuestro caso hemos creado una base de datos llamada Pfc que contiene cuatro tablas.



El siguiente código usa una consulta LINQ para obtener una secuencia IEnumerable de objetos Artículo.

```
Dim db As New PfcDataContext

Dim Productos = From p in db.Articulo Where ArticuloID = 23 Select p
```

El código siguiente muestra cómo obtener un producto de la base de datos, actualizar su precio, y guardar los cambios en la base de datos:

```
Dim db As New PfcDataContext

Dim Producto = (From p in db.Articulo Where ArticuloID = 23 Select p) .Single

Producto.Precio = 14,95

Db.SumitChanges()
```

El código siguiente muestra cómo borrar el artículo 23 de la base de datos.

```
Dim db As New PfcDataContext

Dim Producto = From p in db.Articulo Where ArticuloID = 23 Select p

db.Articulo.RemoveAll(Producto)

Db.SumitChanges()
```

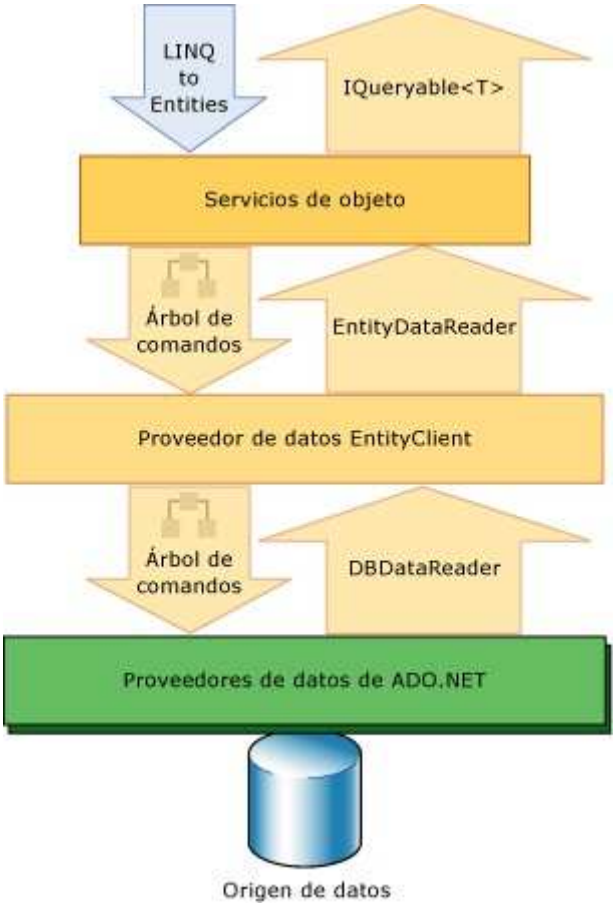
Como hemos podido ver en estos 3 sencillos ejemplos estamos aplicando toda la potencia de SQL a objetos de la aplicación lo que nos permite una gran potencia para la manipulación de datos.

3.7 – LINQ TO ENTITIES

Entity Framework permite a los programadores trabajar con datos en forma de objetos y propiedades específicos del dominio, por ejemplo, con clientes y direcciones, sin tener que pensar en las tablas de las bases de datos subyacentes y en las columnas en las que se almacenan estos datos. LINQ permite a los programadores formular consultas basadas en conjuntos en el código de las aplicaciones, sin tener que usar un lenguaje de consulta independiente. A través de la infraestructura de Servicios de objeto de Entity Framework, ADO.NET expone una vista conceptual común de los datos, incluidos los datos relacionales, como objetos del entorno .NET. Esto hace que la capa de objetos sea un objetivo ideal para la compatibilidad con LINQ. Esta tecnología LINQ, LINQ to Entities, permite a los programadores crear consultas flexibles, pero con establecimiento inflexible de tipos, en el contexto del objeto de Entity Framework mediante el uso de expresiones y de operadores de consulta estándar de LINQ directamente desde el entorno de desarrollo. Las consultas se expresan en el propio lenguaje de programación y no como literales de consulta incrustados en el código de programación, como suele ser el caso en las aplicaciones escritas en Microsoft .NET Framework versión 2.0. El compilador detectará los errores de sintaxis y los errores en los nombres de miembros y los tipos de datos, y los notificará en tiempo de compilación, con lo que se reduce la posibilidad de que se produzcan problemas con los tipos entre Entity Data Model y la aplicación.

Las consultas de LINQ to Entities usan la infraestructura de Servicios de objeto. La clase ObjectContex es la clase primaria para interactuar con un Entity Data Model como objetos de CLR. El programador crea una instancia de ObjectQuery genérica a través del ObjectContext. La clase de ObjectQuery genérica representa una consulta que devuelve una instancia o colección de entidades con tipo. Los objetos entidad devueltos se pueden actualizar y se encuentran en el contexto del objeto. Esto también es así en el caso de los objetos entidad que se devuelven en forma de miembros de tipos anónimos.

Cuando una aplicación usa el modelo EDM, la asignación entre el modelo de datos conceptual y el origen de datos subyacente se controla de forma automática. Un programador puede crear una aplicación de LINQ to Entities sin conocer nada del origen de datos subyacente ni de los métodos concretos que se usan para consultar el origen de datos. Esto también permite cambiar el origen de datos back end sin necesidad de realizar cambios en la aplicación cliente, ya que la mayor parte de las características específicas de las bases de datos se controlan a través de Servicios de objeto.



3.7.1 - Consultas con Linq To Entities

Una operación de consulta de LINQ consta de tres acciones: obtener el origen o los orígenes de datos, crear la consulta y ejecutar la consulta.

Los orígenes de datos que implementan las interfaces genéricas IEnumerable o IQueryable pueden consultarse a través de LINQ. Las instancias de la clase ObjectQuery, que implementa la interfaz genérica IQueryable, actúan como origen de datos para las consultas LINQ to Entities. La clase ObjectQuery genérica representa una consulta que devuelve una instancia o colección de entidades con tipo. El programador crea una instancia de ObjectContext, que es la clase principal para interactuar con un Entity Data Model (EDM) como objetos de CLR.

En la consulta se especifica exactamente la información que se desea recuperar del origen de datos. Una consulta también puede especificar cómo se debe ordenar, agrupar y conformar esa información antes de que se devuelva. En LINQ, una consulta se almacena en una variable. Si la consulta devuelve una secuencia de valores, la propia variable de la consulta debe ser de un tipo que se pueda consultar. Esta variable de consulta no realiza ninguna acción y no devuelve datos; solamente almacena la información de la consulta. Tras crear una consulta debe ejecutarla para recuperar los datos.

En una consulta que devuelve una secuencia de valores, la variable de consulta por sí misma nunca conserva los resultados de la consulta y sólo almacena sus comandos. La ejecución de la consulta se aplaza hasta que la variable de consulta se recorre en iteración en un bucle For Each. Esto se denomina *ejecución aplazada*; es decir, la ejecución de la consulta se produce después de que se cree la consulta. Esto significa que se puede ejecutar una consulta con la frecuencia que se desee. Esto es útil cuando, por ejemplo, se tiene una base de datos que otras aplicaciones están actualizando. En su aplicación puede crear una consulta para recuperar la información más reciente y ejecutar de forma repetida la consulta, devolviendo cada vez la información actualizada.

A diferencia de las consultas aplazadas, que devuelven una secuencia de valores, las consultas que devuelven un valor singleton se ejecutan inmediatamente. Algunos ejemplos de consultas singleton son Count, Max, Average y First. Se ejecutan inmediatamente porque se necesitan los resultados de la consulta para calcular el resultado singleton. También puede usar los métodos ToList o ToArray en una consulta para forzar la ejecución inmediata de una consulta que no crea un valor singleton. Esas técnicas para forzar la ejecución inmediata pueden ser útiles si desea almacenar en memoria caché los resultados de una consulta.

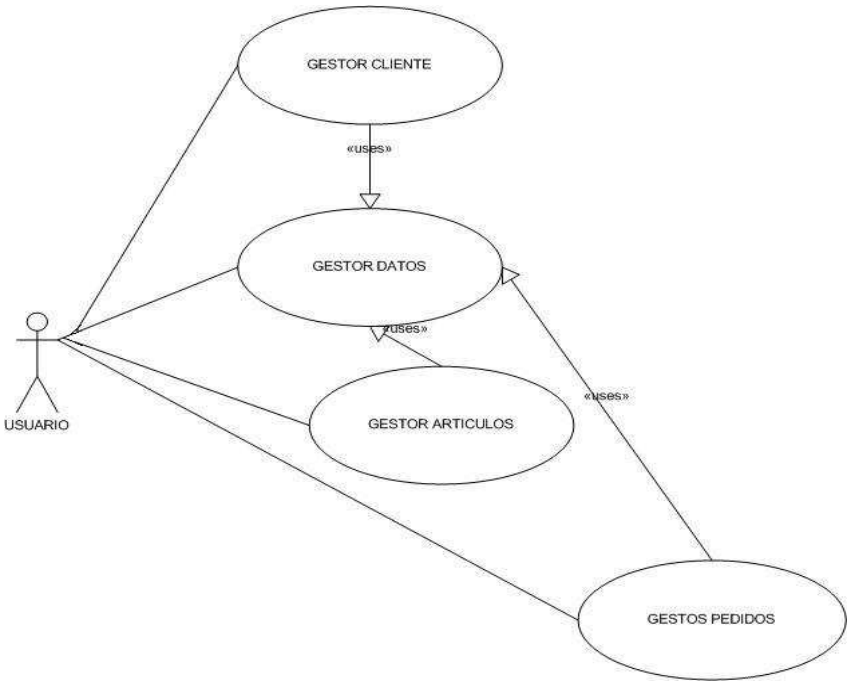
3.7.2 - Sintaxis de expresiones de consulta

Las expresiones de consulta son una sintaxis de consulta declarativa. Esta sintaxis permite a un programador escribir consultas en un lenguaje de alto nivel que tenga un formato similar al de Transact-SQL. Si se utiliza la sintaxis de expresiones de consulta, se pueden realizar incluso operaciones complejas de filtrado, ordenación y agrupamiento en orígenes de datos con código mínimo. En el ejemplo siguiente se usa Select para devolver todas las filas de Articulo y mostrar los nombres de producto utilizando el modelo anterior.

```
Using AWEntities  As New PfcEntities

   Dim productos As ObjectQuery(Of Articulo) = AWEntities.Articulo

   Dim NombreProductos = From p In productos Select p.Descripcion

   Console.WriteLine("Descripción de Articulos")

   For Each Descripcion In NombreProductos

      Console.WriteLine(Descripcion)

   Next

End Using
```

3.9 – DISEÑO DE LA APLICACION

3.9.1 - Modelo de casos de uso



En este caso no se han representado la funcionalidad de la aplicación porque lo que cambia es el método de acceso a datos respetándose el mismo modelo para todos los casos.

