FEED YOUR BRAIN ®

ADO.NET Entity Framework

Aplicaciones y servicios centrados en datos

Unai Zorrilla Castro
Octavio Hernández
Eduardo Quintás

Prólogo de David Salgado, Development Evangelist de Microsoft.

krasis PRESS

ADO.NET Entity Framework

Unai Zorrilla
Octavio Hernández
Eduardo Quintás

Unai:

Para Lucía, centro y significado de mi vida

Octavio:

"I can always find my Cuban skies
In Rosalinda's eyes"
("Rosalinda's eyes", BILLY JOEL,
from "52nd Street", 1978)

A mi esposa Irina, en nuestro 25 aniversario

Eduardo:

"And how can we win
When fools can be kings"
("Knights of Cydonia", MUSE,
from "Back Holes & Revelations", 2006)

ADO.NET ENTITY FRAMEWORK

ISBN: 978-84-935489-9-5
Depósito Legal: C

Impreso en España-Printed in Spain

Prólogo

Lo reconozco, el trabajo con bases de datos siempre me ha provocado una cierta pereza. Desde mis primeros pinitos en el universo de la informática, siempre he encontrado más cercano el trabajo con objetos que con tablas, y más natural la expresión de mis ideas en lenguajes de programación como C++, C# ó Java que en T-SQL o PL/SQL; ni que decir tiene que un diagrama Entidad-Relación distaba generalmente bastante de lo que yo tenía en mente para mi aplicación (que normalmente era un diagrama de clases). Tal vez fueron estas diferencias de representación y tratamiento las que hicieron que me fuese alejando paulatinamente del mundo de las bases de datos y acercándome al mundo del desarrollo.

Puede ser debido al paso del tiempo y la inquietud de investigar en nuevos campos, o tal vez por pura necesidad, pero he ido cediendo y acercándome al mundo de las tablas y de los motores relacionales. Pocas veces tendremos la suerte de tener que diseñar algoritmos puros que no tengan que consumir datos provenientes de alguna fuente de información, porque… ¿qué son las aplicaciones sino meros instrumentos de transformación y manipulación de datos? Hay cosas que hay que asumir cuanto antes para poder seguir avanzando: por muy mal que te lleves con tu compañero de trabajo, con el departamento de IT, con el panadero o con el del taller… ¡vas a tener que verle frecuentemente! Es inevitable, y como es inevitable… por el bien de nuestra salud mental, lo mejor será llevarnos todo lo bien que podamos, aunque eso requiera un sacrificio por nuestra parte, ¿no?

En el tema que nos ocupa, ese acercamiento/sacrificio normalmente pasa por diferentes fases. Primero aprendes a traerte los datos a tu terreno, para saber manipularlos donde eres fuerte, en la capa de negocio, así que te lías la manta a la cabeza y a base de enviar sentencias SQL desde la capa de negocio, te traes los datos y trabajas en tu lenguaje preferido. Luego llega una etapa en la que no quieres los datos, sólo quieres los resultados, así que T-SQL y PL/SQL se convierten en tus mejores amigos y todo lo ves *encapsulable* en el motor de bases de datos; te dedicas a llamar a procedimientos almacenados y a recoger los resultados… pero esos amigos te quitan el control sobre los datos al que estabas acostumbrado, ¡has acabado haciendo casi todo en la base de datos! Es el momento de recuperar el control, te parece que la lógica de tu aplicación no está donde debe estar. Y entonces pasas a trabajar con herramientas ORM. Al más puro estilo ONU, estas herramientas se encargan de lidiar con las diferencias entre la capa de negocio y la capa de acceso a datos. Hablan en sentencias SQL para la capa de datos, y para nosotros en clases, objetos y procedimientos almacenados encapsulados como métodos. Aunque conservan la estructura de la base de datos, y ésta puede no ser la más adecuada para nuestra aplicación, es lo más parecido a programar independientemente de la base de datos y del motor relacional que habíamos conocido. Hasta ahora.

Olvídate de la tabla A o la tabla B: en lugar de trabajar contra una base de datos, vas a empezar a trabajar con algo más cercano. Imagina que la base de datos te presenta la estructura de su contenido en un folio en blanco (relaciones, columnas, datos…) y que tú puedes organizar esa información de la manera que más te convenga para tu aplicación; estás creando un modelo de los datos para tu aplicación. Una vez tengas ese modelo que te abstrae de la representación física de la base de datos, e incluso del motor de base de datos subyacente, tu aplicación se centrará en consultar los datos de ese modelo, y el modelo se encargará de actualizarse en la base de datos. ¡Bienvenido a Entity Framework ☺!

