REVISTA DIGITAL DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES
TELEM@TICA
Revista Telem@tica. Vol. 10. No. 3, septiembre-diciembre, 2011, p. 1-7 ISSN 1729-3804

Mapeo Objeto / Relacional (ORM)

Osmel Yanes Enriquez[1], Hansel Gracia del Busto[2]

[1] Dirección de Servicios TIC (DISERTIC), CUJAE. Ingeniero, yanes@tesla.cujae.edu.cu

[2] Dirección de Servicios TIC (DISERTIC), CUJAE. Ingeniero, hansel@tesla.cujae.edu.cu

Resumen / *Abstract*

Al desarrollar una aplicación siguiendo el paradigma orientado a objetos, los programadores se enfrentan, entre otros desafíos, al problema de la persistencia de los datos, debido a las diferencias que existen entre el modelo relacional y el modelo orientado a objetos. Este problema la mayoría de los casos es solucionado con la ayuda de ciertas herramientas que se encargan de generar de manera automática el acceso a datos, abstrayendo al programador de este problema.

Developing applications following the object-oriented paradigm makes programmers face, among other challenges, the problem of data persistence, due to differences between the relational model and object-oriented model. This problem most of the cases is solved with the help of certain tools that handle automatically data access generation, abstracting the programmer from this problem.

Introducción

El Modelo Relacional es un modelo de datos basado en la lógica de predicado y en la teoría de conjuntos para la gestión de una base de datos. Siguiendo este modelo se puede construir una base de datos relacional que no es más que un conjunto de una o más tablas estructuradas en registros (filas) y campos (columnas), que se vinculan entre sí por un campo en común. Sin embargo, en el Modelo Orientado a Objetos en una única entidad denominada objeto, se combinan las estructuras de datos con sus comportamientos. En este modelo se destacan conceptos básicos tales como objetos, clases y herencia.

Entre estos dos modelos existe una brecha denominada desajuste por impedancia dada por las diferencias entre uno y otro. Una de las diferencias se debe a que en los sistemas de bases de datos relacionales, los datos siempre se manejan en forma de tablas, formadas por un conjunto de filas o tuplas; mientras que en los entornos orientados a objetos los datos son manipulados como objetos, formados a su vez por objetos y tipos elementales.

La gran mayoría de los lenguajes de programación como java o C, tienen un modelo de manejo de datos basado en leer, escribir o modificar registros de uno en uno. Por ello, cuando se invoca el lenguaje de consulta SQL (Standard Query Language) desde un lenguaje de programación es necesario un mecanismo de vinculación que permita recorrer las filas de una consulta a la base de datos y acceder de forma individual a cada una de ellas [3].

Además en el modelo relacional no se puede modelar la herencia que aparece en el modelo orientado a objetos y existen también desajustes en los tipos de datos, ya que los tipos y denotaciones de tipos asumidos por las consultas y lenguajes de programación difieren. Esto concierne a tipos atómicos como integer, real, boolean, etc. La representación de tipos atómicos en lenguajes de programación y en bases de datos pueden ser significativamente diferentes, incluso si los tipos son denotados por la misma palabra reservada, ej.: integer. Esto ocurre también con tipos complejos como las tablas, un tipo de datos básico en SQL ausente en los lenguajes de programación.

Para atenuar los efectos del desajuste por impedancia entre ambos modelos existen varias técnicas y prácticas como los Objetos de Acceso a Datos (Data Acces Objects o DAOs), marcos de trabajo de persistencia (Persistence Frameworks), mapeadores Objeto/Relacionales (Object/Relational Mappers u ORM), consultas nativas (Native Queries), lenguajes integrados como PL-SQL de Oracle y T-SQL de SQL Server; mediadores, repositorios virtuales y bases de datos orientadas a objetos [4].

Las soluciones al problema de la impedancia mencionadas anteriormente presentan ventajas y desventajas que deberán ser evaluadas según las características y requisitos del sistema a desarrollar. En el presente artículo se propone el uso de las herramientas de mapeo Objeto/Relacional como una alternativa a este problema.

Mapeo Objeto/Relacional

El mapeo objeto-relacional es una técnica de programación para convertir datos del sistema de tipos utilizado en un lenguaje de programación orientado a objetos al utilizado en una base de datos relacional. En la práctica esto crea una base de datos virtual orientada a objetos sobre la base de datos

relacional. Esto posibilita el uso de las características propias de la orientación a objetos (esencialmente la herencia y el polimorfismo).

Las bases de datos relacionales solo permiten guardar tipos de datos primitivos (enteros, cadenas de texto, etc.) por lo que no se pueden guardar de forma directa los objetos de la aplicación en las tablas, sino que estos se deben de convertir antes en registros, que por lo general afectan a varias tablas. En el momento de volver a recuperar los datos, hay que hacer el proceso contrario, se deben convertir los registros en objetos. Es entonces cuando ORM cobra importancia, ya que se encarga de forma automática de convertir los objetos en registros y viceversa, simulando así tener una base de datos orientada a objetos [2].

