República de Cuba

Universidad de las Ciencias Informáticas

Facultad 2

"Migración de la Capa de Acceso a Datos del Sistema de Gestión de Emergencias de Seguridad Ciudadana (171)"

TRABAJO DE DIPLOMA

PRESENTADO PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Autor: Rasiel Aponcio Borges.

Tutores: Ing. Yahima Vigo Valdés.

Ing. Daniel Ernesto Vargas.

Ciudad de la Habana, Cuba. Abril de 2009.
"Año del 50 Aniversario del Triunfo de la Revolución"

Dedicatoria

A toda mi familia.

Agradecimientos

A mis padres por su apoyo, ayuda y preocupación constante.

A toda mi familia. Gracias a todos, siempre me han ayudado muchísimo, son lo mejor.

A Roxi por todo su cariño y apoyo.

A mis hermanos(as) de la UCI y a todos esos amigos(as) que son para toda la vida, gracias.

A mis tutores por su ayuda que nunca faltó. Gracias por toda su dedicación.

Mi eterno agradecimiento a la Revolución Cubana por permitirme cumplir mis sueños.

Declaración de Autoría

Declaro que soy el único autor de esta tesis y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____	_____	_____
Rasiel Aponcio Borges	Ing. Yahima Vigo Valdes	Ing. Daniel Ernesto Vargas
Autor	Tutor	Tutor

Resumen

La seguridad ciudadana es uno de los pilares fundamentales desde el punto de vista social que todo país debe garantizar. Una de las acciones llevadas a cabo por la República Bolivariana de Venezuela para asegurar a sus ciudadanos es la construcción del Sistema de Gestión de Emergencias y Seguridad Ciudadana (SIGESC 171).

El SIGESC es un sistema complejo y utiliza tecnologías novedosas, está dividido en varios subsistemas o aplicaciones que interactúan con un gestor de base de datos. Cada una de las aplicaciones informáticas que componen el SIGESC 171 están desarrolladas bajo una arquitectura en capas, entre las cuales se encuentran: la Capa de Interfaz (Acciones y Interfaces de usuarios), Capa de negocio (Lógica de los procesos de negocio) y la Capa de Acceso a Datos (CAD).

La CAD con que cuenta actualmente el SIGESC 171 presenta varios problemas, entre ellos se destacan la carencia de soporte completo a las transacciones que ejecutan procedimientos almacenados en su interior y la aparición de excepciones que provocan demoras considerables en la detección y depuración de errores. Por ello surge la necesidad de migrar la CAD de SIGESC 171 para alcanzar un mayor rendimiento y solucionar los problemas presentados en su primera versión.

El documento que se presenta a continuación recoge los resultados del trabajo investigativo realizado para llevar a cabo el desarrollo de la nueva CAD para el SIGESC 171.

Índice

Introducción

En la década de los '80, la República Bolivariana de Venezuela pasó de ser un país con mínimos niveles de violencia a un país con altas tasas de homicidios con 31.61 asesinatos por cada cien mil habitantes (1). Algunas medidas para dar solución al problema de la inseguridad se encuentra el programa “Caracas Segura 2008” (2), el “Plan 180” (3) y la creación de Centros de Gestión de Emergencias, estos últimos disponen de sistemas automatizados para el proceso de atención de emergencias.

Actualmente no todos los estados de la República Bolivariana de Venezuela disponen del servicio que brindan los Centros de Gestión de Emergencias, además no existe una total coordinación de sus acciones en el momento de dar respuesta a un incidente o fenómeno y algunos actúan de manera independiente.

El Ministerio del Poder Popular para Relaciones Interiores y Justicia promueve la formulación y puesta en marcha del Sistema de Gestión de Emergencia y Seguridad Ciudadana (SIGESC 171) para reducir los índices de violencia e inseguridad. El sistema constituye una de las piedras angulares en el derecho de protección que tienen los ciudadanos según lo establecido en el artículo 55 de la Constitución de la República Bolivariana de Venezuela:

“Toda persona tiene derecho a la protección del estado, a través de los Órganos de Seguridad Ciudadana regulados por Ley, frente a situaciones que constituyan amenazas, vulnerabilidad o riesgos para la integridad física de las personas, sus propiedades, el disfrute de sus derechos y el cumplimiento de sus deberes”. (4)

El SIGESC 171 es un sistema automatizado único para todo el país que debe funcionar en los Centros de Gestión de Emergencias de los diferentes estados. Permitirá el monitoreo de recursos que tienen los centros destinados a la atención de las solicitudes de emergencias, además pretende proveer a los centros de funcionalidades para el control del trabajo del personal y el procesamiento eficiente de las solicitudes de emergencia.

La construcción del SIGESC 171 es una tarea compleja debido al monto de información que se maneja en tiempo real, su prestación de servicios de manera ininterrumpida, interacción con varias aplicaciones, la utilización de tecnologías novedosas como Sistema de Posicionamiento Global (Global Positioning System o GPS), la complejidad de sus procesos de negocio y la seguridad y disponibilidad de la información. El sistema se divide en diferentes subsistemas, que en el caso del SIGESC 171 cada uno es una aplicación informática:

- ✓ Subsistema de Recepción de Llamadas (Operador).

- ✓ Subsistema de Despacho (Despachador).
- ✓ Subsistema de Mapificación.
- ✓ Subsistema de Supervisión de Operadores.
- ✓ Subsistema de Supervisión de Despachadores.
- ✓ Subsistema de Supervisión General.
- ✓ Subsistema de Estadísticas.
- ✓ Subsistema de AVL.
- ✓ Subsistema de Administración y Control de Recursos.
- ✓ Subsistema de Configuración de Operaciones.
- ✓ Subsistema de Administración Informática.

Los subsistemas interactúan con el servidor de base de datos la mayor parte del tiempo porque en él se almacenan las configuraciones de cada uno de los subsistemas y los datos de los usuarios. Por ejemplo el subsistema de Estadísticas que genera reportes a partir de la base de datos que sirven para prevenir sucesos o emitir criterios a partir de análisis estadísticos que evalúen el comportamiento de los índices seleccionados; los subsistemas de Recepción de Llamadas, Despacho y Configuración de Operaciones principalmente registran, modifican y eliminan información de las situaciones de emergencias; el subsistema de Mapificación fundamentalmente hace búsquedas en el servidor y los subsistemas de Administración Informáticas y Administración y Control de Recursos realizan consultas de definición de datos. Cada uno de los subsistemas contiene un módulo principal y cero o varios módulos secundarios y estos a su vez se componen por tres capas, Capa de Interfaz (Acciones y Interfaces de usuarios), Capa de negocio (Lógica de los procesos de negocio) y la Capa de Acceso a Datos (en lo adelante CAD). (5)

La CAD permite gestionar los datos utilizados en los procesos de negocio y abstraerse de la forma en que estos persisten o son obtenidos. Entre las motivaciones que existen para aislar esta lógica en una capa independiente están el desarrollo paralelo de las capas, que a su vez permite que parte del equipo solo se dedique al desarrollo de la lógica de acceso a datos, ganar en modularidad del sistema y aumentar la cohesión de la capa de negocio al no tener que incluir en su lógica implementaciones para el acceso a los datos.

El desarrollo de la CAD del SIGESC 171 requirió del uso de herramientas para acelerar su construcción y que permitieran resolver problemas como:

- ✓ El acceso concurrente de las aplicaciones del SIGESC 171 al servidor de base de datos lo cual provoca que dentro de un escenario de error aparezcan inconsistencias o redundancias en los datos almacenados.
- ✓ La diferencia entre el modelo relacional de la base de datos y el modelo orientado a objetos usado en la implementación de los procesos de negocio.

En respuesta a los problemas mencionados la decisión del equipo de desarrollo fue la utilización de NHibernate como herramienta de mapeo objeto-relacional (ORM) en su versión 1.0, implementar una librería que sirviera de abstracción al complejo conjunto de funcionalidades de NHibernate (NHibernateUtil) y estructurar la CAD de manera que se acoplara lo mejor posible a la arquitectura en capas del SIGESC 171.

El proyecto Hibernate después de su versión 3.0 liberó NHibernate que es la implementación para .NET de la herramienta. Su objetivo es disminuir la diferencia entre el modelo de la base de datos y el modelo de dominio. Para esto se auxilia en archivos XML que describen la forma en que los objetos de ambos modelos deben coincidir (Map o mapear), pero no incluye un mecanismo que describa de forma explícita cómo mapear los procedimientos almacenados, viéndose obligados los programadores a utilizar para el desarrollo de la CAD una versión modificada del framework de persistencia. Esta versión en su implementación carece de un soporte completo a las transacciones que ejecutan procedimientos almacenados en su interior. Otros aspectos que surgen a raíz de la modificación realizada son:

- ✓ Poca descripción de los errores en los XML de mapeo debido a la inexistencia de una integración entre el entorno de desarrollo y NHibernate.
- ✓ Falta de un mecanismo que registre las acciones llevadas a cabo por el framework al ejecutarse un procedimiento almacenado afectando la usabilidad de la herramienta de persistencia y el registro de sucesos al ejecutar procedimientos almacenados.
- ✓ Carencia de cobertura a todos los tipos de datos bases de .NET framework SDK 1.0 como son el caso de enumerativos, tipos booleanos, imágenes y arreglos de byte, haciendo que los programadores tengan que utilizar otras estrategias para el mapeo de estos tipos.
- ✓ Excepciones que provocan demoras considerables en la detección y depuración de errores.

Teniendo en cuenta los problemas anteriormente descritos presentados en la CAD existentes surge la necesidad de realizar una mejora de la CAD y aplicar prácticas y herramientas para obtener una CAD en el SIGESC 171 por lo que se plantea como **problema científico** ¿Cómo

desarrollar la capa de acceso a datos en el Sistema de Gestión de Emergencia y Seguridad Ciudadana (SIGESC 171) que permita una mayor rapidez en la implementación del sistema, una disminución de las líneas de código y corrija los errores de su predecesora?

El **objeto de estudio** de esta investigación son las capas de acceso a datos y el **campo de acción** el subsistema de acceso a datos en los diferentes módulos del Sistema de Gestión de Emergencia y Seguridad Ciudadana.

Se trazó como **objetivo general** migrar la CAD de SIGESC 171 para alcanzar un mayor rendimiento y solucionar los problemas presentados en su primera versión.

Para dar cumplimiento al objetivo general se trazaron los siguientes **objetivos específicos**:

- ✓ Realizar un estudio de los ORM y herramientas que existan para determinar si teóricamente corrigen los problemas presentados en la versión de NHibernate utilizada.
- ✓ Seleccionar las herramientas de acceso a datos y ORM necesarios para automatizar la generación de código y la migración de la CAD de SIGESC 171.
- ✓ Implementar la CAD de un módulo del SIGESC 171 con las herramientas de acceso a datos y framework seleccionados para validar la solución.
- ✓ Documentar el proceso de migración de la CAD al ORM seleccionado para dejar constancia y establecer una base documental sólida para trabajos futuros.

El documento está estructurado esencialmente en 4 partes: Resumen, Introducción, Desarrollo y Conclusiones.

El desarrollo consta con 4 capítulos:

El Capítulo 1: “Fundamentación Teórica” Se describe de manera detallada los principales conceptos que se tratan en el trabajo así como los principales aspectos teóricos. Se realiza un estudio de las principales herramientas y plataformas que se emplean actualmente en el desarrollo de capas de persistencia además de enumerar sus principales puntos fuertes y limitaciones.

El Capítulo 2: “Descripción de la solución propuesta” Se selecciona un ORM hacia el cual se migrará la CAD y se argumenta la selección. Se describe la migración paso a paso teniendo en cuenta los anteriores problemas identificados, manteniendo, y en lo posible, mejorando lo que existe hoy como CAD del SIGESC 171. En el capítulo se describe la integración de la solución con la arquitectura del sistema y aplicar la migración al ORM propuesto.

El Capítulo 3 “Implementación” Se describe las diferentes herramientas, librerías y controles desarrollados para la migración.

El Capítulo 4 “Validación de la solución” Se le realizan pruebas de funcionalidad y rendimiento a la CAD de la aplicación migrada. Se seleccionan funcionalidades que presentaban problemas en la CAD anterior para demostrar que fueron resueltos y por último se realiza una gráfica donde se resume el rendimiento de las funcionalidades seleccionadas.

1 Fundamentación Teórica

En el capítulo se realiza un análisis de los aspectos relacionados con el desarrollo de sistemas multicapas haciendo mayor hincapié en la capa de acceso a datos. Se analizan aspectos teóricos, los principales conceptos y definiciones asociados al dominio del problema y que son necesarios para el desarrollo de la investigación.

Además se hace una caracterización de las herramientas, entorno de desarrollo, lenguaje de programación, framework y alternativas para el desarrollo de la capa de persistencia en ambientes de implementación en capas.

1.1 Base de Datos

Es de costumbre referirse a los datos en una base de datos como "persistentes" (sin embargo no son en realidad persistentes por mucho tiempo). Por persistentes se entiende, intuitivamente, que los datos de la base de datos difieren de otros tipos de datos, tales como datos de entrada, datos de salida, consultas SQL, resultados intermedios y otros datos más generales. Pero precisamente se dice que los datos en una base de datos "persisten" porque, una vez que han sido aceptados por el sistema de base de datos para entrar en la base de datos, estos pueden subsecuentemente ser borrados de esta solamente por una petición explícita del sistema de base de datos, no por un mero efecto de completamiento de ejecución de algún programa. Esta definición de datos persistentes permite dar una definición más precisa de base de datos:

Una base de datos es una colección de datos persistentes que son usados por sistemas de alguna empresa. El término empresa convenientemente se refiere a una organización. Una empresa pudiera ser un individuo (con una pequeña base de datos), o una corporación o similar (con una gran base de datos compartida). (6)

Aclarar que se refiere a sistema de base de datos básicamente como un sistema informático de almacenamiento de datos; en otras palabras es un sistema informático cuyo propósito general es almacenar información y permitir que los usuarios recuperen y actualicen en demanda esta información. La información en cuestión puede ser cualquier cosa que le interese a un individuo u organización. Los componentes principales de un sistema de base de datos son los datos, el hardware, el software y los usuarios. (6)

1.1.1 Base de Datos Relacionales

El modelo relacional no tiene una analogía con el mundo real, más bien está basado en una serie de principios matemáticos principalmente de la teoría de los conjuntos y lógica de predicado. El Dr. E. F. Codd fue el primero en aplicar estos principios al modelado de datos a finales de 1960. El modelo relacional define la manera en que los datos pueden ser representados (la estructura), la manera en que los datos pueden ser protegidos (la integridad) y las operaciones que pueden ser llevadas a cabo sobre los datos (la manipulación). (7)

En términos generales, los sistemas de base de datos relacionales tienen las siguientes características:

- ✓ Todos los datos son representados como un arreglo ordenado conceptualmente en filas y columnas llamadas relaciones.
- ✓ Todos los valores son escalares. Es decir para cualquier fila / columna en particular en la relación hay uno y solamente un valor.
- ✓ Todas las operaciones son llevadas a cabo sobre una relación y el resultado de estas son también una relación (cierre).

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia. Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario esporádico de la base de datos. La información puede ser recuperada o almacenada mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar la información. El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, Structured Query Language o Lenguaje Estructurado de Consultas, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales. Durante su diseño, una base de datos relacional pasa por un proceso al que se le conoce como normalización de una base de datos.

1.1.2 Base de Datos Orientadas a Objetos

El modelo de base de datos orientado a objetos es bastante reciente, y propio de los modelos informáticos orientados a objetos, trata de almacenar en la base de datos los objetos completos (identidad, estado y comportamiento). Su origen se debe a que en los modelos clásicos de datos, como el modelo relacional, existen problemas para representar cierta información, puesto que aunque permiten representar gran cantidad de datos, las operaciones que se pueden realizar con ellos son bastante simples. Las clases utilizadas en un determinado lenguaje de programación orientado a objetos son las mismas clases que serán utilizadas en una base de datos que utiliza el mismo modelo; de esta manera, no es necesario realizar una transformación del modelo de objetos, como sucede cuando se utiliza una herramienta de mapeo objeto relacional. Estos tipos de bases de datos surgen para evitar los problemas que se generan al tratar de representar información, aprovechar las ventajas del paradigma orientado a objetos en el campo de las bases de datos y para evitar transformaciones entre el modelo relacional y el modelo orientado a objetos. (8)

En las bases de datos orientadas a objetos los datos se almacenan como objetos; que no son más que entidades identificables unívocamente que describen tanto el estado como el comportamiento de una entidad del mundo real. El estado de un objeto es descrito mediante

atributos mientras que su comportamiento es definido mediante métodos. Las características de una base de datos orientada a objetos son:

- ✓ La forma de identificar objetos es mediante un identificador único para cada objeto. Generalmente este identificador no es accesible ni modificable para el usuario, que es un modo de aumentar la integridad de entidades y la integridad referencial.
- ✓ Encapsulamiento: cada objeto contiene y define procedimientos o métodos y la interfaz mediante la cual se puede acceder a él y manipularlo. La mayoría de los gestores de bases de datos orientadas a objetos permiten el acceso directo a los atributos incluyendo operaciones definidas por el propio sistema gestor de bases de datos del mismo modelo, los cuales leen y modifican los atributos para evitar que el usuario tenga que implementar una cantidad considerable de métodos cuyo único propósito sea el de leer y escribir los atributos de un objeto. Generalmente, los sistemas gestores de bases de datos orientados a objetos, permiten al usuario especificar qué atributos y métodos son visibles en la interfaz del objeto y cuáles pueden invocarse desde afuera.

En bases de datos orientadas a objetos, los usuarios pueden definir operaciones sobre los datos como parte de la definición de la base de datos. Una operación (llamada función) se especifica en dos partes. La interfaz (o signature) de una operación incluye el nombre de la operación y los tipos de datos de sus argumentos (o parámetros). La implementación (o método) de la operación se especifica separadamente y puede modificarse sin afectar la interfaz. Los programas de aplicación de los usuarios pueden operar sobre los datos invocando a dichas operaciones a través de sus nombres y argumentos, sea cual sea la forma en la que se han implementado. Esto podría denominarse independencia entre programas y operaciones.

1.2 Entorno de Desarrollo en Capas

Una capa es un conjunto de subsistemas que comparten el mismo grado de generalidad y de volatilidad en las interfaces: las capas inferiores son de aplicación general a varias aplicaciones y deben poseer interfaces más estables, mientras que las capas más altas son más dependientes de la aplicación y pueden tener menos interfaces. (9)

El desarrollo en capas es una de las técnicas más comunes en el diseño de software, usada para descomponer un complejo sistema de software en partes que cumplen ciertos requisitos. Si se imagina un sistema en términos de capas como un pastel donde cada capa descansa una sobre otra y cada capa usualmente oculta a su inmediata inferior o sea la capa sobre la cual

descansa. No todas las arquitecturas son así pero la mayoría si ocultan a la capa sobre la cual descansa. (10)

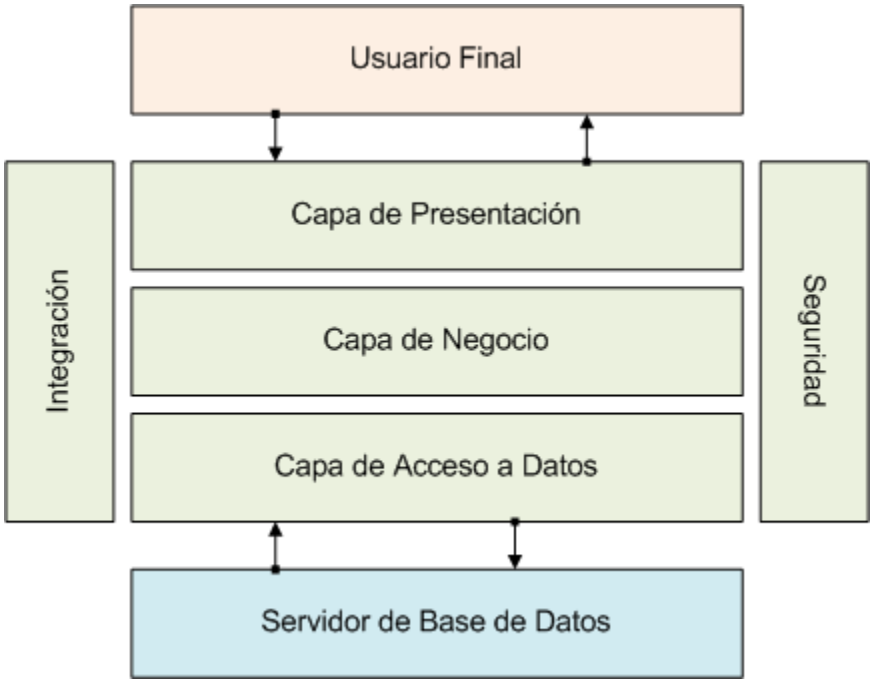


Figura 1 *Ejemplo de Arquitectura basada en capas.*

Descomponer un sistema en capas tiene sus ventajas:

- ✓ Usted puede llegar a comprender una capa completamente como un todo sin estar al tanto del resto de las capas.
- ✓ Se puede "sustituir" capas con implementaciones alternativas de los mismos servicios que brindaba anteriormente o sea se minimiza la dependencia entre las capas.
- ✓ Cada capa te permite usar varios servicios de alto nivel. Pensemos en TCP/IP en el modelo de Interconexión de sistema (OSI) es usado por FTP, Telnet, SSH y HTTP. De otra manera si el modelo no estuviera implementado usando capas cada uno de estos protocolos de alto nivel tendría que implementar su propio protocolo de bajo nivel. (11)

Un sistema de software puede dividirse en cuantas capas se definan en su arquitectura, pero generalmente se suele escuchar hablar de tres capas; presentación, negocio y acceso a datos. Principalmente la Capa de Presentación es la encargada de proveer servicios y mostrar información, la Capa Dominio o Capa de Negocio, como también se conoce, es donde estaría la lógica o el negocio de nuestro sistema. La CAD fundamentalmente se comunica con otros sistemas, que pueden ser otras aplicaciones y sistemas de intercambio de mensajes. Pero en

la mayoría de las aplicaciones empresariales la pieza más grande de origen de datos, son las bases de datos quienes son responsables de almacenar los datos persistentes.

El papel principal de una capa de acceso a datos es la comunicación con varias piezas de la infraestructura que el sistema necesita para su funcionamiento. En el desarrollo de aplicaciones empresariales suelen ser las bases de datos estas piezas.

1.3 Persistencia de Objetos

En el modelo orientado a objetos existen cuatro elementos primarios que si se carece de alguno de ellos el modelo no es orientado a objetos:

- ✓ Abstracción.
- ✓ Encapsulamiento.
- ✓ Modularidad.
- ✓ Jerarquía.

Además hay tres elementos secundarios que son una parte útil del modelo:

- ✓ Tipos (tipificación).
- ✓ Concurrencia.
- ✓ Persistencia.

Al ejecutar cualquier aplicación orientada a objetos se crean y manipulan varios objetos, se tienen objetos transitorios que desaparecerán al final de la sesión y objetos persistentes que continúan existiendo entre las diferentes sesiones. Lo que lleva a definir la Persistencia como la propiedad de un objeto que hace que su existencia trascienda en el tiempo (es decir, el objeto continúa existiendo después de que su creador deja de existir) y/o el espacio (es decir, la posición del objeto varía con respecto al espacio de direcciones en el que fue creado). (12)

1.3.1 Enfoques de Persistencia

Existen varios enfoques de persistencia en el modelo orientado a objetos que los desarrolladores tienen en sus manos para manipular los objetos persistentes. Pueden basarse en el mecanismo de persistencia del lenguaje de programación. Otro enfoque es utilizar los novedosos sistemas de base de datos orientadas a objetos. O bien tener una piel orientada a objeto bajo la cual se oculta una base de datos relacional, esta aproximación es muy utilizada hoy en día debido a la popularidad de las bases de datos relacionales. (13)

Este último enfoque es elegante y compatible con las buenas prácticas de diseño pregonadas en estos últimos 10 años. Consiste en diseñar un modelo de objetos del dominio que represente el cien por ciento de la información que maneja la aplicación y utilizar un framework de Object Relational Mapping (ORM) que resuelva en forma transparente la persistencia de estos objetos contra una base de datos relacional. Utilizar un framework ORM ofrece las siguientes ventajas:

- ✓ Persistencia transparente: Los objetos del dominio no conocen acerca de la base de datos donde son persistidos, el framework lo resuelve en forma automática utilizando archivos de mapeo expresados en XML.
- ✓ Soporte de polimorfismo: Se pueden cargar jerarquías de objetos en forma polimórfica.
- ✓ Soporte de los 3 niveles de mapeo de herencia: Se puede mapear toda una jerarquía de clases a una sola tabla, crear una tabla por cada clase concreta o crear una tabla por cada escalón de la jerarquía.
- ✓ Soporte completo de asociaciones: Los frameworks de ORM soportan el mapeo de todos los tipos de relaciones que pueden existir en un modelo de objetos del dominio (asociaciones 1…1, 1...N, N...M, unidireccionales y bidireccionales).
- ✓ Soporte de carga de objetos Proxy: Se puede cargar objetos que solo contengan la clave del objeto completo.
- ✓ Soporte de caching: En el contexto de una transacción, permite disminuir la cantidad de veces que se encuesta la base de datos, cacheando en memoria los objetos que son accedidos varias veces.
- ✓ Soporte de múltiples dialectos SQL: Es fácil independizarse completamente del tipo de base de datos utilizada. La aplicación puede persistir sus datos en SQL Server, en Oracle, en MySQL, etc. simplemente cambiando la configuración correspondiente.