Hay una relación de include entre los casos de uso gestor cliente, gestor artículos y gestor pedidos con gestor datos que todos formaran parte de este.

Gestor cliente

Resumen de funcionalidad: El gestor cliente es el conjunto clases que me permite crear, borrar o modificar los datos del cliente en la base de datos

Eventos principales:

1. Si se necesita gestionar los datos de un cliente, el gestor nos muestra el formulario de clientes.
2. En este formulario podemos seleccionar el proceso que queremos lanzar, estos podrían ser crear, borrar o modificar un registro de cliente.

Gestor articulo

Resumen de funcionalidad: El gestor artículo es el conjunto clases que me permite crear, borrar o modificar los datos del artículo en la base de datos

Eventos principales:

1. Si se necesita gestionar los datos de un artículo, el gestor nos muestra el formulario de artículos.
2. En este formulario podemos seleccionar el proceso que queremos lanzar, estos podrían ser crear, borrar o modificar un registro de articulo.

Gestor Pedido

Resumen de funcionalidad: El gestor pedido es el conjunto de clases que permite la creación de un pedido del cliente, este gestor maneja las tablas de pedido y líneas de pedido, además de controlar los errores que retorna el gestor de bases de datos porque no respetamos la integridad de la base de datos.

Eventos principales:

1. Si se necesita introducir un pedido el gestor nos muestra el formulario de entrada que nos permitirá crear, borrar o modificar el pedido.
2. El usuario introduce un pedido, el gestor verifica que no exista en cuyo caso lo edita para ser modificado o borrado, al introducir el código de cliente se verifica que exista y nos permite introducir líneas, al introducir líneas también se verifica que exista el articulo.

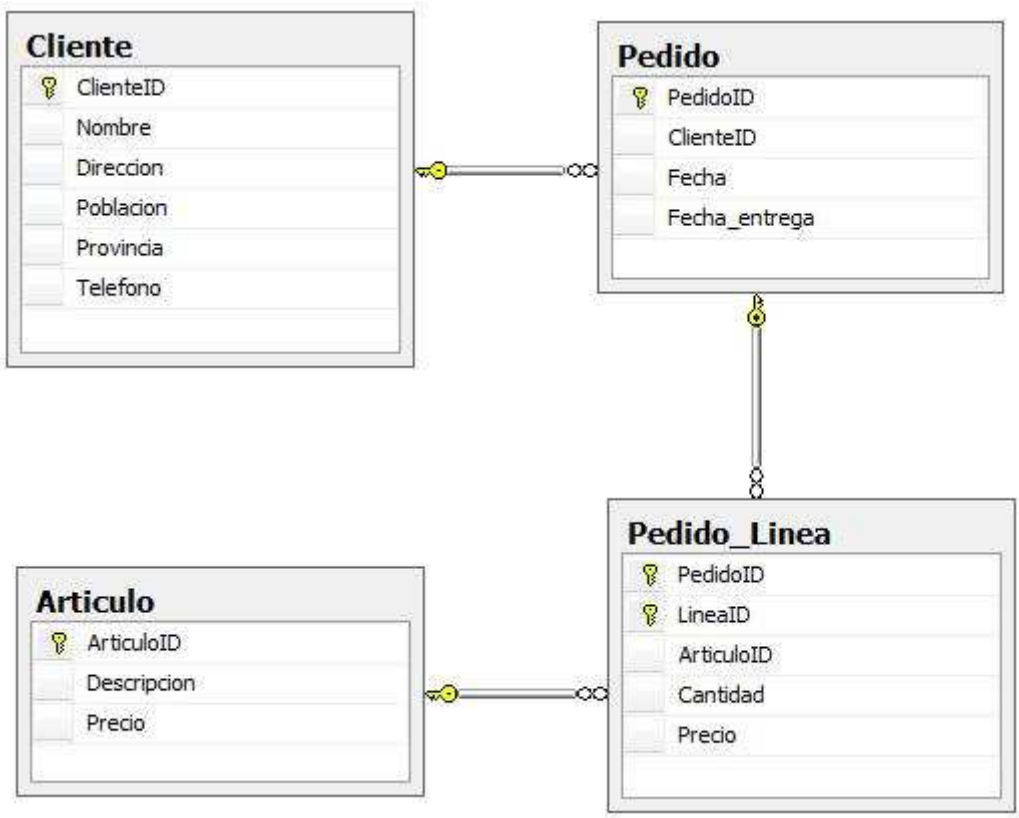
3.9.2 - Modelo Conceptual

La siguiente figura muestra el modelo conceptual del diseño de la base de datos de la aplicación, se han diseñado tres Foreing Keys para mantener la integridad de la base de datos, claves foráneas:

Pedido y Pedido_Linea: Esta clave controla que no se puedan introducir líneas de pedido sin la correspondiente cabecera.

Pedido y Cliente: Esta clave controla que no se puedan introducir ningún pedido si no existe el cliente.

Pedido_Linea y Articulo: Esta clave controla que no se puedan introducir líneas de pedido sin la correspondiente cabecera.



3.9.3 - Diagrama de clases

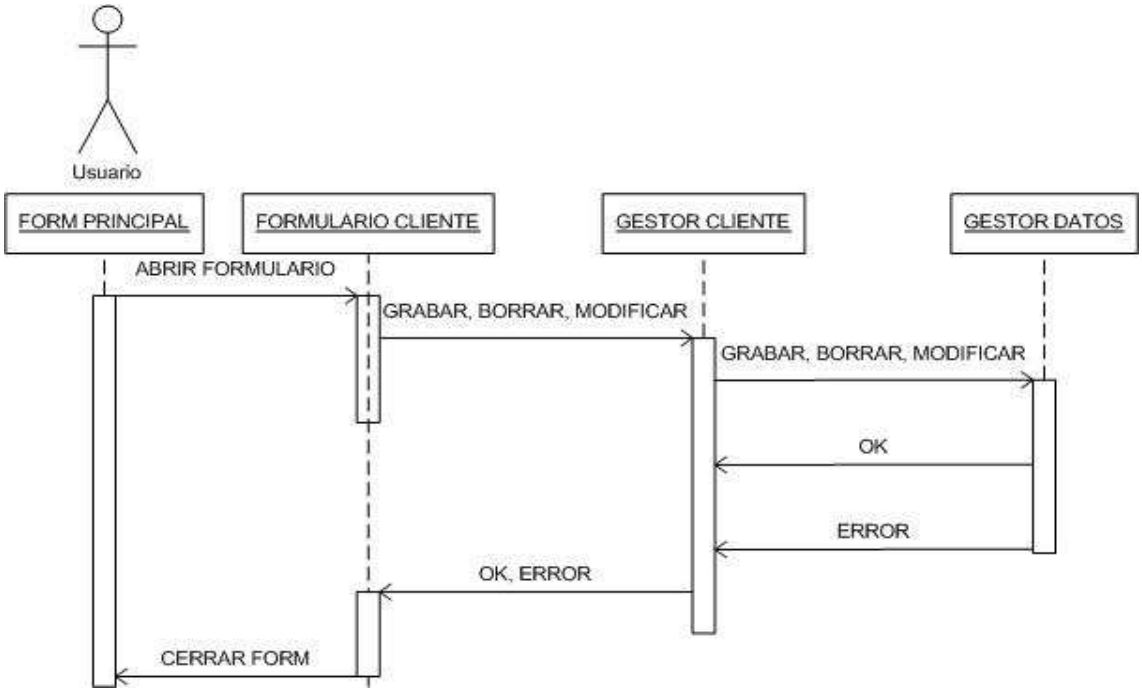
La siguiente figura muestra el diagrama de clases de la aplicación con las relaciones entre clases.

También representa el modelo conceptual que se utilizara para hacer las pruebas con Entity Framewok.

