Revista EIA, ISSN 1794-1237 Número 9, p. 77-90. Julio 2008
Escuela de Ingeniería de Antioquia, Medellín (Colombia)

DIRECTRICES PARA LA CONSTRUCCIÓN DE ARTEFACTOS DE PERSISTENCIA EN EL PROCESO DE DESARROLLO DE SOFTWARE

Juan Bernardo Quintero*
Diana María Hernández**
Andrea Yanza***

RESUMEN

El almacenamiento de datos es uno de los procesos que se halla en casi la totalidad de los sistemas de información. El tratamiento que se les da a las bases de datos durante el desarrollo de software resulta un factor determinante para el buen desempeño de un sistema. La acelerada evolución tecnológica y el desarrollo de la industria del software han conducido al surgimiento de diversas estrategias de modelado y numerosas técnicas de acceso a datos. En este trabajo se precisan los referentes conceptuales y prácticos para la construcción de los productos relacionados con las bases de datos, el acceso a estas y el tratamiento que se les da en la industria a los llamados artefactos de persistencia.

PALABRAS CLAVE: sistema gestor de bases de datos; modelo entidad-relación; mapeo objeto-relacional; patrones; frameworks.

ABSTRACT

Data storage is a process required in almost every information system nowadays; the treatment given to databases during the software development can be a determining factor in order to achieve good performance of a resulting system. The accelerated technological evolution and the progress of the software

* Magíster en Ingeniería Informática, Universidad EAFIT. Jefe de Sistemas, ABC-Flex Ltda., Medellín, Colombia. Docente de Cátedra, Universidad de Antioquia y Universidad EAFIT. jquinte1@eafit.edu.co

** Ingeniera de Sistemas, Universidad de Antioquia. Gerente General, ABC-Flex Ltda. Medellín, Colombia. dianahdez@abcflex.net

*** Magíster (c) en Ingeniería Informática y Docente de Cátedra, Universidad EAFIT. Medellín, Colombia. Integrante del Grupo de Investigación en Ingeniería de Software. ayanza@eafit.edu.co

Artículo recibido 26-III-2008. Aprobado 24-VII-2008
Discusión abierta hasta diciembre de 2008

enterprises have given rise to diverse modeling strategies and several data access techniques. This work explains the practical and conceptual references for the construction of products related to databases, the access techniques to them and the handling that so-called persistence artifacts have at an industrial level.

KEY WORDS: database manager system, entity-relationship model, object-relational mapping, patterns, *frameworks*.

1. INTRODUCCIÓN

Las bases de datos han desempeñado papel protagónico en el proceso de desarrollo de software, por esto en la actualidad se encuentra un número considerable de técnicas de modelado, patrones, *frameworks* y librerías para trabajar lo concerniente al almacenamiento de datos. Los artefactos de persistencia se refieren a los productos construidos durante el proceso de desarrollo de software, permitiendo a los sistemas de información trascender los datos en el tiempo, almacenándolos en bases de datos o archivos.

Durante las décadas de los setenta y ochenta se dio un considerable fortalecimiento de los motores de bases de datos, también llamados SGBD (sistema gestor de bases de datos) o DBMS (*DataBase Management System*), apareció por entonces el llamado paradigma relacional, que puso los datos como el elemento central del proceso de desarrollo de software [1], fue cuando los SGBD evolucionaron hasta ser llamados SGBD relacionales (RDBMS: *Relational DataBase Management System)*.

Sin embargo, en los noventa, el auge de la programación orientada por objetos y el surgimiento de los *frameworks* de mapeo objeto-relacional ocasionó que los desarrolladores de software volcaran su atención sobre otros elementos como los objetos, los componentes o los *frameworks,* los cuales por medio de diferentes estrategias les facilitaban considerablemente las labores, al punto de llevar a casi ignorar la existencia del SGBD. Los artefactos de persistencia se convirtieron en productos que solo eran revisados o modificados cuando el aumento en los volúmenes de información y el acceso concurrente a ellos causaban problemas de rendimiento.

Para concretar la guía planteada en este artículo, se abordan algunos de los principales conceptos del cuerpo del conocimiento de la ingeniería de software: notación, metodología, técnica y herramientas. Las notaciones para el modelado de datos se trabajan en la sección 2, lo referente a la metodología se trata en la sección 3, las técnicas de acceso a datos se abordan en la sección 4, las herramientas más populares para el desarrollo con bases de datos se revisan en la sección 5, y finalmente se muestran las conclusiones y trabajos futuros en la sección 6.

Para ilustrar esta guía, se presenta un caso de aplicación que trata el desarrollo de un portal temático llamado Aldea Contable, aplicación web para apoyar el proceso docente del Departamento de Ciencias Contables de la Universidad de Antioquia.

2. NOTACIONES PARA EL MODELADO DE DATOS

Para construir modelos de datos existen diversas notaciones, las cuales le dan diferente relevancia a cada uno de los elementos que se pueden representar en los modelos. Esta situación hace que una notación específica sea más útil en un contexto que en otro, de acuerdo con las condiciones particulares del proyecto o del grupo de desarrollo.

Aunque en la industria la notación que se utilice para modelar las bases de datos no es un factor que esté determinado por las condiciones del proyecto o del grupo de desarrollo, utilizar una u otra notación puede tener un impacto considerable. A continuación se enuncian los principales aspectos de las siete notaciones más usadas en el modelado de bases de datos.