Por lo general, la mayoría de las aplicaciones persisten su información en una base de datos relacional (RDBMS). Si bien existen otros medios alternativos, como ser una base de datos orientada a objetos (OODBMS), ninguno de estos se compara con la madurez y la popularidad de las bases de datos relacionales. Utilizar un medio relacional ofrece una serie de ventajas muy importantes, por ejemplo:

- ✓ Tecnología madura.
- ✓ Muy eficiente en la grabación y recuperación de grandes volúmenes de datos.
- ✓ Soportan transacciones (la mayoría).
- ✓ Aseguran la integridad de los datos (niveles de aislamiento, locking, etc.).

- ✓ Excelente manejo de la seguridad.
- ✓ Protocolo de consulta estándar (SQL).
- ✓ Buen soporte (muchos administradores).
- ✓ Oferta variada.

1.4 Herramientas de Mapeo Objeto-Relacional (ORM)

Las herramientas ORM son productos de software que surgen con el objetivo de eliminar el diferencial de impedancia (impedance mismatch) entre el mundo relacional y el de la orientación a objetos. Los principales problemas que provoca la diferencia entre el modelo relacional y el paradigma orientado a objetos son:

- ✓ Acceso. En el modelo relacional los atributos pueden ser accedidos y modificados mediante operadores relacionales predefinidos por el lenguaje de consultas (SQL), mientras que en el modelo orientado a objetos permite que cada clase defina la manera en que serán alteradas sus características. Principio de acceso uniforme. (14)
- ✓ Ataduras de Esquemas. Los objetos del modelo orientado a objetos, no siguen ningún esquema puesto que son definidos por el programador, mientras que las tablas en el modelo relacional, deben seguir el esquema entidad-relación.
- ✓ Herencia y polimorfismo. Todo modelo para proclamarse orientado a objeto debe tener estas dos características, mientras que en el modelo relacional el polimorfismo no existe y los mecanismos de herencias se obtienen a través de mecanismos que en ocasiones son muy engorrosos.
- ✓ Identidad de los objetos. En el modelo orientado a objetos los objetos cambian de estado, comportamiento e identidad, esta última es la propiedad que permite diferenciar a un objeto del resto, mientras que en el modelo relacional se necesitan atributos llaves que identifiquen a las tuplas especificados explícitamente por el programador.
- ✓ Estructura y comportamiento. El modelo orientado a objetos se concentra primordialmente en asegurar que la estructura del código sea legible, reutilizable y segura, mientras que el modelo relacional enfatiza en que el comportamiento del sistema una vez en producción, sea eficiente, adaptable y rápido. El modelo relacional hace hincapié en la forma en que los usuarios finales perciben el comportamiento del sistema es mucho más importante.

Un ORM integra las capacidades de los lenguajes orientados a objetos con las bases de datos relacionales. Es una técnica de programación en la cual se vincula la base de datos con la aplicación orientada a objetos, creando objetos virtuales de la base de datos.

Las herramientas de mapeo objeto relacional se encargan no solo de manejar y plantear una solución a las diferencias expuestas, sino que además buscan reducir susceptiblemente el código necesario para llevar a cabo las operaciones de persistencia y recuperación de objetos. Proporcionan además interfaces más simples para el manejo de objetos a través de su propio lenguaje de consulta, y proveen al programador de configuraciones que le permiten optimizar los tiempos de respuesta en sus correspondientes aplicaciones.

1.4.1 Cooperator Framework

Cooperator Framework es por un lado, una librería de clases base, y por otro, una herramienta de generación de código que agiliza el desarrollo de aplicaciones para Microsoft .Net Framework 2.0/3.5. Su principal potencialidad es su sencillez, hace que el desarrollo del acceso a datos sea fácil y transparente. Las principales características son:

- ✓ Uso de entidades para definir el problema a resolver.
- ✓ Modelo totalmente tipado (Capa de Datos y Entidades), esto significa que las clases de persistencia y recuperación de objetos devuelven tipos específicos (Objetos de nuestro dominio), explotando los genéricos.
- ✓ Los objetos pueden enlazarse a los controles de los formularios, tanto en aplicaciones Windows como aplicaciones web aprovechando las ventajas de edición de Visual Studio 2005.
- ✓ Soporta cualquier llave primaria definida en las tablas, sin necesidad de modificarla o crear un campo único en las mismas.
- ✓ Usa procedimientos almacenados por lo que no genera código SQL como la mayoría de los ORM, asegurando un mayor rendimiento.
- ✓ Soporta concurrencia.
- ✓ Genera código de procedimientos almacenados, y Proyectos tanto en C# como en Visual Basic.
- ✓ Mantiene el modelo en un repositorio, el cual puede ser modificado en cualquier etapa del ciclo de desarrollo, permitiendo volver a generar código tanto para nuestra aplicación como los procedimientos almacenados en la base de datos.

- ✓ Licencia BSD (Berkeley Software Distribution). Una de las licencias más permisivas que existen, estando muy cerca del dominio público, dando la libertad de usar, copiar, modificar, compilar y publicar tanto el código fuente como los binarios.

1.4.2 CodeAuthor

CodeAuthor es una herramienta de generación de código en C# 2.0. Entre sus principales características se destacan:

- ✓ Generación completa de la CAD sobre .NET framework 2.0 incluyendo los procedimientos almacenados asociados, pero solo para SQL Server 2005.
- ✓ Soporta las características de .NET 2.0 y 3.5 como son clases parciales, tipos nulos y genéricos.
- ✓ Su filosofía y diseño están basado en que los objetos de acceso a datos deben ser fáciles de usar como un objeto de .NET, además extensible y flexible para adaptarse a cualquier requerimiento.
- ✓ Posee una interfaz simple e intuitiva pero basado en un buen diseño y arquitectura, haciéndolo un robusto framework para C#.
- ✓ Reduce considerablemente el tiempo de desarrollo para aplicaciones web y de escritorio (Windows Forms).
- ✓ Existen dos versiones, una profesional y otra libre, disponibles en dependencia de los requisitos.
- ✓ Genera procedimientos básicos incluyendo los CRUD (en inglés Create, Read, Update y Delete) y búsquedas.
- ✓ Los objetos que son resultados de la generación de código pueden ser serializados para usar en servicios web, para una fácil transferencia o almacenamiento.
- ✓ Completo soporte implícito y explícito a las transacciones.
- ✓ Código comentado completamente.

1.4.3 C# Object Persistent Framework (csopf)

CSOPF es un proyecto cuya meta es hacer que el desarrollo de software sea rápido y fácil de mantener tanto como sea posible. La filosofía es crear un framework muy orientado a desarrolladores. Las características principales son:

- ✓ Auto creación de la base de datos después de correr la aplicación por primera vez incluyendo administración de usuarios y tablas.

- ✓ Soporte para varios gestores de bases de datos relacionales: Microsoft SQL Server, MySql y Firebird.
- ✓ Utiliza atributos para describir las clases.
- ✓ Soporte a relaciones como Uno a uno y Uno a muchos y a operaciones CRUD.
- ✓ Llaves primarias e índices son definido al nivel de la clase y son automáticamente creados en la base de datos.
- ✓ Los detalles de conexión son especificado de dos formas, mediante interfaz de usuario o a través de un archivo XML.
- ✓ Almacenamiento de información para auditoría: Fecha de creación, identificadores creados, fecha de la última modificación, identificador de la última actualización.
- ✓ Consultas parametrizadas para prevenir inyección de SQL.
- ✓ Soporte a transacciones.
- ✓ Compatibilidad con ASP.NET.
- ✓ Compatibilidad para múltiples plataformas (.NET y Mono).

1.4.4 Data Holder

Data Holder es una herramienta de mapeo objeto relacional escrita en C#. Provee encapsulación tipada de datos (Similar a los DataSet de .NET) y persistencia para aplicaciones. Entre sus características claves se encuentran:

- ✓ Soporte de persistencia solo implementado para Microsoft SQL Server 2000 y 2005.
- ✓ Ofrece mecanismos para mapear tablas, vistas, consultas de selección o procedimientos almacenados.
- ✓ Permite crear una estructura jerárquica que contenga múltiples DataHolder.
- ✓ Posee entidades serializables que pueden ser usadas dentro de entornos n-capas.
- ✓ La información de mapeo no se encuentra en la misma clase que los datos, lo cual las hace más reutilizable.
- ✓ Hasta la versión 0.8 su soporte era sobre Microsoft framework 1.1, a partir de la versión 0.9 incluye genéricos de .NET 2.0 y colecciones.

1.4.5 ODX.NET

Open Dataset eXtensions es un framework ORM que provee una capa desconectada flexible de persistencia, tanto para aplicaciones de Windows como móviles. Sus características son:

- ✓ Herencia (ambas variantes sobre una sola tabla o múltiples) y polimorfismo.

- ✓ Relaciones uno a muchos y muchos a muchos.
- ✓ Carga perezosa.
- ✓ Soporta modo desconectado.
- ✓ Múltiples conexiones.
- ✓ Múltiples fuentes de datos.
- ✓ Generación de Proxy en tiempo de ejecución (on-the-fly).
- ✓ Provee de un mecanismo para realizar puntos de salvas (persistent snapshots).
- ✓ Soporte completo a .NET Remoting.
- ✓ Soporta transacciones.
- ✓ Ofrece soporte para Oracle, Postgres, MS Access y MS SQL.
- ✓ Algoritmo para replicación de datos.
- ✓ Soporte para paginado de los resultados.
- ✓ SQL personalizado con protección a inyección.
- ✓ Funciona sobre varias plataformas (Java, Ruby y .NET).
- ✓ Se puede utilizar en casi todas las versiones de .NET framework (v1.1, v2.0 y v3.5) y utiliza las nuevas tecnologías introducidas en cada una de las versiones.

1.4.6 iBATIS

IBATIS toma los mejores atributos e ideas de las tecnologías ORM y Data Mapper encontrando un equilibrio entre ellas, convirtiéndose así en una solución híbrida. Algunas de las ideas que toma son: el soporte para procedimientos almacenados, SQL dinámico, SQL en línea y el Mapeo Objeto Relacional.

Entre sus características más destacadas son:

- ✓ El almacenamiento en caché que permite reducir las veces que se encuesta la base de datos.
- ✓ La carga perezosa que ayuda al rendimiento de los sistemas a la hora de cargar grandes volúmenes de datos (por ejemplo archivos binarios).
- ✓ Brinda soporte a procedimientos almacenados.
- ✓ El código SQL es visible y accesible por los programadores.
- ✓ Soporta transacciones, sesiones y conexiones.
- ✓ Funciona con varios gestores de base de datos (MySQL, Postgres, Oracle, Microsoft Access, Firebird y Microsoft SQL).
- ✓ Libre y de código abierto, está distribuido bajo la licencia Apache License 2.0.

- ✓ Permite el paginado de consultas.
- ✓ Soporta la concurrencia.
- ✓ Posee una curva de aprendizaje elevada.
- ✓ Mapeos sobre archivos XML con estructura muy simple.
- ✓ Permite el uso de múltiples bases de datos y soporta transacciones distribuidas.
- ✓ Soporta el mapeo de objetos compuestos.
- ✓ Manipula una gran variedad de tipos de datos incluyendo colecciones.
- ✓ Es extensible se pueden agregar nuevos manipuladores de tipos de datos.

1.4.7 ADO.NET

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

ADO.NET proporciona acceso coherente a orígenes de datos como Microsoft SQL Server y XML, así como a orígenes de datos expuestos mediante OLE DB y ODBC. Las aplicaciones para usuarios que comparten datos pueden utilizar ADO.NET para conectarse a estos orígenes de datos y recuperar, manipular y actualizar los datos contenidos.

ADO.NET separa el acceso a datos de la manipulación de datos y crea componentes discretos que se pueden utilizar por separado o conjuntamente. También incluye proveedores de datos de .NET Framework para conectarse a una base de datos, ejecutar comandos y recuperar resultados. Los resultados se procesan directamente o se colocan en un objeto DataSet de ADO.NET con el fin de exponerlos al usuario para un propósito específico, combinados con datos de varios orígenes, o de utilizarlos de forma remota entre niveles.

Existen dos componentes de ADO.NET que se pueden utilizar para obtener acceso a datos y manipularlos:

- ✓ Proveedores de datos de .NET Framework. Los proveedores de datos de .NET Framework son componentes diseñados explícitamente para la manipulación de datos y el acceso rápido a datos de sólo lectura y sólo avance.

- ✓ El DataSet de ADO.NET está expresamente diseñado para el acceso a datos independientemente del origen de datos. Como resultado, se puede utilizar con múltiples y distintos orígenes de datos, con datos XML o para administrar datos locales de la aplicación.

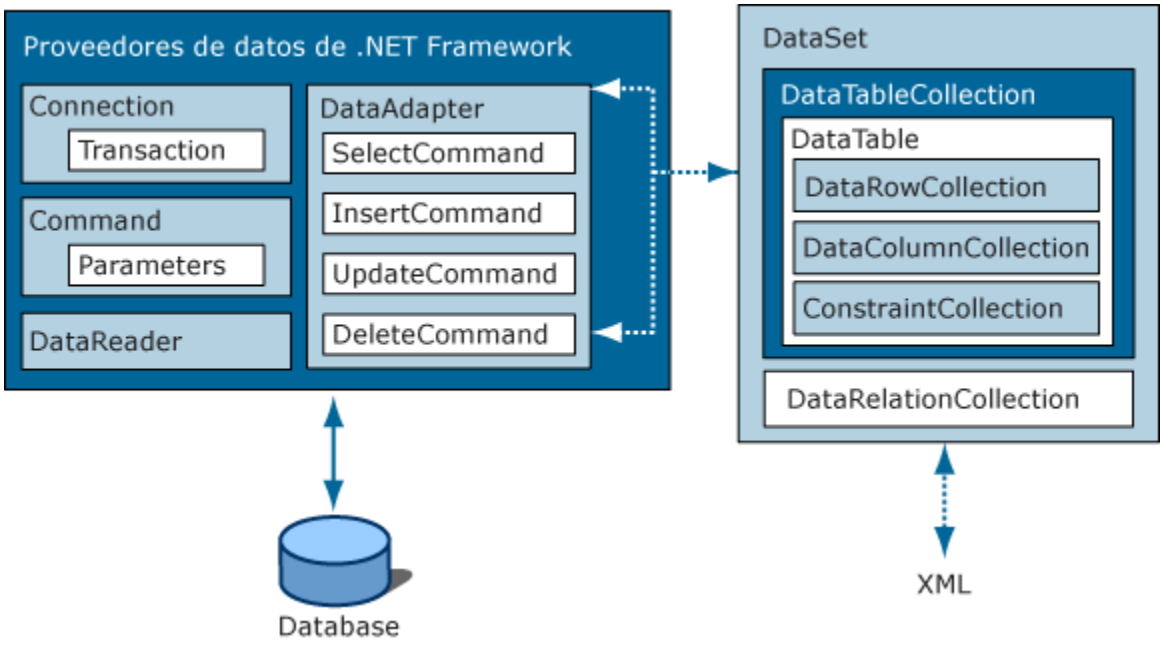


Figura 2 *Componentes principales de ADO.NET.*

1.4.8 NHibernate

NHibernate es la conversión de Hibernate de lenguaje Java a C# para su integración en la plataforma .NET. Genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos con un ligero incremento en el tiempo de ejecución.

NHibernate hace la persistencia de una manera transparente, las clases no tienen que seguir un modelo de programación restrictivo. Las clases persistentes no necesitan implementar ninguna interfaz o heredar de una clase base en especial. Esto hace posible el diseño de la lógica de negocio usando el lenguaje común en tiempo de ejecución de la plataforma .NET (Common Language Runtime o CLR) y un lenguaje orientado a objeto.

Las API (Application Programming Interface) de ADO.NET ofrecen un nivel de abstracción y funcionalidad rudimentario para acceder a casi cualquier base de datos. NHibernate hace uso de las librerías de .NET, incluyendo ADO.NET y encapsula un conjunto de componentes de

software para realizar la persistencia. A continuación se muestran sus principales componentes:

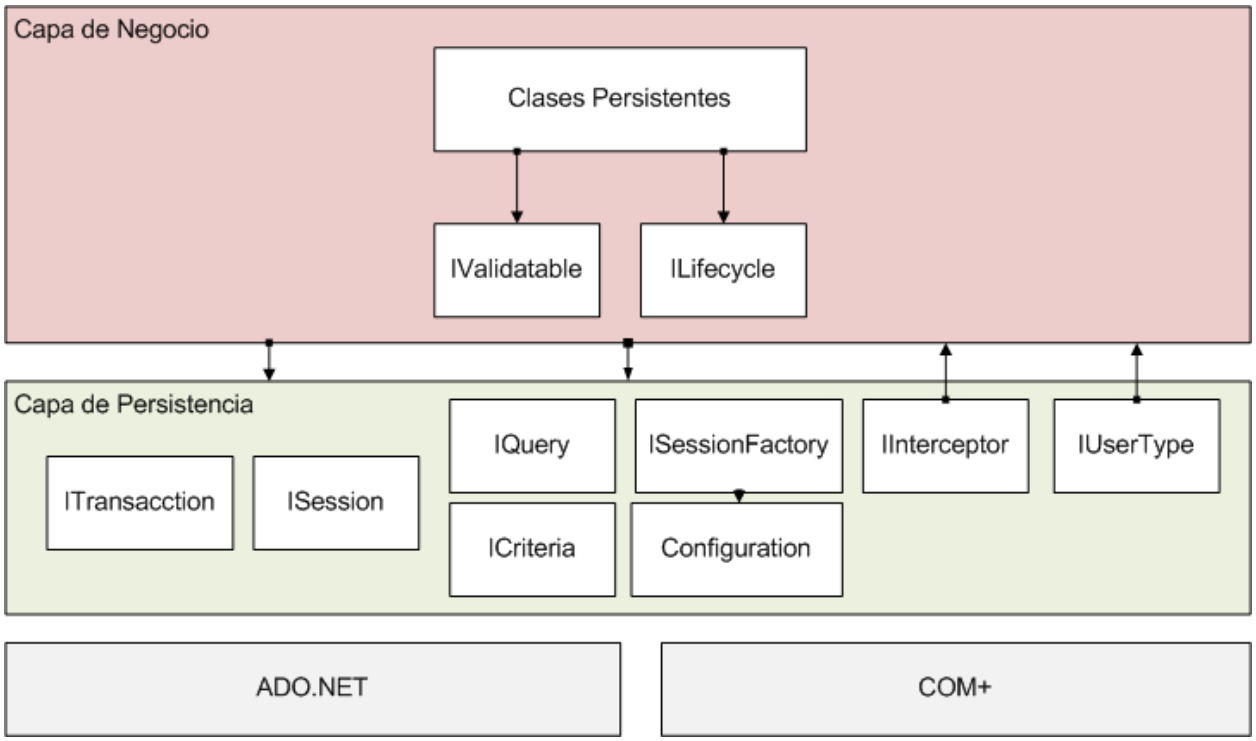


Figura 3 *Componentes generales de la API de NHibernate.*

Básicamente sus componentes se pueden clasificar en cuatro grupos:

- ✓ Componentes que son llamados por la aplicación para realizar operaciones CRUD y consultas. Estos componentes son ISession, ITransaction e ICriteria y de ellas depende básicamente el control de la lógica de negocio de las aplicaciones.
- ✓ Componentes para configurar a NHibernate, el más importante es Configuration.
- ✓ Componentes que permiten a las aplicaciones reaccionar a los eventos que ocurren dentro de NHibernate, tales como IInterceptor, Ilifecycle e IValidatable.
- ✓ Componentes para extender las funcionalidades de NHibernate, estos son IUserType, ICompositeUserType e IIdentifierGenerator.

Y las características claves son:

- ✓ Soporta elementos de los modelos orientados a objetos como son la herencia, polimorfismo, composición y colecciones incluyendo las colecciones genéricas de .NET framework.
- ✓ Brinda soporte para el mapeo de una gran variedad de objetos y colecciones.

- ✓ No realiza generación de código intermedio.
- ✓ Permite la ejecución de SQL personalizado para definir cómo persisten los objetos. En Microsoft SQL Server los procedimientos almacenados están soportados.
- ✓ Soporta la concurrencia pero no sobre todos los escenarios.
- ✓ NHibernate está licenciado bajo la licencia LGPL (Lesser GNU Public License).

1.4.9 Castle ActiveRecord

Castle ActiveRecord fue desarrollado por Castle Project, libre y de código abierto. Es una puesta en práctica del patrón de software ActiveRecord para .NET. Castle ActiveRecord se construye sobre NHibernate, pero libera al programador de escribir los XML para el mapeo entre la base de datos y los objetos, lo cual es necesario al usar NHibernate directamente.

El patrón ActiveRecord sobre el cual está basada la herramienta básicamente consiste en instancias con propiedades que representan cada columna de cada fila de las tablas de la base de datos, y métodos estáticos que actúan sobre todas las filas de manera que se encapsula el acceso a los datos y adicionan lógica de dominio a estos objetos. Es decir los objetos llevan consigo los datos y el comportamiento.

Las principales características que agrega al comportamiento de los objetos el framework:

- ✓ Create: Crea (Graba) una nueva instancia a la base de datos.
- ✓ Update: Guarda la(s) modificación (es) de una instancia ya existente en la base de datos.
- ✓ Save: Graba una instancia en la base de datos. Puede crear o actualizar una instancia dependiendo de la validez del identificador.
- ✓ Delete: Elimina una instancia existente desde la base de datos.
- ✓ Y de forma estática se implementan los siguientes métodos:
- ✓ Find: Retorna una instancia de un tipo específico según su identificador desde la base de datos.
- ✓ FindAll: Retorna todas las instancias de un tipo específico desde la base de datos.
- ✓ FindAllByProperty: Retorna todas las instancias según un valor especificado de alguna propiedad desde la base de datos.
- ✓ FindFirst: Retorna la primera instancia que se encuentra en la primera posición según un criterio de búsqueda.
- ✓ FindOne: Retorna solo una instancia según un criterio de búsqueda. En caso de encontrar más de una lanza una excepción.

- ✓ Exists: Revisa si existe alguna instancia de un tipo específico desde la base de datos.
- ✓ DeleteAll: Elimina todas las instancias de un tipo específico desde la base de datos.
- ✓ CountAll: Retorna la cantidad de instancias de un tipo específico en la base de datos.

En esencia Castle ActiveRecord toma las potencialidades de NHibernate y les realiza un recubrimiento utilizando mecanismos que le brinda el lenguaje C# y la plataforma .NET, pero sin ocultar las funcionalidades de NHibernate nativo, es decir que el programador aun es libre de elegir si usar los atributos de clases o escribir los XML para el mapeo de los datos.

Como muchas otras herramientas de su tipo, Castle ActiveRecord soporta los mecanismos de la programación orientada a objetos pero con la particularidad de que no trata de esconder el hecho de que una base de datos relacional está presente. Como resultado es muy común que generalmente se convine su uso con algún otro patrón de software.

1.5 Comparación entre IBATIS.NET y NHibernate

Parámetros	IBATIS.NET	NHibernate
Soporte a procedimientos almacenados	Si	No
Soporte a transacciones con procedimientos	Si	No
Concurrencia	Si	Si
Filosofía	Data Mapper	ORM
Licencia	Apache License 2.0	LGPL
Paginado	Si	No
Manejo de excepciones	Si	Con problemas
Soporte para Postgres	Si	Si
Logs	Si	Si
Soporte para .NET framework 1.1	Si	Si
Curva de aprendizaje	Elevada	Equipo preparado
Adecuado para el inicio del proyecto	2/3	1/3
Adecuado para el medio/fin del proyecto	3/3	0/3
Simplicidad	2/3	1/3
Grupo de funcionalidades	2/3	3/3
Documentación	1/3	2/3
Comunidad	1/3	2/3

Tabla 1 *Comparación entre IBATIS.NET y NHibernate. (15) (16)*

1.6 IBATIS.NET Solución para la Persistencia en la CAD del SIGESC 171

Existen dos posibles variantes para dar solución a los problemas presentados en la CAD basada en NHibernate 1.0 que hoy se utiliza en el SIGESC. Las variantes son utilizar una versión mejorada de NHibernate o utilizar IBATIS.NET. Para decidir cuál es la solución adecuada para el SIGESC se realizará un análisis teniendo en cuenta los elementos a favor y en contra de ambos framework. La variante de usar una versión de NHibernate mejorada por el

equipo de desarrollo facilitaría la migración debido a que los cambios sobre la CAD no tendrían un gran impacto. Se contaría con el código de la CAD prácticamente escrito, la estructura definida al igual que la integración con la arquitectura del sistema y los ficheros de mapeo solo habría que adaptarlos al nuevo gestor de base de datos que se utilice. Pero aunque se lograra modificar con éxito NHibernate no se puede asegurar que no sea necesario reprogramar algunos elementos de la CAD y los mapeos debido al cambio realizado en la herramienta. Además la modificación tendría que solucionar varios problemas que presenta la CAD del SIGESC 171 actualmente, que son:

- ✓ Las deficiencias en la ejecución de procedimientos almacenados que impiden obtener objetos con colecciones dentro.
- ✓ Falta de soporte para algunos tipos de datos retornados por los procedimientos que obligan a utilizar otras variantes.
- ✓ La falta de descripción en las excepciones que se lanzan a raíz de la ejecución de procedimientos almacenados que hacen difícil la detección de errores y la depuración de las aplicaciones.
- ✓ Errores en la ejecución de transacciones paralelas con procedimientos almacenados.