3.9.4 - Diagramas de secuencias

Diagrama de secuencias de Cliente

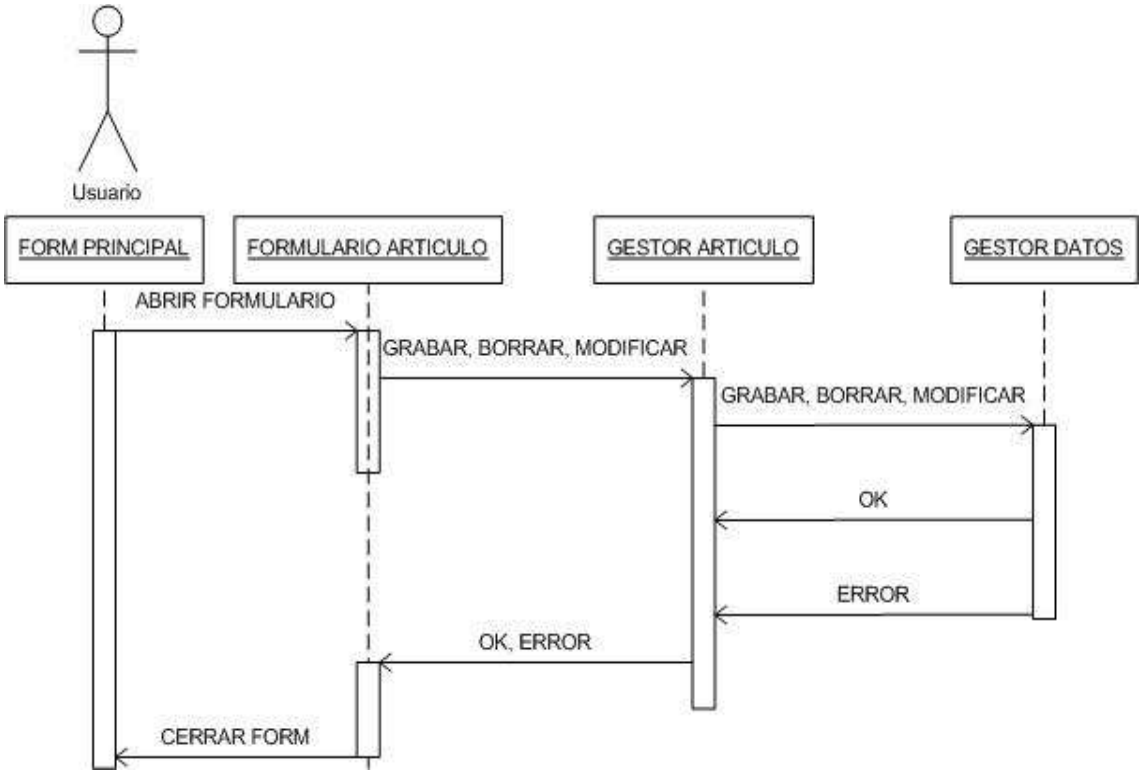
En la siguiente figura se muestra el diagrama de secuencias para la gestión de un cliente.



Al solicitar la apertura del formulario de cliente se pasa el control al formulario, dependiendo de la acción que queramos hacer se pasa el control al gestor de cliente, la única operación que podemos hace sin utilizar el gestor de cliente es el cierre del formulario, una vez transferida la petición el gestor cliente hace la petición al gestor de base de datos que puede estar programado en cualquiera de los sistemas de nuestro estudio, el gestor de datos nos retornara un error o un ok dependiendo si la operación ha terminado correctamente, transfiriendo el control nuevamente al gestor y este notificando al formulario.

Diagrama de secuencias de Articulo

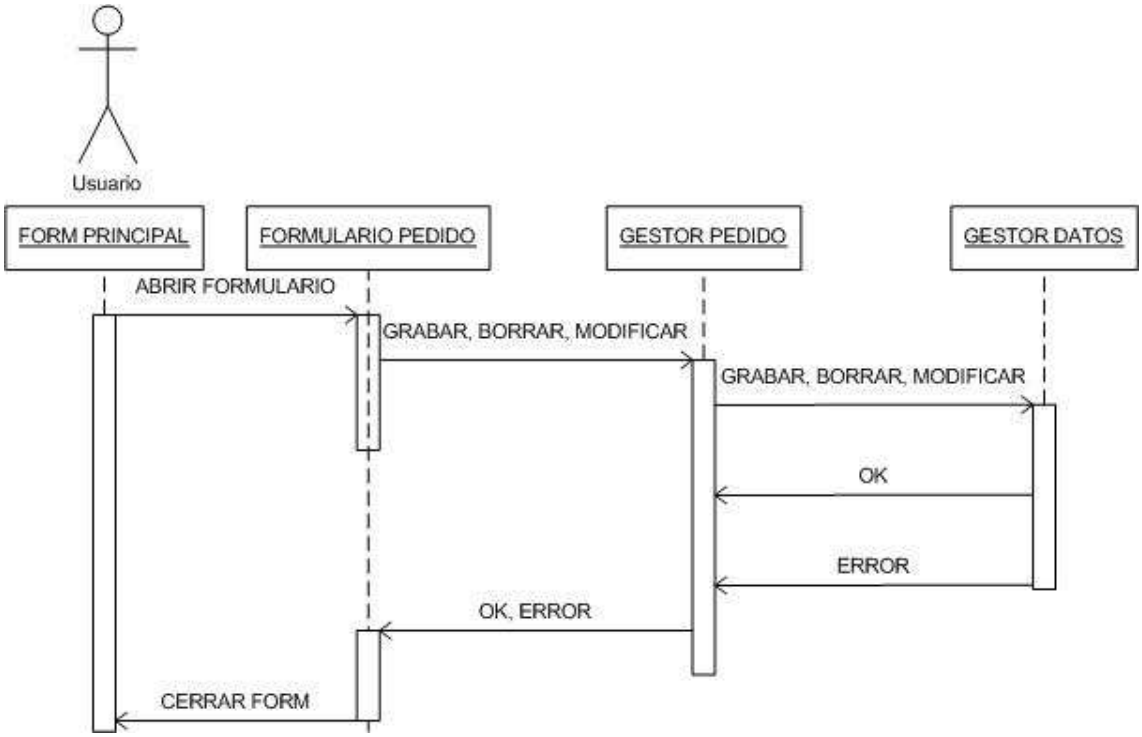
En la siguiente figura se muestra el diagrama de secuencias para la gestión de un articulo.



El funcionamiento de la gestión de un artículo es idéntica a la de un cliente, al solicitar la apertura del formulario de articulo se pasa el control al formulario, dependiendo de la acción que queramos hacer se pasa el control al gestor de artículo, la única operación que podemos hace sin utilizar el gestor de articulo es el cierre del formulario, una vez transferida la petición el gestor articulo hace la petición al gestor de base de datos que puede estar programado en cualquiera de los sistemas de nuestro estudio, el gestor de datos nos retornara un error o un ok dependiendo si la operación ha terminado correctamente, transfiriendo el control nuevamente al gestor y este notificando al formulario.

Diagrama de secuencias de Pedido

En la siguiente figura se muestra el diagrama de secuencias para la gestión de un cliente.



También el funcionamiento del diagrama de secuencias del pedido es muy similar al de clientes y artículos, la principal diferencia radica en el gestor de pedido, que tiene que controlar las cuatro tablas en lugar de una sola.

Al solicitar la apertura del formulario de pedido se pasa el control al formulario, dependiendo de la acción que queramos hacer se pasa el control al gestor de pedido, la única operación que podemos hace sin utilizar el gestor de pedido es el cierre del formulario, una vez transferida la petición el gestor cliente hace la petición al estor de base de datos que puede estar programado en cualquiera de los sistemas de nuestro estudio, el gestor de datos nos retornara un error o un ok dependiendo si la operación ha terminado correctamente, transfiriendo el control nuevamente al gestor y este notificando al formulario.

4 – Implementación

4.1– Desarrollo del aplicativo en ASP

4.1.1 – Descripción general

El enfoque que se ha dado al proyecto no es obtener un producto con grandes funcionalidades, lo que se pretende es documentar los diferentes sistemas de acceso a datos que nos ofrece la actual tecnología .NET.

El proyecto se divide en dos partes perfectamente diferenciadas, el primero de ellos es una implementación del sistema de acceso a datos Entity Framework, en el se recogen las principales consideraciones que se tienen que tener en cuenta para abordar un proyecto de este tipo así como una muestra de la implementación de esta tecnología.

El segundo es la implementación del mismo proyecto pero enfocado al sistema de acceso a datos oData, este sistema se compone de dos proyectos, uno servidor y otro cliente, el proyecto servidor es el encargado del acceso a los datos que se encuentran en la base de datos así como de publicar un acceso a estos mediante un servicio web, la segunda parte del proyecto consta de un acceso al servicio web del proveedor de datos, en nuestro caso la aplicación servidor, y la implementación con la misma interfaz de usuario que el primer proyecto.

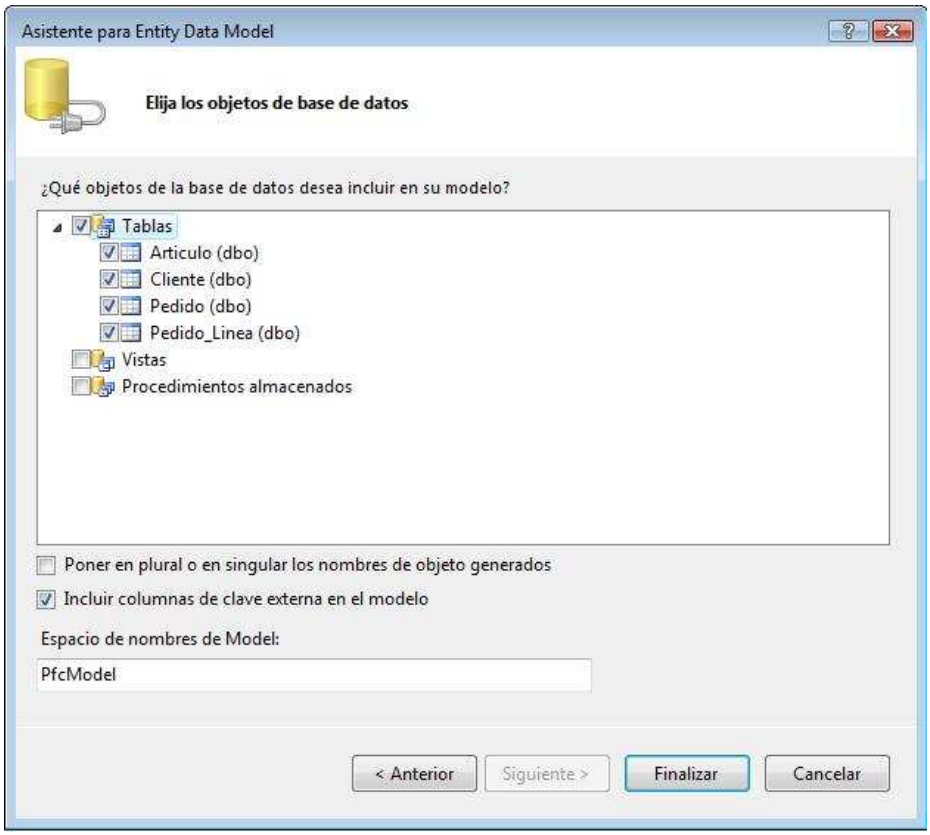
Para la implementación de ambos proyectos se ha utilizado el lenguaje de programación ASP.NET integrado en la plataforma de desarrollo Visual Studio 2010, como gestor de base de datos se utiliza SQL Server 2008, esta elección es importante ya que la nueva tecnología de acceso a datos Entity Framework encuentra problemas con la versión anterior de este gestor de datos el SQL Server 2005, según pruebas realizadas se encontró un problema de compatibilidad con las fechas ya que el nuevo sistema utiliza un tipo de fechas no soportadas por esta versión del producto.