- **Notación Chen:** esta fue la notación que se utilizó inicialmente para la representación de bases de datos. Fue planteada por Peter Chen en los setenta y les da un especial énfasis a las relaciones entre las entidades representándolas con un rombo en el que se pone el nombre de la relación [2].
- **Notación Barker:** fue adoptada por *Oracle Corporation* en sus productos de modelado de datos; goza de mucha popularidad y sirve de referente en la metodología planteada en *CASE*Method* [3].
- **IE** *(Information Engineering):* desarrollada inicialmente por Clive Finkelstein quien luego la refinó con el apoyo de James Martin; aunque es clara e intuitiva, sirve solo para modelos de alto nivel de abstracción (modelos lógicos), pues no permite modelar los atributos de las entidades [4].
- **IDEF1X** *(Data Modeling):* desarrollada por el Departamento de Defensa de los Estados Unidos dentro de una familia de estándares llamada IDEF *(Integrated DEFinition Method);* no obstante que fue planteada para representar modelos físicos, erróneamente se usa en modelos lógicos; para este tipo de modelos es más apropiada la notación de IDEF1 llamada *Information Modeling* [5].
- **ORM** *(Object Role Modeling):* formalizada por Terry Halpin, es una notación muy intuitiva que se centra en representar el papel que desempeña una entidad con respecto a las demás entidades con las que está relacionada, su desventaja radica en que los modelos crecen en tamaño muy rápidamente [6].
- **UML** *(Unified Modeling Language):* si bien es un lenguaje de modelado objetual, se puede extender a través de perfiles para soportar otro tipo de modelos [7]. En la actualidad existen varias propuestas e iniciativas para perfiles de modelado de datos con UML [8-10].
- **XML** *(Extensible Markup Language):* más que una notación, XML es un metalenguaje extensible de etiquetas desarrollado por el W3C *(World Wide Web Consortium)*. Permite describir la gramática de lenguajes específicos por medio de la definición de sus "metadatos", expresando las reglas que los rigen en un documento de definición de tipos (*.DTD) o un archivo de definición del esquema XML (*.XSD). Los datos deben cumplir con las reglas definidas y se almacenan acompañados de etiquetas en el mismo archivo o en un archivo diferente (por lo general *.XML) [11].

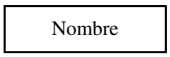
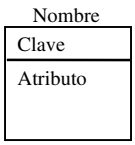
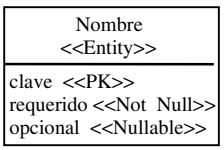
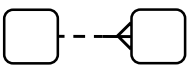
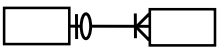
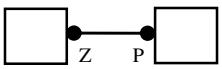
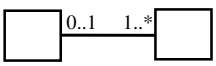
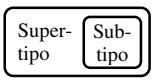
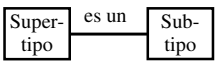
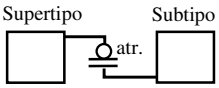
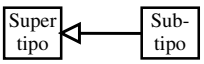
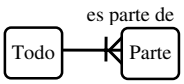
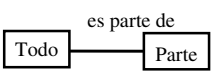
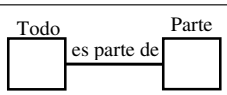
Según las recomendaciones del estudio comparativo de Hay en [12], la notación más apropiada para el análisis es Barker, mientras que en el diseño resulta de gran utilidad usar UML; sin embargo, en algunos casos la notación estará determinada por las herramientas disponibles, las condiciones específicas del grupo de desarrollo o las particularidades del proyecto. En la tabla 1 se presenta un cuadro comparativo que muestra las diferencias entre los principales elementos de las notaciones más utilizadas por herramientas de modelado de datos en la actualidad.

3. LOS MODELOS DE DATOS EN LAS METODOLOGÍAS DE DESARROLLO

Un factor determinante para el tratamiento de los artefactos de persistencia es el nivel de abstracción de los modelos de datos. Según Ambler [13], existen tres estilos para la construcción de modelos relacionados con los datos.

- *Modelo conceptual:* también llamado modelo del dominio, sirve para representar elementos del espacio del problema, frecuentemente antecede o reemplaza al modelo lógico. Se utiliza en las etapas previas al análisis.
- *Modelo lógico:* sirve para analizar los conceptos del dominio y las relaciones entre éstos. Se utiliza en la etapa de análisis.
- *Modelo físico:* sirve para representar los elementos que se dispondrán en el SGBD o esquema inter-

Tabla 1. Comparativo de los elementos en las principales notaciones de modelado de datos

Elemento de Modelado	Notación			
	Barker	**IE**	**IDEFIX**	**UML**
Entidad y características de los atributos				
Relación con multiplicidad de 0..1 a 1..n				
Herencia				
Agregación/ Composición				

no de la base de datos. Se utiliza en la etapa de diseño.

Sin embargo, en el Proceso Unificado de Desarrollo (RUP: *Rational Unified Process*) [14] los modelos de datos se reducen a un artefacto que se construye en el flujo de trabajo del análisis y diseño.

En propuestas como las planteadas en CADM *(CASE Applications Development Method)* [15], los modelos de datos tienen papel protagónico, convirtiéndose en artefactos que rigen el proceso de desarrollo y la generación de código. La metodología presentada en CADM se deriva del método CASE (CASE***** Method) [3] y plantea diez etapas de las que se destacan el análisis, el diseño y la construcción. La notación usada en CADM es una adaptación de la notación Barker que posibilita la diferenciación de los modelos lógicos y físicos.

3.1 Análisis

En el análisis se construyen artefactos cercanos al dominio del problema; tienen carácter declarativo, centrándose en "qué" se pretende hacer. El principal artefacto del análisis en CADM es el modelo Entidad-Relación (diagrama E-R), que se refiere a un modelo lógico. Para la construcción de un diagrama E-R, se explora gramaticalmente la narrativa del negocio, buscando elementos propios del modelo de la siguiente manera:

- **Entidades.** Se refieren a los conceptos, personas, objetos o cosas de los cuales el sistema necesite guardar información. Las forman los sustantivos más relevantes encontrados en la narrativa del negocio. Por tratarse de un concepto o abstracción, suele utilizarse un sustantivo en singular para nombrar la entidad. Se simbolizan usando un rectángulo con los bordes redondeados.
- **Atributos.** Constituyen cada una de las características asociadas a una entidad; gramaticalmente los atributos se buscan en los adjetivos que califican a las entidades. Se simbolizan colocando su nombre dentro del rectángulo de la respectiva entidad. Los tipos de atributos son los siguientes:
  - Claves primarias. Identificadores que hacen que una presencia de la entidad sea diferente

de las demás. Se simbolizan con el signo número.