Para corregir estos problemas es necesario realizar un estudio profundo del framework NHibernatate y de otros que soportan la ejecución de procedimientos almacenados o dan solución a los problemas planteados anteriormente. Esto tomaría mucho tiempo y esfuerzo por parte del equipo de desarrollo.

Usar un framework como iBATIS.NET proporcionaría:

- ✓ Soporte completo para la ejecución de procedimientos almacenados.
- ✓ Trabajo con transacciones funcional.
- ✓ Tratamiento de errores bien clasificado y estable.
- ✓ Mecanismo de mapeo sencillo y que acelera el proceso de desarrollo considerablemente.

iBATIS.NET posee una curva de aprendizaje elevada y es adecuado en el medio y fin de la etapa de desarrollo lo que facilitaría el proceso de rescribir la CAD. La forma de inicialización y configuración de iBATIS se asemeja a NHibernate por lo que la estructura e integración con la arquitectura no sufrirán cambios dramáticos. El proceso de rescritura de procedimientos y código en la CAD no contendrá mucho esfuerzo lógico por parte de los programadores y el proceso de migración recaerá en la adaptación del equipo de desarrollo a iBATIS.

1.7 Plataforma .NET

Es un conjunto de iniciativas de Microsoft que ofrecen un modelo de programación de código administrado para la creación de aplicaciones en clientes de Windows, servidores y dispositivos móviles. Los desarrolladores pueden usar .NET para crear muchos tipos de aplicaciones: aplicaciones web, de servidor, de cliente inteligente, de consola y de base de datos. .NET es capaz de abstraer gran cantidad de lenguajes(C#, Visual Basic, Delphi (Object Pascal), C++, J#, Perl, Python, Fortran y Cobol.NET).

Microsoft .NET Framework SDK

.NET Framework simplifica el desarrollo de aplicaciones en un entorno altamente distribuido. El diseño de .NET Framework está enfocado a cumplir los objetivos siguientes:

- ✓ Proporcionar un entorno coherente de programación orientada a objetos, en el que el código de los objetos se pueda almacenar y ejecutar de forma local, distribuida en Internet o ejecutar de forma remota.
- ✓ Proporcionar un entorno de ejecución de código que reduzca lo máximo posible la implementación de software y los conflictos de versiones.
- ✓ Ofrecer un entorno de ejecución de código que garantice la ejecución segura del mismo, incluso del creado por terceras personas desconocidas o que no son de plena confianza.
- ✓ Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos en los que se utilizan secuencias de comandos o intérpretes de comandos.
- ✓ Ofrecer al programador una experiencia coherente entre tipos de aplicaciones muy diferentes, como las basadas en Windows o en el Web.
- ✓ Basar toda la comunicación en estándares del sector para asegurar que el código de .NET Framework se puede integrar con otros tipos de código.

.NET Framework contiene dos componentes principales: Common Language Runtime (CLR Motor de Tiempo de Ejecución) y la biblioteca de clases de .NET Framework. CLR es el fundamento de la tecnología. Se puede considerar como un agente que administra el código en tiempo de ejecución y proporciona servicios centrales, como la administración de memoria, la administración de subprocesos y la interacción remota. El mismo al tiempo que aplica una seguridad estricta a los tipos y otras formas de especificación del código que garantizan su seguridad y solidez. De hecho, el concepto de administración de código es un principio básico

del motor de tiempo de ejecución. El código destinado al motor de tiempo de ejecución se denomina código administrado, a diferencia del resto de código, que se conoce como código no administrado. La biblioteca de clases, el otro componente principal de .NET Framework, es una completa colección orientada a objetos de tipos reutilizables que se pueden emplear para desarrollar aplicaciones.

1.8 Lenguajes

Lenguaje C#

C# es un lenguaje de programación sencillo y moderno, orientado a objetos y con seguridad de tipos. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg. C# combina la gran productividad de los lenguajes de desarrollo rápido de aplicaciones (RAD, Rapid Application Development) con la eficacia de C++. El código de C# es compilado como código administrado, lo cual le permite beneficiarse de los servicios del CLR. Estos servicios incluyen interoperabilidad, recolección de basura, seguridad y gestión de versiones.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programar usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes. Se suele decir que C# es el lenguaje nativo de .NET.

La sintaxis y estructuración de C# es muy similar a la C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en este lenguaje a C# y su aprendizaje a los desarrolladores habituados a este. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic.

Lenguaje XML

XML (Extensible Markup Language), significa lenguaje de marcas generalizado. Es un lenguaje usado para estructurar información en un documento o en general en cualquier fichero que contenga texto, como por ejemplo ficheros de configuración de un programa o una tabla de datos. Ha ganado muchísima popularidad en los últimos años debido a que es un estándar abierto y libre. El XML fue propuesto en 1996, y la primera especificación apareció en 1998. Desde entonces su uso ha tenido un crecimiento acelerado, que se espera que continúe durante los próximos años. Hoy en día es un lenguaje por excelencia usado en casi todas las aplicaciones para representar datos en ficheros o para guardar datos en ficheros de configuración de aplicaciones.

XML constituye una manera más avanzada de representar la información, cuya principal novedad consiste en permitir compartir los datos con los que se trabaja a todos los niveles, por todas las aplicaciones y soportes. XML juega un papel muy importante en el mundo informático actual, ya que es el lenguaje que permitirá compartir la información de una manera segura, fiable y fácil. Este lenguaje de etiquetas ofrece múltiples ventajas a la hora de representar información, entre ellas se puede mencionar:

- ✓ Es extensible, por lo que una vez diseñado el documento XML y puesto en producción, igual es posible extenderlo con la adición de nuevas etiquetas de manera que los antiguos consumidores de la vieja versión todavía puedan entender el nuevo formato.
- ✓ Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarlo. Mejora grandemente la compatibilidad entre aplicaciones.

Los desarrolladores del lenguaje tuvieron unos objetivos bien marcados a la hora de diseñarlo, como por ejemplo:

- ✓ Es directamente utilizable sobre Internet y aplicaciones Web.
- ✓ Es soportado por una amplia variedad de aplicaciones.
- ✓ Es prácticamente fácil implementar editores y procesadores de código XML.
- ✓ El número de características opcionales en XML es mínimo.
- ✓ Los documentos XML son fácilmente legibles por los desarrolladores y razonablemente claros.
- ✓ Los documentos XML son fáciles de crear.

El lenguaje XML es un estándar extensible, multiplataforma y basado en texto para la representación de información de una manera estructurada y legible.

Lenguaje SQL

SQL (Structured Query Language), Lenguaje de Consultas Estructurado, es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Reúne las características del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar información de una base de datos de una forma sencilla.

El SQL explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones. Es un lenguaje que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros permite una alta productividad en codificación.

Admite fundamentalmente dos modos de uso:

- ✓ Un uso interactivo, destinado principalmente a los usuarios finales avanzados u ocasionales, en el que las diversas sentencias SQL se escriben y ejecutan en línea de comandos, o un entorno semejante.
- ✓ Un uso integrado, destinado al uso por parte de los programadores dentro de programas escritos en cualquier lenguaje de programación anfitrión. En este caso el SQL asume el papel de sublenguaje de dato.

Conclusiones

En este capítulo se han expresado los conceptos principales que se necesitan para comprender los elementos que maneja la investigación y se describen la arquitectura en capas y en especial la capa de persistencia con sus diferentes enfoques. Se expresan algunas de las soluciones de software existentes para el desarrollo de acceso a datos de los sistemas. Además se describe la plataforma y los lenguajes utilizados. También se propone una de las herramientas analizadas como parte de la solución para los problemas de la anterior CAD existentes en el SIGESC 171.

2 *Descripción de la Solución Propuesta*

En el capítulo se describen la propuesta de solución de la CAD del SIGESC 171. Se especifican los elementos de instalación y configuración de la CAD y se define la estructura de la nueva CAD. Además se especifican los problemas existentes en la CAD anterior y se plantea cómo solucionarlos. También se detallan mecanismos nuevos a usar en la CAD como es el caso del almacenamiento en caché y la creación de manipuladores de tipos personalizados.

2.1 IBATIS.NET en el SIGESC 171

IBATIS es un framework de la capa de persistencia que acepta el código SQL haciendo más fácil escribirlo y capaz de integrarse a software orientados a objetos.

El lenguaje estructurado de consulta (SQL) ha existido por mucho tiempo, desde hace 35 años que Edgar F. Codd sugirió la idea de que los datos podían ser normalizados dentro de juegos de tablas relacionadas. Muchas compañías han invertido en sistemas de administración de RDBMS, hoy en día se puede afirmar que las bases de datos relacionales y el lenguaje estructurado de consulta han soportado la prueba del tiempo, convirtiéndose en una piedra angular del desarrollo de aplicaciones.

iBATIS está basado en la idea de que hay valor en las bases de datos relacionales y el código SQL. En muchos casos se ve que una aplicación es reescrita en un nuevo lenguaje de programación y la base de datos y el código SQL se mantienen en gran parte iguales.

2.1.1 Funcionamiento

Por lo general las plataformas de desarrollo ofrecen bibliotecas para acceder a las bases de datos, ya sea a través de sentencias SQL o procedimientos almacenados. Los desarrolladores encuentran varias complicaciones a la hora de trabajar. Algunas de las complicaciones a las que se enfrentan los desarrolladores son:

- ✓ Separar el código SQL del código de programación.
- ✓ Pasar parámetros de entrada y recoger los datos de los parámetros de salida.
- ✓ Separar las clases de acceso a datos de la lógica de negocio.
- ✓ Almacenamiento en caché de los datos que son usados con frecuencia.
- ✓ Administración de las transacciones y la concurrencia.

iBATIS es un “data mapper” con algunas características de ORM que resuelve las complicaciones antes mencionadas. Martin Fowler describe el patrón Data Mapper de la forma siguiente:

Una capa de Mappers mueve los datos entre los objetos y la base de datos manteniendo independientes el uno del otro y del propio mapper. (11)

Este framework de persistencia permite mapear las consultas SQL con objetos. Las consultas SQL son desacopladas de la aplicación poniéndolas en archivos XML. IBATIS relaciona objetos con procedimientos almacenados o sentencias SQL usando un descriptor XML.

La idea básica del framework es reducir significativamente la cantidad de código que un desarrollador normalmente necesita para acceder a una base de datos relacional con el uso de archivos XML que contienen las consultas SQL. Su simplicidad es la mayor ventaja sobre otras herramientas ORM.

Su objetivo no es lograr una abstracción de la base de datos específica a utilizar, sino más bien dar control completo sobre el SQL empleado, permitiendo que el desarrollador aproveche toda la potencia de la base de datos, lo que incluye la sintaxis SQL. (15)

iBATIS para ejecutar cualquier consulta o procedimiento pasa por tres etapas principales. Cada una de ellas se resume a continuación:

- ✓ Se proveen los parámetros ya sean objetos o tipos nativos. Los parámetros pueden ser utilizados para establecer valores en tiempo de ejecución a las consultas o procedimientos almacenados. Si no se necesitan parámetros pueden ser omitidos.
- ✓ Ejecutar la funcionalidad pasando los parámetros y el nombre del mapeo que se especificó en el archivo XML correspondiente a los mapeos. iBATIS prepara la sentencia SQL o el procedimiento almacenado, establece los valores en tiempo de ejecución de los parámetros, ejecuta la sentencia o procedimiento y devuelve el resultado.
- ✓ Después que es ejecutado la sentencia o procedimiento, en caso de una actualización el número de filas es devuelto. En caso de una consulta, un objeto, o una colección de objetos son devueltos. Al igual que los parámetros, los objetos retornados pueden ser objetos, tipos primitivos o colecciones.

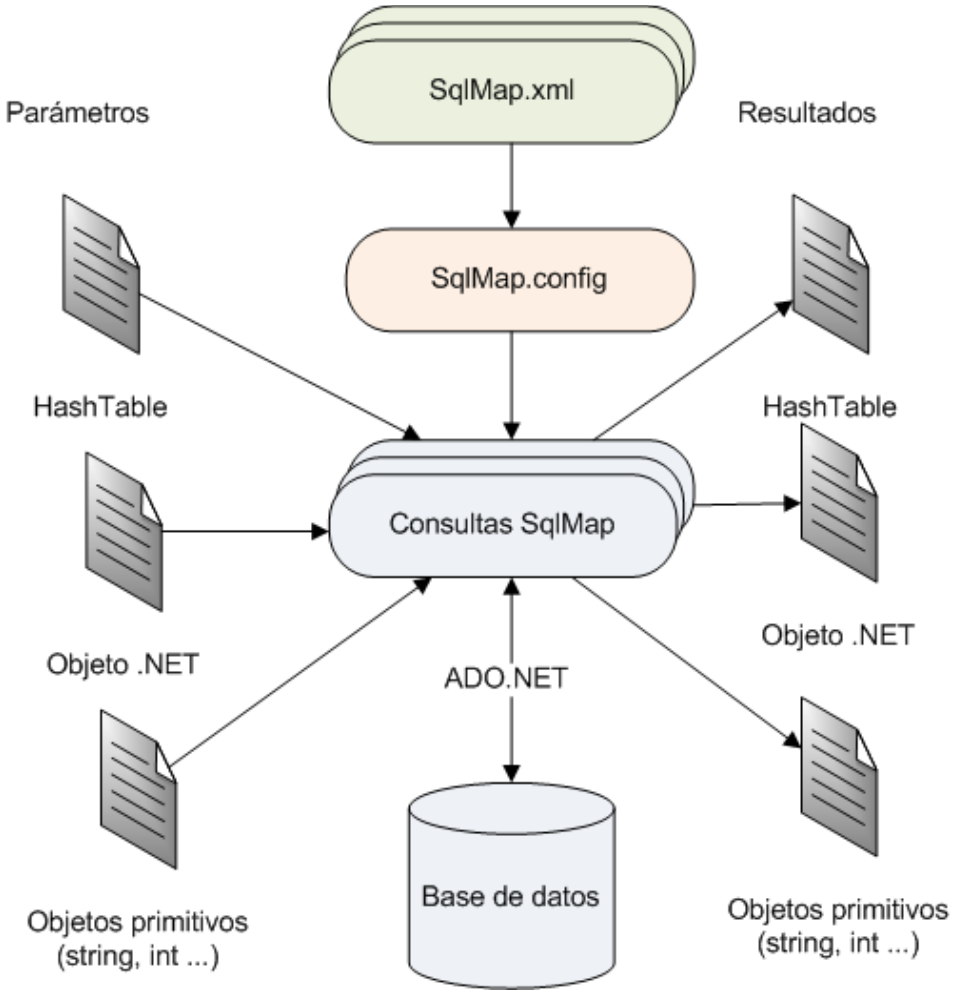


Figura 4 *Funcionamiento de iBATIS.*

2.1.2 Instalación

Para usar iBATIS es necesario incluir las referencias a los siguientes ensamblados en la CAD:

- ✓ IBatisNet.DataMapper.dll
- ✓ IBatisNet.Common.dll (implicado)
- ✓ Castle.DynamicProxy.dll (implicado)

Véase que las bibliotecas Common y DynamicProxy son implicadas, esto significa que es necesario agregarlas porque son usadas por la biblioteca DataMapper.

En la CAD del SIGESC 171 se utilizará mediante la biblioteca IbatisUtil una instancia única del DataMapper, por lo que sólo es necesario agregar la referencia a IBatisNet.DataMapper.dll. IBatisNet.Common.dll y Castle.DynamicProxy.dll se utilizan en tiempo de ejecución, pero la plataforma .NET resuelve las dependencias por sí sola.

Además se adicionan tres tipos de XML para organizar las configuraciones del framework. Estos ficheros son:

- ✓ providers.config: Es un archivo usado por el framework para buscar la definición del proveedor de base de datos.
- ✓ SqlMap.xml: Contiene la información de los mapeos. La CAD contendrá uno o varios ficheros de este tipo.
- ✓ SqlMap.config: Es el archivo de configuración de iBATIS. En él se especifica la ubicación de los SqlMap.xml y del providers.config. También se incluyen otras configuraciones como el almacenamiento en caché. Es necesario un archivo de este tipo por instancia de IBATIS, es decir por cada fuente de datos que se utilice en la aplicación.

La estructura de carpetas de la CAD y convención de nombre de estos archivos se define en el epígrafe **Especificación de la estructura de la CAD**.

Por último es necesario integrar iBATIS.NET con Visual Studio para obtener una mayor descripción de los errores en los archivos de configuración, habilitar el auto completamiento a la hora de diseñarlos y validarlos en tiempo de ejecución. Para esto es necesario copiar los archivos SqlMap.xsd, SqlMapConfig.xsd y providers, en la dirección especificada en la Tabla 2 Dirección de los esquemas. Esta dirección puede variar en dependencia del lugar donde se encuentra instalado la versión de Visual Studio y el idioma del sistema operativo.

Visual Studio.NET 2003 y Windows en Inglés
C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Packages\schemas\xml

Tabla 2 *Dirección de los esquemas.*

2.1.3 Elementos de Configuración

iBATIS se configura mediante un archivo XML que proporciona los detalles de la fuente de datos, ficheros de mapeo, y otras características como el almacenamiento en caché y transacciones. En tiempo de ejecución se invoca un método que provee la biblioteca de clases de IBATIS para leer y validar el archivo de configuración. Después de haber validado el archivo, una instancia del DataMapper es retornada. El archivo de configuración generalmente es nombrado SqlMap.config.

Los elementos que se pueden especificar para la configuración del framework son diversos. La CAD de las aplicaciones del SIGESC 171 no usan todos estos elementos, solo los que se explican a continuación.

2.1.3.1 Elemento “providers”

A un sistema de base de datos se accede a través de un proveedor. IBATIS.NET Data Mapper contiene un archivo llamado providers.config donde se almacena la descripción de los proveedores de datos soportados por el framework.

A cada proveedor es posible especificarle si se encuentra habilitado o no a través del atributo “enabled”. Además mediante el atributo “default” un proveedor se establece como por defecto, esto significa que en caso de no especificar en la configuración ninguno se usa el establecido.

En el archivo de configuración de IBATIS (SqlMap.config) los proveedores pueden ser incluidos de tres formas diferentes. La Tabla 3 Atributos del elemento "providers". explica cada una de ellas.

Atributo	Descripción
resource	Especifica la ruta relativa del archivo que contiene la configuración de los proveedores de datos.
url	Especifica la ruta absoluta donde se encentra la configuración de los proveedores de datos.
embedded	Especifica que las configuraciones van a ser cargadas de un recurso embebido dentro de un ensamblado. La sintaxis es: “[espacio de nombres.]Nombre del recurso, el nombre del ensamblado que contiene el recurso”. Por ejemplo: embedded="Recursos.providers.config, MiApp.Datos"

Tabla 3 *Atributos del elemento "providers".*

2.1.3.2 Elemento “setting”

Existen un conjunto de parámetros que son configurables en iBATIS. Estos consisten en opciones u optimizaciones para cada instancia del Data Mapper. Cada uno de estos parámetros toma un valor booleano y posee un valor por defecto, por lo que se pueden omitir el elemento “settings” o cualquiera de sus atributos. Los atributos del elemento “setting” y lo que ellos controlan se describen en la Tabla 4 Atributos para el elemento "setting"..

Atributo	Descripción
cacheModelsEnabled (por defecto true)	Habilita o deshabilita todos los modelos de caché para el Data Mapper. Es útil para la depuración.
useStatementNamespaces (por defecto false)	Con esta opción habilitada es necesario referirse a los mapeos a través de su nombre completo. Útil cuando se tienen declaraciones con identificadores iguales en diferentes archivos

	de mapeo.
validateSqlMap (por defecto false)	Habilita o deshabilita la validación de los archivos de mapeo. Es útil para la depuración.
useReflectionOptimizer (por defecto true)	Habilita o deshabilita el uso de reflexión para acceder a las propiedades de los objetos de C#.

Tabla 4 *Atributos para el elemento "setting".*

2.1.3.3 Elemento “typeAlias”

El elemento “typeAlias” permite definir nombres cortos para los tipos de datos en lugar de especificar el nombre largo para cada una de las clases. Requiere de dos atributos “alias” que es el identificador único para el tipo y “type” que es el nombre completo del tipo. La sintaxis es type="[espacio de nombre.clase], [nombre del ensamblado], Version=[versión], Culture=[cultura], PublicKeyToken=[public token]".

Existe un grupo de alias ya predefinidos en iBATIS. Se muestran en la Tabla 5 Listado de alias definidos por IBATIS.

Tipo de .NET	Alias de IBATIS
System.ArrayList	list
System.Boolean	Boolean, bool
System.Byte	Byte, byte
System.Char	Char, char
System.DateTime	dateTime, date
System.Decimal	Decimal, decimal
System.Double	Double, double
System.Guid	guid
System.Hashtable	map, hashmap, hashtable
System.Int16	Int16, short, Short
System.Int32	Int32, int, Int, integer, Integer
System.Int64	Int64, long, Long
System.SByte	SByte, sbyte
System.Single	Float, float, Single, single
System.String	String, string
System.TimeSpan	N/A
System.UInt16	Short, short
System.UInt32	Uint, uint
System.UInt64	Ulong, ulong

Tabla 5 *Listado de alias definidos por IBATIS.*

2.1.3.4 Elemento “typeHandler”

El elemento “typeHandler” permite la configuración y uso de manipuladores de tipos personalizados. Véase el epígrafe **Tipos de datos**. A cada manipulador se le especifican tres

atributos, la Tabla 6 Atributos del elemento typeHandler. muestra los nombres y las descripciones de cada uno de ellos.

Atributo	Descripción
type	Nombre del tipo a manipular
dbType	Indica el tipo de dato (del proveedor) a manejar
callback	El alias o el nombre de la clase que implementa la interfaz ITypeHandlerCallback.

Tabla 6 *Atributos del elemento typeHandler.*

2.1.3.5 Elemento "database"

En "database" se encapsulan elementos que configuran el gestor de base de datos para ser usado por el framework. Los elementos son "provider" y "datasource".

El elemento "provider" especifica el proveedor que se va a utilizar, este elemento es opcional. Pero si se desea se puede especificar el nombre de un proveedor de los que se encuentra en el archivo providers.config.

En el elemento "datasource" se especifica la cadena de conexión con que el framework se conectará a la base de datos.

2.2 Tipos de datos

Una de las formas de extender las funcionalidades de iBATIS es implementar manipuladores de tipos de datos que permiten corregir dificultades en bases de datos, drivers y/o tipos de datos no estándares.

Entre los gestores de bases de datos existen varios elementos comunes, como el lenguaje SQL que es utilizado para acceder a los datos, pero existen otros elementos que diferencian el trabajo de un SGBD a otro. Ellos implementan sus propios lenguajes de consulta y sus propios tipos de datos. Muchos tipos de datos como BLOB (binary large objects) y CLOB (character large objects) son soportados por la mayoría de las bases de datos, pero son manejados de manera diferente por cada driver.

Por tanto es difícil para iBATIS framework implementar un manipulador de tipos común para todas las bases de datos. Para lidiar con esta situación, IBATIS soporta la creación de manipuladores de tipo (en inglés type handler) que permite especificar la manera en que ciertos tipos serán manejados. A través de un manipulador de tipo se puede especificar cómo mapear de la base de datos a un tipo del CLR, incluso se pueden redefinir los tipos existentes.

2.2.1 Tipos de datos soportados por iBATIS

IBATIS soporta varios tipos de datos. En las Tabla 7 Tipos de los gestores soportados en los ResultMap y ParameterMap. y Tabla 8 Tipos del CLR soportados en los ResultMap y ParameterMap. se listan los tipos de datos que tienen un manejador de tipos ya definido en iBATIS.

CLR Type	iBATIS support	SqlDbType	OleDbType	OdbcType	OracleType
Byte[]	Yes	Binary, Image, VarBinary	Binary, VarBinary	Binary, Image, VarBinary	Raw
Boolean, bool?	Yes	Bit	Boolean	Bit	Byte
Byte, byte?	Yes	TinyInt	-	TinyInt	Byte
DateTime, DateTime?	Yes	DateTime, SmallDateTime	Date	Date, DateTime, SmallDateTime, Time	DateTime
char, char?	Yes	Not supported	Char	Char	Byte
Decimal, decimal?	Yes	Decimal, Money, SmallMoney	Decimal, Currency, Numeric	Decimal, Numeric	Number
Double, double?	Yes	Float	Double	Double	Double
Guid, Guid?	Yes	UniqueIdentifier	Guid	UniqueIdentifier	Raw
Int16, Int16?	Yes	SmallInt	SmallIInt	SmallInt	Int16
Int32, Int32?	Yes	Int	Integer	Int	Int32
Int64, Int64?	Yes	BigInt	BigInt	BigInt	Number
Single, Single?	Yes	Real	Single	Real	Float
String	Yes	Char, Nchar, NVarchar, Text, VarChar	Char, VarChar	Char, NChar, NText, NVarChar, Text, VarChar	NVarChar, VarChar
TimeSpan	No	Not supported	DBTime	Time	DateTime
UInt16, UInt16?	yes	Int	-	-	UInt16
UInt32, UInt32?	yes	Decimal	-	-	UInt32
UInt64,	yes	Decimal	-	-	Number

UInt64?