4.2– El proyecto PfcEntity

4.2.1 – Integración con el gestor de datos

Es uno de los puntos más simples y mejor implementados, para la creación del modelo de entidad se dispone de un asistente que con unos simples pasos trasforma el modelo conceptual implementado en la base de datos en un conjunto de clases directamente instanciables con la función de persistencia integrada, dicho de esta manera no parece gran cosa pero si se ha programado anteriormente con Ado.NET uno se puede dar cuenta que el número de horas empleadas baja considerablemente, hasta el punto que puedas empezar a plantearte el abandono de las anteriores tecnologías en pro de esta nueva.

Para hacer la integración del modelo conceptual y que genere el modelo de entidad se tiene que agregar al proyecto un nuevo objeto, en este caso “ADO NET Entity data Model”, esto lanza un asistente que nos preguntara que base de datos queremos utilizar y que tablas utilizaremos para generar el modelo.



4.2.2 – El fichero Model1.edmx

El asistente de Entity Framework genera un fichero en el proyecto que en nuestro caso se llama Model1.edmx, el cual contiene todos los modelos generados automáticamente a partir de la definición de la base de datos, por ejemplo en nuestro caso así se genero la clase Articulo automáticamente.

```
#Region "Entidades"

''' <summary>
''' No hay documentación de metadatos disponible.
''' </summary>
<EdmEntityTypeAttribute(NamespaceName:="PfcModel", Name:="Articulo")>
<Serializable()>
<DataContractAttribute(IsReference:=True)>
Public Partial Class Articulo
    Inherits EntityObject
    #Region "Método de generador"

    ''' <summary>
    ''' Crear un nuevo objeto Articulo.
```

```
    ''' </summary>
    ''' <param name="articuloID">Valor inicial de la propiedad
ArticuloID.</param>
    ''' <param name="precio">Valor inicial de la propiedad Precio.</param>
    Public Shared Function CreateArticulo(articuloID As Global.System.Int32,
precio As Global.System.Decimal) As Articulo
        Dim articulo as Articulo = New Articulo
        articulo.ArticuloID = articuloID
        articulo.Precio = precio
        Return articulo
    End Function

    #End Region
    #Region "Propiedades primitivas"

    ''' <summary>
    ''' No hay documentación de metadatos disponible.
    ''' </summary>
    <EdmScalarPropertyAttribute(EntityKeyProperty:=true, IsNullable:=false)>
    <DataMemberAttribute()>
    Public Property ArticuloID() As Global.System.Int32
        Get
            Return _ArticuloID
        End Get
        Set
            If (_ArticuloID <> Value) Then
                OnArticuloIDChanging(value)
                ReportPropertyChanging("ArticuloID")
                _ArticuloID = StructuralObject.SetValidValue(value)
                ReportPropertyChanged("ArticuloID")
                OnArticuloIDChanged()
            End If
        End Set
    End Property

    Private _ArticuloID As Global.System.Int32
    Private Partial Sub OnArticuloIDChanging(value As Global.System.Int32)
    End Sub

    Private Partial Sub OnArticuloIDChanged()
    End Sub

    ''' <summary>
    ''' No hay documentación de metadatos disponible.
    ''' </summary>
    <EdmScalarPropertyAttribute(EntityKeyProperty:=false, IsNullable:=true)>
    <DataMemberAttribute()>
    Public Property Descripcion() As Global.System.String
        Get
            Return _Descripcion
        End Get
        Set
            OnDescripcionChanging(value)
            ReportPropertyChanging("Descripcion")
            _Descripcion = StructuralObject.SetValidValue(value, true)
            ReportPropertyChanged("Descripcion")
            OnDescripcionChanged()
        End Set
    End Property
```

```
    Private _Descripcion As Global.System.String
    Private Partial Sub OnDescripcionChanging(value As Global.System.String)
    End Sub

    Private Partial Sub OnDescripcionChanged()
    End Sub

    ''' <summary>
    ''' No hay documentación de metadatos disponible.
    ''' </summary>
    <EdmScalarPropertyAttribute(EntityKeyProperty:=false, IsNullable:=false)>
    <DataMemberAttribute()>
    Public Property Precio() As Global.System.Decimal
        Get
            Return _Precio
        End Get
        Set
            OnPrecioChanging(value)
            ReportPropertyChanging("Precio")
            _Precio = StructuralObject.SetValidValue(value)
            ReportPropertyChanged("Precio")
            OnPrecioChanged()
        End Set
    End Property

    Private _Precio As Global.System.Decimal
    Private Partial Sub OnPrecioChanging(value As Global.System.Decimal)
    End Sub

    Private Partial Sub OnPrecioChanged()
    End Sub

    #End Region
    #Region "Propiedades de navegación"

    ''' <summary>
    ''' No hay documentación de metadatos disponible.
    ''' </summary>
    <XmlIgnoreAttribute()>
    <SoapIgnoreAttribute()>
    <DataMemberAttribute()>
    <EdmRelationshipNavigationPropertyAttribute("PfcModel",
"FK_Pedido_Linea_Articulo", "Pedido_Linea")>
     Public Property Pedido_Linea() As EntityCollection(Of Pedido_Linea)
        Get
            Return
CType(Me,IEntityWithRelationships).RelationshipManager.GetRelatedCollection(Of
Pedido_Linea)("PfcModel.FK_Pedido_Linea_Articulo", "Pedido_Linea")
        End Get
        Set
            If (Not value Is Nothing)
                CType(Me,
IEntityWithRelationships).RelationshipManager.InitializeRelatedCollection(Of
Pedido_Linea)("PfcModel.FK_Pedido_Linea_Articulo", "Pedido_Linea", value)
            End If
        End Set
    End Property

    #End Region
End Class
```

Podemos apreciar en esta clase como el generador no solo crea la clase Articulo, sino que también la crea en el namespace PfcModel, crea todos los métodos y las asociaciones con otras tablas que en el modelo conceptual están representadas mediante Foreing Keys, como podemos observar es espectacular la cantidad de código que ahorramos teclear utilizando este asistente.

4.2.3 – Acceso a las clases

Para el proyecto Entity Framework se utiliza ASP.NET MVC, con lo cual el control de la página se hace en la clase HomeControler, para instanciar las clases y poder usarlas en nuestras páginas asp.net como por ejemplo podría ser la página que lista los clientes en pantalla.

Tenemos que crear una referencia a PfcEntities que es la clase que contiene nuestras entidades y la asignamos a _db, después simplemente la instanciamos y con el método ".ToList()" nos retorna una lista de objetos cliente que ya pueden ser utilizados para mostrarlos en pantalla.

```
Public Class HomeController
Inherits System.Web.Mvc.Controller
Dim _db As PfcEntities

Public Sub New()
    _db = New PfcEntities()
End Sub

Function Cliente() As ActionResult
    ViewData.Model = _db.Cliente.ToList()
    Return View()
End Function
```

Una vez que tenemos la lista vamos a mostrarla en pantalla, en este caso ya tenemos el código en lugar de VB en ASP, en la primera línea podemos ver que la pagina hereda la lista de clientes (`Inherits="System.Web.Mvc.ViewPage(Of List(Of PfcEntity.Cliente))"`), ahora únicamente tenemos que ejecutar un bucle por todos los elementos de la lista para ir mostrando los atributos de la clase cliente en una tabla, esta operación se simplifica enormemente si ponemos un objeto "GridView" en la pagina y asignamos la lista a su método ".DataSource" ya que se generan automáticamente las columnas y el contenido con lo cual queda poco por hacer en la pagina, pero he preferido este método porque quería comprobar si podía acceder a elementos individuales de la lista y poder asignar atributos de estos clases a otros elementos de la pagina, como pueden ser los botones de editar o borrar que contienen el código de cliente para enviar acciones a otros formularios.

```
<%@ Page Language="VB"  MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage(Of List(Of PfcEntity.Cliente))" %>

<asp:Content ID="clienteTitle" ContentPlaceHolderID="TitleContent"
runat="server"> Cliente </asp:Content>

<asp:Content ID="clienteContent" ContentPlaceHolderID="MainContent"
runat="server">


    <form id="form1" runat="server">

    <div>

    <table border="0" cellpadding="0" cellspacing="0">
        <tr>
        <th>ID Cliente</th>
        <th>Nombre </th>
        <th>Dirección </th>
        <th>Población </th>
        <th>Provincia </th>
        <th>Cod. Postal </th>
        <th>Telefono </th>
        <th> </th>
        </tr>

        <%  For Each m In Model%>
            <tr>
            <th><%= m.ClienteID%></th>
            <td><%= m.Nombre%></td>
            <td><%= m.Direccion%></td>
            <td><%= m.Poblacion%></td>
            <td><%= m.Provincia%></td>
            <td><%= m.CPostal%></td>
            <td><%= m.Telefono%></td>
            <td>
            <%= Html.ActionLink("Editar", "EditCliente", New With {.id =
m.ClienteID})%>
            <%= Html.ActionLink("Borrar", "DeleteCliente", New With {.id =
m.ClienteID})%>
            </td>
            </tr>
```

```
        <% Next%>
    </table>
    <%= Html.ActionLink("Añadir Cliente", "AddCliente")%>
    <%= Html.ActionLink("Menu Principal", "Index", "Home")%>

    </div>
    </form>
</asp:Content>
```

4.2.4 – El método Add

Para incluir una nueva clase Cliente en la entidad también disponemos de un método que hace que esta operación sea muy simple, además cuando hace el intento de grabación de los datos se hace una comprobación de integridad referencial, es decir, que no se incumpla ninguna restricción de claves en la base de datos, esto se hace más patente en el método “Delete” ya que existe una restricción la cual no permite la eliminación de un cliente si se utiliza en un pedido y si se intenta el sistema retorna un mensaje de error.