Con el libro que tienes en tus manos, aprenderás a sacarle el máximo partido a esta nueva tecnología destinada a marcar un antes y un después en la relación entre la capa de negocio y la capa de acceso a datos. ¿Qué es Entity Framework? ¿Cómo crear esos modelos de información? ¿Cómo consultar la información de esos modelos desde la aplicación? Éstas son solo algunas de las preguntas a las que encontrarás aquí la respuesta, y es que tanto tú como yo tenemos la suerte de que los autores del libro no se quedan en la superficie: son una especie de *buceadores tecnológicos*. Cuando tengas las respuestas a las preguntas descritas anteriormente y te encares con tus primeros proyectos en Entity Framework, empezarás a apreciar la capacidad de *buceo* de los autores. Volverás al libro para ver las mejores prácticas en la herencia, cómo sacarle el máximo partido a una consulta, cómo llegar incluso a definir modelos más allá de lo que te permiten las herramientas visuales de diseño editando el propio XML que los define.

Estos *buceadores*, Unai, Octavio y Eduardo, realizan una gran aportación a las comunidades de desarrolladores, nos ayudan a través de sus blogs, sus conferencias, artículos… ¡y encima tienen tiempo para trabajar y para escribir libros! Personalmente, tengo la suerte de conocerles y de haber compartido momentos de trabajo y momentos de ocio con todos ellos, incluyendo hasta *piques* sobre cómo se comportará el compilador de C# en presencia de cierto código, sin que valga compilarlo para ver qué pasa, ¿verdad, Unai? ¿Esto sería ocio o trabajo?

Estimado lector, confío en que este libro te será de gran utilidad, tanto en tu fase de descubrimiento de qué es Entity Framework y para qué puedes utilizarlo, como en los retos que surjan más adelante en el día a día del uso de la tecnología. Autores (amigos), gracias por vuestra aportación a la comunidad y por vuestra dedicación al mundo del software; así da gusto trabajar. Familiares y amigos de los autores, gracias por tener esa increíble paciencia y por tenerlos siempre con una sonrisa en la cara; sin vosotros, esto no tiene sentido.

Happy Hacking!

David Salgado
blogs.msdn.com/DavidSalgado
Development Evangelist, Microsoft Ibérica

Autores

Unai Zorrilla Castro
Microsoft MVP y MCTS
Actualmente, Unai es Development Advisor para grandes cuentas en Plain Concepts, empresa de la cual es socio fundador, y asiduo colaborador de Microsoft Ibérica en talleres para arquitectos, jornadas de formación y giras de producto. Es también tutor de CampusMVP y ha publicado multitud de artículos técnicos en revistas especializadas como dotNetManía, MSDN Online y MTJ .NET.

Octavio Hernández Leal
C# MVP, MCT y MCSD
Ingeniero en Informática con más de veinte años de experiencia en formación, consultoría y desarrollo de aplicaciones. Autor del libro "C# 3.0 y LINQ" y de múltiples artículos relacionados con .NET para revistas especializadas. Es además tutor de CampusMVP.

Eduardo Quintás
Eduardo es Ingeniero en Informática y trabaja como Técnico Superior en el área de Innovación Tecnológica de la Universidad de la Coruña. Realiza consultorías de desarrollo para PlainConcepts, colabora habitualmente en **geeks.ms** y está especializado en el desarrollo de soluciones web en .NET, bases de datos y arquitecturas orientadas a servicios.

Contenido

Prólogo de los autores

Escribir un libro es una tarea ardua, que conlleva muchas horas de dedicación. Si sumamos a esto lo difícil que es hablar sobre una tecnología que acaba de salir al mercado y sobre la que no existe una experiencia de uso previa, se imaginará usted la cantidad de trabajo que ha sido necesario para sacar esta obra adelante. Desde los primeros momentos en que pensamos en escribir este libro, a la vista de las primeras betas de un producto que nos maravillaba, los autores deseábamos crear una obra plural, en la que interviniera la mayor cantidad de gente posible, aportando sus ideas y sus consejos. Creemos firmemente que este deseo se ha convertido en realidad, y que usted tiene entre las manos un ejercicio de sinceridad sobre una tecnología que puede marcar un antes y un después en su forma de desarrollar y de abordar los proyectos de software centrados en datos.

Contenido del libro

El libro consta de seis capítulos, más cinco apéndices de referencia.