Tabla 7 *Tipos de los gestores soportados en los ResultMap y ParameterMap.*

CLR Type	Object/Map Property Mapping	Result Class/Parameter Class	Type Alias
System.ArrayList	Yes	Yes	list
System.Boolean	Yes	Yes	Boolean, bool
System.Byte	Yes	Yes	Byte, byte
System.Char	Yes	Yes	Char, char
System.DateTime	Yes	Yes	dateTime, date
System.Decimal	Yes	Yes	Decimal, decimal
System.Double	Yes	Yes	Double, double
System.Guid	Yes	Yes	guid
System.Hashtable	Yes	Yes	map, hashmap, hashtable
System.Int16	Yes	Yes	Int16, short, Short
System.Int32	Yes	Yes	Int32, int, Int, integer, Integer
System.Int64	Yes	Yes	Int64, long, Long
System.SByte	Yes	Yes	SByte, sbyte
System.Single	Yes	Yes	Float, float, Single, single
System.String	Yes	Yes	String, string
System.TimeSpan	Yes	Yes	N/A

System.UInt16	Yes	Yes	Short, short
System.UInt32	Yes	Yes	Uint, uint
System.UInt64	Yes	Yes	Ulong, ulong
Nullable<bool>	Yes	Yes	bool?
Nullable<byte>	Yes	Yes	byte?
Nullable<char>	Yes	Yes	char?
Nullable<DateTime>	Yes	Yes	DateTime?
Nullable<decimal>	Yes	Yes	decimal?
Nullable<double>	Yes	Yes	double?
Nullable<Int16>	Yes	Yes	Int16?
Nullable<Int32>	Yes	Yes	Int32?
Nullable<Int64>	Yes	Yes	Int64?
Nullable<SByte>	Yes	Yes	SByte?
Nullable<Single>	Yes	Yes	Single?
Nullable<UInt16>	Yes	Yes	UInt16?
Nullable<UInt32>	Yes	Yes	UInt32?
Nullable<UInt64>	Yes	Yes	UInt64?

Tabla 8 *Tipos del CLR soportados en los ResultMap y ParameterMap.*

2.2.2 Interfaz ITypeHandlerCallback

Para implementar un manipulador de tipo solamente se necesita implementar parte de sus funcionalidades, que están definidas en una interfaz llamada ITypeHandlerCallback.

La interfaz ITypeHandlerCallback permite especificar cómo se manipulará el tipo personalizado antes de que los parámetros sean establecidos en el IDbCommand y después que los valores sean devueltos por el IDataReader.

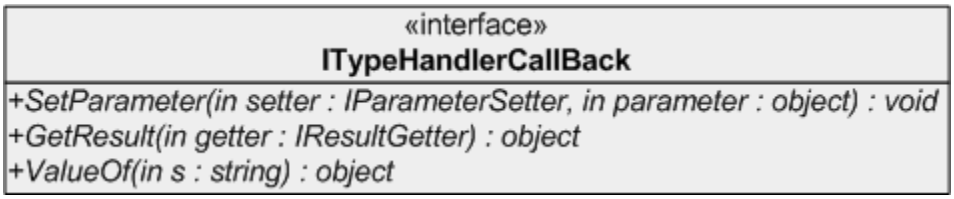


Figura 5 *Interfaz ITypeHandlerCallback.*

SetParameter toma dos argumentos, la interfaz IParameterSetter para cambiar los valores en el IDbCommand y el valor a ser cambiado. El método describe cómo se procesará el valor de un parámetro antes de ser añadido los parámetros del IDbCommand. Permite hacer cualquier conversión necesaria antes de que IBATIS comience a trabajar. Si se necesita se puede tener acceso a los parámetros mediante la propiedad setter.DataParameter.

GetResult toma un argumento de la interfaz IResultGetter, para obtener el valor desde el IDataReader. El método permite procesar el valor del resultado devuelto por la base de datos justamente después que ha sido recuperado por IBATIS y antes de ser utilizado en su resultClass, resultMap o listClass. Si se necesita se puede tener acceso a los IDataReader mediante la propiedad getter.DataReader.

ValueOf permite comparar la representación de un valor en forma de cadena de caracteres con un valor esperado y que puede ser manejado. Este método se utiliza para traducir valores nulos en tipos que pueden ser comparados. Si el manipulador de tipo no soporta valores nulos o no existe una representación en forma de cadena razonable para él (por ejemplo, tipo archivo) se devuelve la cadena dada. Cuando se encuentra un valor inesperado se lanza la excepción apropiada para el caso.

2.2.3 Registrar un manipulador personalizado de un tipo de dato

Existen dos maneras de configurar un manipulador personalizado para que pueda ser utilizado por IBATIS.

- ✓ Agregar el manipulador dentro de la propiedad <alias> del fichero de mapeo (Sqlmap.xml) en el cual se va a usar. De esta manera se le especifica explícitamente a IBATIS que utilice el manipulador para un resultado dentro de un parameterMap o resultMap determinado.

- ✓ Agregar el manipulador a la configuración de IBATIS (Sqlmap.config) dentro de la propiedad <typeHandlers>. De este modo el framework utiliza el manipulador para cualquier parameterMap o resultMap que lo referencien.

2.2.4 Solución

Teniendo en cuenta los elementos analizados anteriormente, se propone la solución para la migración de los tipos de datos. Todos los tipos de datos primitivos que anteriormente NHibernate daba soporte continuarán siendo manipulados por iBATIS de igual forma lo que posibilita que la integración entre la CAD y la Capa de Negocio no sufra cambios. Existen varios casos en los cuales iBATIS brinda una mejor solución para el trabajo con los tipos de datos:

- ✓ Enumerativos: En NHibernate es necesario utilizar otros mecanismos para lograr el mapeo de los enumerativos. Sin embargo iBATIS ofrece soporte a través de los manipuladores de tipos.
- ✓ Booleanos: Es necesario tratarlos como valores enteros usando NHibernate mientras que iBATIS les brinda soporte.
- ✓ BLOB y CLOB: La versión de NHibernate utilizada en el sistema no ofrece soporte a estos tipos de datos. iBATIS mapea estos tipos como arreglos de byte lo cual no requiere ninguna implementación adicional.

2.3 Especificación de la estructura de la CAD

Para definir la estructura de la CAD utilizando iBATIS es necesario mantener algunos aspectos existentes en la solución anterior que vienen dado por la arquitectura definida para el sistema y las particularidades para NHibernate. Entre estos elementos se encuentran la arquitectura de los módulos, la integración entre capas y el flujo de información entre ellas.

En el SIGESC 171 los módulos son conjuntos de funcionalidades agrupadas que dan solución a una lógica de negocio determinada. Un módulo es totalmente independiente y está constituido por diferentes capas. (5)

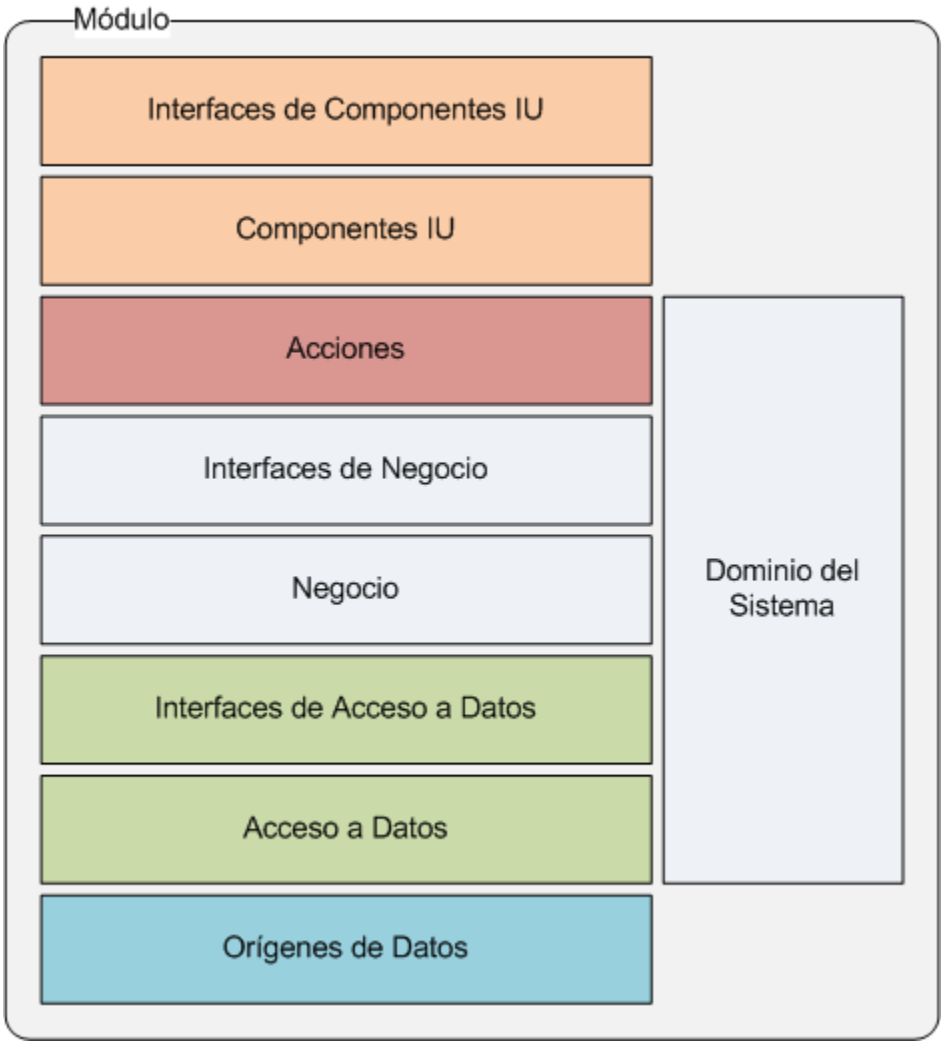


Figura 6 *Estructura de un Módulo.*

2.3.1 Capa de Acceso a Datos

Cada capa dentro de un módulo tiene una arquitectura bien definida y similar en todos los casos. Los puntos fundamentales que definen a cada una de las capas son la integración entre ellas, el flujo de información y su configuración.

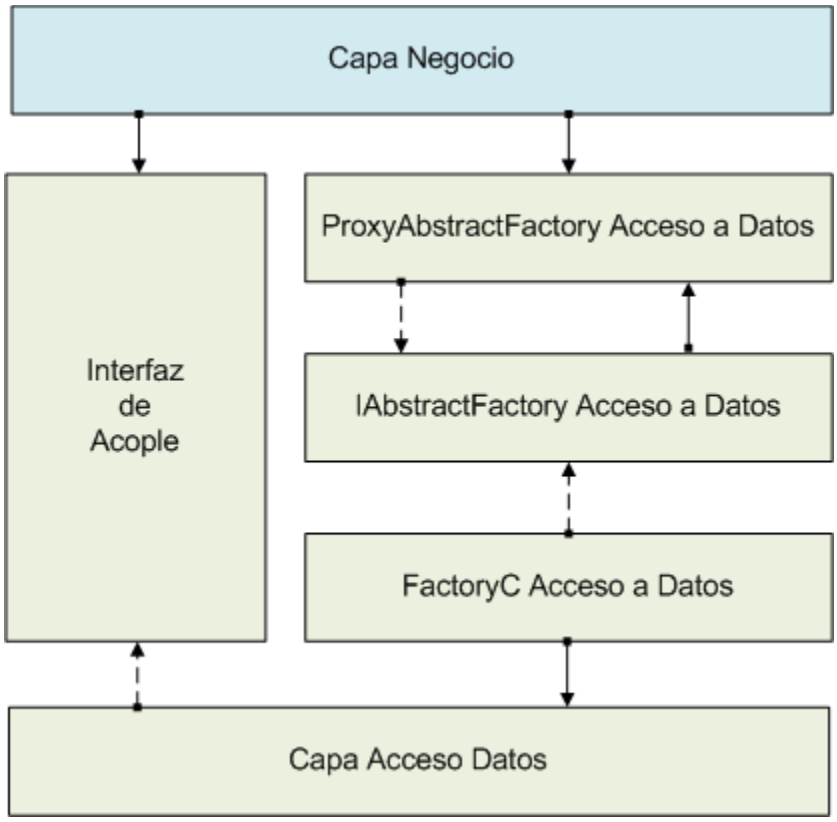


Figura 7 *Vista general de la Capa de Acceso a Datos. (5)*

2.3.1.1 Integración entre CAD y Capa de Negocio

La integración entre capas trata fundamentalmente el problema de lograr que las capas sean totalmente desacopladas, con el objetivo de lograr el mantenimiento, el desarrollo en paralelo y una mayor cohesión. La arquitectura del SIGESC 171 especifica para lograr estos objetivos la aplicación de un conjunto de patrones de diseño.

- ✓ Fachada: Cada capa expondrá una fachada (interfaz de acople) con el conjunto de funcionalidades que necesita la capa inmediata superior.
- ✓ AbstractFactory: En la fachada se definirán fábricas abstractas, por familias de objetos, con el objetivo de definir (no implementar) cómo se van a agrupar los objetos de dicha capa para ser creados por una fábrica concreta que forma parte de la capa en cuestión.
- ✓ Puente: Brinda un mecanismo dinámico, utilizando reflexión, para obtener objetos.
- ✓ Proxy (proxy remoto): es el encargado de abstraer la forma en que se obtiene la fábrica concreta utilizando un “Puente”.
- ✓ Factory: El proxy implementará el patrón “Factory” para obtener el “Puente”, permitiendo cambiarlo de ser necesario.

2.3.1.2 Flujo de Información entre CAD y Capa de Negocio

Teniendo en cuenta que en una arquitectura en capa la capa superior conoce a la inferior el flujo de información en la Capa de Acceso a Datos es solo en una dirección, de arriba hacia abajo (de Negocio a Acceso a Datos). La comunicación se realiza a través de métodos que expone la CAD y la capa de negocio los utiliza.

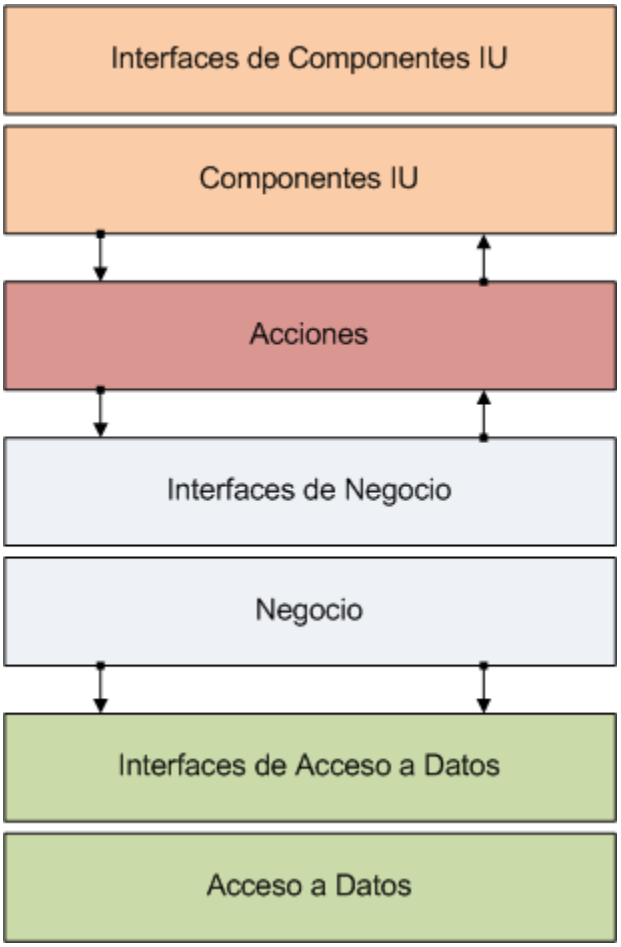


Figura 8 *Flujo de información. (5)*

2.3.1.3 Ubicación de las configuraciones en la CAD

La nueva solución se ajusta a las funcionalidades de iBATIS, dejando a un lado la anterior definición dado que fue creada bajo la filosofía de funcionamiento de NHibernate.

La configuración de la CAD se especifica mediante ficheros XML que brindan la información acerca de los proveedores de datos, la base de datos y los mapeos. Los archivos están estructurados dentro de la CAD de la siguiente manera.

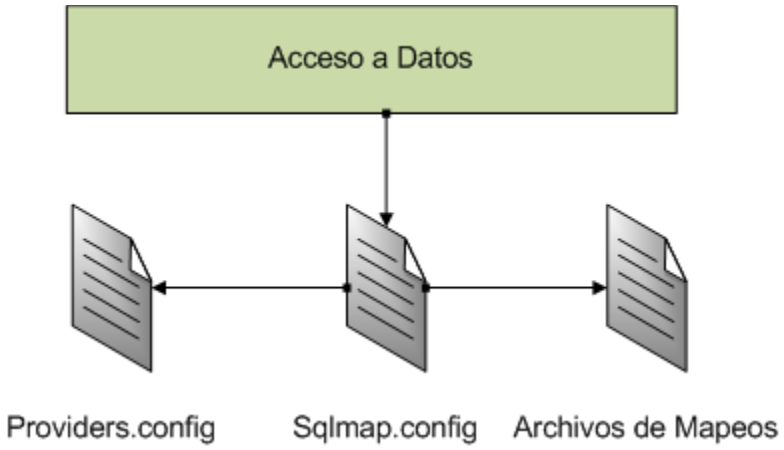


Figura 9 *Configuración de la CAD.*

2.3.2 Estructura de directorios de la CAD

La estructura general (ver Anexo 5 Nueva estructura de la CAD de un módulo del SIGESC.) que se propone para la CAD de cada módulo de SIGESC 171 contiene los siguientes elementos:

- ✓ Clases de Transferencia y Clases de Acceso a Datos.
- ✓ Configuraciones.
- ✓ Manipuladores de tipos personalizados.

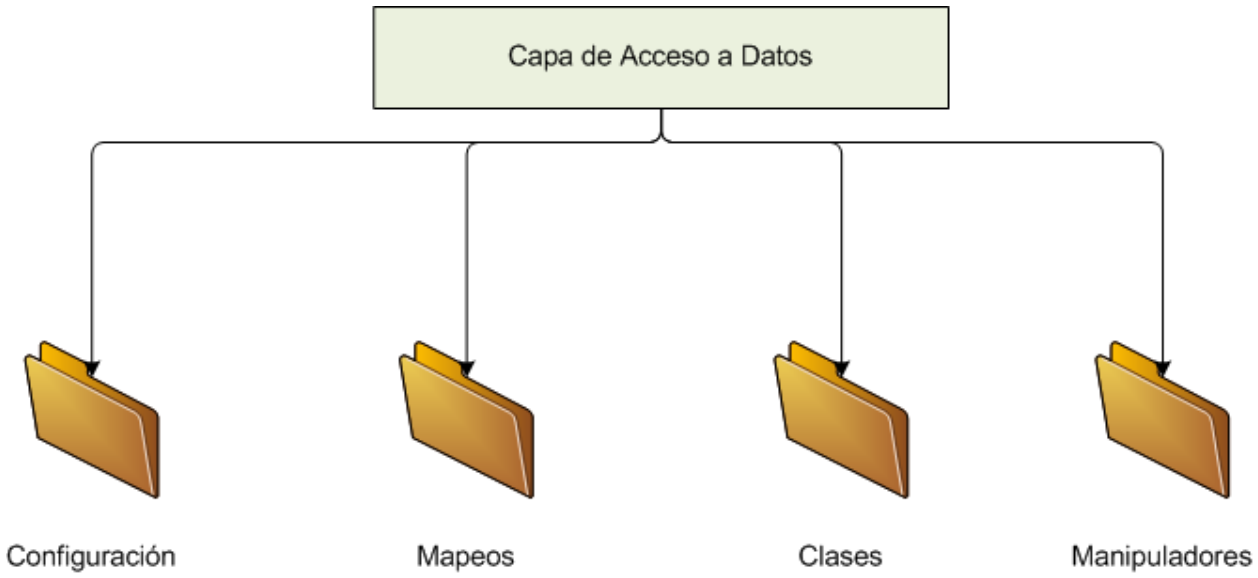


Figura 10 *Estructura de carpetas de la CAD.*

2.3.2.1 Clases de Transferencia y Acceso a Datos

Las clases de transferencia surgen debido a la diferencia que existe entre el modelo de Domino (así sea de sistema o de módulo) con respecto a la información obtenida de la Base de Datos. Esta diferencia aumenta además por la lógica de procedimientos del sistema. Es por eso que es necesario aplicar el patrón DAO.

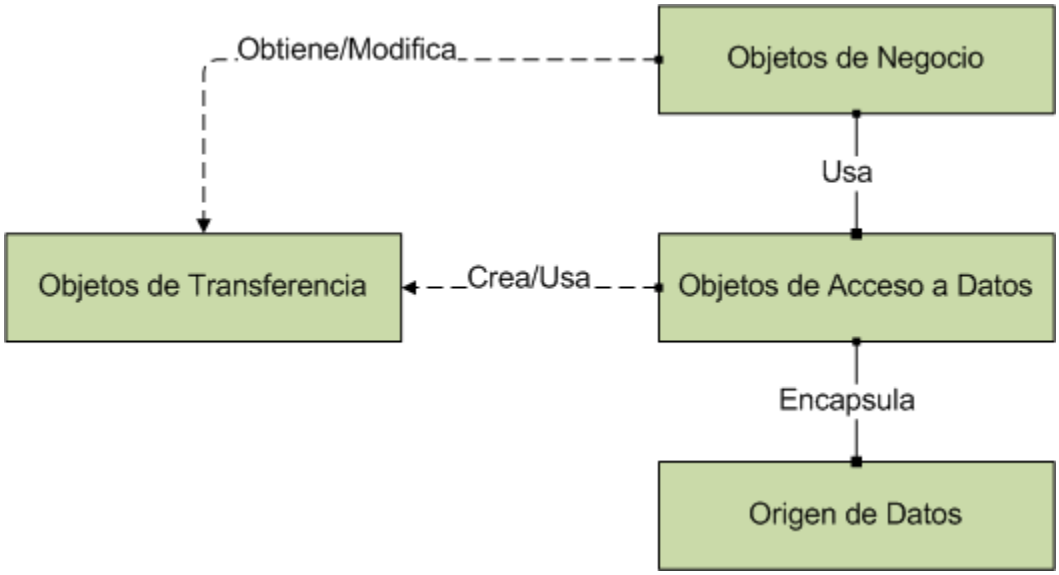


Figura 11 *Implementación del Patrón DAO.*

- ✓ Objetos de Negocio: Es el objeto que necesita acceder a la fuente de datos para poder almacenar o consultar datos. Es decir los objetos que se comunican con la CAD a través de los proxies de acceso a datos.
- ✓ Objetos de Acceso a Datos: Abstraen a los Objetos de Negocio de los detalles del acceso a la fuente de datos. Son las clases de acceso a datos.
- ✓ Origen de Datos: Representa la implementación de la fuente de datos en sí. El origen de datos del cual se extrae información, es decir la Base de Datos en concreto.
- ✓ Objetos de Transferencia: Es un objeto intermedio entre los Objetos de Negocio y los Objetos de Acceso a Datos.

2.3.2.2 Configuraciones

En la CAD existen dos grupos de configuraciones, configuraciones del framework de persistencia y de los archivos de mapeo. Anteriormente se definió la estructura y ubicación de estas configuraciones dentro de la CAD.

- ✓ Configuraciones del framework de persistencia: Las configuraciones del framework consisten en dos archivos XML denominados sqlmap.config y providers.config. En estos archivos se especifican las configuraciones del framework y de los proveedores de datos. Ver epígrafe **Elementos de Configuración**.

  Archivos de Mapeo: Los archivos de mapeo se organizan por cada clase de acceso a datos. Entre ellos pueden existir relación, es decir se puede referenciar un mapeo que exista en otro archivo. Lo anterior permite establecer agrupaciones entre ellos, aunque se debe evitar el uso abusivo de esta funcionalidad. Por lo general lo más práctico sería agrupar los mapeos comunes en un mismo archivo. Ver epígrafe Capítulo 2: Descripción de la Solución Propuesta
- ✓ IBATIS.NET en el SIGESC 171.

Mantener agrupados todos los archivos de mapeo evita que estén esparcidos por toda la estructura de la CAD. Los archivos de mapeo no se deben agrupar junto con las clases de acceso a datos porque estos trabajan por separado y menos agruparlos con otros archivos de configuración. Organizarlos de esa manera complicaría la configuración. Los archivos de mapeo no tienen una categorización interna por la cual es necesario agruparlos según su funcionalidad y usando nombres sugerentes. Este enfoque ayuda a navegar a través de los archivos de mapeos y comprender su lógica.

2.3.3 Convenciones de Nombres

En IBATIS hay varias cosas que necesitan un nombre, por ejemplo statements, result maps, parameter maps, SQL maps, y archivos XML. Por tanto es necesario definir pautas para nombrar estos elementos.