En este caso también el control de la página se hace en la clase HomeControler, para instanciar las clases y poder usarla en nuestras páginas asp.net en este caso la pantalla para agregar un cliente nuevo.

La diferencia principal de este caso con el anterior es ahora utilizamos dos métodos de la entidad que se han generado automáticamente:

```
_db.AddToCliente(clienteToAdd)
_db.SaveChanges()
```

Este método solo necesita un nuevo objeto “Cliente” asignarlo como parámetro y después llamar al método “.SaveChanges()”, con esta dos simples instrucciones se soluciona todo el problema de la persistencia, a continuación podemos ver la función completa “AddCliente”, con lo cual podemos comprobar la simplicidad del conjunto, ya que la misma función nos permite el control de errores validando los campos del formulario y mostrando un mensaje de error si se viola alguna condición.

```
Public Function AddCliente(ByVal form As FormCollection)
        Dim clienteToAdd As New Cliente()

        ' Deserializamos
        TryUpdateModel(clienteToAdd, New String() {"Nombre", "Direccion",
"Poblacion", "Provincia", "CPostal", "Telefono"}, form.ToValueProvider())

        ' Validacion
        If String.IsNullOrEmpty(clienteToAdd.Nombre) Then
            ModelState.AddModelError("Nombre", "El Nombre es necesario!")
        End If

        ' Si es valido se graba el cliente en la BD
        If (ModelState.IsValid) Then
            _db.AddToCliente(clienteToAdd)
            _db.SaveChanges()
            Return RedirectToAction("Cliente")
        End If

        ' En caso contrario se refresca la vista
        Return View(clienteToAdd)
    End Function
```

4.2.5 – El método Update

Para modificar los atributos de una clase en la entidad también disponemos de un método que hace que esta operación sea muy simple, al igual que la anterior cuando hace el intento de sobrescribir los datos se hace una comprobación de integridad referencial, por ejemplo no puedo cambiar el ID de cliente en un pedido si no existe ese ID de cliente en la base de datos, lo cual el sistema retorna un mensaje de error.

En este caso también el control de la página se hace en la clase HomeControler, para instanciar las clases y poder usarla en nuestras páginas asp.net en este caso la pantalla para editar un cliente existente.

En este caso difiere del anterior en que ya no utilizamos dos métodos, en este caso solo utilizamos uno. Anteriormente se ha visto como podemos acceder a una colección de objetos, para pode modificar un objeto concreto Entity Framework también dispone de métodos específicos para tal fin, a continuación mostramos el método para acceder a un objeto concreto y la llamada para que el objeto se modifique en la base de datos:

```
Cliente = _db.Cliente.First(Function(m) m.ClienteID = id)

_db.SaveChanges()
```

Esto es posible debido que el objeto “Cliente” ya existe en la entidad y lo único que hacemos es modificar algún atributo por lo tanto solo tenemos que llamar al método “.SaveChanges()”.

```
Public Function EditCliente(ByVal id As Integer)
        ' Cogemos el cliente a modificar
        Dim clienteToUpdate As Cliente = _db.Cliente.First(Function(m)
m.ClienteID = id)
        ViewData.Model = clienteToUpdate
        Return View()
    End Function

    <AcceptVerbs(HttpVerbs.Post)> _
    Public Function EditCliente(ByVal form As FormCollection)

        ' Get movie to update
        Dim id As Integer = Integer.Parse(form("ClienteID"))
        Dim clienteToUpdate As Cliente = _db.Cliente.First(Function(m)
m.ClienteID = id)

        ' Deserialize (Include white list!)
```

```
        TryUpdateModel(clienteToUpdate, New String() {"Nombre", "Direccion",
"Poblacion", "Provincia", "CPostal", "Telefono"}, form.ToValueProvider)
        ' Validate
        If String.IsNullOrEmpty(clienteToUpdate.Nombre) Then
            ModelState.AddModelError("Nombre", "El Nombre es necesario!")
        End If

        ' If valid, save movie to database
        If (ModelState.IsValid) Then
            _db.SaveChanges()
            Return RedirectToAction("Cliente")
        End If
        ' Otherwise, reshow form
        Return View(clienteToUpdate)
    End Function
```

4.2.6 – El método Delete

Para eliminar una clase de la entidad también disponemos de un método Delete, al igual que la anterior cuando hace el intento de eliminar una clase se hace una comprobación de integridad referencial, por ejemplo no puedo eliminar un cliente si esta en uso en un pedido, lo cual hace que el sistema retorna un mensaje de error.

Este caso tenemos que acceder específicamente al objeto que queremos eliminar igual que en el caso anterior pero utilizaremos el método"DeleteObject".

```
        Dim clienteToDelete As Cliente = _db.Cliente.First(Function(m)
m.ClienteID = id)

        _db.DeleteObject(clienteToDelete)

        _db.SaveChanges()
```

Es la función con menos líneas ya que no necesita hacer ninguna comprobación del contenido del objeto para intentar el borrado, solo se tiene en cuenta la restricciones de la base de datos.

```
Public Function DeleteCliente(ByVal id As Integer)
        ' Cogemos cliente a borrar
        Dim clienteToDelete As Cliente = _db.Cliente.First(Function(m)
m.ClienteID = id)

        ' Borrar
        _db.DeleteObject(clienteToDelete)
        Try
            'si es una FK no la puede borrar
            _db.SaveChanges()
        Catch ex As Exception

        End Try

        ' Mostramos vista de cliente
        Return RedirectToAction("Cliente")
    End Function
```

4.3– Desarrollo del aplicativo en ASP

4.3.1 – El proyecto PfcServicioOData

Este proyecto ASP solo consta de un formulario que contiene la publicación del servicio oData, este proyecto siempre tiene que estar en funcionamiento ya que es el proveedor datos de la aplicación cliente, aquí nuevamente podemos observar que realmente soluciona lo que pretendía este protocolo ya que el Protocolo de datos abierto (OData) permite la creación de servicios de datos basada en HTTP, lo cual queda claro que lo consigue con una gran simplicidad, también ser publicado y editado por clientes Web mediante simples mensajes HTTP y también se consigue este punto aportando una gran flexibilidad y opciones en el acceso a los datos a través de los servicios expuestos..

4.3.2 – Integración con el gestor de datos

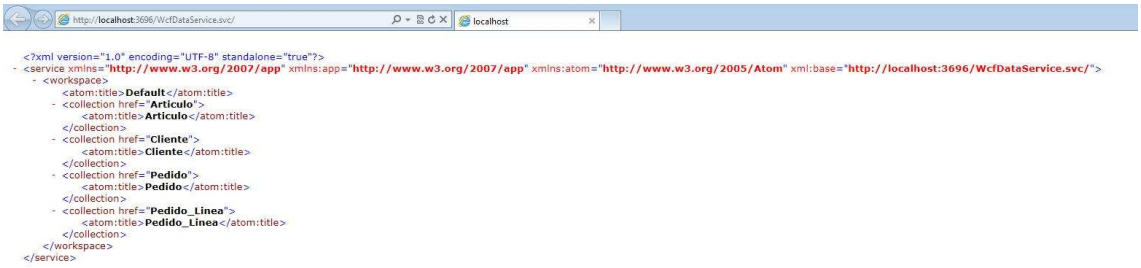
Este punto es similar al anterior porque también para la creación del modelo de entidad se utiliza el asistente que con unos simples pasos trasforma el modelo conceptual implementado en la base de datos en un conjunto de clases directamente instanciables con la función de persistencia integrada, hasta aquí es similar pero ahora crear un servicio de datos es realmente sencillo, a continuación mostramos todo el código necesario para crearlo.

```
Imports System.Data.Services
Imports System.Data.Services.Common
Imports System.Linq
Imports System.ServiceModel.Web

Public Class WcfDataService Inherits DataService(Of PfcEntities)

 Public Shared Sub InitializeService(ByVal config As DataServiceConfiguration)
    config.SetEntitySetAccessRule("*", EntitySetRights.All)
    config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2
 End Sub

End Class
```

Lo primero es poner el origen de datos, en nuestro caso es el modelo creado por el asistente PfcEntities, después solo tenemos que crear la regla de acceso mediante la instrucción “`config.SetEntitySetAccessRule("*", EntitySetRights.All)`” donde el “*” significa todas las tablas de la base de datos, en el caso de restringir el acceso a las tablas tendríamos que cambiar el asterisco por el nombre de la tabla que queremos dar permiso, el segundo parámetro indica los privilegios de acceso otorgados a la tabla, los privilegios permitidos son:

Parámetro	Descripción
All	Todas las lecturas y escrituras son permitidas
AllRead	Todas las lecturas son permitidas
AllWrite	Todas las escrituras son permitidas
None	No tiene permitido el acceso
ReadMultiple	Permite lectura múltiple de líneas
ReadSingle	Permite lectura de una línea
WriteAppend	Crear nuevos datos está permitido
WriteDelete	Borrar datos etas permitido
WriteMerge	Se permite actualización de combinación
WriteReplace	Se permite actualización de reemplazar

Al ejecutar la aplicación podemos ver la pantalla mostrando que el servicio está en marcha.