Entre las ventajas que ofrecen los ORM se encuentran: rapidez en el desarrollo, abstracción de la base de datos, reutilización, seguridad, mantenimiento del código, lenguaje propio para realizar las consultas. No obstante los ORM traen consigo algunas desventajas como el tiempo invertido en el aprendizaje. Este tipo de herramientas suelen ser complejas por lo que su correcta utilización requiere un espacio de tiempo a emplear en conocer su funcionamiento adecuado para posteriormente aprovechar todo el partido que se le puede sacar. Otra desventaja es que las aplicaciones suelen ser algo más lentas. Esto es debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.

En el mapeo objeto-relacional encontramos el uso de algunos patrones de diseño como el Repository y el Active Record.

Patrón Repository

El patrón Repository utiliza un repositorio para separar la lógica que recupera los datos y los mapea al modelo de entidades, de la lógica del negocio que actúa en el modelo. El repositorio media entre la capa de fuente de datos y la capa de negocios de la aplicación; encuesta a la fuente de datos, mapea los datos obtenidos de la fuente de datos a la entidad de negocio y persisten los cambios de la entidad de negocio a la fuente de datos. En la figura 1 se muestra un esquema patrón Repository.

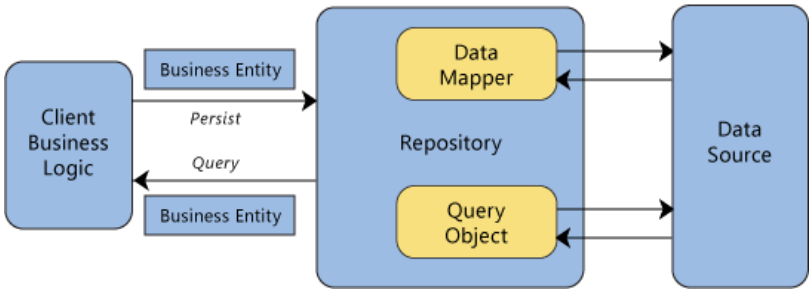

Figura 1.

Los repositorios son puentes entre los datos y las operaciones que se encuentran en distintos dominios. Un repositorio elabora las consultas correctas a la fuente de datos y mapea los resultados a las entidades de negocio expuestas externamente. Los repositorios eliminan las dependencias a tecnologías específicas proveyendo acceso a datos de cualquier tipo [1].

El patrón de diseño Repository puede ayudar a separar las capas de una aplicación web ASP.NET ya que provee una arquitectura de 3 capas separadas, lo que mejora el mantenimiento de la aplicación y ayuda a reducir errores. Además facilita las pruebas unitarias.

Patrón Active Record

Active Record es un patrón en el cual, el objeto contiene los datos que representan a una fila (o tupla) de nuestra tabla o vista, además de encapsular la lógica necesaria para acceder a la base de datos. De esta forma el acceso a datos se presenta de manera uniforme a través de la aplicación (lógica de negocio + acceso a datos en una misma clase). En la figura 2 se muestra un esquema del patrón Active Record.

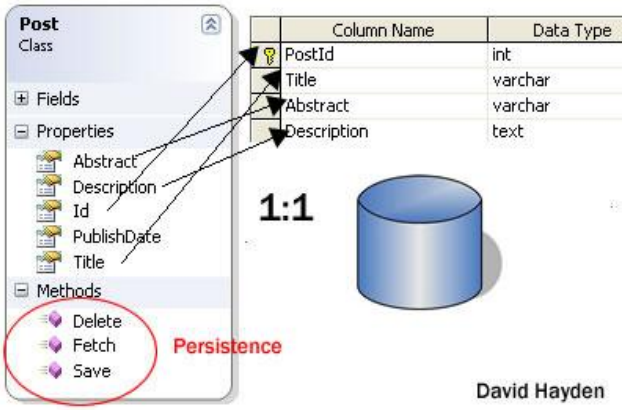

Figura 2.

Una clase Active Record consiste en el conjunto de propiedades que representa las columnas de la tabla más los típicos métodos de acceso como las operaciones CRUD (Create, Read, Update, Delete), búsqueda (Find), validaciones, y métodos de negocio [5].

Este patrón constituye la aproximación más obvia, poniendo el acceso a la base datos en el propio objeto de negocio. De este modo es evidente como manipular la persistencia a través de él mismo. Gran parte de este patrón viene de un Domain Model y esto significa que las clases están muy cercanas a la representación en la base de datos. Cada Active Record es responsable de sí mismo, tanto en lo relacionado con persistencia como en su lógica de negocio [6].

SubSonic y NHibernate

Las herramientas ORM (Object-Relational Mapping) se utilizan para convertir los datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, a través de un motor de persistencia. Esto se traduce en que las herramientas ORM permiten almacenar objetos en una base de datos de forma permanente, y extraerlos cuando sea necesario.

Subsonic es una herramienta ORM libre, de código abierto creada por Rob Conery. Inspirado por Ruby on Rails. Subsonic toma un acercamiento minimalista al código y enfatiza convención sobre configuración. Aunque se inspira en Ruby on Rails, no es una parte de él, sino que toma las ideas de Ruby on Rails y las adapta a la plataforma ya existente ASP.NET. [7]

A diferencia de otros ORM, SubSonic usa el método de la generación y compilación del nivel de acceso de datos en lugar de realizar una asignación basada en reflexión en el tiempo de ejecución. Al igual que muchos ORM, SubSonic soporta los sistemas de bases de datos más comunes, entre ellos: SQL Server, MySQL, SQLite, Oracle, IBM DB2, PostgreSQL, SQL CE, VistaDB y también hace muy fácil la migración del acceso a datos entre estos [8].

NHibernate es la conversión de Hibernate de lenguaje Java a C# para su integración en la plataforma .NET. Al igual que Hibernate, NHibernate es una herramienta de ORM que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML). Hibernate fue una iniciativa de un grupo de desarrolladores dispersos alrededor del mundo conducido por Gaving King.

Al usar NHibernate para el acceso a datos el desarrollador se asegura de que su aplicación es agnóstica en cuanto al motor de base de datos a utilizar en producción, pues NHibernate soporta los más habituales en el mercado: MySQL, PostgreSQL, Oracle, MS SQL Server, etc. Sólo se necesita cambiar una línea en el fichero de configuración para que podamos utilizar una base de datos distinta.

Ambas herramientas tienen sus ventajas y desventajas, pero generalmente la selección de uno u otro es una cuestión de preferencia personal.

CONCLUSIONES

El uso de un ORM es una alternativa sumamente efectiva a la hora de trasladar el modelo conceptual (orientado a objetos) al esquema relacional nativo de las bases de datos SQL. Evita la inclusión de sentencias SQL embebidas en el código de la aplicación, lo que a su vez facilita la migración hacia otro sistema gestor de bases de datos. Incorpora una capa de abstracción entre el modelo relacional físico y la capa de negocios de la aplicación. Al ser realizado, en esta capa, de manera automática la conversión de instrucciones orientadas a objetos, a sentencias SQL, minimiza la ocurrencia de errores humanos.

De cualquier modo utilizar un ORM no debe ser considerado una panacea, sino que debe usarse a discreción; teniendo en cuenta las particularidades de cada problema a modelar. En determinados casos no es recomendable el uso de un ORM, sobre todo cuando se imponen tiempos de respuesta mínimos o se requiere una menor sobrecarga. En estos casos lo más conveniente es el uso de un microORM; evitando siempre que sea posible las inyecciones de SQL Inline.

Lo anteriormente expuesto libera a los desarrolladores de aplicaciones de la responsabilidad de conocer las múltiples variantes de SQL que existen en función del gestor de bases de datos que se utilice. No obstante, en escenarios en que se necesite hacer un uso más eficiente del sistema de almacenamiento de información, personalizado de acuerdo a las necesidades de la aplicación, y se escoja como gestor de bases de datos una variante NoSQL no es necesaria la utilización de un ORM.

En resumen, en dependencia del gestor de bases de datos a emplear, se recomienda siempre que sea posible y sea factible la utilización de un ORM.

REFERENCIAS

[1] MSDN, The Microsoft Developer Network. Local Help, Visual Studio 2008. Disponible en: http://msdn.microsoft.com/en-us/

[2] Introducción a Object-Relational Mapping (ORM). Iván Guardado. Mayo 2010. Disponible en: http://web.ontuts.com/tutoriales/introduccion-a-object-relational-mapping-orm/

[3] Introducción al Desarrollo de Aplicaciones con Bases de Datos. Alarcos. Departamento de Tecnologías y Sistemas de Información de la Universidad de Castilla. Enero 2007. Disponible en: http://alarcos.esi.uclm.es/doc/aplicabbdd/Documentos/teoria/introducci%C3%B3n%20al%20desarrollo%20aplicaciones%20con%20bases%20de%20datos.pdf

[4] Impedance Mismatch. Kazimierz Subieta. Enero 2008. Disponible en: http://www.sbql.pl/Topics/ImpedanceMismatch.html

[5] Patrones de Acceso a Datos: Active Record. Grimpi IT. Febrero 2009. Disponible en: http://grimpidev.wordpress.com/2008/11/22/patrones-de-acceso-a-datos-active-record/

[6] Active Record. Néstor Salceda. Septiembre 2009. Disponible en: http://es.debugmodeon.com/articulo/active-record

[7] Getting Started with SubSonic. Scott Kuhl. Noviembre 2006. Disponible en:

http://geekswithblogs.net/scottkuhl/archive/2006/11/16/97298.aspx

[8] Autocompletado del código SQL, herramientas de subversión, desarrollo ágil, y más. James Avery. Febrero 2008. Disponible en: http://msdn.microsoft.com/es-es/magazine/cc164246.aspx