- Los primeros cinco capítulos cubren ampliamente la gran mayoría de las características del Marco de entidades de ADO.NET (ADO.NET Entity Framework). Comenzando por un capítulo de introducción a la tecnología, luego se continúa con un capítulo centrado en la creación de modelos conceptuales de entidades, que describe desde los apartados más básicos hasta las posibilidades más avanzadas por medio de ejemplos claros y prácticos. Los siguientes capítulos introducen al lector en las distintas posibilidades que la tecnología ofrece para consultar y actualizar los datos de un modelo conceptual, todo ello de una forma ordenada, tal y como se fueron concibiendo los distintos subsistemas que componen el Marco de entidades, y explicando las ventajas e inconvenientes de cada uno de ellos.

- Para finalizar, el último capítulo del libro ofrece una introducción a ADO.NET Data Services, una aplicación práctica del Marco de entidades desarrollada por Microsoft para exponer la infraestructura que conforma esta tecnología a través de servicios REST.

Sobre los ejemplos

El código fuente de todos los ejemplos del libro se encuentra disponible para su descarga en el sitio Web del editor, **www.krasispress.com**. Para compilarlo y ejecutarlo, deberá tener instalada cualquier edición de Visual Studio 2008 o incluso Visual C# Express 2008, siempre que incluyan el Service Pack 1, en el que se incorporó el

Marco de entidades a .NET Framework, y los asistentes que facilitan el uso de esta tecnología a Visual Studio 2008.

La casi totalidad de los ejemplos del libro se apoya en una base de datos de SQL Server 2005 diseñada para almacenar la información que utiliza y genera una tienda online que vende productos digitales (software, música, vídeos, etc.), a la que hemos llamado "Media Amazon", o más sucintamente **MAmazon**. Encontrará una copia de seguridad de esta base de datos junto con el código de los ejemplos.

Agradecimientos

Los autores quieren aprovechar la ocasión para agradecer a los siguientes colectivos e individuos:

- Al equipo de desarrollo del Marco de entidades de ADO.NET por el gran trabajo realizado; trabajo al que seguirán sin duda alguna nuevas y excitantes versiones que ya estamos deseando ver y asimilar. En especial, vaya nuestro agradecimiento a **Danny Simmons**, **Alex James**, **Diego Vega** y **Pablo Castro**, cuyas respuestas y comentarios han ayudado en gran medida a enriquecer esta obra.
- A todo el grupo DPE de Microsoft Ibérica, por su continuo apoyo a todo lo relacionado con el desarrollo con .NET Framework en general y a la escritura de este libro en particular. Nuestro agradecimiento a **David Carmona**, **David Salgado**, **José Murillo**, **Alfonso Rodriguez**, **Beatriz Ordóñez**, **Ethel García**, **Isabel Gómez**, **Salvador Sánchez**, **César de la Torre**, **Antonio Gómez** y **Aurelio Porras**, así como también a **Cristina González Herrero**, que realiza una encomiable labor como responsable del programa MVP de Microsoft.
- A todos los miembros de esa gran familia en expansión que es **Plain Concepts** (**www.plainconcepts.com**), y en particular de aquellos que han participado activamente en la creación (aportando ideas o ejemplos) y revisión de este libro, como **Yamil Hernández**, **Cristina González Muñoz** y **Vicente García**.
- A quienes han participado en la revisión desinteresada de este libro, aportando enriquecedores comentarios: **Isabel Gómez**, **David Salgado**, **Augusto Ruiz** y **Alberto Población**.
- Por último, no nos gustaría dejar pasar la ocasión de agradecer a todos aquellos amigos (más que clientes ya) que se han atrevido a adoptar de nuestra mano esta tecnología en sus proyectos, y gracias a los cuales hemos obtenido una gran cantidad de *feedback* que sin dudas nos ha ayudado a comprender las necesidades y preocupaciones que los desarrolladores sienten al enfrentarse por primera vez a una tecnología novedosa como es el Marco de entidades, y a saber lo que los lectores podrían necesitar de un libro como éste. A la gente de **Acciona**, **Enyca**, **Hispanitas**, **Taya**, etc. - ¡muchas gracias!

Madrid, agosto de 2008

CAPÍTULO 1

Introducción

En la primera parte de este capítulo, trataremos de introducir al lector en los distintos argumentos y consideraciones que han sido tenidos en cuenta y servido como base para el desarrollo del **Marco de entidades de ADO.NET** (ADO.NET Entity Framework, que abreviaremos en lo adelante por **EF**), así como en el conjunto de conceptos que han surgido alrededor de éste. A continuación, veremos qué es EF, cómo es su arquitectura y cuáles son los elementos fundamentales que forman parte de él.

¿POR QUÉ EL MARCO DE ENTIDADES?