2.3.3.1 Nombres de los Statements

Los statements deben seguir las mismas convenciones que se usan para nombrar los métodos en el lenguaje de programación que se esté utilizando. En caso de C# un ejemplo sería ActualizarPersona o DetallesDePersona. Usar esta convención no solo ayuda a mantener la consistencia, sino que también permite vincular los métodos con los statements.

2.3.3.2 Nombre de los Parameter Maps

Por lo general los parameter maps no son muy utilizados, pues en las consultas en línea no es necesario proveerles un nombre; pero en el SIGESC 171 todo el acceso a los datos se realiza a través de procedimientos almacenados lo cual aumenta el uso de los parameter maps. Por lo

general no se utilizan los mismos parámetros para la selección, modificación o inserción por lo que es recomendable añadir un sufijo “Param”. Por ejemplo:

<select id="ObtenerPersona" parameterMap="ObtenerPersonaParam" ... >

2.3.3.3 Nombre de los Result Maps

Los Result Maps están sujetos a un tipo de dato y su reutilización es muy frecuente. Por esta razón se recomienda usar el nombre del tipo al cual estén asociados y además añadirles el sufijo “Result”. Por ejemplo:

<resultMap id="PersonaResult" type="Persona">

2.3.3.4 Nombres de los CacheModels

Los CacheModels son configuraciones de los diferentes tipos de caché que se utilizarán. Para nombrarlos se comenzará con un nombre que se asocie a los Statement que lo utilizan y se le agrega el sufijo “Cache”. Por ejemplo:

<cacheModel id="UsuarioCache" implementation="LRU">

2.3.3.5 Nombres de los archivos XML

Anteriormente habías clasificado los archivos de configuración en dos formas, archivos de mapeo y archivos de configuración del framework.

El archivo principal de configuración de framework puede llamarse como se desee, pero se recomienda llamarlo Sqlmap.config. Si se tienen diferentes archivos de configuración como es el caso del SIGESC 171 es necesario añadirle como prefijo el nombre del módulo al cual pertenecen. Por ejemplo LlamadaSqlMap.config.

Los archivos de mapeo como convención van a estar organizados por cada clase de acceso a datos, de esta manera lo correcto sería nombrarlos como la clase que los involucra. Por ejemplo ADGestionarUsuario.xml. En caso de estar trabajando con más de un origen de datos pudiera añadirse como prefijo el nombre de este. Por ejemplo PostgresADGestionarUsuario.xml.

2.3.3.6 Nombre de los Manipuladores de Tipo

Los manipuladores de tipo que se definan en la CAD se deben agrupar en una misma ubicación. Para nombrarlos solo es necesario incluirle como sufijo la palabra “TypeHandler” después del nombre que el desarrollador seleccione.

2.4 Sesiones y Transacciones

Las sesiones en iBATIS son un contenedor para conexión y una transacción de ADO.NET. En IBATIS la interfaz IDalSession implementa la interfaz IDisposable lo que permite utilizar la sintaxis using de C#. Las sesiones no pueden anidarse, en caso de una llamada a un BeginTransaction u OpenConnection dentro de un mismo hilo más de una vez, esto provocará una excepción. En otras palabras cada hilo puede tener a lo sumo una sesión abierta por instancia de SqlMapper.

Las API de IBATIS ofrecen métodos para demarcar los límites de una conexión, estos son OpenConnection y CloseConnection. Por defecto cuando se invoca cualquier método de las API de iBATIS se abre y cierra una conexión automáticamente. Esto significa que cada llamada a estos métodos es una simple unidad de trabajo. Para muchos casos esto puede ser suficiente, pero cuando se desea que varias instrucciones sean ejecutadas como una sola unidad de trabajo, es decir que fallen o tengan éxito como grupo, es necesario una transacción explícita.

Una transacción en términos simples es una unidad de trabajo que generalmente involucra varias operaciones que deben tener éxito o fallar como grupo. Es decir si una acción en la transacción falla todas las demás se deben deshacer y dejar los datos en un estado consistente.

iBATIS soporta varios ámbitos de transacciones:

- ✓ Automáticas: Para acciones simples que no requieran explícitamente de una transacción.
- ✓ Locales: Simple y de ámbito estricto que involucra varias acciones, pero solamente una base de datos.
- ✓ Globales: Compleja y de amplio ámbito que involucra varias acciones y varias Bases de Datos.

2.4.1 Características de las transacciones

Existen ciertas propiedades o características para un sistema capaz de manejar el procesamiento de transacciones, conocidas como ACID (por sus siglas en inglés Atomicity, Consistency, Isolation and Durability). Consisten en:

- ✓ Atomicidad: es la característica que garantiza que todas las acciones en una transacción tengan éxito o fracasen como un mismo grupo.

- ✓ Coherencia: está relacionado con las restricciones que ponen los esquemas de base de datos para asegurar la integridad. La coherencia requiere que antes o después de una transacción la base de datos se encuentre en un estado coherente. Una base de datos se encuentra en un estado coherente cuando las restricciones de integridad, llaves foráneas y llaves únicas se cumplen.
- ✓ Aislamiento: es la propiedad que garantiza que las transacciones no entren en conflicto unas con otras.
- ✓ Durabilidad: define que después de ejecutada una transacción los datos deben ser seguros. Incluso si se produce un fallo del sistema después de la operación, los datos deben ser seguros.

2.4.2 Transacciones Automáticas

Muchos de los drivers para conectarse a las bases de datos implementan el "autocommint mode". Este se encarga de reafirmar automáticamente cada una de las operaciones que se ejecuten sobre el servidor de base de datos. IBATIS no soporta este modo directamente, en cambio permite el uso de las transacciones automáticas. Las transacciones automáticas permiten ejecutar una operación sobre el servidor sin preocuparse de demarcar las transacciones. La operación se ejecutará en una transacción pero el desarrollador no tiene que preocuparse explícitamente de comenzar, completar o terminar con ella.

2.4.3 Transacciones Locales

Son el tipo más común de transacciones y las que generalmente se usan. Una transacción local es una transacción que está contenida dentro de una aplicación e involucra solamente a un recurso, que generalmente es una base de datos relacional.

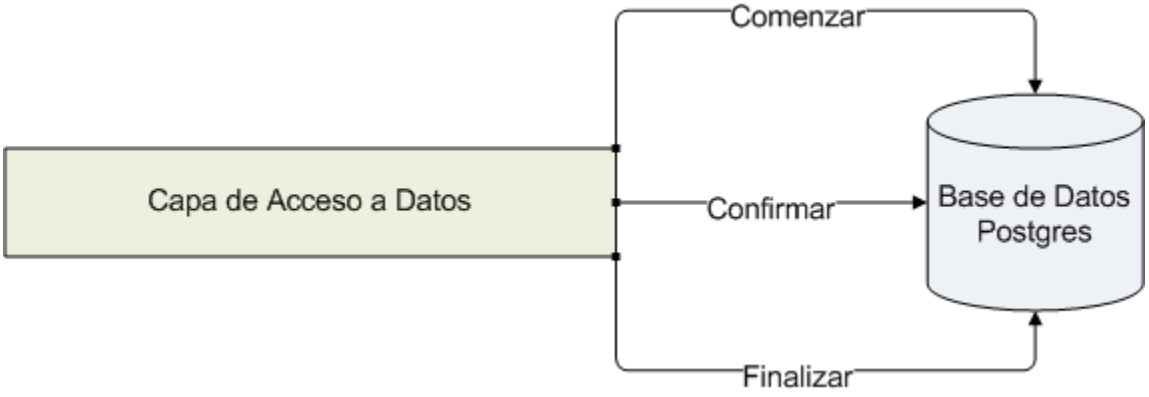


Figura 12 *Transacción Local.*

2.4.4 Transacciones Globales

El SIGESC 171 cuenta con una base de datos centralizada por lo cual parecería que el uso de transacciones globales no es necesario, pero no es así. Como el SIGESC es un sistema que ofrece servicios de manera ininterrumpida, las transacciones globales pueden servir de ayuda en caso de la migración de servidor de base de datos. Parte de los datos pueden estar almacenados en el servidor antiguo y otra parte en el nuevo servidor sin alterar la integridad de los datos. Todo ello puede ser posible ya que las transacciones globales pueden involucrar varias bases de datos. Afortunadamente IBATIS no agrega ninguna complejidad adicional a la hora de usar las transacciones globales.

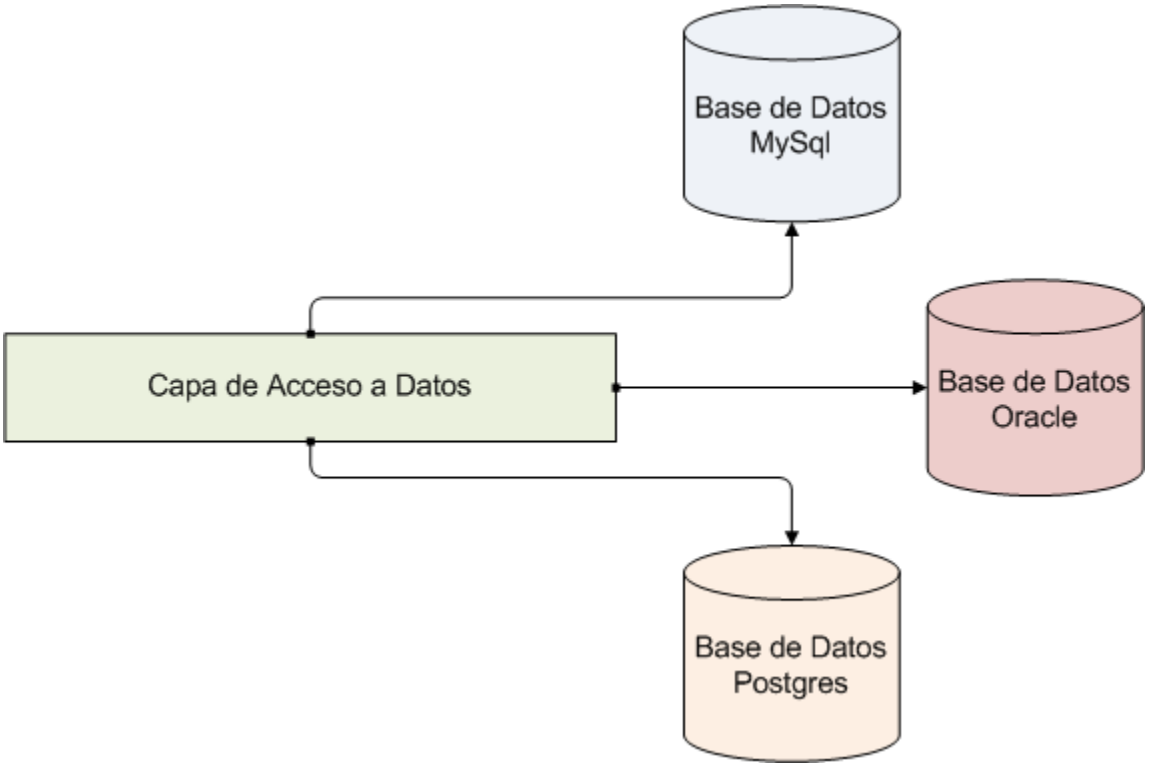


Figura 13 *Transacción global.*

2.4.5 Solución

Teniendo en cuenta los elementos analizados anteriormente se propone la solución para el problema de las transacciones. Para los escenarios presentes en el sistema el tipo de transacción que se utilizará con iBATIS son las transacciones locales. Este tipo de transacciones es la que más se asemeja a la utilizada con NHibernate. Su uso es sencillo,

solamente es necesario especificar el momento de iniciar, reafirmar y cancelar la transacción con los métodos correspondientes brindados por iBATIS.

2.5 Almacenamiento en Caché

En iBATIS el almacenamiento en caché se centra en los datos obtenidos por la capa de persistencia. El mecanismo es muy simple y robusto, totalmente basado en configuraciones que eliminan lo engorroso de manejar la caché directamente.

La mayoría de las cachés que los desarrolladores incorporan en sus aplicaciones son datos que no cambian con frecuencia. Estos datos son los que se ven con frecuencia en los menús desplegables o listas de selección, como son los nomencladores y búsquedas. Pero la caché va más allá de datos de solo lectura, también se pueden almacenar objetos para leer y escribir sobre ellos. Como toda tecnología la caché introduce algunos problemas, el principal es cuándo saber que el contenido almacenado en la memoria es antiguo o reciente. Es bastante fácil escribir reglas simples como el tiempo que tarda la caché en limpiarse, pero no es tan fácil cuando la ejecución de varios procesos deberían invalidar el contenido de la memoria caché. A medida que crece la relación entre el código que está en ejecución y los datos almacenados en memoria se hace más difícil mantener la integridad de la caché. Cuando se llega a este punto de complejidad el problema de rendimiento que resuelve la caché se torna menos convincente y se comienza a ver como una tarea molesta. Por tal motivo iBATIS se centra en proveer estrategias e implementaciones de caché para la CAD solamente y lo hace a través de configuraciones fáciles de manejar.

Considerando la filosofía de iBATIS y su diferencia con las demás soluciones de persistencia que existen, la memoria caché puede llegar a ser un punto delicado para quienes usan herramientas ORM. iBATIS está construido sobre la idea de mapear consultas a objetos no de mapear tablas de la base de datos a objetos. Los mecanismos de caché de los ORM tradicionales como mapean las tablas a objetos mantienen la identificación (OID) al igual que las base de datos mantienen la identificación de una fila dentro en una tabla de la base de datos. Esto significa que si dos resultados diferentes devuelven el mismo objeto, el objeto se almacena en caché una sola vez. En iBATIS no es así porque es un framework centrado en los datos es decir se centra en los resultados de las consultas y no en la identificación de los objetos.

Los modelos de caché son los encargados en iBATIS de ayudar a evitar el manejo manual de los resultados que se encuentran en la memoria caché y las dependencias de estos resultados.

2.5.1 Modelos de Caché

Un modelo de caché es una configuración de caché, más específicamente es la base sobre la cual todas las implementaciones de caché en iBATIS están definidas. Los modelos de configuración de la caché se define dentro de un archivo de mapeo y puede ser utilizado por más de una consulta.

Una configuración de un modelo de caché se define dentro de un elemento cacheModel que contiene los siguientes atributos:

- ✓ Id (requerido): Este valor especifica el identificador exclusivo que se hace referencia en las consultas que usen este modelo de caché.
- ✓ Implementation (requerido): Tipo de caché que se desea utilizar. Algunos alias predefinidos son MEMORY, LRU y FIFO.
- ✓ ReadOnly (opcional): Cuando se establece a true, esto indica que el caché se utiliza únicamente como una caché de sólo lectura. Los objetos recuperados de una caché de sólo lectura no deberían permitir que sus propiedades cambien.
- ✓ Serialize (opcional): Especifica cómo los datos deben ser copiados después de obtenidos. Si se pone en true significa que se hace una copia de los objetos devueltos.

2.5.1.1 Atributo Implementation

Los modelos de caché utilizan un mecanismo extensible para soportar diferentes tipos de caché. El tipo de caché que se desea utilizar se especifica en el atributo “Implementation” del elemento “cacheModel”. El nombre de la clase que se especifique debe implementar la interfaz ICacheController o ser uno de los tres alias predefinidos por el framework. Al tipo de caché se le pueden adicionar algunos parámetros de configuración mediante el elemento “property” que se encuentra dentro del cuerpo de “cacheModel”.

Los tres alias predefinidos en la distribución de .NET son:

- ✓ MEMORY: Utiliza las referencias a los objetos para manejar el comportamiento de la caché. Es decir, el recolector de basura es quien determina qué objeto se mantiene en la memoria caché o cuáles se eliminan. Solamente se le puede aplicar una propiedad nombrada “reference-type” que puede tomar diferentes valores. Ver Tabla 9 Tipos que se pueden especificar para MEMORY..
- ✓ LRU: Utiliza el algoritmo LRU (Menos Recientemente Usado) para determinar cómo los objetos se eliminan automáticamente de la caché. Cuando se llena la caché se elimina

el objeto que hace más tiempo no se accede. De esta manera los objetos que se usan más a menudo permanecerán en memoria sin riesgo de ser eliminados. Es recomendable usar esta estrategia para listados paginados, enumerativos y claves de búsquedas. Solamente se le puede aplicar una propiedad nombrada “CacheSize”, que es un entero que indica la cantidad de objetos que se pueden almacenar en caché al mismo tiempo. Es válido aclarar que un objeto puede ser desde una cadena hasta una colección, por lo que es necesario tener cuidado para no desatar una excepción de límite de memoria.

- ✓ FIFO: Utiliza un algoritmo de FIFO (Primero en entrar, Primero en Salir) para determinar cómo los objetos se eliminan de la caché. Cuando la caché se llena en su totalidad se elimina el objeto más antiguo en la caché. Este tipo es recomendable para consultas que se realizan en un corto periodo de tiempo de manera muy seguida y después no se vuelven a ejecutar dentro de un tiempo determinado. Solamente se le aplica la propiedad “CacheSize” y se corren los mismos riesgos que en el tipo LRU.

Tipo	Descripción
WEAK (por defecto)	Este tipo es probablemente la mejor elección en la mayoría de los casos. Además de aumentar el rendimiento de los resultados populares, libera la memoria para ser utilizada en la asignación de otros objetos asumiendo que los resultados no estén en uso.
SOFT (solo en Java)	Este tipo reducirá el riesgo de quedarse sin memoria en caso de que los resultados no están actualmente en uso y la memoria es necesaria para otros objetos.
STRONG	Este tipo garantiza que los resultados permanezcan en memoria hasta que la caché no sea explícitamente borrada. Es ideal en casos que los resultados son muy pequeños, absolutamente estáticos y se utilizan muy a menudo.

Tabla 9 *Tipos que se pueden especificar para MEMORY.*

2.5.1.2 Atributo ReadOnly

La etiqueta “cacheModel” proporciona un atributo “readOnly”. Este atributo es simplemente un indicador que especifica cómo el modelo de caché debe recuperar y almacenar en caché los objetos. Si se establece en true indica que la caché no va a ser alterada por la aplicación. Si el atributo se establece en falso asegura que los resultados pueden ser leídos y modificados. Por defecto el atributo se encuentra con valor true.

2.5.1.3 Atributo Serializar

El atributo “serializar” se usa para especificar cómo los atributos se van a guardar en caché una vez que los objetos sean devueltos. Cuando el tributo toma valor true a cada objeto devuelto se le hace una copia. Esto significa que el objeto tendrá el mismo valor, pero no es la misma instancia. Así se asegura que el objeto almacenado en caché nunca sea devuelto. Es válido llamar la atención que los objetos no se serializan en el disco, sino que se crean copias que están en la memoria caché.

2.5.1.4 Combinando ReadOnly con Serialize

Es necesario entender bien los resultados que provocan la combinación del atributo “readOnly” y “Serialize”:

ReadOnly	Serialize	Resultado	Razón
True	False	Bueno	La más rápida de las combinaciones. Devuelve una instancia compartida de la caché lo cual puede provocar problemas si es mal utilizada.
False	True	Bueno	Rápida recuperación de los objetos de la caché. Devuelve una copia de los objetos almacenados.
False	False	Peligroso	La caché solo existe para el hilo de la llamada es decir no puede ser usada por otros hilos.
True	True	Malo	Funciona igual a la combinación readOnly false y serialize true salvo que no tiene ningún sentido semántico.

Tabla 10 *Diferentes combinaciones de ReadOnly con Serialize.*

2.5.2 Liberar la Caché

Cada implementación de los modelos de caché ofrecen etiquetas que posibilitan liberar su contenido. El desarrollador es el encargado de especificar cuándo y cómo desea que los objetos sean eliminados. Las etiquetas “flushOnExecute” y “flushInterval” se encargan de proveer las funcionalidades que especifican cuándo la caché se debe limpiar.

A la etiqueta “flushOnExecute” se le especifica una consulta o procedimiento mapeado y permite que la caché se elimine después de ejecutado. Esto es útil para cuando se tienen resultados que deben actualizarse a raíz de una modificación en la base de datos subyacente. Se pueden tener dentro de un modelo de caché varios elementos del tipo “flushOnExecute”. En caso de utilizar un procedimiento o consulta que se encuentre en otro fichero de mapeo es necesario especificarle el espacio de nombres al que pertenece.

La otra etiqueta usada para manipular los elementos de la caché se denomina “flushInterval”. Es un poco más simple que la anterior, no depende de la configuración de otros elementos a no ser el tiempo. Básicamente lo que realiza este mecanismo de vaciado de la caché es esperar a que transcurra el tiempo especificado para limpiar la memoria. El intervalo comienza una vez que la configuración es cargada y continúa hasta que la aplicación es terminada. Los valores de los intervalos pueden especificarse en horas, minutos, segundos y milisegundos mediante los siguientes atributos:

- ✓ Hours (opcional): El número de horas que deben transcurrir antes de limpiar la caché.
- ✓ Minutes (opcional): El número de minutos que deben transcurrir antes de limpiar la caché.
- ✓ Seconds (opcional): El número de segundos que deben transcurrir antes de limpiar la caché.
- ✓ Milliseconds (opcional): El número de milisegundos que deben transcurrir antes de limpiar la caché.

2.5.3 Solución

El almacenamiento de objetos en caché es un mecanismo que no se explota en la versión anterior de la CAD. El escenario principal a utilizar es en la carga de los nomencladores del sistema debido a que son datos que se modifican con poca frecuencia y son utilizados frecuentemente. La implementación del modelo de caché que se propone para la solución es LRU con los atributos Serialize en “true” y readOnly en “false”. El método de liberar la caché para la solución es flushOnExecute y se especifican los procedimientos que alteran los datos almacenados en caché.

2.6 Tratamiento de Excepciones

Para entender el término de excepciones es necesario definir que es el fracaso de una rutina. Una rutina no es una sucesión arbitraria de instrucciones, sino la implementación de una cierta especificación.

Cuando las rutinas rompen el contrato se dice que fracasan.

- ✓ Fracaso: Una llamada a una rutina tiene éxito si termina su ejecución en un estado en el que satisface el contrato de la rutina. Fracasa si no tiene éxito.
- ✓ Las excepciones son fracasos que interrumpen la ejecución de una determinada rutina.

Formalmente:

- ✓ Excepciones: Es un suceso en tiempo de ejecución que puede causar que una rutina fracase.

Es necesario aclarar que fracaso y excepción son dos términos diferentes: todo fracaso es el resultado de una excepción, pero no toda excepción produce un fracaso.

Para el tratamiento disciplinado de excepciones existen dos principios fundamentales:

- ✓ Reintento: intentar cambiar las condiciones que condujeron a la excepción y ejecutar de nuevo la rutina desde el comienzo.
- ✓ Fracaso (también conocido como pánico organizado): limpiar el entorno, terminar la llamada e informar del fallo a quien hiciera la llamada.

2.6.1 Excepciones en .NET

En .NET Framework, una excepción es un objeto derivado de la clase Exception. La excepción se inicia en un área del código en que se produce un problema. La excepción asciende por la pila hasta que la aplicación la controla o el programa se detiene.

El motor de tiempo de ejecución utiliza un modelo de control de excepciones basado en objetos de excepción y bloques de código protegidos. Cuando se produce una excepción, se crea un objeto Exception que la representa.

El motor de tiempo de ejecución crea una tabla de información de excepciones para cada ejecutable. Cada método del ejecutable tiene una matriz de información de control de excepciones asociada (que puede estar vacía) en la tabla de información de excepciones. Cada entrada de la matriz describe un bloque de código protegido, los filtros de excepción asociados a ese código y los controladores de excepción (instrucciones Catch). Esta tabla de excepciones es muy eficiente y no supone una disminución del rendimiento en el tiempo del procesador ni en el uso de memoria si no se produce una excepción. Los recursos sólo se utilizan cuando se produce una excepción.

La tabla de información de excepciones representa cuatro tipos de controladores de excepciones para los bloques protegidos:

- ✓ Un controlador Finally que se ejecuta cada vez que se cierra el bloque, tanto si se produce por el flujo de control normal como por una excepción no controlada.
- ✓ Un controlador de errores que se debe ejecutar si se produce una excepción, pero no se ejecuta cuando termina el flujo de control normal.

- ✓ Un controlador filtrado por el tipo que controla las excepciones de una clase especificada o de cualquiera de sus clases derivadas.
- ✓ Un controlador filtrado por el usuario que ejecuta código especificado por el usuario para determinar si el controlador asociado debe controlar la excepción o si dicha excepción se debe pasar al siguiente bloque protegido.

Cuando se produce una excepción, el tiempo de ejecución inicia un proceso de dos pasos. El motor de tiempo de ejecución busca la matriz del primer bloque protegido que:

- ✓ Protege una región que contiene la instrucción que se está ejecutando actualmente.
- ✓ Contiene un controlador de excepciones o un filtro que controla la excepción.

Si hay una coincidencia, el motor de tiempo de ejecución crea un objeto Exception que describe la excepción. A continuación, el tiempo de ejecución ejecuta todas las instrucciones Finally o de error entre la instrucción en la que se produjo la excepción y la instrucción que controla la excepción. Tenga en cuenta que el orden de los controladores de excepción es importante: primero se evalúa la excepción situada más al interior. Además que los controladores de excepciones pueden tener acceso a las variables locales y a la memoria local de la rutina que detecta la excepción, pero se pierden los valores intermedios del momento en que se inicia la excepción.