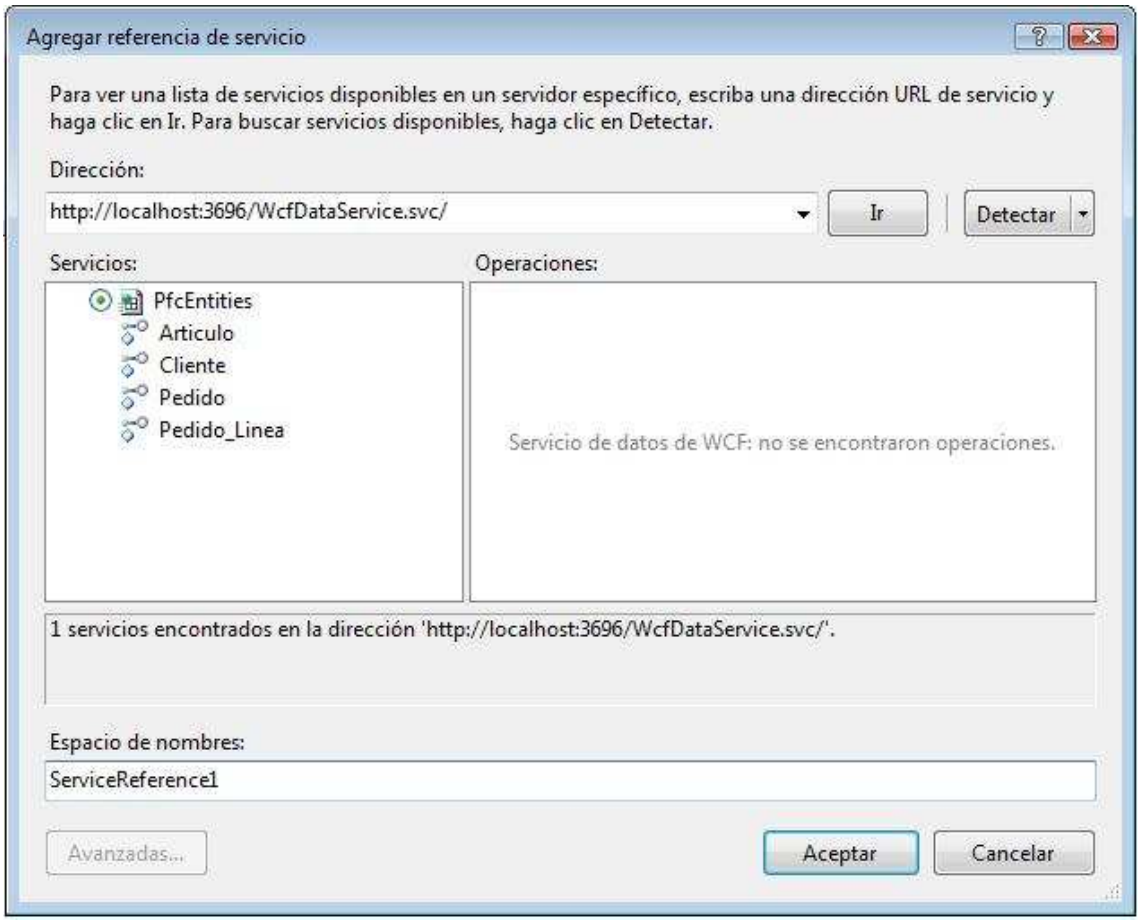


4.3.3 – El proyecto PfcClienteOData

Este proyecto ASP es el consumidor de servicios que el proyecto anterior pone a disposición de los clientes del web, básicamente es un proyecto aparentemente igual que el anterior pero el acceso a los datos se realiza conectando con el servicio a través de una URL, lo que permite una deslocalización entre el origen de datos y el cliente que lo consume.

4.3.4 – Consumiendo un web service

El primer paso es agregar un servicio web a nuestro proyecto, esto se hace incluyendo en el proyecto un nuevo elemento en “agregar referencia de servicio”, aparece un asistente que a partir de la URL donde esta publicado el servicio, el asistente nos mostrara todos los recursos disponibles en la URL que publica el servicio, en nuestro caso podemos ver las entidades que hemos generado automáticamente en el servicio.



4.3.5 – Programando con las referencias

Para poder utilizar las clase que nos proporciona el servicio oData, lo primero que tenemos que hacer es importar la referencia que nos ha generado automáticamente el asistente "`Imports PfcClienteOData.ServiceReference1`", ahora ya tenemos acceso a todas las clases publicadas.

Para crear un objeto tenemos que referenciarlo con la dirección Url donde esta publicado el servicio, en este proyecto se ha creado una variable de sesión para no tener que repetir la escritura de la Url cada vez que se hace referencia a una clase, en nuestro caso se ha creado en el fichero Global.asax para que se cree automáticamente al iniciar la sesión, la instrucción que utilizamos para hacer referencia al servicio es "`Dim PfcDbEnt As ServiceReference1.PfcEntities = New ServiceReference1.PfcEntities(New Uri(Session("Uri")))`" donde Session("Uri") es la variable anteriormente comentada, el contenido de esta variable es "http://localhost:3696/WcfDataservice.svc", a continuación podemos ver el código para asignar todos los clientes a un GridView utilizando el método "DataSource", simplemente como a tenemos la referencia, lo podemos hacer en una sola línea "`GridView1.DataSource = PfcDbEnt.Cliente`".

```
Imports PfcClienteOData.ServiceReference1
Imports System.Data.Services.Client

Public Class Cliente
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
        ' Creacion de DataServiceContext usando el servicio URI.
        Dim contexto As DataServiceContext = New DataServiceContext(New
Uri(Session("Uri")))
        ' Utilizamos la referencia al servicio de datos
        Dim PfcDbEnt As ServiceReference1.PfcEntities = New
ServiceReference1.PfcEntities(New Uri(Session("Uri")))
        ' Mostramos todos los clientes en el grid
        GridView1.DataSource = PfcDbEnt.Cliente
        GridView1.DataBind()
    End Sub

    Protected Sub new_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
Nuevo.Click
        'Si el cliente es nuevo la variable de sesion es nothing
        Session("ObjCli") = Nothing
        Response.Redirect("clienteEdit.aspx")
    End Sub

    Private Sub GridView1_RowCommand(ByVal sender As Object, ByVal e As
System.Web.UI.WebControls.GridViewCommandEventArgs) Handles GridView1.RowCommand
        ' Creacion de DataServiceContext usando el servicio URI.
        Dim contexto As DataServiceContext = New DataServiceContext(New
Uri(Session("Uri")))
        ' Utilizamos la referencia al servicio de datos
        Dim PfcDbEnt As ServiceReference1.PfcEntities = New
ServiceReference1.PfcEntities(New Uri(Session("Uri")))
        ' Seleccionamos el codigo de cliente
        Dim codigo As String = GridView1.Rows(e.CommandArgument).Cells(0).Text
        ' Buscamos el cliente
        Dim dcli As ServiceReference1.Cliente = (From r In PfcDbEnt.Cliente Where
r.ClienteID = codigo Select r).First()
        Select Case e.CommandName
            Case "Editar"
                'Si el cliente no es nuevo la variable de sesion contiene la
clase cliente
                Session("ObjCli") = dcli
                Response.Redirect("clienteEdit.aspx")
            Case "Borrar"
                ' eliminamos el cliente
                Try
                    contexto.AttachTo("Cliente", dcli)
                    contexto.DeleteObject(dcli)
                    contexto.SaveChanges()
                Catch ex As Exception

                End Try
                Response.Redirect("cliente.aspx")
        End Select
    End Sub
End Class
```

Como podemos ver en el código el cambio principal radica en el sistema de instanciar las clases que contienen los datos, a partir de este punto los métodos add, update y delete son similares a la del anterior proyecto.

5 – CONCLUSIONES

Estamos delante de un avance importante en cuanto a la tecnología de acceso a datos, con la tecnología ADO.NET tradicional el coste de programación aumenta ya que se tienen que montar todos datasets y datareaders necesarios, también se tiene que montar todas las consultas SQL para la manipulación de datos, esto significa un coste en tiempo importante, pero por otra parte aunque se hayan sacado nuevas tecnologías de acceso a datos el uso de estas no excluye el uso de ADO.NET, sino que pueden convivir ambas complementándose.

El uso de DataSet se podía ver como una base de datos en memoria que contiene los datos consultados, esta se puede manipular y además conserva la misma estructura de la fuente de datos, además existe el objeto DataRelation que permite crear relaciones entre las DataTables de las que se compone un DataSet, todo estos objetos se sustituyen por entidades cuando trabajamos con Entity Framework lo que nos proporciona una abstracción mayor entre el código y la persistencia de los datos. Esta abstracción crea un avance cualitativo tanto para la programación como en los tiempos de desarrollo.

Entity Framework es un avance tanto en concepto como facilidad de programación, básicamente no separamos entre las parte de programación con objetos y la parte de persistencia, simplemente programamos igual y él se encarga de la persistencia, esto reduce la cantidad de código y solo centrarte en programar.

Aunque no es objeto de estudio en este proyecto el rendimiento de Entity Framework comparado con DataSets deja mucho que desear, cuando en un aplicativo el rendimiento es el factor predominante los Datasets es la mejor solución para la implementación de acceso a datos, aunque no ha que descartar una solución hibrida que combine la velocidad de los Datesets para puntos críticos del aplicativo y la sencillez y la potencia de gestionar nuestro sistema de persistencia como entidades sin preocuparnos de la persistencia.