Mientras se redactaba este libro, ya estábamos dando charlas de introducción con las primeras betas del producto. Uno de los principales objetivos de estas sesiones era el de intentar hacer llegar a los asistentes los distintos razonamientos y objetivos que han dado pie al surgimiento de esta nueva tecnología, así como de una nueva forma de trabajo. Una de las diapositivas más sencillas, y sin embargo más ilustrativas, de esas presentaciones consistía únicamente en una frase similar a *"Los desarrolladores no somos fontaneros"*. Seguramente a usted, estimado lector, esto le parecerá una 'verdad de Perogrullo', y no le faltará razón; pero aunque no se lo crea, esta verdad la olvidamos con demasiada facilidad. A lo largo del tiempo, los desarrolladores hemos ido aprendiendo, creando y mejorando muchos conceptos de los lenguajes orientados a objetos (OO), que son aplicados constantemente en todos nuestros desarrollos. Elementos como la herencia de clases y el polimorfismo son esenciales en la mayoría de los desarrollos, puesto que nos facilitan y nos abren nuevas y mejores vías de trabajo; numerosos son, además, los libros y documentación acerca de los patrones de diseño y cómo estos se aplican en las diferentes implementaciones que realizamos. Desde un punto de vista de calidad, nuestro trabajo además se evalúa utilizando métricas conocidas como la 'Profundidad de la herencia', 'Complejidad del código' o el 'Grado de mantenibilidad', muy ligadas a los lenguajes OO.

Hasta el presente, la mayoría de los desarrollos que involucraban el acceso a una fuente de datos como Microsoft SQL Server, Oracle u otro motor relacional no hacían uso de estos conceptos y/o métricas de calidad, debido a la estrecha ligazón de los modelos relacionales con el código que los utiliza. Esta relación propicia que, en demasiadas ocasiones, el desarrollador no pueda exponer realmente todas sus capacidades y las de los lenguajes en los que se apoya para su trabajo.

Imagine que en una misma sala juntáramos a un profesional de bases de datos y a un desarrollador de un lenguaje OO, y a ambos se les expusiera un mismo problema con el fin de que trataran de resolverlo. Si, una vez terminado el ejercicio, nos planteáramos revisar los resultados, nos daríamos cuenta de las diferentes técnicas y razonamientos que ambos habrían aplicado. Por un lado, el desarrollador habría aplicado algunos, o muchos, de los conceptos mencionados anteriormente, como la herencia de clases o el polimorfismo; mientras que el profesional de bases de datos habría hecho uso de otros conceptos, tales como la normalización o desnormalización de tablas, el uso de tipos adecuados, índices y otros elementos familiares en ese mundo.

Desajuste de impedancias

Estas diferencias entre los modelos relacionales y modelos de objetos conforman lo que se conoce habitualmente como 'desajuste de impedancias' (*impedance mismatch*). Existen muchas diferencias entre ambos mundos, algunas muy obvias como, por ejemplo, la diferencia en el concepto de **relación** en el mundo relacional y en un mundo orientado a objetos. En los modelos relacionales, las relaciones se establecen mediante la duplicación de los datos en distintas tablas, de tal forma que, si deseamos relacionar un tupla de una tabla B con una tupla de una tabla A, deberemos establecer en la tabla B una columna que contenga un valor que permita identificar al elemento de la tabla A con el que la queremos relacionar. Sin embargo, en los lenguajes de programación orientados a objetos como C# o Visual Basic las relaciones no necesitan apoyarse en la duplicidad de datos; por seguir con el ejemplo, bastaría con poner una referencia en el objeto B al objeto A con el que se desea establecer una relación, es decir, una asociación.

Salvar este desajuste de impedancias ha sido uno de los objetivos centrales de múltiples teorías y tecnologías a lo largo de estas últimas décadas.

Diseños guiados por modelos de dominio

Domain Driven Design (DDD)
Este término lo acuñó por primera vez **Eric Evans**, en su libro "Domain Driven Design: Tackling Complexity in the Heart of Software" (Addison-Wesley, 2004).

A lo largo de los últimos años, la comunidad de desarrolladores ha ido creando y evolucionando una filosofía de trabajo centrada en la definición de modelos conceptuales y dominios de trabajo, en la que el reto que realmente se intenta resolver consiste en separar el problema de las distintas técnicas y argucias de desarrollo habituales para los programadores. Los diseños guiados por modelos de dominio proponen centrarse en el modelo conceptual o dominio de trabajo para resolver el problema. En realidad, DDD no puede considerarse como una metodología en sí misma, ni por supuesto como un tipo de tecnología; no es más que una manera de pensar, una forma de abordar el desarrollo de los proyectos que tiene como prioridad el problema a resolver. Es curioso como esta verdad tan simple muchas veces queda relegada a un segundo plano cuando hablamos de tecnología.