Si no hay ninguna coincidencia en el método actual, el motor de tiempo de ejecución busca en todos los llamadores del método actual y va subiendo hasta recorrer toda la pila. Si ningún llamador tiene una coincidencia, el motor de tiempo de ejecución permite que el depurador tenga acceso a la excepción. Si el depurador no se asocia a la excepción, el motor de tiempo de ejecución provoca el evento UnhandledException. Si no hay agentes de escucha para el evento UnhandledException, el motor de tiempo de ejecución vuelca un seguimiento de pila y cierra el programa.

2.6.2 Excepciones en IBATIS

IBATIS proporciona algunas clases para manipular las excepciones lanzadas por el framework. Las principales son:

- ✓ ConfigurationException: Se lanza cuando ocurre alguna excepción en el proceso de configuración del framework.
- ✓ DataMapperException: Se lanza cuando ocurre alguna excepción en el componente SqlMapper.

La jerarquía de excepciones en iBATIS es la siguiente:

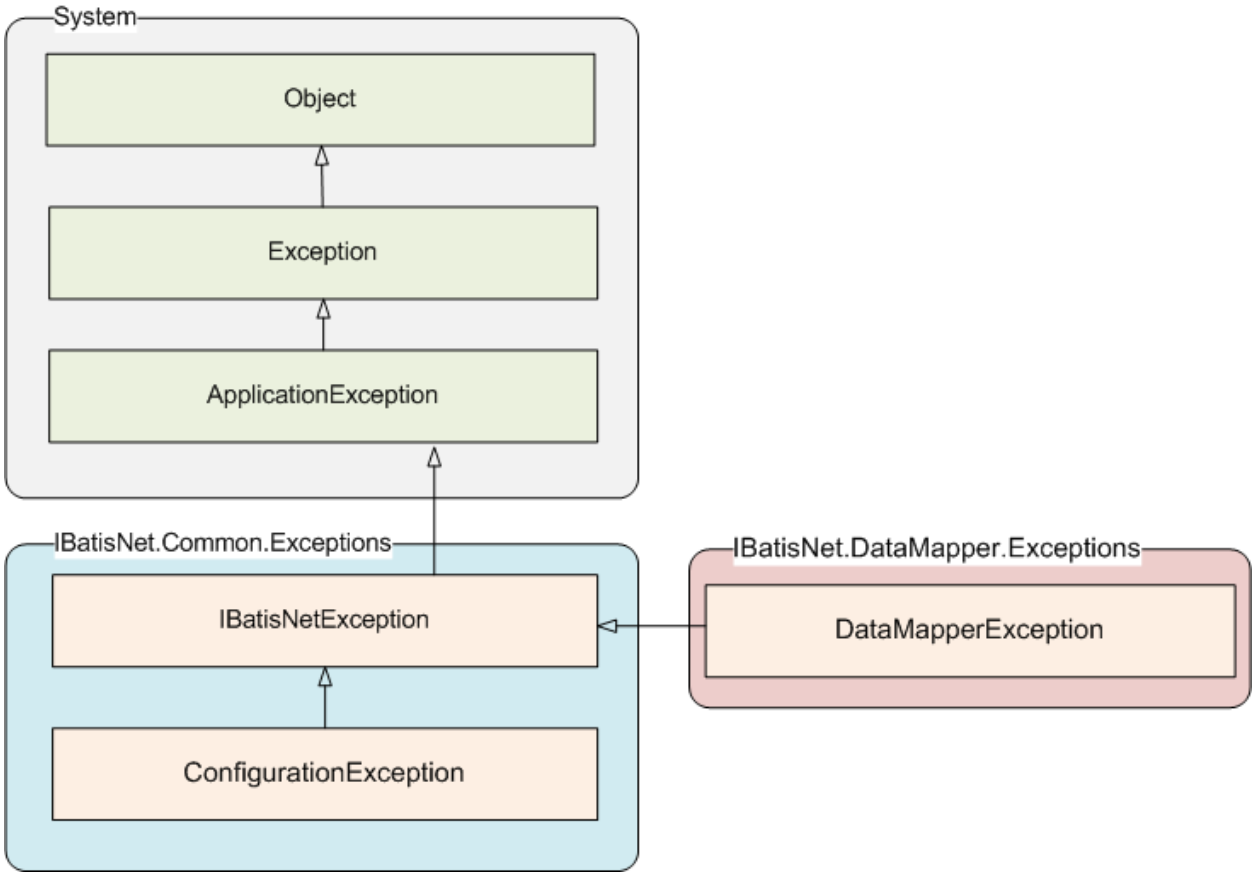


Figura 14 *Jerarquía de excepciones en IBATIS.*

2.6.3 Solución

Teniendo en cuenta los elementos analizados anteriormente se propone una solución para el tratamiento de excepciones. Hacer uso de un conjunto bien diseñado de bloques de código de control para hacer que el sistema sea mucho más fuerte y menos propenso a interrupciones. A continuación se proporcionan algunas sugerencias sobre los procedimientos recomendados para controlar excepciones:

- ✓ Saber cuándo configurar un bloque Try/Catch. Por ejemplo, se puede comprobar mediante programación si se da una condición que es probable que ocurra sin utilizar el control de errores. En otras situaciones, es aconsejable utilizar el control de excepciones para detectar una condición de error. En el ejemplo siguiente se usa una instrucción if para comprobar si se ha cerrado una conexión. Se puede usar este método en lugar de iniciar una excepción si la conexión no está cerrada.

```
if(conn.State != ConnectionState.Closed)
conn.Close();
```

En el ejemplo siguiente, se inicia una excepción si la conexión no está cerrada.

```
try {
conn.Close();
}
catch (InvalidOperationException ex) {
//Hacer algo o ignorar el error.
}
```

El método que se elija depende de la frecuencia con que se espera que se produzca el evento. Si el evento es muy excepcional y es un error (por ejemplo, un final de archivo inesperado), es mejor utilizar el control de excepciones porque se ejecuta menos código en el caso normal. Si el evento ocurre con frecuencia, es mejor utilizar el método de programación para comprobar si hay errores. En este caso, si se produce una excepción, se tarda más en controlarla.

- ✓ Utilice bloques try/finally en torno a código que podría generar una excepción y centralice las instrucciones Catch en una ubicación. De esta manera, la instrucción Try genera la excepción, la instrucción finally cierra o desasigna recursos y la instrucción Catch controla la excepción desde una ubicación central.
- ✓ Ordene siempre las excepciones de los bloques Catch de la más específica a la menos. Con esta técnica se controla la excepción específica antes de que pase a un bloque Catch más general.
- ✓ Termine los nombres de clases de excepción con la palabra "Exception". Por ejemplo:

```
public class MyFileNotFoundException : ApplicationException {
}
```

- ✓ Utilice al menos los tres constructores comunes cuando cree sus propias clases de excepción.
- ✓ Utilice, en la mayoría de los casos, los tipos de excepción predefinidos. Defina nuevos tipos de excepción sólo para escenarios de programación. Introduzca una nueva clase de excepción que permita al programador adoptar acciones distintas en código basado en la clase de excepción.
- ✓ No derive excepciones definidas por el usuario de la clase base Exception. Para la mayoría de las aplicaciones, derive excepciones personalizadas de la clase ApplicationException.

- ✓ Utilice mensajes de error gramaticalmente correctos, incluida la puntuación de cierre. Cada oración de una cadena descriptiva de una excepción debe acabar con un punto.
- ✓ Proporcione propiedades de Exception para el acceso mediante programación. Incluya información adicional en una excepción (además de la cadena descriptiva) sólo si hay un escenario de programación en que dicha información es útil.
- ✓ Diseñe las clases de manera que nunca se inicie una excepción en el uso normal. Por ejemplo, una clase FileStream expone otra forma de determinar si se ha llegado al final del archivo. Así se evita la excepción que se inicia si se lee más allá del final del archivo. En el ejemplo siguiente se muestra cómo leer hasta el final del archivo.

```
class FileRead {
void Open() {
FileStream stream = File.Open("miArchivo.txt", FileMode.Open);
byte b;

// ReadByte retorna -1 si EOF.
while ((b == stream.ReadByte()) != true) {
// Hacer algo.
}
}
}
```

- ✓ El seguimiento de pila comienza en la instrucción en que se inicia la excepción y termina en la instrucción Catch que detecta la excepción. Tenga esto en cuenta para decidir dónde ubicar una instrucción throw.
- ✓ Utilice métodos de generador de excepciones. Es habitual que una clase inicie la misma excepción desde distintos lugares de su implementación. Para evitar el exceso de código, use métodos auxiliares que creen la excepción y la devuelvan. Por ejemplo:

```
class File {
string fileName;
public byte[] Read(int bytes) {
if (!ReadFile(handle, bytes))
throw NewFileIOException();
}

FileException NewFileIOException() {
```

```
string description = // Construrir la cadena, incluir el nombre del archivo.
return new FileException(description);
}
}
```

- ✓ Inicie excepciones en lugar de devolver un código de error.
- ✓ Cuando inicie una excepción, elimine los resultados intermedios. Los autores de llamadas deben poder asumir que no se producen efectos no deseados cuando se inicia una excepción desde un método.

2.7 Registro de sucesos en la CAD

La manera de registrar las acciones que se producen entre la CAD y el servidor de base de datos es a través de un mecanismo interno que ofrece iBATIS. El mecanismo de registro de sucesos está construido sobre la API Apache Log4Net. Existen tres implementaciones diferentes para registrar los sucesos en la CAD:

NoOpLoggerFA, ConsoleOutLoggerFA, TraceLoggerFA, pero también se pueden usar componentes externos como la API Apache Log4Net mediante la implementación Log4NetLoggerFA.

Solo se explicarán tres implementaciones que son las que proporciona IBATIS.

- ✓ NoOpLoggerFA es la implementación para registrar las acciones que utiliza la CAD por defecto. Consiste en ignorar todos los sucesos.
- ✓ ConsoleOutLoggerFA envía todos los mensajes a la consola (Console.Out).
- ✓ TraceLoggerFA envía todos los mensajes a System.Diagnostics.Trace.

El mecanismo de registro se configurará añadiendo un archivo de configuración en el ensamblado con el elemento <iBATIS>. Para esto se debe haber especificado en el elemento <sectionGroup> la sección correspondiente a la configuración de los registros.

A cada una de las implementaciones que proporciona IBATIS se le pueden especificar un conjunto de argumentos. Ellos son:

- ✓ showLogName: Valor booleano que especifica si se muestra el nombre.
- ✓ showDataTime: Valor booleano que especifica si se muestra la fecha en que aconteció el suceso.
- ✓ level: Nivel al cual se desea registrar las acciones. Por defecto su valor es All.

- ✓ dateTimeFormat: Una cadena que contiene el formato con que se desea mostrar la fecha del suceso.

2.7.1 Solución

Teniendo en cuenta los elementos analizados anteriormente se propone una solución para el problema del registro de sucesos en la CAD. Se utilizará la implementación ConsoleOutLoggerFA proporcionada por iBATIS. Los argumentos a especificar para esta implementación fueron analizados anteriormente y los valores que se proponen son:

Argumento	Valor
showLogName	True
showDataTime	True
level	Debug
dateTimeFormat	`yyyy/MM/dd HH:mm:ss`

Tabla 11 *Valores de los argumentos.*

Conclusiones

En el presente capítulo se describe la solución propuesta, que consiste en un subsistema de acceso a datos que corrige los errores de su predecesor y lo mejora en algunos aspectos. Se le dieron respuesta a cada uno de los problemas que existían en la CAD anterior. En cada epígrafe se brinda la solución a cada uno de estos problemas y se abordaron algunos elementos que pueden ser de ayuda para el mejoramiento de la CAD. Quedando así definida la estructura del subsistema de acceso a datos y su integración con la arquitectura del SIGESC 171.

3 Implementación

En este capítulo se describen la librería IbatisUtil, herramientas y controles implementados para migrar el subsistema de acceso a datos del módulo seleccionado. Se muestran los diagramas de clases. Además se explica el funcionamiento y se exponen las principales funcionalidades de cada uno de los componentes.

3.1 iBatisUtil

La librería IbatisUtil se encarga de proporcionar una instancia única del SqlMapper definido en el archivo de configuración sqlmap.config de la CAD. Su implementación está basada en el patrón Singleton y se auxilia de algunas clases útiles que brinda iBATIS.

Principales funcionalidades

IniciarIbatisNet: Valida el sqlmap.config, establece los datos del proveedor de acceso a datos y la cadena de conexión e inicia la instancia de SqlMapper definida. En caso de ocurrir algún error lanza una excepción del tipo ConfigurationException.

GetInstancia: Retorna la instancia del SqlMapper definida en el sqlmap.config.

Diagrama de clases

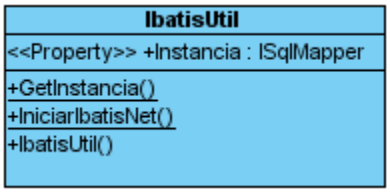


Figura 15 *Diagrama de clases de IbatisUtil.*

Funcionamiento

La primera vez que se solicita una instancia del SqlMapper, IbatisUtil invoca el método IniciarIbatisNet. A través de la clase DomSqlMapBuilder el archivo de configuración sqlmap.config es buscado dentro de los recursos embebidos en los ensamblados. Posteriormente se establecen las propiedades, como son cadena de conexión, proveedor de datos y se construye la instancia del framework. En las posteriores llamadas se retorna la misma instancia. Es necesario destacar que la configuración solo se valida si se establece como “true” la propiedad validateSqlMap en el sqlmap.config.

3.2 Herramientas

Para ayudar al proceso de migración se implementaron dos herramientas, que ayudaron a automatizar algunas tareas.

3.2.1 Postgres2IbatisXML

Postgres2IbatisXML es una herramienta que se encarga de generar el fichero de mapeo a partir del script de la base de datos. La herramienta busca las funciones dentro de un fichero SQL y la muestra al desarrollador, el cual selecciona las funciones que desea incluir en el fichero XML de mapeos de iBATIS.NET. Al final del proceso de generación se muestra conformado el archivo de mapeo y brinda la posibilidad de exportar o copiar al portapapeles dicho XML.

Principales funcionalidades

- ✓ Cargar Script: Carga un script SQL y lo analiza mostrando las funciones que estén contenidas en este.
- ✓ Seleccionar Todos: Selecciona todas las funciones que se encuentran dentro del script para generar el mapeo correspondiente de estas.
- ✓ Generar XML: Genera el archivo XML de mapeo y lo visualiza en un control.
- ✓ Copiar al Portapapeles: Copia el contenido completo del XML de mapeo generado para el portapapeles de Windows.
- ✓ Guardar Fichero: Exporta como un fichero XML todo el contenido del mapeo.

Diagrama de clases

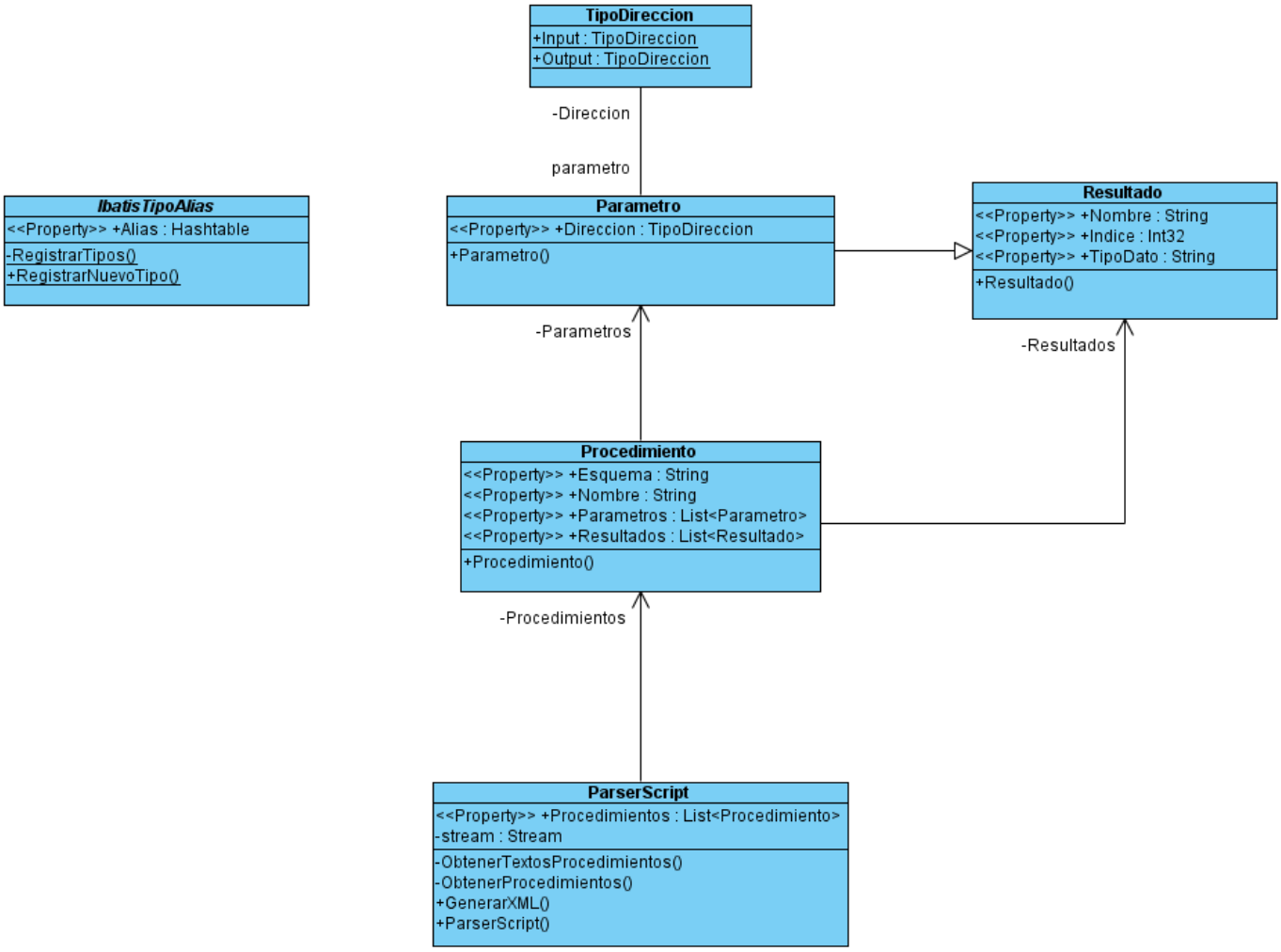


Figura 16 *Diagrama de clases de Postgres2IbatisXML.*

Funcionamiento

Postgres2IbatisXML necesita para generar el XML de mapeo que el script SQL esté en determinado formato, para ello es necesario exportarlo con la herramienta SQL Manager 2007 for PostgreSQL siguiendo los pasos mostrados a continuación.

- ✓ Se selecciona en el menú de archivo la opción herramienta y a continuación extraer base de datos.
- ✓ Se selecciona una fuente de datos y se mantiene desmarcada la opción de extraer todos los metadatos y datos. Ver Anexo 1 Seleccionar la base fuente de datos..
- ✓ A continuación después de seleccionar la opción guardar hacia un archivo, se especifica la ubicación donde se desea guardar el script. Ver Anexo 2 Especificar la ubicación..

- ✓ Posteriormente se selecciona la opción de extraer la estructura de la base de datos solamente y a continuación en el próximo formulario es necesario seleccionar en la opción objetos a extraer las funciones. Luego se agregan las funciones que se deseen extraer. Ver Anexo 3 Seleccionar las funciones..
- ✓ Por último se seleccionan las opciones como muestra la figura y se genera el script. Ver Anexo 4 Seleccionar conversión de los tipos de datos char y varchar..

Después que el script SQL es cargado mediante las funcionalidades de la clase ParserScript se extraen los nombres de las funciones y se muestran para que se seleccionen las que se desean incluir en el XML de mapeo. Una vez seleccionadas las funciones se genera el archivo.

3.2.2 NHibernate2iBATIS

NHibernate2iBATIS es una herramienta que se encarga de migrar los archivos de código fuente de la CAD escritos basados en el framework NHibernate para iBATIS.NET. Su eficacia está dada según el seguimiento que los programadores de acceso a datos le hayan dado a las pautas del diseño.

Principales funcionalidades

- ✓ Cargar CS: Carga un archivo de código fuente y muestra las invocaciones a procedimientos y funciones que contiene este.
- ✓ Generar CS: Genera un nuevo archivo de código fuente migrado para iBATIS.NET.
- ✓ Copiar al Portapapeles: Copia todo el contenido del archivo que se generó para el portapapeles de Windows.
- ✓ Guardar Fichero: Exporta el contenido del archivo generado para un fichero de código fuente.

Diagrama de clases

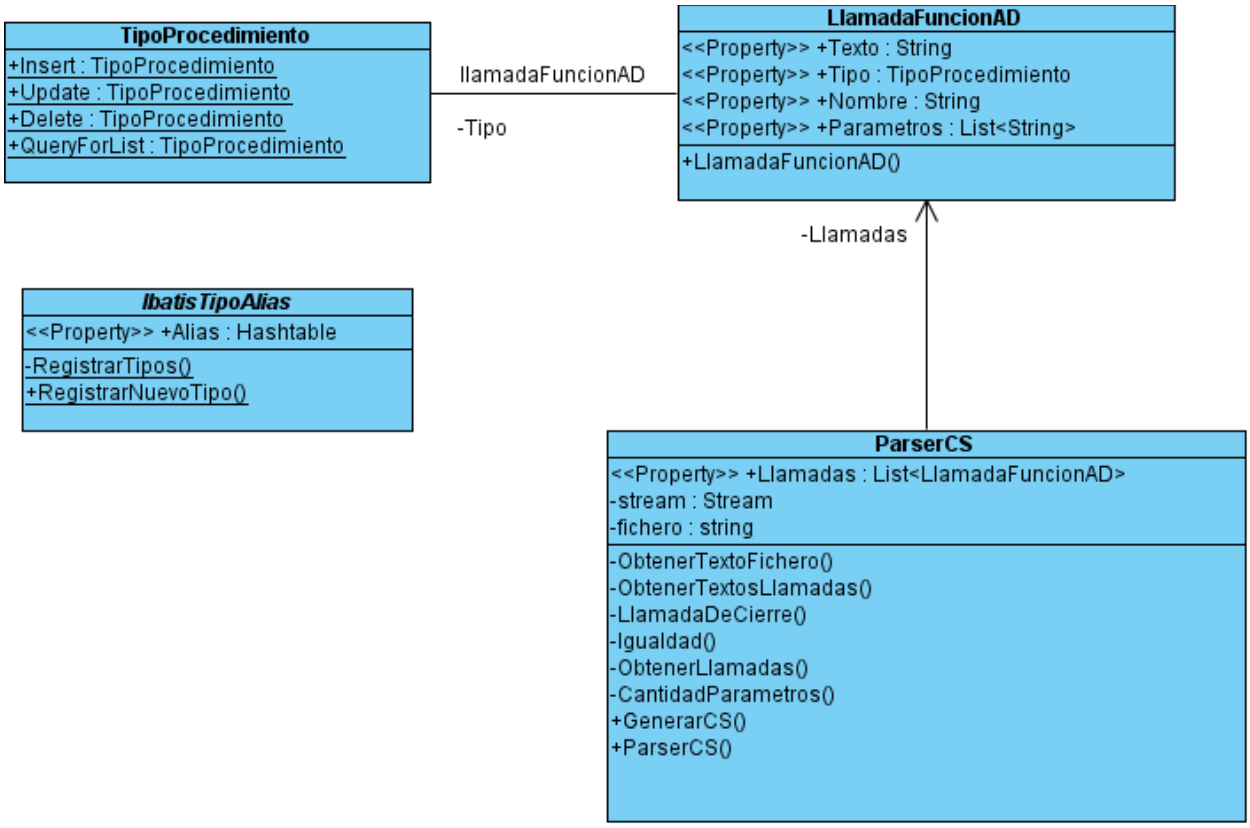


Figura 17 *Diagrama de clases de NHibernate2iBATIS.*

Funcionamiento

NHibernate2iBATIS después de cargar el archivo fuente utiliza la clase ParserCS para obtener las llamadas a funciones o procedimientos. Para generar el archivo de código fuente de la CAD para iBATIS.NET es necesario clasificar estas invocaciones a procedimientos o funciones que existen dentro del fichero cargado. Para clasificar las invocaciones se da clic derecho sobre ellas y aparecerá un menú contextual. En el menú contextual se selecciona el tipo de invocación. Posteriormente se genera el archivo fuente de la nueva CAD.

3.3 ProveedorDatosPostgresNomenclador

ProveedorDatosPostgresNomenclador es la implementación para PostgresSQL del proveedor de datos que utilizan los controles nomencladores del SIGESC 171. Su objetivo principal consiste en obtener el resultado de una función, tabla o vista dado un origen de datos (cadena de conexión y esquema).

Principales funcionalidades

ObtenerTablaDimamicaNomenclador: Dado el nombre y la lista con los valores de los parámetros de una función, ejecuta esta y devuelve un DataTable con el resultado.

ObtenerTablaNomenclador: Tiene dos variantes, la primera dado el nombre de una tabla devuelve en un DataTable todos los datos de ella. La otra sobrecarga se le especifica además del nombre de la tabla, la columna y el valor por el cual se desea filtrar el contenido de la tabla y como resultado devuelve un DataTable con los datos.