- Obligatorios. Se simbolizan con un círculo relleno o con un asterisco.
- Opcionales. Se simbolizan con un círculo hueco o con un espacio en blanco.
- Claves candidatas. Son aquellas que sirven de alternativa a la clave primaria.

- **Relaciones.** Muestran la forma en que dos entidades se asocian; gramaticalmente las relaciones se pueden encontrar dentro de la narrativa del negocio como verbos conjugados, preposiciones, conjunciones u otros elementos que unen dos de los sustantivos que representan entidades. Las relaciones se representan como líneas que unen las dos entidades implicadas y tienen principalmente dos características:
  - *Multiplicidad*. Se refiere al número de presencias de una entidad que puede tener la otra, se lee llegando y se representa con una línea simple para denotar uno, o con tres líneas en forma de pata de gallo *(crow's foot* en inglés*)* para denotar muchos.
  - *Obligatoriedad*. Determina si ante una presencia de una entidad deberá o podrá haber presencias de la otra entidad, se lee saliendo y se representa con una línea continua para denotar obligatoriedad o con una línea punteada para denotar que es opcional.

En el caso de aplicación, la Aldea Contable dispone de nueve módulos para apoyar los diferentes frentes del proceso docente; por limitaciones de espacio, se ilustra solo el uso de los artefactos de persistencia en el módulo "Revista Contaduría", que corresponde al apoyo virtual de una publicación impresa que se entrega periódicamente en el Departamento de Ciencias Contables de la Universidad de Antioquia.

El diagrama E-R de la figura 1 muestra las entidades y relaciones del módulo "Revista Contaduría". En este diagrama se observa como la relación de muchos a muchos entre las entidades *ARTICULO* y *AUTOR* presenta un conflicto de pérdida de información que necesita ser resuelto con el surgimiento de la entidad de intersección *AUTORIA_ARTICULO;* en esta entidad se ubican los atributos *ROL* y *POSICION,* que en caso de situarse en cualquiera de las dos entidades involucradas en la relación ocurrirían

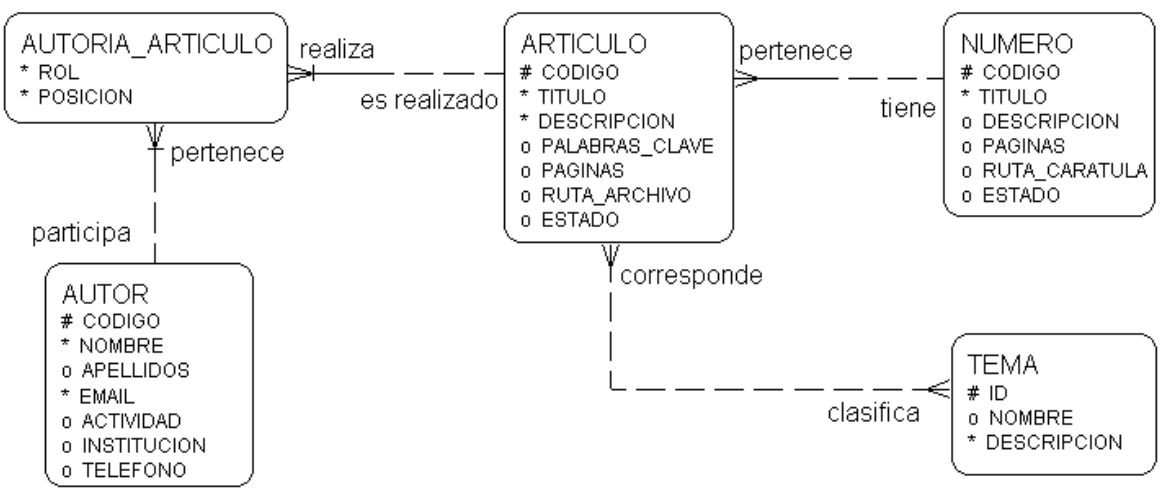


Figura 1. Modelo de análisis del módulo "Revista Contaduría" de la Aldea Contable

problemas de multiplicidad. Un caso contrario se presenta en la relación de muchos a muchos entre las entidades *ARTICULO* y *TEMA*, que no necesita ser tratada como un conflicto, pues no tienen información propia que experimente problemas de multiplicidad.

La entidad *AUTORIA_ARTICULO* se denomina débil con respecto a *NUMERO* y *ARTICULO*, pues su clave primaria se supone compuesta por las claves de las otras dos entidades y no necesita ser representada dentro de dicha entidad. Esta situación se representa con una línea cruzada en la relación del lado de la entidad débil; este tipo de relaciones se llaman vinculantes *(Identifying Relationship)*.

3.2 Diseño

En el diseño se construyen artefactos cercanos a la solución o a los objetos del SGBD, tiene carácter imperativo, centrándose en el "cómo" de la solución. El principal artefacto del diseño en CADM es el llamado esquema de datos o modelo del servidor, que se refiere a un modelo físico. Un esquema de datos se construye a partir de la transformación del diagrama E-R, teniendo en cuenta las siguientes consideraciones:

- **Tablas.** Cada una de las entidades modeladas en el análisis representa una tabla en el diseño. Por tratarse de la necesidad de almacenamiento, su nombre suele ser el sustantivo que viene del análisis pero en plural, pues en una tabla se almacenan muchos registros. Se simbolizan con un rectángulo con los bordes rectos.
- **Columnas.** Cada uno de los atributos del diagrama E-R se transforma en una columna del esquema de datos; la columna conserva las características funcionales que tenía el atributo, salvo que algunas veces se le adiciona el tipo de dato.
- **Claves foráneas.** Cada relación en el diagrama E-R genera una clave foránea en el esquema de datos y el surgimiento de una o varias columnas en la tabla del extremo de la relación con multiplicidad de muchos. El número de columnas que surgen está determinado por el número de atributos de la clave primaria de la otra tabla.

Es importante recalcar que las columnas que surgen en el diseño, producto de las relaciones entre entidades en el análisis solo se representan en los modelos de datos y no se representan en los diagramas E-R, pues en el análisis se acepta que las relaciones tienen identidad propia y por ellas viaja información.

El esquema de datos de la figura 2 muestra el modelo físico de la base de datos del caso de aplicación. En este diagrama se observa la tabla *TEMARIO_ARTICULOS* resolviendo la relación de muchos a muchos que se presentaba entre las entidades que sirven de base para las tablas *TEMAS* y *ARTICULOS*. En el caso de aplicación, todas las claves foráneas transfieren la clave primaria de la tabla con multiplicidad uno a la tabla con multiplicidad muchos, exceptuando la clave foránea *ART_NUM_FK*, en la cual la columna *NUMERO* pasa a la tabla *ARTICULOS*, pero no como clave primaria, ya que no proviene de relaciones vinculantes como las demás.

3.3 Construcción

El principal artefacto de la construcción en CADM lo constituyen los scripts de creación de los objetos de la base de datos. Estos scripts se llaman DDL (*Data Definition Language)* y se construyen a partir de la transformación del esquema de datos, teniendo en cuenta las siguientes consideraciones:

- Cada tabla se convierte en una sentencia "*CREATE TABLE…*" que debe tener una definición de columna, con su respectivo tipo y restricciones, por cada una de las columnas del esquema de datos.
- Cada clave primaria se convierte en una cláusula "… PRIMARY KEY…" en la creación o alteración de tabla, y los valores de esta columna no podrán repetirse. En caso de que la clave sea compuesta,

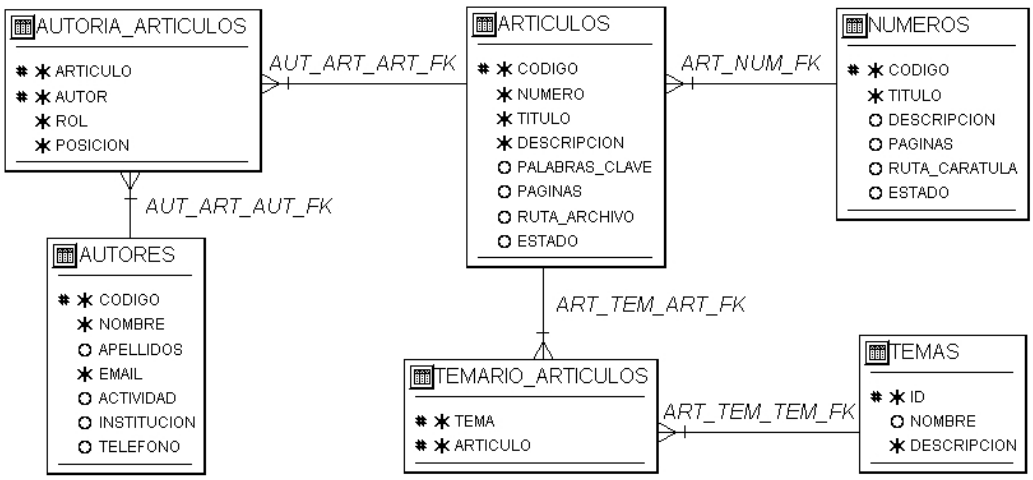


Figura 2. Modelo de diseño del módulo "Revista Contaduría" de la Aldea Contable

el orden de las columnas suele definir el orden de la columna en el índice asociado.

- Cada clave foránea se convierte en una cláusula "… *REFERENCES*…" en la creación o alteración de tabla de la clave foránea, hacia la tabla de la clave primaria que se referencia. Es importante recalcar que solo se pueden referenciar columnas o combinaciones de ellas que sean únicas *(PRIMARY KEY o UNIQUE)*.
- Cada objeto creado en la base de datos debe considerar el cálculo de espacios de almacenamiento, para evitar problemas de rendimiento en el acceso a datos. Esta práctica se llama *sizing* de esquema, y los cálculos que se realizan dependen en gran medida de las características específicas del SGBD utilizado.

Si en lugar de un motor relacional, se cuenta con un SGBD orientado por objetos, se puede utilizar UML como técnica de modelado de datos, teniendo en cuenta ciertas restricciones para que cada elemento en el modelo encuentre correspondencia directa en elementos de las bases de datos [16].

4. TÉCNICAS PARA EL ACCESO A DATOS

A continuación se ilustran los mecanismos de acceso a datos como las API, librerías de abstracción, el patrón DAO y las técnicas de mapeo objeto-relacional.

4.1 API *(Application Programming Interface)*

Una API (interfaz de programación de aplicaciones) es un conjunto de rutinas y especificaciones de comunicación entre componentes de software. Existen varios tipos de API: nativas de un sistema operativo como ODBC en Windows, nativas de un lenguaje como JDBC con Java y nativas de un SGBD como OCI con Oracle. Enseguida se presentan algunas de las API de mayor utilización para el acceso a bases de datos.

- **ODBC** *(Open Data Base Connectivity)*. Es un estándar planteado por Microsoft cuyo objetivo es hacer posible el acceso a cualquier dato de cualquier aplicación, sin importar qué SGBD lo almacene; ODBC logra esto al insertar una capa intermedia llamada manejador (Driver) de bases

de datos, entre la aplicación y el SGBD; el propósito de esta capa es traducir las consultas de datos de la aplicación en comandos que el SGBD pueda interpretar. Desde la versión 2.0 el estándar soporta SQL (Structured Query Language) [17]. Para conectarse a la base de datos se crea en el sistema operativo un origen de datos (DSN: Data Source Name) dentro del cual se definen los parámetros, la ruta y las características de la conexión, según las especificaciones del fabricante.

- **JDBC** *(Java Data Base Connectivity)*. Permite la ejecución de sentencias sobre bases de datos desde Java, independiente del sistema operativo donde se esté ejecutando o del SGBD utilizado [18]. Provee servicios para acceder a una base de datos utilizando SQL como lenguaje de consulta y manipulación de datos.

  Un manejador *(Driver)* de conexión hacia un SGBD específico es un conjunto de clases que implementan las interfaces Java definidas en JDBC y que utilizan los métodos de registro para definir un localizador de una base de datos (URL: Uniform Resource Locator). En la ejecución, el programa debe correr con la librería apropiada para su SGBD y accede a éste estableciendo una conexión y utilizando el localizador a la base de datos con los parámetros específicos.

  La API de persistencia recientemente desarrollada para la plataforma Java EE se llama JPA *(Java Persistence API),* uno de cuyos principales objetivos es unificar la manera en que funcionan las utilidades de mapeo objeto-relacional.

- **OCI** *(Oracle Call Interface)*. Permite ejecutar sentencias SQL para la consulta y manipulación de datos en un SGBD Oracle, desde un lenguaje de programación como C o PHP [19]. Las librerías OCI no necesitan ser recompiladas para su utilización y se tratan de igual forma que las aplicaciones que no son para bases de datos. Las funciones de OCI son:

  – Hacer un análisis sintáctico de las sentencias SQL.

  – Abrir un cursor (espacio de memoria para guardar la respuesta a una consulta).

  – Introducir variables del cliente en la memoria compartida del servidor.

  – Ejecutar las sentencias SQL en el espacio de memoria del cursor.

  – Descargar en la aplicación cliente los registros de datos solicitados.

  – Cerrar los cursores y, de esta forma, liberar los espacios de memoria reservados.

4.2 Librerías de abstracción de acceso a datos

Adicionalmente a los servicios de una API, una librería de abstracción provee un modelo de componentes que enmascara la complejidad del acceso a datos, permite usar instrucciones de alto nivel y facilita al desarrollador cambiar el origen de los datos.

Las librerías de abstracción están determinadas por el lenguaje de programación que utilicen, por esto una API como JDBC también encaja dentro de esta categoría; pero teniendo en cuenta el bajo nivel de sus instrucciones, se enmarcó dentro de la categoría de las API.

A continuación se muestran las librerías de abstracción de acceso a datos más usadas en los lenguajes de Microsoft y en PHP.

- **Microsoft.** Para los diferentes lenguajes de programación usados por las plataformas de Microsoft existen las siguientes alternativas [20]:

  – Microsoft DAO. Los objetos de acceso a datos de Microsoft constituyeron un modelo de componentes, usado por excelencia para el acceso a bases de datos Microsoft Access.

  – RDO. Los objetos de datos remotos permiten el desarrollo de interfaces que interactúan directamente con orígenes de datos ODBC en máquinas remotas.

- UDL (Universal Data Link). Son archivos que proveen una interfaz común para especificar los atributos de una conexión basada en ODBC o en OLE DB.
- ADO, ADO.NET (ActiveX Data Objects). Son interfaces de programación de alto nivel para orígenes de datos soportados por ODBC y OLE DB.

- **PHP.** Junto con Java, C y VB, PHP es uno de los lenguajes más usados en la actualidad [21]. Dispone de las siguientes librerías de abstracción de acceso a datos [22]:
  - ADOdb, una popular librería de código abierto que también cuenta con una versión para el lenguaje Python, disponible en la comunidad de SourceForge.
  - PEAR DB y PEAR MDB2, dos versiones de las librerías de abstracción de acceso a datos de la comunidad de PHP. PEAR es la sigla *PHP Extension and Application Repository*.
  - PDO *(PHP Data Objects)*, reciente estrategia de PHP para usar un núcleo común de métodos de acceso a datos, con independencia del SGBD utilizado.

4.3 El patrón DAO *(Data Access Objects)*

Un patrón de software presenta una solución probada a un problema recurrente en un contexto específico [23]. El patrón DAO permite abstraer y encapsular el acceso a una fuente de datos, independizando el almacenamiento persistente de la lógica de negocio y control. El DAO administra las conexiones, recupera y almacena información de la fuente de datos, actuando como un adaptador entre los componentes de negocio y el origen de los datos. Los elementos del patrón DAO son: [24]

- *BusinessObject:* objeto cliente que requiere acceso a la fuente de datos *(Servlet)*.
- *DataAccessObject:* abstracción de la implementación de acceso a datos *(Class)*.
- *DataSource:* implementación de una fuente de datos (DBMS, XML, etc.).
- *TransferObject:* objeto usado por la clase DAO para el transporte de datos *(Bean)*.

El diagrama de la figura 3 ilustra el uso del patrón DAO en el caso de aplicación de la Aldea Contable para el acceso a los datos de la tabla Temas.

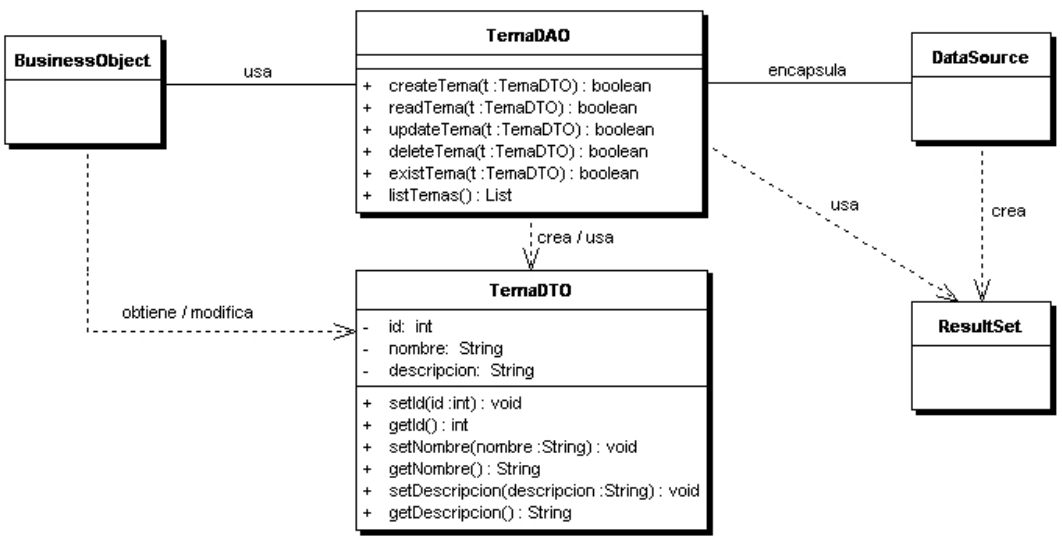


Figura 3. Diagrama de clases del patrón DAO en la tabla Temas de la Aldea Contable

La clase TemaDAO se encarga de implementar sobre la tabla Temas las tradicionales operaciones CRUD, denominadas así por las iniciales de *Create, Read, Update* y *Delete;* en ocasiones se llaman operaciones CRUDEL [25], adicionando *Exist* para verificar la existencia de un objeto y *List* para recuperar todos los datos de una tabla. La clase TemaDTO contiene objetos planos con los datos para insertar o que fueron sacados de la tabla Temas. En la figura 3 se ilustra además la clase ResultSet, que es creada por la clase DataSource cuando se necesita recuperar múltiples registros y se usa por la clase TemaDAO para listar los registros de la tabla Temas.

4.4 Mapeo objeto-relacional *(ORM: Object Relational Mapping)*

Se refiere a la técnica de transformar las representaciones de los datos de un modelo de objetos en un modelo de datos con un esquema basado en SQL [26]. Una solución ORM está compuesta de cuatro partes:

- Una API encargada de realizar las operaciones CRUD para los objetos de las clases persistentes. Dichas operaciones se corresponden con instrucciones DML *(Data Manipulation Language)* en el SGBD.
- Un lenguaje o API para especificar consultas *(Queries)* que se refieran a clases y propiedades de clases.
- Una utilidad para especificar metadatos de mapeo.
- Técnicas de optimización de transacciones como el chequeo sucio automático *(automatic dirty checking),* el cual detecta cambios en los objetos y actualiza su estado en la base de datos sin necesidad de escribir sentencias *update;* estas técnicas están acompañadas de estrategias para la recuperación de los registros de las tablas involucradas en una clave foránea sobre objetos en memoria, tales como:
  - Carga perezosa *(Lazy Loading)*. Se cargan primero los registros de la tabla maestra y posteriormente, si se necesitan, los registros de la tabla detalle.
  - Carga proactiva *(Eager Loading)*. Mediante un join se cargan de una vez los registros de la tabla maestra y los de la tabla detalle.

Para la implementación de una solución que utilice el mapeo objeto-relacional, se dispone de cuatro niveles de adaptabilidad.

- *Relacional puro*. La aplicación completa, incluyendo la interfaz de usuario, está diseñada con el modelo relacional y operaciones basadas en SQL. Las aplicaciones en esta categoría generalmente tienen un alto uso de procedimientos almacenados.
- *Mapeo de objetos ligero*. Las entidades se representan como clases que se mapean manualmente a tablas relacionales. El manejo manual de SQL/JDBC se oculta de la lógica de negocio usando patrones de diseño, como el patrón DAO. Esta aproximación es ampliamente conocida y es adecuada para aplicaciones con un número pequeño de entidades o aplicaciones con modelos genéricos de datos.
- *Mapeo de objetos medio*. La aplicación se diseña alrededor del modelo objetual. El SQL se genera en tiempo de desarrollo utilizando una herramienta de generación de código o en tiempo de ejecución por medio de un *framework*. Este tipo de mapeo es adecuado para aplicaciones con transacciones complejas, en particular cuando se requiere portabilidad entre diferentes bases de datos. Estas aplicaciones comúnmente usan el patrón *Active Record,* y rara vez se utilizan procedimientos almacenados.
- *Mapeo de objetos completo*. Soporta modelos de objetos sofisticados que tengan composición, herencia y polimorfismo. La capa de persistencia se implementa de manera transparente en *frameworks* como Hibernate [27] o utilizan la técnica de *scaffold* para desplegar automáticamente los formularios para las operación CRUD.

Las opciones para adoptar una solución ORM se basan en el uso de implementaciones de la especificación JDO o en la utilización de *frameworks* ORM.

JDO *(Java Data Objects)*. Estándar desarrollado por Java Community Process, que define las interfaces y clases para manejar objetos persistentes del dominio (objetos de negocio y aplicación). Su funcionamiento se basa en las siguientes premisas:

- Mediante un archivo XML se indica qué clases son persistentes.
- Se establecen los parámetros de configuración para la ejecución de la persistencia: clases que implementan JDO, archivo con los parámetros de conexión, descriptores de persistencia (habitualmente se sitúan en los directorios META-INF o WEB-INF para aplicaciones web) y la versión modificada de las clases persistentes con su versión original.
- Se procesan las clases persistentes compiladas y se les agrega código mediante el *Enhancer,* que se encarga de hacer que cada clase persistente implemente la interfaz *PersistenceCapable*. Las nuevas .class son compatibles con cualquier implementación de JDO.
- El objeto de la clase *PersistenceManager* se encarga de establecer el contexto de la conexión (jdo.properties) y las transacciones y realiza las operaciones internas de persistencia (CRUD). La recuperación selectiva de instancias se lleva a cabo mediante el lenguaje de consulta JDOQL [28].

Entre las implementaciones de JDO se encuentran: KODO de BEA, JDO Genie de Hemisphere Technologies, JRelay de Object Industries GmbH, Versant y JPox.

Frameworks ORM. Un *framework* provee una arquitectura definida en la cual otra aplicación puede ser organizada y desarrollada [29]. Uno de los *frameworks* ORM más conocidos es Hibernate, que apoya la construcción de la capa de acceso a datos en aplicaciones Java, soportando una amplia variedad de SGBD: Oracle, DB2, MySQL, PostgreSQL, SAP DB, entre otras. En cualquier aplicación que utilice Hibernate aparecen cuatro objetos básicos:

- *Configuration*. Contiene la información necesaria para realizar la conexión a la base de datos. Lee dos tipos de archivos XML: un archivo de propiedades (Hibernate.properties) en el cual se especifican los parámetros de conexión al SGBD y un archivo de mapeo (*.hbm.xml) para cada clase que requiera persistir, indicando las correspondencias clases/tablas y propiedades/columnas.
- *SessionFactory*. Es una fábrica de *Sessions*. Un objeto *Configuration* tiene toda la información necesaria para crear una *SessionFactory*.
- *Session*. Es la principal interfaz entre la aplicación Java e Hibernate. Mantiene las conversaciones entre la aplicación y la base de datos. Permite adicionar, modificar y borrar objetos en la base de datos.
- *Transaction*. Permite las operaciones de confirmar y deshacer transacciones.

Hibernate proporciona un lenguaje de consulta orientado a objetos llamado HQL *(Hibernate Query Language)*. Se diferencia del SQL en dos aspectos: permite hacer preguntas basadas en objetos y propiedades y no en tablas y columnas; no permite manipular datos, sólo se encarga de recuperar objetos; las operaciones de actualización, inserción y borrado las llevan a cabo los objetos *Session* [30].

Otros de los frameworks de mapeo objeto-relacional más usados son: TopLink de Oracle, Torque y SimpleORM.

5. HERRAMIENTAS

Los SGBD se encargan exclusivamente del almacenamiento y manejo de los datos; su operación dentro del desarrollo de un sistema de información se ve complementada por la participación de otros tipos de herramientas como las listadas a continuación.

- **Herramientas de acceso a SGBD.** Sirven para escribir e interpretar instrucciones SQL, DDL o DML, que se envían al motor de bases de datos para ser ejecutadas. Entre las más comunes se tienen: SQL Navigator y TOAD de Quest Software, SQL Developer de Oracle y MySQL Query Browser de MySQL AB.
- **Herramientas de administración de SGBD.** Una base de datos se puede administrar con un subconjunto de instrucciones SQL, en especial DCL (Data Control Language), sin embargo, el volumen de instrucciones de administración es considerable; es en este punto en donde las herramientas que apoyan las labores de administración se hacen útiles. En algunos casos los SGBD vienen acompañados de herramientas de acceso y de administración. Entre las más usuales están: SQL Server Enterprise Manager, Oracle Enterprise Manager (OEM) y phpMyAdmin para MySQL.
- **Herramientas de modelado de datos.** Sirven para construir modelos relacionados con los datos, permiten la generación de código y, en algunos casos, la transformación de modelos lógicos a modelos físicos. Entre las de mayor uso tenemos: DBDesigner de FabForce.net, Designer de Oracle, ERwin de Computer Associates y ER/Studio de Embarcadero Technologies.
- **Herramientas de generación de código de acceso a datos.** Sirven para generar el código de la capa de persistencia a partir de un esquema de la base de datos. Por lo general, basan el código generado en el uso del patrón DAO. Entre las más conocidas figuran: FireStorm/DAO de Code Futures Software, DAOGen de TitanicLinux.Net y MyDB Studio de H2LSoft.

La Ingeniería Inversa es una funcionalidad presente en la mayoría de herramientas, que permite conectarse a uno o varios tipos de SGBD para extraer de sus metadatos la información necesaria para la construcción de modelos, generación de código o definición de *scripts* para la creación de objetos de la base de datos.

6. CONCLUSIONES Y TRABAJOS FUTUROS

El área de las bases de datos es uno de los frentes del desarrollo de software que ha alcanzado una mayor madurez; características como la transformación de modelos y la generación de código para el acceso a datos y para la creación de los objetos de la base de datos indican que las herramientas que apoyan la construcción de artefactos de persistencia cumplen con los criterios para ser consideradas herramientas MDA *(Model Driven Architecture)* [31].

Un interesante frente de trabajo es la construcción de herramientas de código abierto para apoyar procesos de desarrollo basados en métodos que se centran en los artefactos de persistencia como CADM *(CASE Applications Development Method)* [15] y la reciente *Agile Data Method* [32].

En el caso de aplicación de la Aldea Contable se utilizó una aproximación metodológica basada en RUP [14]; sin embargo, se les puso énfasis a los modelos de datos y a los artefactos de persistencia, lo que agilizó en gran medida el desarrollo del proyecto. Todos los modelos construidos durante el desarrollo de la Aldea Contable se encuentran disponibles para la comunidad informática en la dirección: http://contaduria.udea.edu.co/index.php?page=Principal.Documentacion; para los modelos de datos solo se publican los modelos físicos construidos con DBDesigner de fabFORCE.net usando la notación IE; para los modelos lógicos y físicos de este artículo se utilizó Designer de Oracle y la notación Barker.

REFERENCIAS

1. Codd, E. F. A relational model of data for large shared data banks. Communications of the ACM, Association for Computing Machinery Inc. Vol. 13, No. 6, Junio de 1970, pp. 377-387.
2. Chen, P. The entity-relationship model: towards a unified view of data. ACM Transactions on Database Systems, Vol. 1, No 1, Marzo de 1976, pp. 9-36.
3. Barker, R. CASE*Method: entity relationship modeling. Wokingham, England: Addison-Wesley, 1990.

4. Finkelstein, C. Strategic systems development. Sydney: Addison-Wesley, 1992.

5. Bruce, T. Designing quality databases with IDEF1X information models. New York: Dorset House, 1992. ISBN 0932633188.

6. Halpin, T. Information modeling and relational databases. San Francisco: Morgan Kaufmann, 2001.

7. Booch, G., Rumbaugh, J. y Jacobson, I. El lenguaje unificado de modelado. Madrid: Addison Wesley, 1999. ISBN 8478290281.

8. Naiburg, E. and Maksimchuk, R. UML for database design. Addison-Wesley Object Technology Series, 2001. ISBN: 9780201721638.

9. Ambler, S. A UML profile for data modeling. [documento electrónico] Ambysoft Inc. 2001. (Citada: agosto 16 de 2007) http://www.agiledata.org/essays/umlDataModelingProfile.html

10. Object Management Group. Request for proposal, Information Management Metamodel (IMM). OMG, 2007.

11. World Wide Web Consortium. Extensible markup language (XML) 1.0, 4 ed. [documento electrónico]. W3C Recommendation, 2006. (Citada: agosto 16 de 2007) http://www.w3.org/TR/REC-xml/

12. Hay, D. A comparison of data modeling techniques. [documento electrónico]. Essential Strategies, Inc, 1997. (Citada: agosto 16 de 2007) http://www.essentialstrategies.com/publications/modeling/compare.htm

13. Ambler, S. Data modeling 101. [documento electrónico]. Ambysoft Inc, 2006. (Citada: agosto 16 de 2007) http://www.agiledata.org/essays/dataModeling101.html

14. Booch, G., Rumbaugh, J. y Jacobson, I. El proceso unificado de desarrollo de software. Madrid: Addison Wesley, 2000. ISBN 8478290362.

15. Dorsey, P. y Koletzke, P. Manual de Oracle Designer. España: McGraw-Hill, 1997. 556 p.

16. Dorsey, P. y Hudicka, J. Oracle 8: Diseño de bases de datos con UML. México: McGraw-Hill Interamericana, 1999. 394 p.

17. Microsoft Corporation. ODBC - Open database connectivity overview. [documento electrónico]. Microsoft Corporation, 2007. (Citada: agosto 9 de 2007) http://support.microsoft.com/kb/110093/en-us.

18. Sun Developer Network (SDN). JDBC overview. [documento electrónico]. Sun Microsystem, 2007. (Citada: agosto 9 de 2007) http://java.sun.com/products/jdbc/overview.html

19. Oracle Technology Network. Oracle call interface: general information. [documento electrónico]. Oracle Corporation, 2007. (Citada: agosto 9 de 2007) http://www.oracle.com/technology/tech/oci/index.html

20. Microsoft Developer Network. ADO programmer's guide. [documento electrónico]. MSDN. (Citada: agosto 30 de 2007) http://msdn2.microsoft.com/enus/library/ms807642.aspx

21. TIOBE Software. TIOBE programming community index. [documento electrónico]. The Coding Standards Company. (Citada: septiembre 13 de 2007) http://www.tiobe.com/tpci.htm

22. Holloway A. and Jones C. Underground PHP and Oracle manual 2007. 195 p.

23. Gamma, E., Helm R., Johnson R. and Vlissides J. Design patterns, elements of reusable object-oriented software. Reading MT: Addison Wesley, 1995.

24. Sun Developer Network (SDN). Core J2EE Patterns: Data Access Object. [documento electrónico]. Sun Microsystem, 2002. (Citada: agosto 31 de 2007) http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

25. Kruger, S., Parlione, M., Mills, D., McNena, S., Healy, M. and Biava, F. Engineering WS-I compliant web services for IBM Lotus Domino V8. [documento electrónico]. IBM, 2007. (Citada: septiembre 16 de 2007) http://www.ibm.com/developerworks/lotus/library/domino8-WS-I/ index.html?ca=drs-

26. Red Hat Middleware. Hibernate reference documentation v 3.2.2. [documento electrónico]. Red Hat, 2007 (Citada: agosto 23 de 2007) http://www.hibernate.org/hib_docs/reference/en/pdf/hibernate_reference.pdf

27. Bauer, C. and King, G. Hibernate in action. [documento electrónico]. (Citada: agosto 23 de 2007) http://dev2dev.bea.com.cn/bbs/servlet/D2DServlet/download/126-34474-202400-2852/HibernateInAction.pdf

28. Mármol, J. Persistencia de objetos. JDO, Solución Java. [documento electrónico] (Citado: agosto 27 de 2007) http://dis.um.es/~jmolina/Persistencia%20de%20Objeto%20JDO.pdf

29. Fayad, M., Schmidt, D. and Johnson, R. Building application frameworks. Addison-Wesley, 1st ed., 1999.

30. Roses, F. Introducción a Hibernate. [documento electrónico]. (Citada: agosto 27 de 2007). www.froses.com/Assets/Files/Articles/Hibernate_Introduccion_es.pdf

31. Quintero, J. y Anaya, R. Marco de referencia para la evaluación de herramientas basadas en MDA. Memorias del X Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software, 2007. p. 225-238. ISBN 978-980-325-323-3.

32. Ambler, S. The vision of the Agile Data Method [documento electrónico] Ambysoft Inc, 2007. (Citada: septiembre 19 de 2007) http://www.agiledata.org/essays/vision.html