En cuanto a oData se trata de una forma de exponer los datos a través de internet pero internamente está utilizando un de los dos sistemas anteriormente mencionados, con lo cual no podemos comparar este sistema ya que Entity Framework o Ado.net son parte de oData, de todas formas el potencial de esta tecnología es enorme, directamente se me ocurre que puede ser un sustituto o complemento de la programación web, entiéndase con ello los programas que utilizan una página de interfaz de usuario como podría ser un programa escrito en ASP, por ejemplo podría crear un software programado por ejemplo con Windows Forms pero el acceso a

datos lo hace de forma remota contra un servicio oData, evito problemas de compatibilidad entre navegadores, los datos que recibo del servicio oData son fácilmente almacenables en una base de datos local, lo cual me permitiría trabajar off-line en el caso de que no disponga de conexión, es decir podría introducir datos en el programa y cuando este disponga de conexión se puede sincronizar con el origen de datos volcando todo el trabajo realizado en el servidor.

También hemos podido ver dos sistemas para la manipulación de datos, Linq to Sql y Linq to Entities, en algunas publicaciones se comenta de que Linq to Sql ya está obsoleto por la aparición de Linq to Entities pero mientras se mantenga el desarrollo de ADO.NET estará vigente ya que permite facilitar cualquier manipulación de datos de una forma sencilla para el programador, este apartado merecería un proyecto por si mismo ya que la tecnología Linq es mucho más completa, ya que lo que se pretende con esta tecnología es permitir que todo el código hecho en Visual Studio (incluidas las llamadas a bases de datos, datasets, XMLs) sean también orientados a objetos. Antes de LINQ, la manipulación de datos externos tenía un concepto más estructurado que orientado a objetos. Además LINQ trata de facilitar y estandarizar el acceso a dichos objetos.

En la implementación sí que se ha producido un importante avance ya que si bien al principio te tienes que acostumbrar al nuevo modelo de datos después resulta fácil e intuitivo, me hubiera gustado poder desarrollar un aplicativo completo y mucho más complejo para poder evaluar hasta qué punto es una ventaja o un inconveniente, ya que yo habitualmente programo con acceso a dato tradicional y realmente no tengo grandes problemas, por ejemplo una duda que me ha quedado es que pasaría si cuando creó una lista de objetos estos son un par millones, por ejemplo en alguna ocasión utilizando un DataSet normal se ha llegado a hacer un Query de ese tamaño y realmente no ha tenido problema.

Tengo que reconocer que esta vez Microsoft me ha dado una grata sorpresa con esta tecnología, esta compañía nos tiene acostumbrados a grandes lanzamientos que al final son lentos, no funcionan como deberían o simplemente no gusta, pero con esta tecnología ha recuperado un poco de la confianza perdida, sobre todo con oData, la facilidad con que se pueden crear servicios es increíble, la publicación de los servicios también es muy sencilla, también Entity Framework me ha sorprendido por la cantidad de código que se genera automáticamente liberando al programador de esta pesada tarea.

Por último no dejar atrás toda la tecnología Linq, complemento ideal para toda esta tecnología de acceso a datos, cuesta creer que no se inventara antes, las consultas nos proveen de una flexibilidad para manipular los datos que ahorra muchísimo tiempo de programación.

Bibliografía

Riordan, Rebecca M. Aprenda ADO.NET ya: McGraw-Hill

Zorrilla Castro, Unai. ADO.NET Entity Framework 4.0 - Aplicaciones y servicios centrados en datos: Kassis Press

Zeeshan Hirani. Entity Framework Learning Guide

Julia Lerman (2009). Programming Entity Framework: O'Reilly Media

Larry Tenny, Zeeshan Hirani. Entity Framework 4.0 Recipes. A Problem-Solution Approach: Apress

David Sceppa (2006). Programming Microsoft ADO.NET 2.0 Core Reference: Microsoft Press

Fuentes web principales

http://www.canalvisualbasic.net/manual-net/ado-net/

http://msdn.microsoft.com/es-es/library/bb399572.aspx

http://weblogs.asp.net/scottgu/archive/2010/07/16/code-first-development-with-entity-framework-4.aspx

http://alexjimenez.wordpress.com/2010/08/31/entity-framework-guas-bsicas-y-el-enfoque/

http://msdn.microsoft.com/es-es/library/bb399567.aspx

http://www.codeproject.com/KB/database/DatabaseAcessWithAdoNet1.aspx

http://www.orbitone.com/en/blog/archive/2010/06/09/odata-and-wcf-data-services.aspx

http://programming4.us/database/2006.aspx

ANEXO

Manual de Usuario de la aplicación

Consideraciones Previas

Antes de empezar seguir la instrucciones de instalación y configuración incluidas en el documento Instalacion.doc

La entrega consta de tres soluciones. NET

- PfcEntity, proyecto ASP MVC con el sistema de acceso a datos Entity Framework.
- PfcServicioOData, proyecto servidor de datos que crea la interfaz oData
- PfcClienteOData, proyecto cliente del servicio de datos

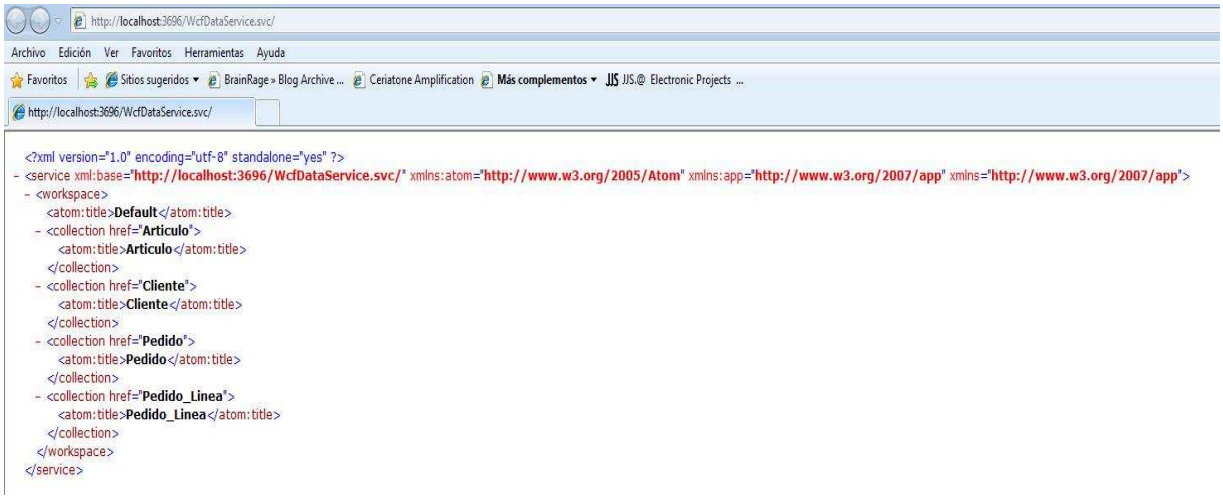
El manual de usuario para la solución PfcEntity y PfcClienteOData es el mismo ya que se ha utilizado el mismo concepto para la interfaz de presentación al usuario.

Puesta en marcha

PfcEntity, este proyecto no tiene ninguna especificación en especial, abrir el proyecto y ejecutar desde el Visual Studio 2010, la página de inicio es index.aspx.

OData, consta de dos proyectos, PfcServicioOData que es el proyecto que enlaza con la base de datos y expone los datos en un puerto, PfcClienteOData es el que proyecto que utiliza los datos expuestos en el servicio y hace de interfaz de usuario.

Primero abrir la solución PfcServicioOData con Visual Studio 2010 y ejecutar la solución, debe de mostrar una pantalla como la de la siguiente figura.



A continuación minimizar el explorador y el Visual Studio, abrir una segunda instancia del Visual Studio y abrir la solución PfcClienteOData, antes de ejecutarlo asegurarse de que la página de inicio sea default.aspx, esto se puede conseguir dándole un doble clic al formulario default como si quisiéramos editarlo en el explorador de soluciones de esta forma cuando ejecutamos la solución Visual Studio comienza con el formulario activo.

Menú principal

En la siguiente figura se muestra el menú principal de la aplicación

Botón Clientes, este botón nos muestra la gestión de cliente en la cual podremos crear borrar o editar cualquier cliente.

Botón Artículos, este botón nos muestra la gestión de artículos en la cual podremos crear borrar o editar cualquier artículo.

Botón Pedidos, este botón nos muestra la gestión de pedidos en la cual podremos crear borrar o editar cualquier pedido así como sus líneas.

Clientes

Esta pantalla muestra una lista de todos los clientes, desde la cual podemos dar de alta, editar o borrar un cliente.

Botón Página principal, nos lleva al menú principal de la gestión.

Botón Nuevo Cliente, nos permite entrar en el formulario para crear un nuevo cliente

Botón Editar, nos permite entrar en el formulario para modificar los datos de un cliente.

Botón Borrar, elimina un cliente siempre y cuando las restricciones lo permitan (no se puede borrar un cliente que se utiliza en un pedido.

Nuevo cliente

Este es el formulario de creación de un cliente.

Botón Página principal, nos lleva al menú principal de la gestión.

Botón Aceptar, graba el cliente en la base de datos.

Botón Salir, volvemos a la lista de clientes sin gravar.

Edición Cliente

Este es el formulario que permite modificar los datos de un cliente.

Botón Página principal, nos lleva al menú principal de la gestión.

Botón Aceptar, graba el cliente modificando los datos en la base de datos.

Botón Salir, volvemos a la lista de clientes sin modificarlo.

Artículos

Esta pantalla muestra una lista de todos los artículos, desde la cual podemos dar de alta, editar o borrar un artículo.

Botón Página principal, nos lleva al menú principal de la gestión.

Botón Nuevo Artículo, nos permite entrar en el formulario para crear un nuevo artículo.

Botón Editar, nos permite entrar en el formulario para modificar los datos de un artículo.

Botón Borrar, elimina un articulo siempre y cuando las restricciones lo permitan (no se puede borrar un artículo que se utiliza en un pedido.

Nuevo Articulo

Este es el formulario de creación de un cliente.

Botón Página principal, nos lleva al menú principal de la gestión.

Botón Aceptar, graba el artículo en la base de datos.

Botón Salir, volvemos a la lista de artículos sin gravar.

Edición Artículo

Este es el formulario que permite modificar los datos de un cliente.

Botón Página principal, nos lleva al menú principal de la gestión.

Botón Aceptar, graba el artículo modificando los datos en la base de datos.

Botón Salir, volvemos a la lista de artículos sin modificarlo.

Pedidos

Esta pantalla muestra una lista de todos los pedidos, desde la cual podemos darlo de alta, editarlo, borrarlo o acceder a las líneas que contiene.

Botón Página principal, nos lleva al menú principal de la gestión.

Botón Nuevo Pedido, nos permite entrar en el formulario para crear un nuevo pedido.

Botón Editar, nos permite entrar en el formulario para modificar los datos de un pedido.

Botón Borrar, elimina un pedido y todas las líneas de pedido asociadas a él.

Botón Líneas, nos permite ver las líneas asociadas al pedido.

Nuevo Pedido

Este es el formulario de creación de la cabecera de un pedido.

Botón Página principal, nos lleva al menú principal de la gestión.

Botón Aceptar, graba la cabecera del pedido en la base de datos.

Botón Salir, volvemos a la lista de pedidos sin gravar.

Edición Cabecera Pedido

Este es el formulario que permite modificar los datos de la cabecera de un pedido.

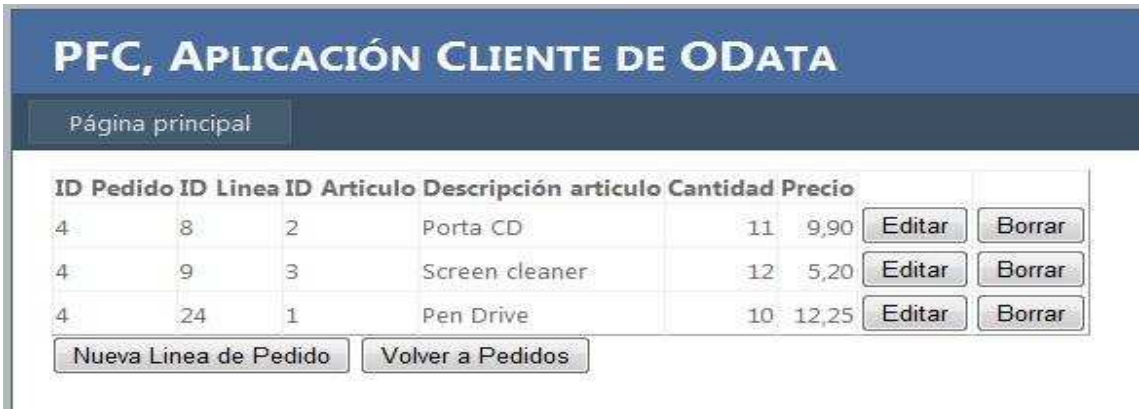
Botón Página principal, nos lleva al menú principal de la gestión.

Botón Aceptar, graba la cabecera del pedio modificando los datos en la base de datos.

Botón Salir, volvemos a la lista de pedidos sin modificarlo.

Líneas de Pedido

Esta pantalla muestra una lista de todas las líneas que contiene un pedido, desde la cual podemos crear una nueva, editarla o borrarlas líneas que contiene.



Botón Página principal, nos lleva al menú principal de la gestión.

Botón Nueva Línea de Pedido, nos permite entrar en el formulario para añadir una nueva línea al pedido.

Botón Volver a Pedidos, nos permite volver a la lista de pedidos.

Botón Editar, nos permite entrar en el formulario para modificar la línea del pedido.

Botón Borrar, elimina la línea del pedido.

Añadir línea al pedido

Este es el formulario para añadir una línea nueva al pedido.

Botón Página principal, nos lleva al menú principal de la gestión.

Botón Aceptar, graba la línea del pedido en la base de datos.

Botón Salir, volvemos a las líneas de pedido sin gravar.

Editar línea de pedido

Este es el formulario que permite modificar la línea de un pedido.

Botón Página principal, nos lleva al menú principal de la gestión.

Botón Aceptar, graba la línea del pedio modificando los datos en la base de datos.

Botón Salir, volvemos a las líneas del pedido sin modificarlo.

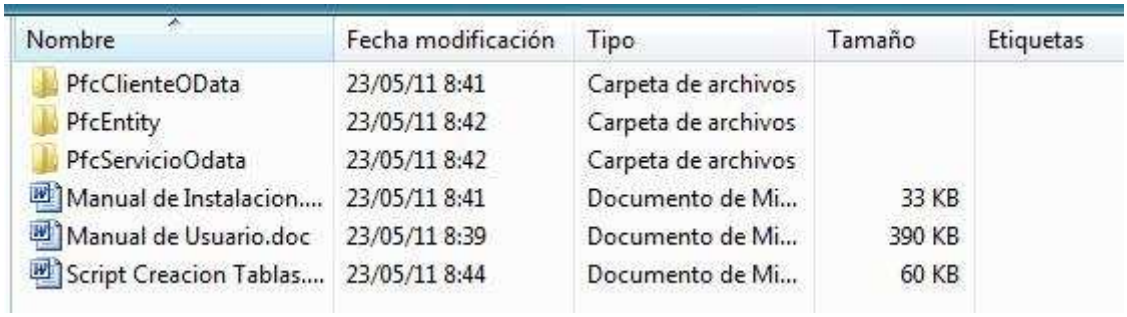
Manual de instalación

Requerimientos básicos

Los requerimientos básicos es tener instalado Visual Studio 2010 y Sql Server 2008 o la versión express, en pruebas realizadas con Sql Server 2005 se han producido problemas de incompatibilidad con el formato de fecha produciéndose un error en el programa.

Instalación

Descomprimir el fichero “jponceh_PFC_PAC3.rar”, creara una carpeta Pfc con el contenido siguiente:



Nombre	Fecha modificación	Tipo	Tamaño	Etiquetas
PfcClienteOData	23/05/11 8:41	Carpeta de archivos		
PfcEntity	23/05/11 8:42	Carpeta de archivos		
PfcServicioOdata	23/05/11 8:42	Carpeta de archivos		
Manual de Instalacion....	23/05/11 8:41	Documento de Mi...	33 KB	
Manual de Usuario.doc	23/05/11 8:39	Documento de Mi...	390 KB	
Script Creacion Tablas....	23/05/11 8:44	Documento de Mi...	60 KB	

Relación de ficheros contenidos

Manual de Instalacion.doc, este documento.

Manual de Usuario.doc, instrucciones para el funcionamiento de las gestiones.

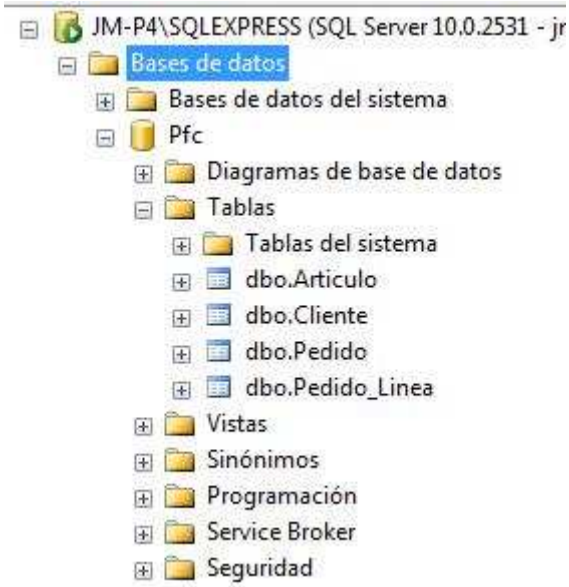
“Script Creacion Tablas.doc”, script que genera la base de datos y datos de ejemplo.

Carpeta PfcEntity, contiene la solución de acceso a datos Entity Framework

Carpetas PfcServicioOData y PfcClienteOData, contienen la solución de acceso a datos OData, funciona las dos soluciones al mismo tiempo en dos instancias de Visual Studio ya que una hace de servidora de datos y la otra de cliente.

Creación de la base de datos

En el Sql Server 2008 pulsar nueva consulta, copiar el contenido del documento “Script creación Tablas.doc” y pulsar ejecutar, esto creara la base de datos Pfc que contiene cuatro tablas.



Configuración de la cadena de conexión a la base de datos

En el proyecto PfcEntity y PfcServicioOData, que son los dos que tienen acceso a la base de datos la cadena de conexión se encuentra en el fichero web.config que es accesible abriendo los proyectos con Visual Studio 2010, la configuración es similar en los dos proyectos

La línea afectada en el web config es :

```
<add name="PfcEntities" connectionString=
```

En la constante connectionString haremos los cambios de configuración

Opción 1, Autenticación de Windows

El cadena de conexión no necesita usuario ni contraseña, solamente necesita que tenga el nombre del servidor de BD correctamente.

```
Data Source=JM-P4\SQLEXPRESS;Initial Catalog=Pfc;Integrated Security=True
```

Lo único que tenemos que hacer en este caso es modificar el parámetro Data Source y poner el nombre de nuestro servidor.

Opción 2, Autenticación de SQL SERVER

En este caso la cadena de conexión necesita el nombre del servidor de la BD, usuario y contraseña, hay que sustituir todo el trozo anterior por uno similar al siguiente.

```
Data Source= JM-P4\SQLEXPRESS;Initial Catalog=Pfc;User ID=sa;Password=contraseña;
```

Data Source, nombre del servidor de base de datos.

Initial Catalog, nombre de la BD, no tocar.

User ID, nombre del usuario de SQL Server, por defecto “sa” (system administrator).

Password, contraseña de acceso.