Diagrama de clases

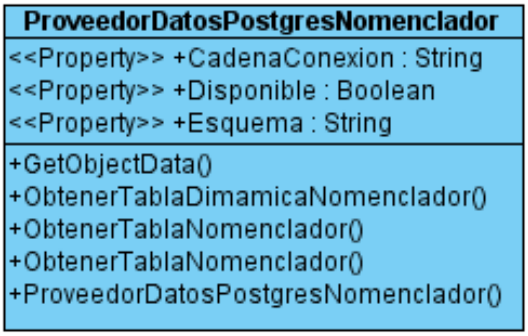


Figura 18 *Diagrama de clases de ProveedorDatosPostgresNomenclador.*

Funcionamiento

ProveedorDatosPostgresNomenclador funciona como todos los componentes en Visual Studio .NET y al igual que todo componente tiene la ventaja de hacer muy sencillo el proceso de establecer los valores de sus propiedades. Para comenzar su funcionamiento es necesario especificarle la cadena de conexión y el esquema de base de datos del cual se extraerán los datos. Una vez especificado esto se selecciona el tipo de origen de datos y el proveedor queda listo para ser usado. Para su uso solo es necesario especificarle en la propiedad correspondiente al proveedor de datos de cualquier control nomenclador el ProveedorDatosPostgresNomenclador.

Conclusiones

En el presente capítulo se representaron los modelos de clases de cada una de las herramientas, controles y librerías de clase que se utilizaron en la implementación. También se describió cada una de ellas y se expusieron sus funcionalidades.

4 Validación de la Solución

En este capítulo se validan los componentes implementados en la capa de persistencia que permiten el almacenamiento y recuperación de objetos persistentes en las entidades o tablas que conforman la base de datos para el Módulo de Administración Informática.

4.1 Pruebas de unidad con NUnit

En programación, una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

NUnit se presenta como una herramienta muy útil dentro del marco del desarrollo orientado a pruebas (TDD), permitiendo realizar las pruebas unitarias de forma automatizada y cubriendo todos los casos de uso que deban implementar las clases de la aplicación. La escritura de las pruebas y la ejecución de los métodos que deben pasarlas correrán a cargo de NUnit.

NUnit es una herramienta que se encarga de analizar ensamblados generados por .NET, interpretar las pruebas inmersas en ellos y ejecutarlas. Utiliza atributos personalizados para interpretar las pruebas y provee además métodos para implementarlas. En general, NUnit compara valores esperados y valores generados, si estos son diferentes la prueba no pasa, caso contrario la prueba es exitosa.

NUnit carga en su entorno un ensamblado y cada vez que lo ejecuta lo recarga. Esto es útil porque se pueden tener ciclos de codificación y ejecución de pruebas simultáneamente, así cada vez que se compile no tiene que volver a cargar el ensamblado al entorno de NUnit si no que este siempre obtiene la última versión del mismo. NUnit ofrece una interface simple que informa si una prueba o un conjunto de pruebas fallaron, pasaron o fueron ignoradas.

El plan de ejecución de las pruebas con NUnit es el que se muestra en la Tabla 12 Plan de ejecución de las pruebas en NUnit.:

Inicialización		
TestFixture		
Pruebas		
SetUp	TestMethod 1..n	TearDown
Restablecimiento		
TestFixtureTearDown		

Tabla 12 *Plan de ejecución de las pruebas en NUnit.*

Las figuras Figura 19 Prueba fallida con NUnit. y Figura 20 Prueba satisfactoria con NUnit. muestran ejemplos de pruebas utilizando la interfaz visual de NUnit.

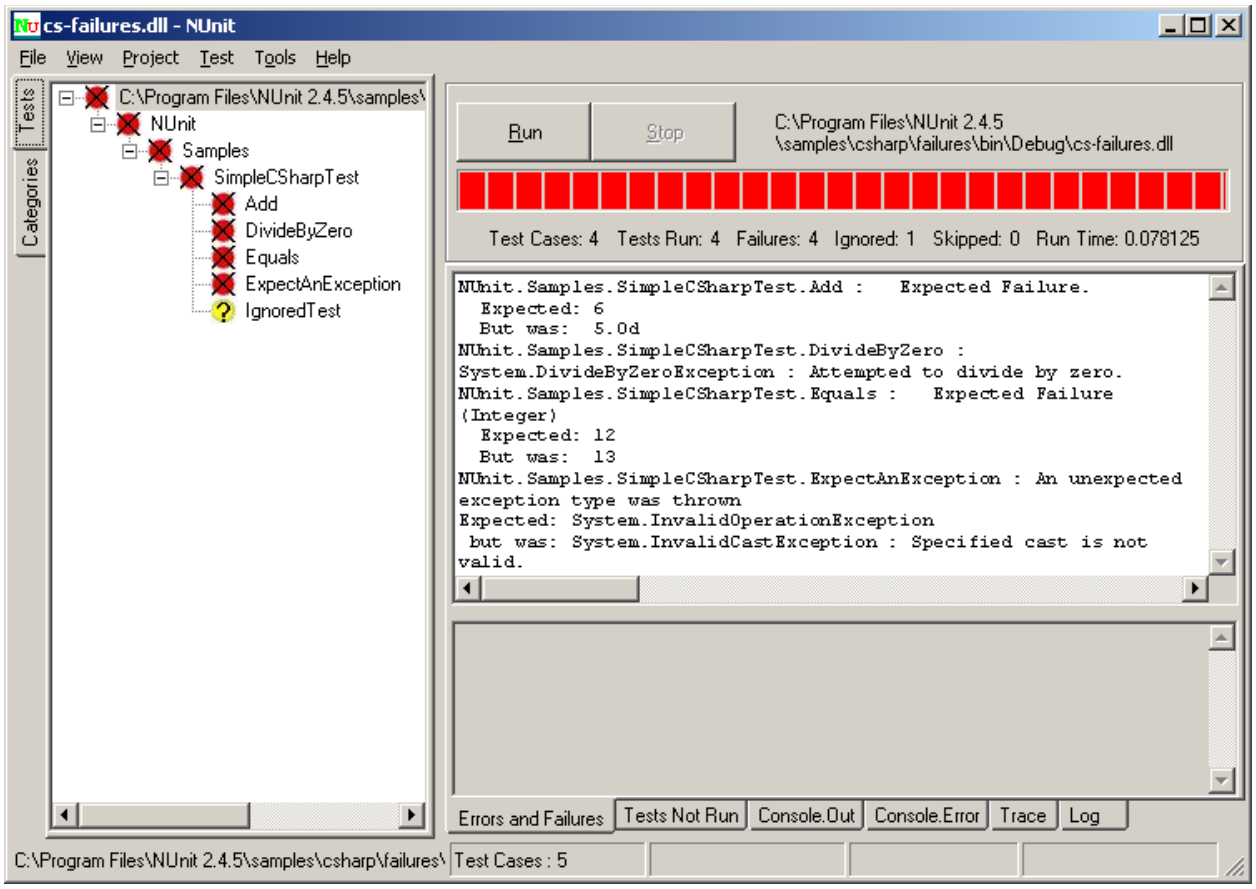


Figura 19 *Prueba fallida con NUnit.*

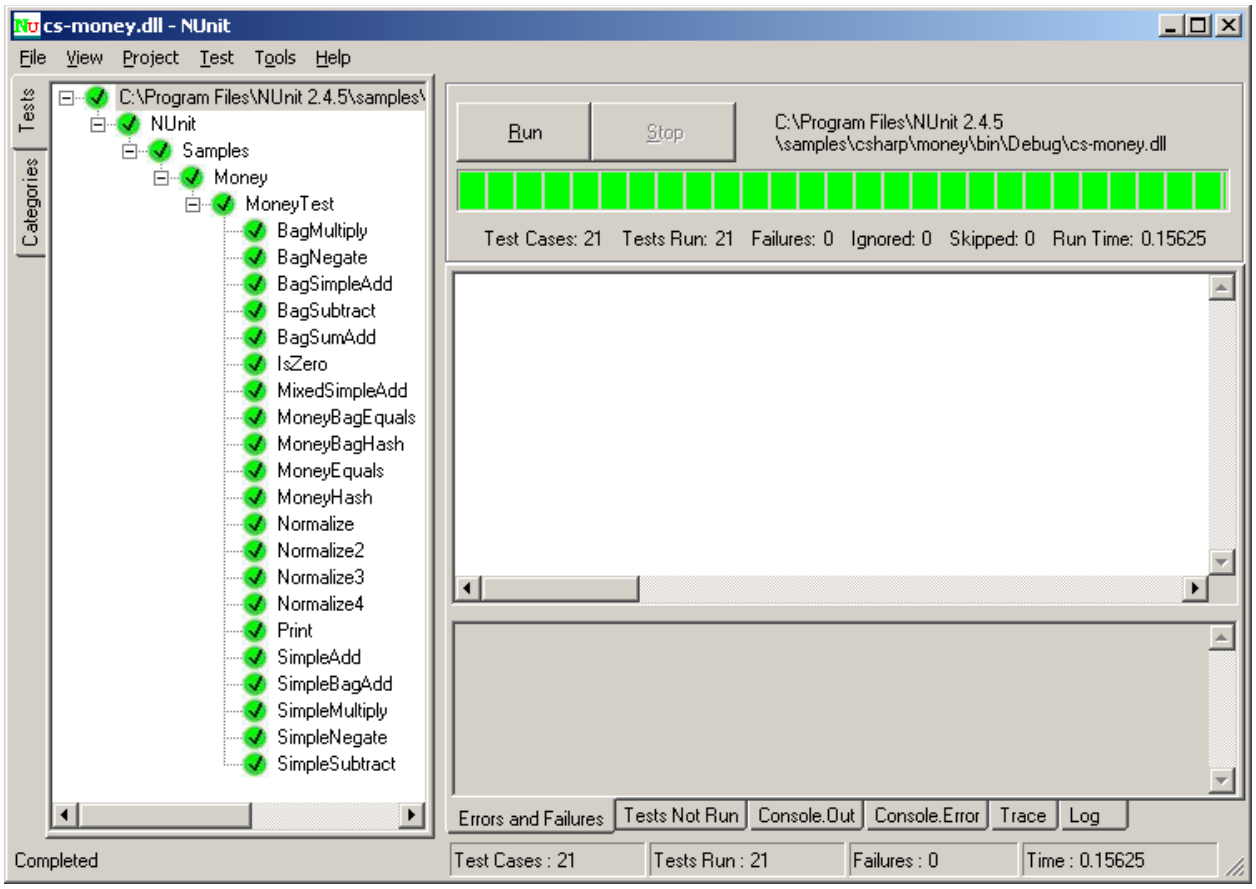


Figura 20 *Prueba satisfactoria con NUnit.*

4.2 Atributos de NUnit

Atributos cuyo ámbito de uso es a nivel de clase:

- ✓ TestFixture: Se usa para indicar que una determinada clase contiene métodos de prueba. Entre las restricciones para la clase están que deberá contar con un constructor por defecto (siendo válido el que proporciona automáticamente .Net) y que debe ser pública para que NUnit pueda acceder a ella para la ejecución de las pruebas.

Atributos cuyo ámbito de uso es a nivel de método:

- ✓ Test: Su uso es indicar que un determinado método es un método de prueba. Entre las restricciones para el método de prueba estarán que no devuelva ningún valor (void) y no reciba parámetros, y que la clase a la que pertenezca esté marcada con el atributo TestFixture.

- ✓ ExpectedException: Indica que el método va a producir una excepción del tipo indicado en el atributo (exactamente ese tipo, no otro distinto, aunque herede del mismo). La prueba se considera superada si al ejecutarlo lanza la excepción esperada.
- ✓ TestFixtureSetUp: Inicializa el entorno de pruebas antes de ejecutarlas, es decir, establece las condiciones necesarias para que las pruebas se realicen en un ambiente idóneo. Se ejecutará antes que cualquier otro método. Sólo puede haber un método marcado con el atributo TestFixtureSetUp dentro de una clase TestFixture.
- ✓ TestFixtureTearDown: Se encarga de restaurar el entorno creado para las pruebas por el método marcado con TestFixtureSetUp. Sólo puede haber un método marcado con este atributo dentro de la clase.
- ✓ Setup: Inicializa y establece el entorno antes de la ejecución de cada prueba, es decir, justo antes de cada método de prueba que se ejecute. Generalmente se encarga de inicializar variables y dejar el entorno “limpio” para la ejecución de la prueba. Sólo puede haber un método marcado como Setup dentro de cada clase de pruebas.
- ✓ TearDown: Su función es restaurar el entorno tras la ejecución de cada prueba unitaria. Sólo puede haber un método marcado como TearDown dentro de cada clase de pruebas.

A nivel de método y clase se tienen los siguientes atributos:

- ✓ Category: Otra opción que tenemos es agrupar las pruebas según categorías, de modo que podamos realizar la ejecución de un determinado bloque de pruebas simplemente indicando la categoría a la que pertenece la prueba. Para ello haremos uso del atributo Category, que puede usarse de forma conjunta a los atributos TestFixture o Test, es decir, a nivel de clase o de método.
- ✓ Explicit: Obliga a indicar durante el proceso de pruebas que se incluya una determinada prueba en el mismo. Es decir, hay que indicar explícitamente que deseamos ejecutar una determinada prueba, siendo ignorada por NUnit en caso contrario.
- ✓ Ignore: Es usado para ignorar una determinada prueba y no incluirla en el plan de pruebas. Al ejecutar NUnit, se marcará con amarillo, indicando que hay código sin probar. La ventaja de esto es que el código estará incluido en el ensamblado, ya que es compilado, pero lo excluimos de las pruebas igual que si todo el código del método o la clase hubiese sido comentado.

4.3 Pruebas a Funcionalidades

Para probar la solución propuesta se seleccionaron un conjunto de funcionalidades que forman parte del Módulo de Administración Informática. La mayor parte de estas funcionalidades involucran problemas que se solucionaron o mejoras que se hicieron a la CAD.

Los problemas o mejoras fueron probados en una estación de trabajo con los requisitos establecidos en el manual de despliegue del SIGESC 171. Las pruebas fueron encaminadas a detectar los siguientes problemas:

- ✓ Obtención de objetos compuestos: Este tipo de prueba va encaminada a detectar errores a la hora de obtener objetos que contengan dentro otros objetos o colecciones de estos.
- ✓ Colecciones: Probar fundamentalmente la manipulación de colecciones por parte de la capa de acceso a datos.
- ✓ Inserción: Probar el método object Insert(string statementName, object parameterObject) de iBATIS.NET en la CAD.
- ✓ Eliminación: Probar el método int Delete(string statementName, object parameterObject) de iBATIS.NET en la CAD.
- ✓ Actualización: Probar el método int Update(string statementName, object parameterObject) de iBATIS.NET en la CAD.
- ✓ Tiempo de respuesta: Tiene como objetivo probar el tiempo de respuesta de funcionalidades que involucran varias operaciones consecutivas.
- ✓ Rendimiento: El objetivo de esta prueba es detectar problemas en la CAD relacionadas con el manejo de volúmenes de datos.
- ✓ Transacciones: Probar las transacciones locales en la CAD, fundamentalmente transacciones sencillas que no involucran grandes cantidades de operaciones pero que contengan procedimientos almacenados en su interior.
- ✓ Transacciones Paralelas: El objetivo es probar las transacciones en entornos multihilos.
- ✓ Cargar: Probar el método object QueryForObject(string statementName, object parameterObject) de iBATIS.NET en la CAD.
- ✓ Tipos de dato: Encaminada a probar los diferentes tipos de datos que incluían problemas en la versión anterior de la CAD.

- ✓ Funcionalidad: Su objetivo es probar funcionalidades reales del SIGESC 171 utilizando la CAD. Entre ellas se encuentran funcionalidades que incluyen la invocación de procedimientos almacenados con consultas DDL.

4.4 Funcionalidades seleccionadas

Las funcionalidades que se seleccionaron para realizar los tipos de pruebas descritos en el epígrafe **¡Error! No se encuentra el origen de la referencia.** son las siguientes:

- ✓ CargarOficina: Consiste en recuperar una oficina de la base de datos. El objeto oficina del Dominio de SIGESC 171 contiene múltiples campos (nombre, tipos…), colecciones de objetos (teléfonos) y otros objetos (dirección).
- ✓ CargarTeléfonos: Esta funcionalidad recupera de la base de datos una colección de objetos que representa los teléfonos de una oficina.
- ✓ EliminarTeléfonos: Involucra una consulta de eliminación en la base de datos.
- ✓ InsertarTeléfonoOficina: Involucra una consulta de inserción en la base de datos.
- ✓ ModificarOficina: Involucra una consulta de actualización en la base de datos.
- ✓ BuscarIP: Consiste en determinar si un determinado número de IP se encuentra registrado en la base de datos. El resultado de la funcionalidad es un valor de tipo bool.
- ✓ CargarPunto: Carga un objeto punto que contiene campos de diferentes tipos de datos.
- ✓ NombreOficina: Devuelve una cadena con el nombre de la oficina especificada por parámetro.
- ✓ ModificarPunto: La funcionalidad realiza la llamada a un procedimiento almacenado que se encuentra en el ámbito de una transacción local de iBATIS.NET.
- ✓ AsignarFoto, ActualizarFoto y CargarFoto: Son funcionalidades que se encargan de gestionar las fotos de los usuarios en el gestor.
- ✓ CrearUsuarioGestor: Crea un usuario en el gestor de base de datos utilizando una consulta DDL.
- ✓ BuscarUsuarioGestor y EliminarUsuarioGestorPrueba: Buscan y eliminan usuarios en el gestor de base de datos respectivamente.

Existen varios escenarios de pruebas que no se encuentran en las funcionalidades seleccionadas pues es muy difícil encontrar un módulo en el SIGESC 171 que contenga todos los escenarios de prueba posibles. Para dar solución a esto se recrearon casos de pruebas que involucraban estos escenarios, como es el caso de:

- ✓ Probar el tiempo de respuesta, debido a que en el SIGESC no existe una funcionalidad que encueste varias veces la base de datos.
- ✓ Probar transacciones paralelas, debido a que el modulo de administración informática no se desarrolla en un entorno multihilo.
- ✓ Probar tipos de datos, debido a que el módulo Administración Informática posee una gran variedad de tipos de datos pero no incluye a todos los soportados por la CAD.

4.5 Resultados de las pruebas

A continuación la Tabla 13 Pruebas con NUnit. y el Gráfico 1 Tiempo de respuesta de los casos de prueba. muestran el resultado de las pruebas realizadas a las funcionalidades seleccionadas.

No.	Nombre del Método	Objetivo	Tiempo(s)
1	CargarOficinaPrueba	objeto compuesto	0.016
2	CargarOficinaRendimiento	Rendimiento	1.734
3	CargarTelefonosPrueba	Colecciones	0.016
4	EliminarTelefonosPrueba	Método eliminar	0.016
5	InsertarTelefonoOficinaPrueba	Método insertar	0.016
6	InsertarTelefonoTransaccionesParalelasPrueba	Transacciones	2.031
7	InsertarTelefonoVelocidadPrueba	Tiempo	12.234
8	ModificarOficinaPrueba	Método modificar	0.188
9	BuscarIPPrueba	Tipo booleano	0.109
10	CargarPuntoPrueba	Tipos de datos	0.047
11	NombreOficinaPrueba	Método cargar	0.010
12	ModificarPuntoPrueba	Transacciones simples	0.047
13	AsignarFotoTipoDatoPrueba	Insertar tipo Imagen	0.938
14	ActualizarFotoPrueba	Actualizar tipo Imagen	0.109
15	CargarFotoTipoDatoPrueba	Cargar tipo Imagen	0.078
16	CrearUsuarioGestorPrueba	Consulta DDL dinámica	0.125
17	BuscarUsuarioGestorPrueba	Funcionalidad	0.031
18	EliminarUsuarioGestorPrueba	Funcionalidad	0.016
19	CargarUsuarioPrueba	Objeto compuesto	0.219

Tabla 13 *Pruebas con NUnit.*

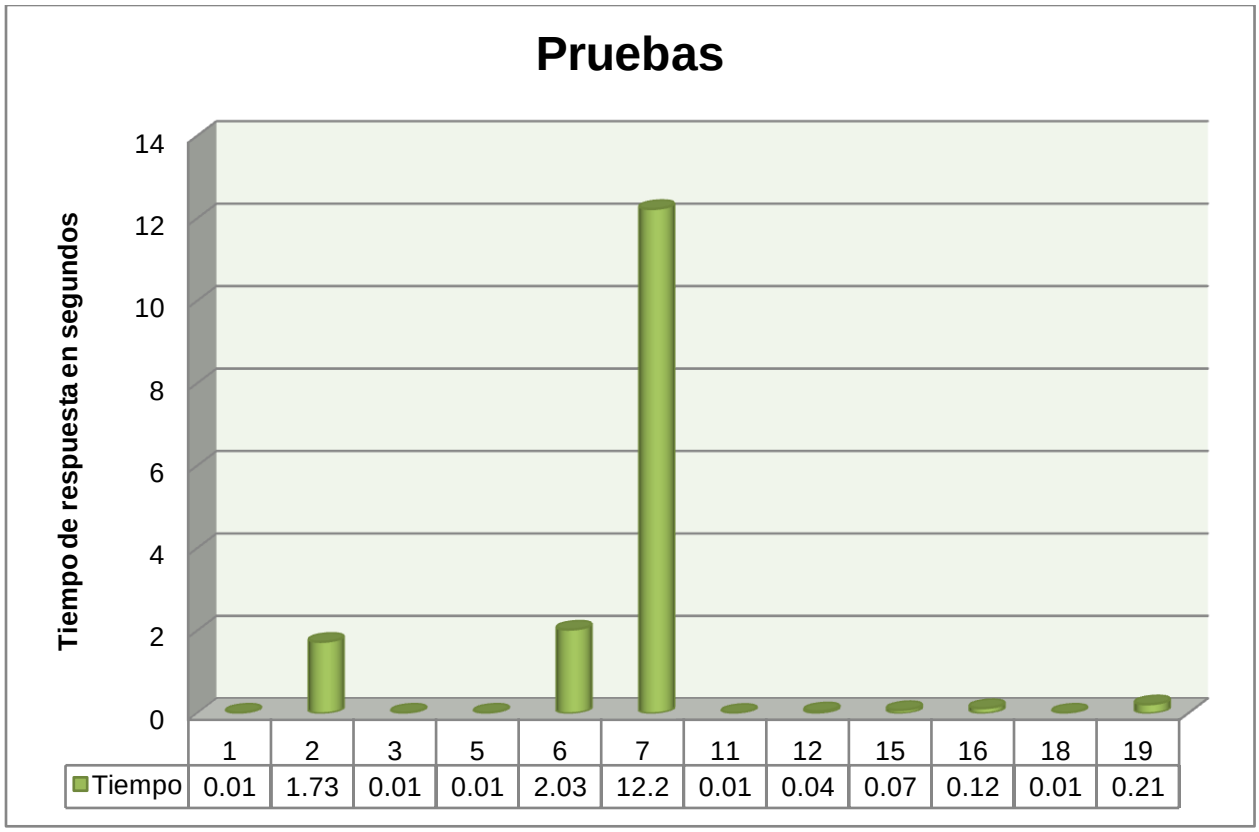


Gráfico 1 *Tiempo de respuesta de los casos de prueba.*

Conclusiones

En el presente capítulo se explicaron aspectos del desarrollo guiado por pruebas. Se realizo una descripción del framework NUnit. Además se implementaron las pruebas para comprobar cada uno de los problemas detectados en la anterior CAD. Se realizaron pruebas al módulo de Administración Informática y a los controles implementados.

Conclusiones Generales

En el presente trabajo titulado “Migración de la Capa de Acceso a Datos del Sistema de Gestión de Emergencias de Seguridad Ciudadana (171)” se le ha dado cumplimiento al objetivo general y a cada uno de los objetivos específicos planteados.

Se efectuó un estudio de los conceptos relacionados con la persistencia de objetos y se analizan herramientas de persistencia libres y de código abierto. Se fundamenta la propuesta hecha para la utilización de iBATIS.NET y se describe la plataforma y lenguajes utilizados.

Se explica el funcionamiento del framework a utilizar, se explica su instalación y configuración. Queda definida la solución para cada uno de los problemas detectados en la anterior CAD. Se describe la estructura general de la capa y las convenciones de nombre que se van a utilizar.

Se implementa el Módulo de Administración Informática utilizando la solución propuesta. Se explican cada una de las herramientas utilizadas para llevar a cabo el proceso de migración del módulo. También se implementan controles y proveedores de datos para el SIGESC 171 y una librería para interactuar con el framework de persistencia.

Se describe el funcionamiento y los elementos del framework para pruebas Nunit. Se implementan clases de pruebas para algunos casos de uso del SIGESC 171. Además se realizan pruebas que compruebas funcionalidades, rendimientos, problemas existentes en la CAD anterior, mejoras realizadas y características del framework de persistencia

En resumen se cumplieron cada uno de los objetivos trazados en la investigación quedando implementado el Módulo de Administración Informática, herramientas y librerías para acelerar el proceso de migración de los demás módulos y controles para el SIGESC 171.