Definición de modelo
Perdónenos, estimado lector, pero la definición de modelo tiene tantas acepciones, que no sabríamos decantarnos por una única de ellas. En nuestro escenario, un **modelo** representa un sistema de abstracciones creado a partir de un sistema real que puede tener distintas presentaciones, código, diagramas, etc.

Con el fin de apoyar estas ideas, desde la comunidad de desarrolladores empezaron a surgir nuevos patrones de diseño que se fundamentan en los modelos conceptuales, o que tienen como objetivo resolver problemas habituales cuando se empieza a aplicar esta forma de pensar para realizar nuestros trabajos de desarrollo. Algunos de estos patrones se presentan a continuación, puesto que serán de gran ayuda para entender muchos de los aspectos esenciales de EF.

Patrones en diseños guiados por dominios

Por supuesto, esta sección no pretende abordar todos los patrones de diseño conocidos y usados en el desarrollo de aplicaciones; ni siquiera presentaremos estos patrones de una forma académica. Nuestro objetivo aquí consiste única y exclusivamente en presentar y explicar de forma somera algunos de los patrones de diseño más habituales, los cuales probablemente ya conocerá, y que están muy ligados con EF.

Value Object

El patrón Value Object, también conocido con otros nombres, como Data Transfer Object (DTO), es uno de los patrones más conocidos y usados en arquitecturas de múltiples capas. Martin Fowler, en su libro "Patterns Of Enterprise Application Architecture" (Addison-Wesley, 2002), define un DTO como "un objeto que transporta datos entre procesos con el fin de reducir el número de llamadas". Cuando trabajamos sobre sistemas distribuidos, los costes asociados a las llamadas suelen ser bastante altos, por lo que uno de los objetivos principales es reducir el número de llamadas que se realizan a esos sistemas. Un DTO permite mantener una colección

de datos, más o menos relacionados, y transportarlos de una vez en las llamadas a las interfaces remotas, por lo que debe de ser un objeto que se pueda seriar. Seguramente, si usted conoce o ha trabajado alguna vez con ADO.NET, podrá identificar rápidamente a `DataSet` o `DataTable` como clases típicas para objetos DTO.

Uno de los principales inconvenientes de los objetos DTO dentro de los modelos de dominio suele estar relacionado con la transferencia de estos objetos, puesto que, en la mayoría de ocasiones, éstos estarán conectados mediante asociaciones a otros objetos que también deberían seriarse. Por otro lado, los programadores acostumbran a compartir la capa de negocio con los clientes de los sistemas remotos, de tal forma que, equivocadamente, estos objetos no solamente son contenedores de datos, sino que además incluyen cierta lógica de negocio que en teoría no deberían incorporar.

De forma usual, los campos de un DTO son tipos primitivos (**`string`**, **`int`**, `DateTime`, etc.) u otros DTO (formando lo que se conoce como **tipos complejos**), que permiten representar la estructura de los datos que deseamos transportar.

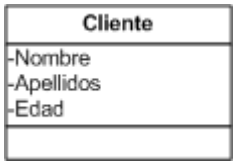

Figura 1: Value Object

Lazy Loading

El patrón Lazy Loading (literalmente, "carga retardada") es otro de los patrones más utilizados cuando hablamos de modelos conceptuales. Lazy Loading permite retardar la carga de un determinado objeto o colección de objetos (carga que suele ser costosa) justamente hasta el momento en el que el objeto o la colección de objetos son necesarios. Este patrón contribuye, sin duda alguna, a mejorar la eficiencia en las operaciones de programación habituales.

Imagine que usted ha construido dos DTO para representar las estructuras de datos `Cliente` y `Pedido`, en las que ha incluido una asociación dentro del DTO `Cliente` con una colección de pedidos; es decir, ha establecido una relación `1:N` entre `Cliente` y `Pedido`. Si cargáramos de la base de datos los datos de un determinado cliente, tendríamos dos opciones: por un lado, cargar simultáneamente todos sus datos y los de los pedidos asociados con él (*eager loading*); por el otro, solamente cargar los datos del cliente, sin cargar por el momento sus pedidos (*lazy loading*). Esta última opción suele ser lo más deseable en la mayoría de las ocasiones, puesto que en muchas operaciones solamente necesitaremos tener en cuenta los datos propios del cliente. Llegado el momento en el que los pedidos del cliente sean

necesarios, se dispondrá de un método que permita cargarlos y ponerlos a disposición de quien los desea usar.

Veremos, a lo largo de este libro, como EF dispone de mecanismos para realizar cargas retardadas (la opción por defecto), o indicar la carga instantánea de relaciones. Nuevamente, recomendamos leer a Martin Fowler para profundizar más sobre este conocido patrón de diseño.

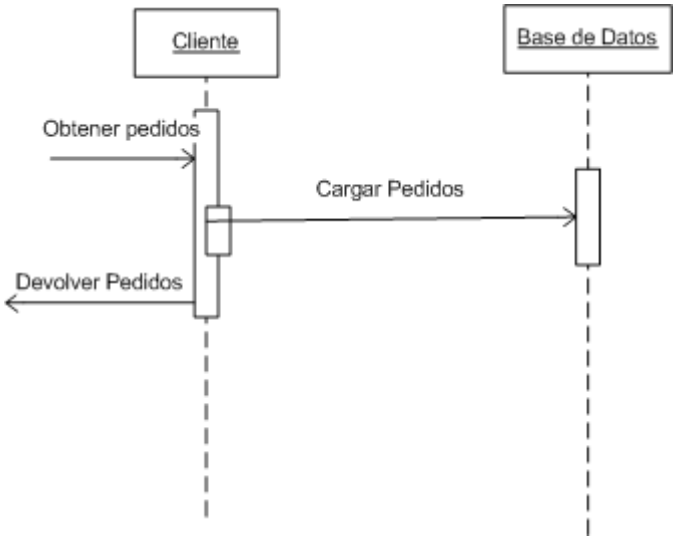

Figura 2: Lazy Loading

Data Mapper

Los objetos y las bases de datos relacionales disponen de diferentes mecanismos para estructurar los datos. Muchos de los conceptos relacionados con los objetos como herencia, polimorfismo y/o colecciones no tienen una representación directa en los modelos relacionales. Cuando se construye un modelo de objetos para resolver un determinado problema, es totalmente lógico usar las capacidades anteriores; sin embargo, esto introduce una gran complejidad cuando se intenta "cuadrar" los esquemas de objetos con los esquemas de los modelos relacionales. El patrón Data Mapper tiene como objetivo separar las estructuras de los objetos de las estructuras de los modelos relacionales y realizar la transferencia de datos entre ambos. Con el uso de un Data Mapper, los objetos son ignorantes del esquema presente en la base de datos y, por supuesto, no necesitan hacer uso de código SQL.

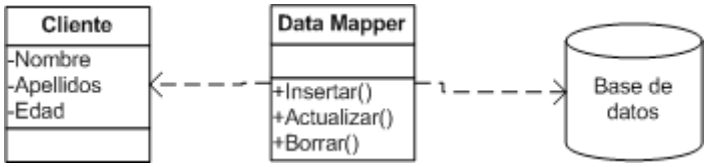

Figura 3: Data Mapper

Unit of Work

Cuando se trabaja con datos fuera de una base de datos mediante objetos DTO, es importante, por no decir esencial, poder llevar un seguimiento de las operaciones que se realizan sobre estos datos, cuándo se cambian los valores de algunas de sus propiedades, cuándo se elimina o inserta algún DTO, etc. Por supuesto, se podría hacer una llamada a la base de datos cada vez que se realiza una operación sobre un determinado dato; pero esto al final provocaría un exceso de llamadas a la base de datos, que ralentizarían la ejecución normal del sistema. El patrón Unit of Work permite mantener y llevar el seguimiento de todas las operaciones que se realizan contra una estructura de datos en un determinado proceso de negocio, para que una vez acabado el proceso de negocio se puedan volcar los resultados de todas las operaciones realizadas a la base de datos.

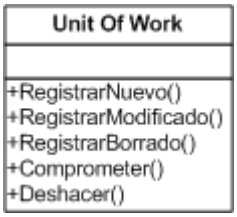

Figura 4: Unit of Work

PI, POCO y otras ideas

Hasta este momento, hemos visto como DDD nos propone centrarnos en un dominio conceptual como principal punto de partida para resolver los problemas; además, hemos hecho una presentación somera de algunos patrones de diseño muy relacionados con DDD, como Value Object (fíjese que éstos son los objetos de trabajo de los dominios conceptuales), Data Mapper y Unit of Work. Estos tres patrones tienen también mucho que ver con otro concepto conocido como PI (Persistence Ignorance, o Ignorancia de la persistencia), el cual propugna el trabajo con objetos DTO que solamente conozcan del dominio conceptual en el que están creados y para nada tengan que saber sobre el almacenamiento subyacente; lo que ayuda a centrarse realmente en los problemas, sin tener en cuenta artefactos propios de una determinada tecnología. Hacer que nuestros objetos DTO sean ignorantes de la tecnología, y de una determinada clase o interfaz base, es lo que se conoce como la construcción de

objetos POCO (Plain Old CLR Objects), concepto de sobra conocido en marcos de trabajo como NHibernate, así como en otras plataformas de desarrollo como Java, donde estos objetos se denominan POJO (Plain Old Java Objects).

El trabajo con objetos POCO es realmente una tarea ardua y que supone algún sacrificio en desarrollo, ya que al no poder implementar clases o interfaces base o hacer uso de una tecnología concreta, tampoco puede aprovecharse aquello que todos esos elementos nos aportan. Algunos autores relajan los requisitos de un objeto POCO permitiendo que éstos puedan implementar algún tipo de contrato o interfaz; esta relajación se conoce como IPOCO.

En la primera versión de EF no tenemos la posibilidad de trabajar con objetos POCO, aunque es una promesa del grupo de producto que esta posibilidad sea viable a partir de la segunda versión. No obstante, en esta primera versión sí podemos trabajar con objetos IPOCO, aunque ésta no es la opción por defecto y deberemos ser nosotros los que realicemos la construcción de los objetos y la implementación de las interfaces para que éstos puedan trabajar con el motor del Marco de entidades. Por defecto, EF se decanta por otra filosofía, conocida en la teoría como Prescriptive Classes (Clases prescriptivas), y genera objetos que heredan de una clase base predeterminada, que en el caso de EF se llama `EntityObject`.

¿QUÉ ES EL MARCO DE ENTIDADES?

El Marco de entidades de ADO.NET (ADO.NET Entity Framework, EF) es un marco de trabajo para la plataforma .NET que permite superponer varias capas de abstracción sobre el almacén relacional con el fin de hacer posible una programación más conceptual (basada en los conceptos del dominio con el que se trabaja) y de reducir a una mínima expresión el desajuste de impedancias causado por las diferencias entre los modelos de programación relacional y orientado a objetos.

Por una parte, es de destacar que EF es una parte integral de ADO.NET a partir de .NET Framework 3.5 SP1. Más exactamente, EF incluye un nuevo proveedor de ADO.NET, llamado **Entity Client**, que habilita el acceso a los modelos conceptuales. A lo largo de este libro, iremos viendo la uniformidad conceptual entre este proveedor y el resto de los proveedores de ADO.NET. Por otra parte, como todo marco de trabajo, EF incluye dos componentes fundamentales:

- **Recursos para el entorno de desarrollo**, y en particular un asistente para el diseño visual de modelos de entidades dentro de Visual Studio y la generación de código a partir de éstos.
- **Librería**. Los tipos que componen el Marco de entidades se implementan físicamente en el ensamblado `System.Data.Entity.dll`. La organización lógica de esos tipos es tal, que sus espacios de nombres se anidan dentro del espacio `System.Data`, como se muestra en la tabla a continuación. Todo esto refuerza la idea de la pertenencia de EF a la familia de ADO.NET.

Espacio de nombres	Contenido añadido por EF
`System.Data`	Tipos básicos de EF, como `EntityObject`, `EntityKey` o `EntityState`.
`System.Data.Common`	Tipos relacionados con la implementación de proveedores de EF.
`System.Data.Common.CommandTrees`	Tipos que se utilizan en los árboles de expresiones de LINQ to Entities.
`System.Data.EntityClient`	Implementación del proveedor ADO.NET de EF. Contiene tipos cuyos nombres le resultarán muy familiares, como `EntityConnection`, `EntityCommand` y `EntityDataReader`.
`System.Data.Mapping`	Tipos que se utilizan en el mapeo entre entidades lógicas y físicas.
`System.Data.Metadata.Edm`	Tipos que permiten acceder a los metadatos (definiciones) de los modelos.
`System.Data.Objects`	Tipos que implementan el modelo de programación que hace posible un trabajo basado en objetos contra datos provenientes de una base de datos relacional. En particular, en este espacio de nombres reside la clase `ObjectQuery<T>`, que implementa `IQueryable<T>` y por tanto sirve como origen para las consultas integradas sobre modelos de entidades (para una introducción a LINQ, consulte el Apéndice).
`System.Data.Objects.Classes`	Tipos que representan los principales conceptos que se utilizan en los modelos. Normalmente, no es necesario interactuar directamente con las clases de este espacio, pues el generador de código crea versiones más específicas de ellas.

Tabla 1: Espacios de nombres relacionados con EF

Arquitectura y componentes

EF se apoya en seis elementos fundamentales construidos encima de ADO.NET 2.0, tal y como se puede apreciar en la siguiente figura.

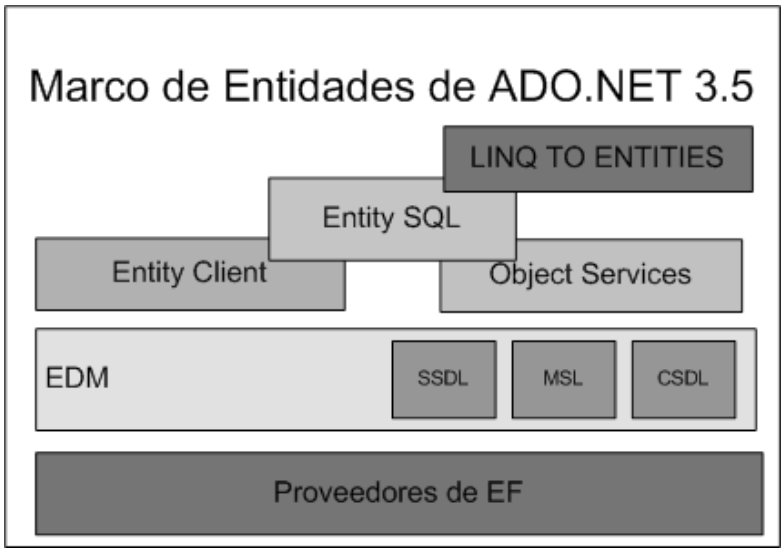

Figura 5: Componentes de la arquitectura de EF

A continuación presentamos los conceptos fundamentales relacionados con cada uno de estos elementos (de abajo hacia arriba), entrando más profundamente en cada uno de ellos en los posteriores capítulos que acompañan este libro.

Proveedores específicos de EF

Una de las características más atractivas de EF es su 'agnosticismo' con relación a la base de datos contra la que se trabaja. Tal y como hemos comentado anteriormente, EDM se encarga de implementar el patrón Data Mapper entre las entidades de nuestro modelo conceptual y el esquema físico de la base de datos subyacente. Más adelante veremos cómo el trabajo que se realiza sobre estas entidades es traducido directamente al dialecto específico de la base de datos con la que estemos trabajando, gracias a los **proveedores de datos de EF**. Por defecto, en la primera versión de esta tecnología tendremos un proveedor específico para SQL Server, en sus versiones 2000, 2005 y 2008, ésta última con alguna limitación para los nuevos tipos espaciales introducidos. Además de estas bases de datos, podremos trabajar también con SQL Server Compact Edition, en versión 3.5 SP1 o superior.

Nota:

Si no conoce SQL Server Compact Edition y tiene interés en saber qué es y cómo funciona esta base de datos "in process", le recomendamos la lectura del libro de **José M. Torres** titulado "SQL Server 2008 Compact Edition. Aplicaciones *smart-client* para escritorio y dispositivos móviles", publicado por esta misma editorial.

feed your brain®

campus MVP

- ☑ Sin tener que desplazarte
- ☑ Sin romper tu ritmo de trabajo
- ☑ Preguntándole a los que más saben

infórmate ya

902 876 475 www.campusmvp.com

krasis

Microsoft GOLD CERTIFIED Partner | Learning Solutions Custom Development Solutions

> “[...] me alegra ver que alguien pueda escribir de una forma tan accesible, a la vez que precisa, sobre Entity Framework.”
>
> **Diego Vega**
> Program Manager, Entity Framework Team
> Microsoft Corporation

Este libro es la primera obra que se publica en el mundo sobre ADO.NET Entity Framework, el futuro del acceso a datos. Entity Framework superpone varias capas de abstracción por encima de los almacenes de datos, de forma que nos permite trabajar con los conceptos de nuestra aplicación y olvidarnos de las diferencias entre los modelos de programación relacional y orientada a objetos.

La obra consta de seis capítulos más cinco apéndices, que cubren ampliamente la gran mayoría de las características de Entity Framework:

- El libro comienza con un capítulo de introducción a la tecnología, para luego centrarse en la creación de modelos conceptuales de entidades, que describe desde los apartados más básicos hasta las posibilidades más avanzadas por medio de ejemplos claros y prácticos.

- Los siguientes capítulos introducen al lector en las distintas posibilidades que la tecnología ofrece para consultar y actualizar los datos de un modelo conceptual, todo ello de una forma ordenada, tal y como se fueron concibiendo los distintos subsistemas que componen Entity Framework, y explicando las ventajas e inconvenientes de cada uno de ellos.

- Para finalizar, el último capítulo del libro ofrece una introducción a ADO.NET Data Services, una aplicación práctica de Entity Framework desarrollada por Microsoft para exponer la infraestructura que conforma esta tecnología a través de servicios REST.

Este libro, aparte de ser una herramienta de aprendizaje, se convertirá en un manual de referencia que siempre querrá tener a mano.

NIVEL - Intermedio/Avanzado

MVP
Microsoft® Most Valuable Professional

campus MVP
www.campusmvp.com

krasis PRESS
www.krasis.com