Recomendaciones

Se recomienda:

- ✓ Hacer uso de los mecanismos de herencia que brinda el servidor de base de datos objeto-relacional PostgresSQL.
- ✓ Aplicar modelos de caché a los escenarios donde intervengan nomencladores u otros datos que no varían frecuentemente.
- ✓ Utilizar los mecanismos de carga perezosa que brinda el framework iBATIS para los casos en que se extraigan grandes volúmenes de datos del servidor.
- ✓ Tener en cuenta la posible aplicación del mecanismo Rowhandler que brinda iBATIS para optimizar la CAD.
- ✓ Implementar manipuladores de tipos personalizados para facilitar y acelerar el desarrollo de la CAD en el SIGESC 171. Algunos de estos pueden ser para enumerativos, imágenes o objetos del dominio que sean utilizado frecuentemente como es el caso de la dirección.

Referencias Bibliográficas

1. [Online] http://www.nationmaster.com/graph/cri_mur_percap-crime-murders-per-capita.

2. Ministerio de Poder Popular para Relaciones Interiores y Justicia. [Online] http://www.mij.gov.ve/spip.php?page=seguridad-ciudadana&id_rubrique=37.

3. Plan 180 - Propuesta Para Reducir la Inseguridad en Venezuela en 180 Dias. [Online] http://www.chacao.gov.ve/plan180/presentacion.htm.

4. **Venezuela.** *Artículo 55 de la Constitución de la República Bolivariana de Venezuela.* 2002.

5. **Tellez, Eddy Sánchez.** *Tesis Especificación de la Arquitectura Base del SIGESC.* Ciudad de la Habana : s.n., 2008.

6. **Date, C J.** *An Introduction to Database Systems. 8th Edition.* s.l. : Addison-Wesley Publishing Company, 2003.

7. **Riordan, R M.** *Designing Relational Database Systems.* s.l. : Ms Press, 1999.

8. **Devarakonda, R S.** *Object-Relational Database Systems.www.acm.org/crossroads/xrds7-3/ordbms.html.*

9. **Jacobson, I and Booch, G.** *Proceso Unificado de Desarrollo de Software.* Madrid : Addison-Wesley, 2000.

10. **Brown.** *Enterprise Java Programming with IBM Websphere.* s.l. : Addison-Wesley, 2001.

11. **Fowler, M, et al.** *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002.

12. **Booch, G.** *Análisis y Diseño Orientado a Objetos con Aplicaciones. 2ª Ed.* s.l. : Addison-Wesley, 1996.

13. **Cabrera, M.** [Online] 2005. http://mcabrera.datacenter1.com/articles/dotNET/nhibernate/#_Toc107999126.

14. **Myers, B.** *Object-Oriented Software Construction.* s.l. : Interactive Software Engineering Inc. (ISE), 1997.

15. **Gianneschi, Fabrizio.** *JDBC vs iBATIS A case study.* s.l. : Atlantis S.p.A., 2004.

16. **Kuaté, P H, et al.** *NHibernate in Action.* s.l. : Manning, 2008.

17. **Johnson, Rod, et al.** *Professional Java Development with the Spring Framework.* 2005.

18. [Online] 2002. http://www.dei.uc.edu.py/tai2002/BDOO/index.html.

19. **Begin, C, Googin, B and Meadors, L.** *iBATIS in Action.* s.l. : Manning, 2007.

20. MSDN. [Online] Microsoft Corporation. http://msdn.microsoft.com/es-es/library/h43ks021(VS.80).aspx.

21. Code Author. [Online] http://www.codeauthor.org/index.htm.

22. C# Object Persistent Framework . [Online] http://csopf.sourceforge.net/index.php.

23. Data Holder. [Online] http://dataholder.sourceforge.net/.

24. Open Dataset eXtensions. [Online] http://www.codeplex.com/odx/Wiki/View.aspx?title=Documentation&referringTitle=Home.

25. **Microsoft.** *Documentación de .Microsoft .NET Framework SDK.*

Bibliografía

1. [Online] http://www.nationmaster.com/graph/cri_mur_percap-crime-murders-per-capita.

2. Ministerio de Poder Popular para Relaciones Interiores y Justicia. [Online] http://www.mij.gov.ve/spip.php?page=seguridad-ciudadana&id_rubrique=37.

3. Plan 180 - Propuesta Para Reducir la Inseguridad en Venezuela en 180 Dias. [Online] http://www.chacao.gov.ve/plan180/presentacion.htm.

4. **Venezuela.** *Artículo 55 de la Constitución de la República Bolivariana de Venezuela.* 2002.

5. **Tellez, Eddy Sánchez.** *Tesis Especificación de la Arquitectura Base del SIGESC.* Ciudad de la Habana : s.n., 2008.

6. **Date, C J.** *An Introduction to Database Systems. 8th Edition.* s.l. : Addison-Wesley Publishing Company, 2003.

7. **Riordan, R M.** *Designing Relational Database Systems.* s.l. : Ms Press, 1999.

8. **Devarakonda, R S.** *Object-Relational Database Systems.www.acm.org/crossroads/xrds7-3/ordbms.html.*

9. **Jacobson, I and Booch, G.** *Proceso Unificado de Desarrollo de Software.* Madrid : Addison-Wesley, 2000.

10. **Brown.** *Enterprise Java Programming with IBM Websphere.* s.l. : Addison-Wesley, 2001.

11. **Fowler, M, et al.** *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002.

12. **Booch, G.** *Análisis y Diseño Orientado a Objetos con Aplicaciones. 2ª Ed.* s.l. : Addison-Wesley, 1996.

13. **Cabrera, M.** [Online] 2005. http://mcabrera.datacenter1.com/articles/dotNET/nhibernate/#_Toc107999126.

14. **Myers, B.** *Object-Oriented Software Construction.* s.l. : Interactive Software Engineering Inc. (ISE), 1997.

15. **Gianneschi, Fabrizio.** *JDBC vs iBATIS A case study.* s.l. : Atlantis S.p.A., 2004.

16. **Kuaté, P H, et al.** *NHibernate in Action.* s.l. : Manning, 2008.

17. **Johnson, Rod, et al.** *Professional Java Development with the Spring Framework.* 2005.

18. [Online] 2002. http://www.dei.uc.edu.py/tai2002/BDOO/index.html.

19. **Begin, C, Googin, B and Meadors, L.** *iBATIS in Action.* s.l. : Manning, 2007.

20. MSDN. [Online] Microsoft Corporation. http://msdn.microsoft.com/es-es/library/h43ks021(VS.80).aspx.

21. Code Author. [Online] http://www.codeauthor.org/index.htm.

22. C# Object Persistent Framework . [Online] http://csopf.sourceforge.net/index.php.

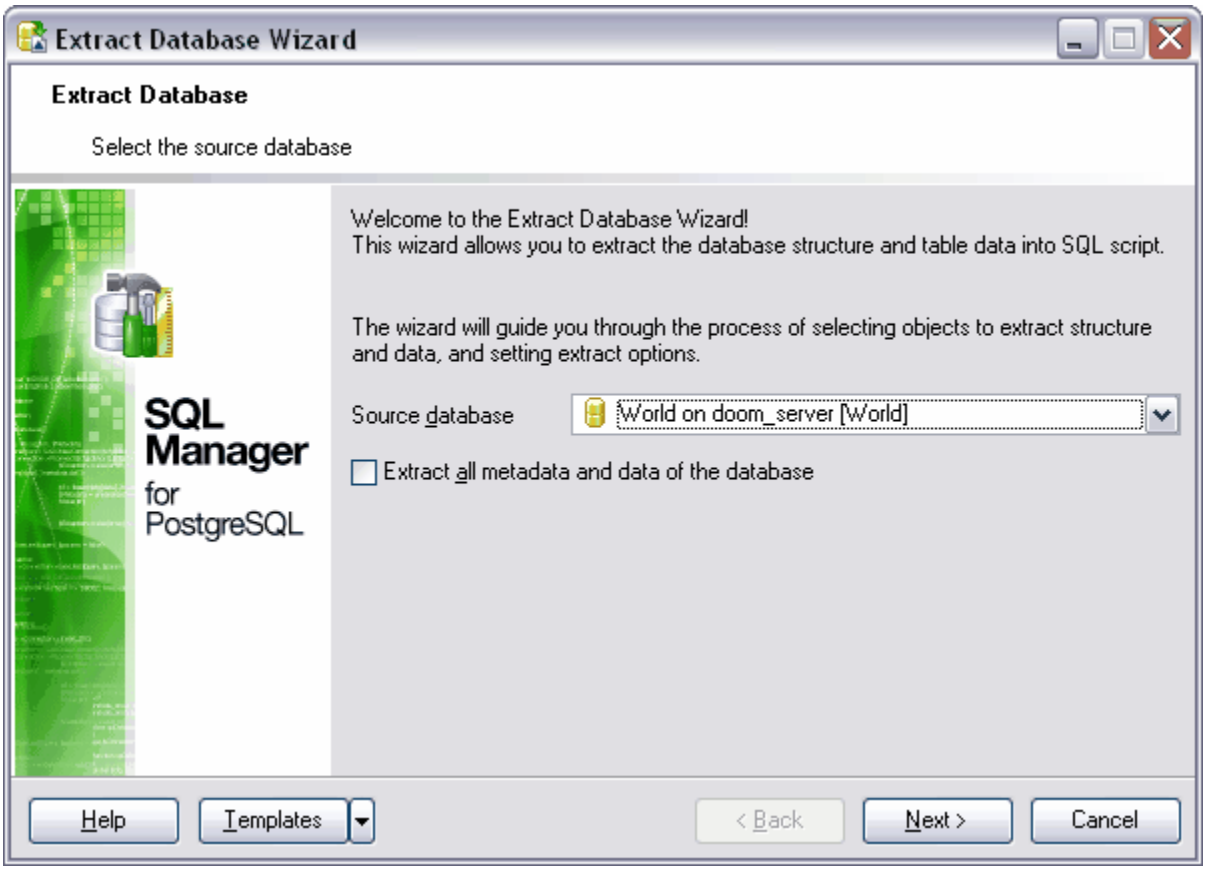
23. Data Holder. [Online] http://dataholder.sourceforge.net/.

24. Open Dataset eXtensions. [Online] http://www.codeplex.com/odx/Wiki/View.aspx?title=Documentation&referringTitle=Home.

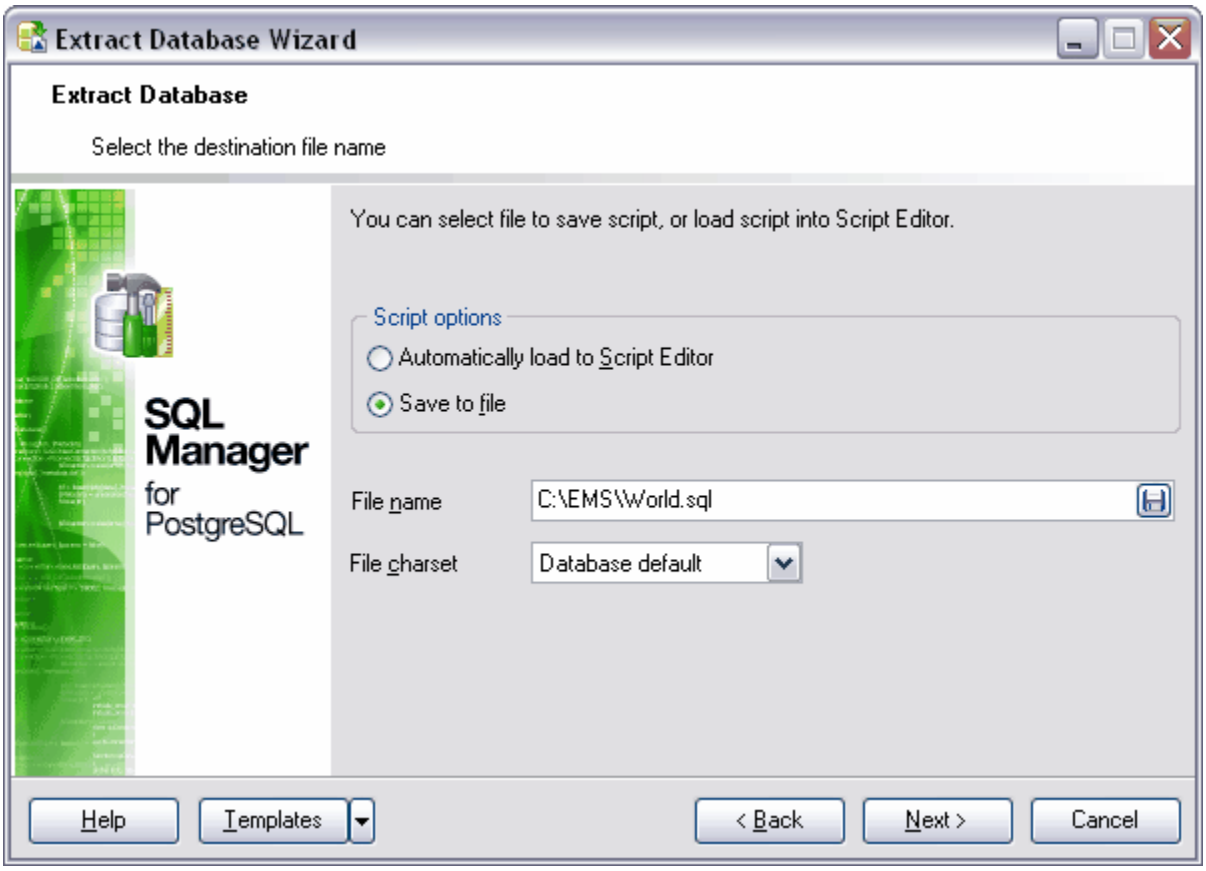
25. **Microsoft.** *Documentación de .Microsoft .NET Framework SDK.*

Anexos

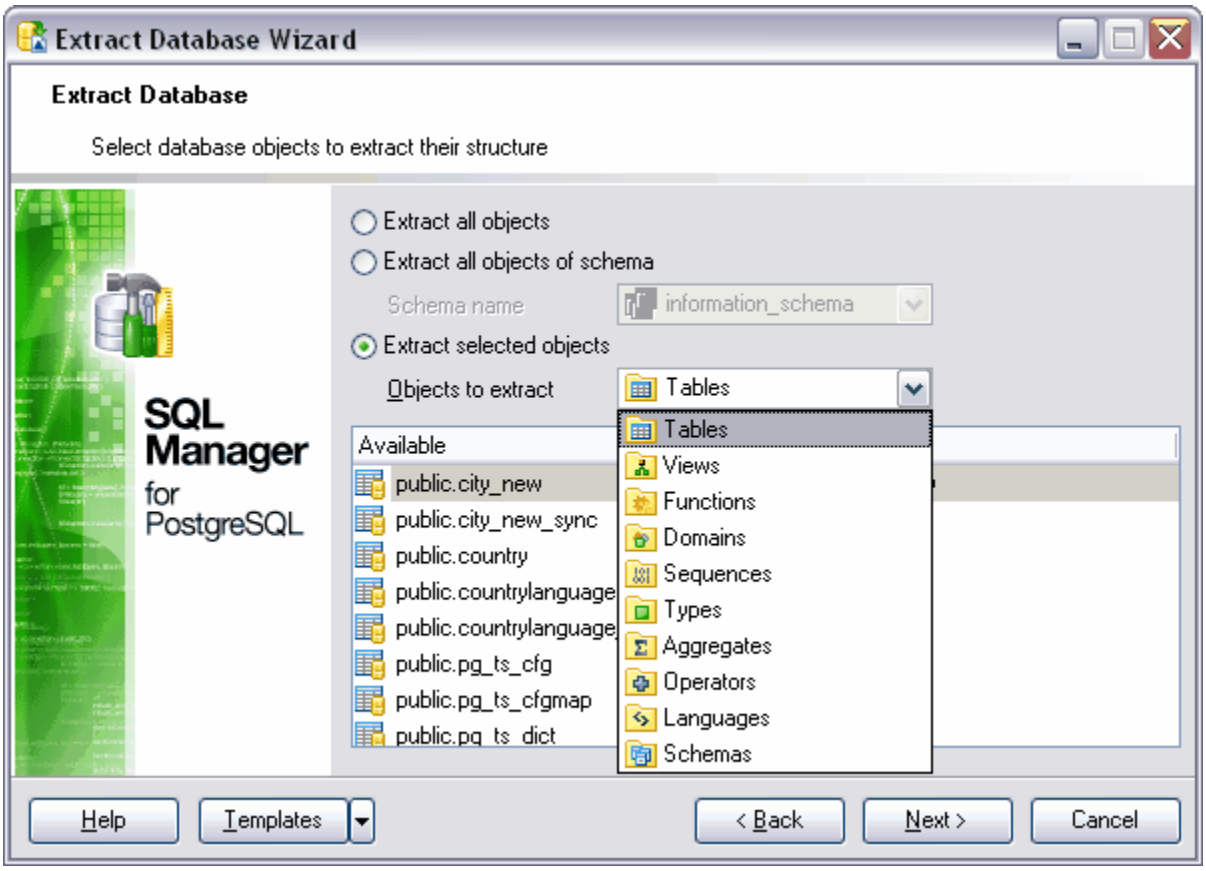
Anexo 1 Seleccionar la base fuente de datos.



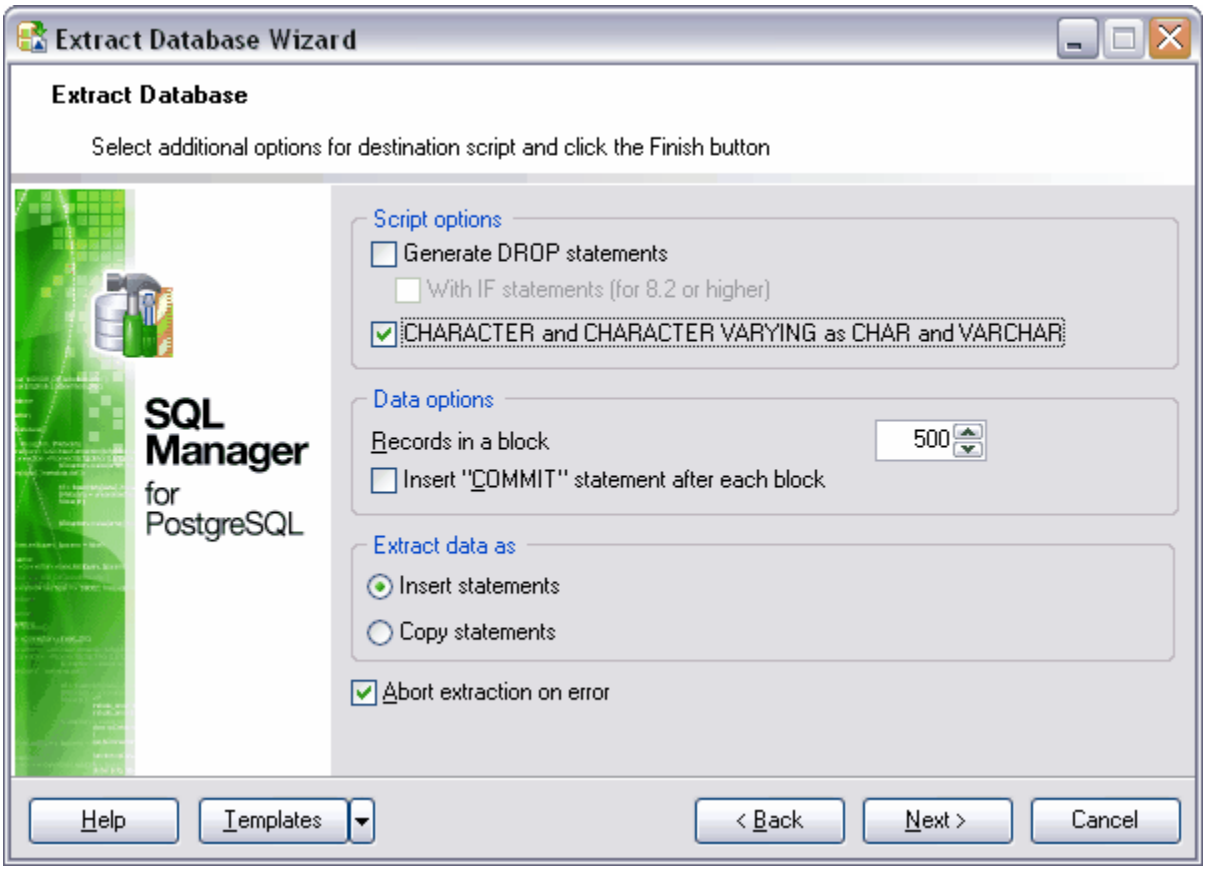
Anexo 2 Especificar la ubicación.



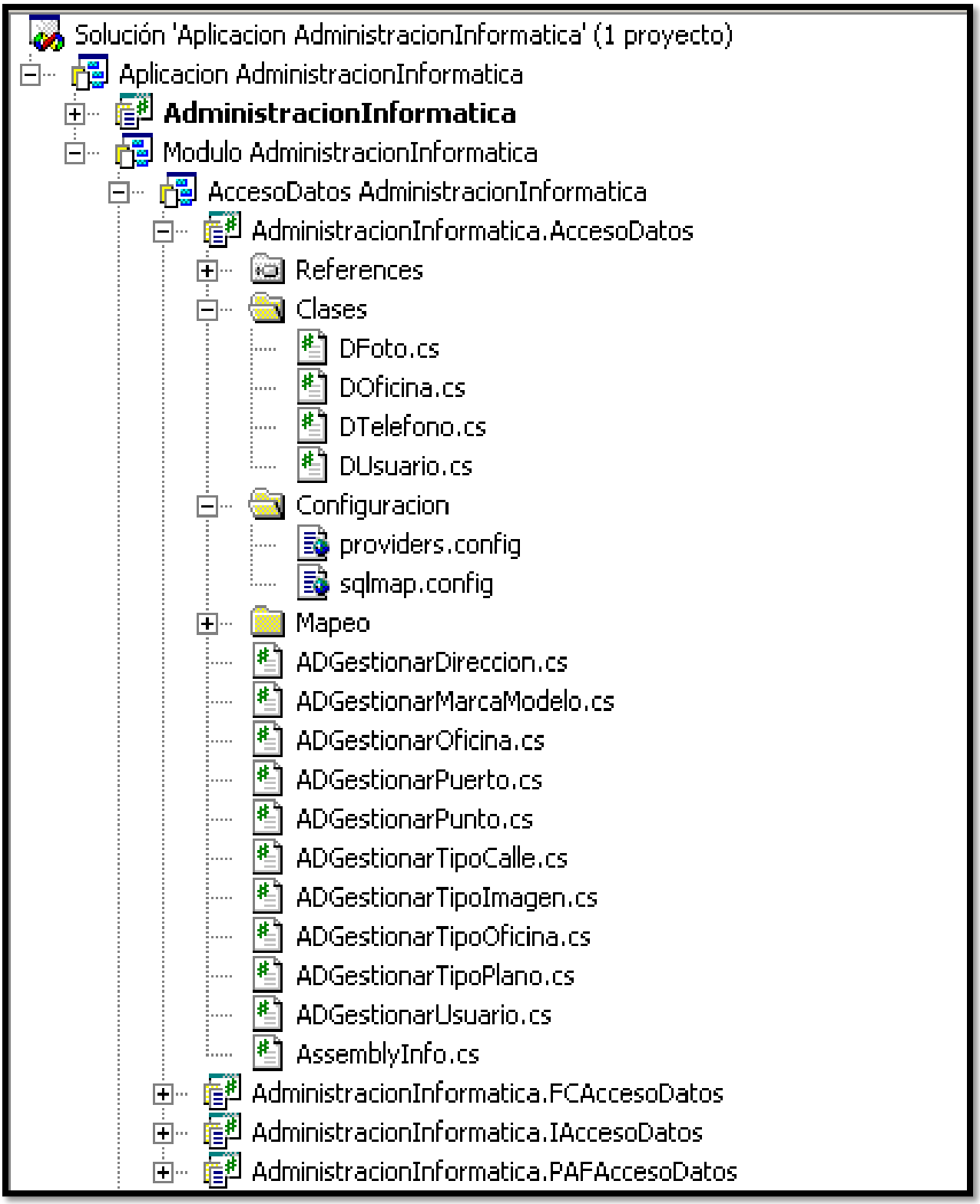
Anexo 3 Seleccionar las funciones.



Anexo 4 Seleccionar conversión de los tipos de datos char y varchar.



Anexo 5 Nueva estructura de la CAD de un módulo del SIGESC.



Glosario de Términos

Excepción: Una indicación de error que puede ser lanzada cuando una situación de error es detectada y manejada por un manejador de excepciones en cualquier parte del programa.

Nomenclador: Catálogo, lista o repertorio de nombres o sujetos.

Framework: en el desarrollo de software, es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Mapper: En español mapeador es el encargado de realizar el mapeo. Esta funcionalidad permite que cuando creemos un registro, desde el contexto de un registro relacionado con él, algunos de los campos del nuevo registro aparezcan ya cubiertos con valores extraídos del registro relacionado.

Data Mapper: El patrón consiste en una capa de Mappers que proporciona una forma sencilla de interacción de datos entre los objetos y bases de datos relacionales.

ORM: Object-Relational mapping o Mapeo Objeto-relacional es una técnica de programación utilizada para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional.