ESCUELA POLITÉCNICA DEL EJÉRCITO

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UN MOTOR DE PERSISTENCIA GENÉRICO PARA .NET

Previa a la obtención del título de:

INGENIERO DE SISTEMAS E INFORMÁTICA

POR: **SANDRA MARIBEL PILACUÁN ALEGRÍA,**
MERCY GEOVANNA PINARGOTE ALARCÓN

SANGOLQUI, 30 DE NOVIEMBRE DE 2005

CERTIFICACIÓN

Certifico que el presente trabajo fue realizado en su totalidad por la Sra. SANDRA MARIBEL PILACUAN ALEGRÍA y la Sra. MERCY GEOVANNA PINARGOTE ALARCÓN como requerimiento parcial a la obtención del título de INGENIERAS DE SISTEMAS E INFORMÁTICA.

_____

30 de noviembre de 2005

_____	_____
ING. PABLO ALMEIDA.	ING. CECILIA HINOJOSA.
DIRECTOR	CODIRECTORA

DEDICATORIA

A mi pequeño hijo que es la luz de mi vida.

Mercy

A mi madre Elba, por su ternura, su dedicación y todos los esfuerzos que realizó para lograr este proyecto, que en realidad es suyo.

A mi hermano Henry, por la motivación que ha significado para mi y también por su amor y apoyo incondicional.

A mi padre Jaime^{+}, por templar mi carácter y por enseñarme, los principios que hoy forman parte de mi vida diaria.

A mis compañeros y amigos en especial a Raquel con quienes compartimos gratos y imborrables recuerdos de nuestra vida estudiantil.

Maribel

AGRADECIMIENTOS

Agradezco infinitamente a mis padres por todo el amor y apoyo brindados durante mis estudios y principalmente en la culminación de mi carrera.

A mi esposo por su cariño, ayuda e ideas con las que he logrado realizar un buen trabajo.

A mis profesores director y codirector de la tesis que han trabajado junto con nosotras para concluir satisfactoriamente nuestro proyecto de tesis.

Mercy

AGRADECIMIENTOS

A la Escuela Politécnica del Ejército, mi sentimiento de gratitud por acogerme en sus aulas y formarme como profesional íntegra y comprometida.

Por la beca estudiantil que facilitó mis estudios por varios semestres y me permitió adquirir experiencia profesional como practicante de sus laboratorios de computación.

A todas la personas de los laboratorios de computación, por compartir conmigo sus conocimientos y por el apoyo incondicional que recibí.

A nuestro director de tesis Ing. Pablo Almeida por su apoyo y la acertada dirección.

A nuestra querida codirectora de tesis, Ing. Cecilia Hinojosa, por su asesoramiento científico y estímulo permanente para culminar este proyecto.

Maribel

Índice de contenidos.

Listado de Tablas

Listado de figuras

RESUMEN

La tesis presenta la problemática de la persistencia de los objetos de las aplicaciones, y las características de una solución para persistir objetos.

De manera general se documenta las herramientas de la metodología de desarrollo de software, los elementos de apoyo dentro de la plataforma de desarrollo y las diferencias dentro del lenguaje de base de datos.

Para conocer más de cerca el problema de la persistencia, se expone una serie de conceptos y términos fundamentales en torno al problema de hacer perdurar los objetos más allá de la ejecución de las aplicaciones que los crean. Se identifica un conjunto de requisitos, que servirá de base para elaborar una capa de persistencia en .Net.

Después de exponer la parte teórica se desarrolla la especificación de requerimientos tanto para el motor de persistencia como para el prototipo, capítulos en los que se lista la funcionalidad que se espera y se muestran los acuerdos del alcance, para luego derivar en los análisis y diseño respectivos.

A continuación se realizan la implementación y pruebas del software desarrollado, con el fin de comprobar su correcta funcionalidad.

Finalmente se describen las conclusiones y recomendaciones a las que hemos llegado al terminar el proyecto de tesis.

CAPÍTULO I

GENERALIDADES

1.1. Introducción

Las aplicaciones informáticas constan de dos componentes principales que colaboran para llevar a cabo la funcionalidad que el usuario desea. El primero de estos componentes es el gestor de datos, que guarda la información necesaria para operar la aplicación, en forma de datos en disco. El segundo de estos componentes es el programa propiamente dicho, que recupera estos datos de la base de datos, realiza los cálculos necesarios y presenta los resultados deseados al usuario.

Para que estos dos componentes puedan funcionar deben poder comunicarse intercambiando datos, pero la evolución de estos dos componentes ha sido divergente, de forma que es difícil que colaboren en una misma aplicación y que tengan independencia, así es que los gestores de datos utilizan desde los años 70 a la actualidad un modelo llamado “relacional” que se ha convertido en un estándar para las aplicaciones de software, en cambio los lenguajes de programación utilizan un modelo llamado “orientado a objetos”. Es por esto que aparece un conflicto a la hora de unir estos dos componentes en una aplicación. Una solución a este problema es utilizar un motor de persistencia que permita que la aplicación desarrollada sea independiente del gestor de datos que se utilizará para almacenar los datos.

El motor de persistencia traduce entre los dos formatos de datos: de registros a objetos y de objetos a registros. Cuando el programa quiere grabar un objeto llama al motor de persistencia, que traduce el objeto a registros y llama al gestor de datos para que guarde estos registros. De la misma manera, cuando el programa quiere recuperar un objeto, el gestor de datos recupera los registros correspondientes, los cuales son traducidos en formato de objeto por el motor de persistencia. El programa solo ve que puede guardar objetos y recuperar objetos, como si estuviera programado para un gestor de datos orientado a objetos. El gestor de datos solo ve que guarda registros y recupera registros, como si el programa estuviera dirigiéndose a ella de forma relacional. Es decir, cada uno de los dos componentes trabajan con el formato de datos que le resulta más natural y es el motor de persistencia el que actúa de traductor entre los dos modelos, permitiendo que los dos componentes se comuniquen y trabajen conjuntamente.

1.1 Antecedentes

La mayoría de los sistemas computacionales más nuevos se han construido basados en la programación orientada a objetos pero no se ha definido correctamente cual es el código que debe ir en cada uno de los objetos.

El uso de Java tiene muchos años en el mercado y su característica es el de ser de código abierto por lo que existen varios motores de persistencia difundidos y probados en grandes aplicaciones; en contraste en nuestro país es nuevo el desarrollo de software en .Net., existe mucho desconocimiento de los usos y ventajas de tener un motor de persistencia genérico y reutilizable para el rápido desarrollo de aplicaciones.

De la experiencia de varias empresas que actualmente desarrollan software a medida en nuestro medio se sabe que cada empresa tiene que pasar por un período de 3 a 6 meses hasta estabilizar su propia capa de persistencia sin contar con el tiempo adicional del mantenimiento respectivo.

1.2 Posición actual

Los nuevos sistemas se han construido basados en la programación orientada a objetos pero no se ha definido correctamente cual es el código que debe ir en cada uno de los objetos que pertenecen al sistema, por ejemplo una clase contiene métodos para interactuar con el usuario, interactuar con la base de datos, así como para manejar procesos lógicos del negocio, para mejorar la forma en la que se desarrollan los sistemas es necesario que se implemente bajo una arquitectura en capas que cuente al menos con:

1. Capa de presentación, que contiene las clases que implementan la comunicación con el usuario.
2. Capa lógica, que contiene las clases que implementan la lógica de la aplicación.
3. Capa de persistencia, que aloja los datos que necesitan ser preservados por la aplicación (archivos, gestor de datos, etc.).

Para el manejo de la capa de persistencia existen actualmente desarrollados y probados varios motores de persistencia para la plataforma Java y muy pocos para la plataforma .NET. Entre los motores de persistencia más completos de código abierto para Java podemos destacar: Hibernate, Castor, Torque, OJB y

Cayenne, algunos de estos traen una versión para .NET pero ninguna de ellas garantiza su desenvolvimiento en aplicaciones comerciales o de gran alcance. Entre los comerciales, podemos destacar TopLink, Cocobase y FastObjects, pero los costos de estas herramientas representan rubros muy altos en los presupuestos para el desarrollo de aplicaciones.

Excepto por Objectspaces para .NET 2004 que ha sido anunciada por Microsoft y ORM.NET, que es un producto de tipo comercial para .NET.[1] , no existen en el mercado otras opciones a la hora de decidirse a utilizar un motor de persistencia. Debido a esto el presente proyecto de tesis presenta una alternativa válida, que puede impactar positivamente en un proyecto de software.

1.3 Justificación

Realizar un motor de persistencia puede reducir el código de una aplicación sustancialmente, haciéndola menos costosa de desarrollar, además que el código que se obtiene programando de ésta manera es más limpio y sencillo, por lo tanto, más fácil de mantener y más robusto. El motor de persistencia no solo simplifica la programación, sino que permite hacer ciertas optimizaciones de rendimiento.

Empleando un motor de persistencia en empresas de desarrollo de software cada integrante se encargará de implementar los requerimientos de su producto, es decir, un programador solo se encarga de programar y no de manejar la base de datos, mejorando el tiempo de desarrollo, así como un administrador del gestor de datos se encargará de hacer lo que mejor sabe administrar el gestor de datos,

1 Tomado de: www.programacion.net/bbdd/articulo/joa_persistencia/art. Motores de Persistencia.

sin tener que preocuparse por si introduce errores en la programación por cambios que haya realizado en el gestor de datos. Un motor de persistencia permite que el administrador del gestor de datos realice los cambios necesarios en el gestor de datos sin afectar al programador.

Con un motor de persistencia bien realizado los administradores del gestor de datos deben poder mover tablas, modificar el nombre de las tablas, modificar el nombre de las columnas, y reorganizar las tablas sin afectar el lugar donde se las esté usando.

Escribir el código para un motor de persistencia se lo realizará una sola vez y este motor podrá ser utilizado en el desarrollo de nuevas aplicaciones de software como un componente externo que manejará la persistencia y permitirá un desarrollo más rápido de la aplicación.

1.4 Alcance

La presente tesis cubre el análisis, diseño e implementación de un motor de persistencia genérico para .Net.

Se entregará un prototipo que permita mostrar la funcionalidad y las características del motor de persistencia.

El desarrollo de este motor de persistencia, plantea encapsular el comportamiento necesario para hacer objetos persistentes, es decir, para leer, escribir y suprimir en el almacenamiento permanente de los objetos de una

aplicación. Para garantizar que se logre un motor de persistencia robusto se requieren las siguientes funciones:

1. Varios tipos de mecanismos de persistencia, un mecanismo de persistencia es cualquier tecnología que se pueda utilizar para almacenar permanentemente los objetos para una actualización, una recuperación, o eliminación. Los mecanismos posibles de la persistencia incluyen ficheros "planos", gestores de datos relacionales, gestores de datos objeto-relacionales, gestores de datos jerárquicos, gestores de datos de la red.
2. Encapsulamiento, completa del mecanismo de persistencia, idealmente se debe enviar solo mensajes de eliminación, recuperación o actualización respectivamente. El motor de persistencia debe manejar el resto.
3. Acciones de Multi – objeto, una capa robusta de persistencia debe poder apoyar la recuperación de muchos objetos simultáneamente.
4. Transaccionalidad, esto está relacionado con el punto 3, porque se puede borrar, guardar o recuperar varios objetos en una sola transacción.
5. Extensibilidad, un motor de persistencia debe ser flexible para permitir incorporar nuevas clases en la aplicación.
6. Identificar objetos, todo objeto debe tener un identificador del objeto (OID) que identifique únicamente un objeto.
7. Manejo de Cursores, que permitan una conexión lógica al mecanismo de persistencia usando una recuperación controlada. Es decir recuperar solo lo que el usuario necesita en ese momento.
8. Utilizar Proxies para la representación de otros objetos que contiene la información que identifica a dicho objeto.

9. Recuperación de registros, el motor de persistencia debe estar en la capacidad de recuperar registros.
10. Soportar múltiples arquitecturas, como las empresas van cambiando de arquitecturas centralizadas a arquitectura en capas el motor de persistencia debe apoyar en estos cambios.
11. Soportar varias versiones de bases de datos; La versión de base de datos debe ser transparente para los programadores.
12. Soportar múltiples conexiones simultáneas; Un motor de persistencia debe poder apoyar conexiones múltiples, simultáneas a diferentes motores de persistencia.
13. Permitir diversas estrategias para tener acceso a una base de datos relacional, mediante drivers nativos y no nativos; Una buena capa de la persistencia apoyará los más comunes, Tales como: conectividad abierta de la base de datos (ODBC), la conectividad de la base de datos de Java (JDBC), y los drivers nativos provistos por el vendedor de la base de datos.
14. Manejar consultas en lenguaje estructurado de consultas (SQL), el SQL embebido dentro de la aplicación debe ser la excepción, no la norma.

Este motor de persistencia puede ser utilizado por aplicaciones con arquitectura cliente / servidor en dos ó más capas.

Los resultados que se esperan conseguir con este proyecto en las etapas del ciclo de vida del software son:

Etapa de análisis y diseño se obtendrá:

- Especificación de requerimientos.
- Diagrama de casos de uso.
- Diagrama de interacción (secuencia).
- Diagrama de actividad para determinados procesos.
- Diagrama de clases.
- Diccionario de datos.

Etapa de desarrollo se obtendrá:

- Un motor de persistencia de objetos genérica para .Net.
- Código fuente de la aplicación.
- Prototipo de demostración de la aplicación.
- Archivo ejecutable.

Etapa de pruebas

- Pruebas de unidad.
- Pruebas de validación.
    i. Salidas que el usuario reconoce del sistema.
    ii. Métodos de caja negra.
    iii. Casos derivados del modelo de comportamiento.

Etapa de implantación

- Instalación del motor de persistencia de objetos genérica para .Net.
- Pruebas de integración:

i. Demostración mediante una aplicación prototipo la funcionalidad de la capa de persistencia.

La aplicación prototipo estará desarrollada bajo una arquitectura en capas y demostrará como es posible comunicarse con diferentes gestores de datos sin cambiar el código escrito para el motor de persistencia, los gestores de datos con los que interactuara la aplicación prototipo serán Oracle, SQL Server 2000, DB2 se mostrará como el motor de persistencia soporta conexiones simultáneas a diferentes gestores de datos así como recuperar varios objetos simultáneamente, se demostrará como maneja la transaccionalidad sin perjudicar el rendimiento de la aplicación prototipo.

Se incorporarán nuevas clases dentro de la aplicación prototipo para mostrar la facilidad que el motor de persistencia entrega a los usuarios en ésta labor y como excepción a la regla se mostrará como el motor de persistencia puede manejar consultas SQL.

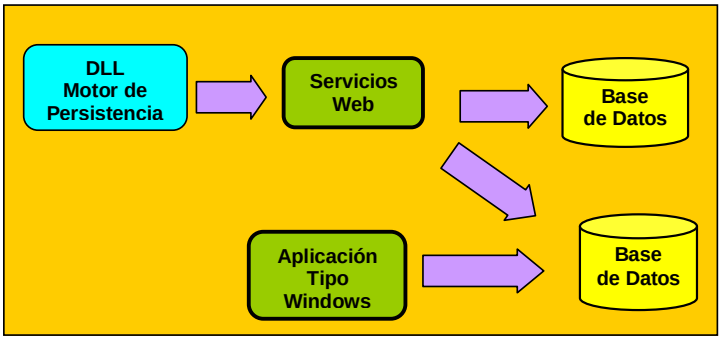


Figura 1.1: Esquema de trabajo de un motor de persistencia.

1.5 Objetivos

General

- Realizar el análisis, diseño, e implementación de un motor de persistencia para .Net, para simplificar las operaciones de mantenimiento (Inserción, actualización, eliminación), de consulta y conexión con el gestor de datos.

Específicos

- Transparentar la inserción, actualización y eliminación de datos en el gestor de datos.
- Aplicar patrones de diseño para el desarrollo de un motor de persistencia.
- Desarrollar un prototipo en *n* capas que permita la facturación de productos para demostrar la funcionalidad del motor de persistencia para .Net, desarrollado en ésta tesis.
- Acceder a la plataforma de datos a través del motor de persistencia para realizar las operaciones CRUD del prototipo planteado.

Con formato: Numeración y viñetas

~~1~~ CAPÍTULO II

MARCO TEÓRICO

~~1.1~~ 2.1 Metodología OMT

Con formato: Numeración y viñetas

La ingeniería de software es un proceso para la producción ordenada de software, usando una colección de técnicas y convenciones de notación. Una metodología es una serie de pasos, con técnicas y notación asociadas a cada paso. Los conceptos y notación que soportan la metodología OMT se presentan en el modelado de objetos, dinámico y funcional. Los pasos para la producción de software son organizados en un ciclo de vida que consiste de varias fases de desarrollo. OMT abarca sólo la primera parte de ellos, es decir, análisis, diseño e implantación. En esta tesis se va a utilizar OMT, como guía metodológica para comprobar el cumplimiento de la fase de análisis.

UML es un lenguaje visual de modelación que permite mostrar la visión del análisis que se realiza con OMT y se lo utilizará como lenguaje de modelado. En la tabla equivalencia de modelos entre UML y OMT se puede identificar la relación entre los modelos de UML y OMT[2.]

~~1.3~~ 2.2 Metodología OMT 2

Con formato: Numeración y viñetas

La metodología Object Modeling Tecnique (OMT) 2 es la segunda versión de OMT que es publicada por James Rumbaugh en 1996 en la que adopta algunas

2 Tomado de : www.ifi.uio.no/in219/verktoy/ doc/html/doc/user/mg/dgmsomt.html

de las técnicas de Booch, OMT 2 es una metodología de segunda generación introduce cambios en el modelo de objetos para hacerlo compatible con UML.

OMT es una metodología orientada a objetos muy difundida que se hace cargo de todo el ciclo de vida del software, es una de las metodologías de análisis y diseño orientada a objetos más maduras y eficientes que existen en la actualidad. Parte de la idea de utilizar los mismos conceptos y la misma notación a lo largo de todo el ciclo de vida común a todas las fases, a través de tres modelos que capturan los aspectos estáticos, dinámicos y funcionales que combinados proveen una descripción completa del software. Trabaja con el modelo espiral o un proceso evolutivo con una separación no rígida entre las fases del desarrollo.

Tabla 2.1: Tabla de equivalencia de modelos entre UML y OMT

Diagrama de actividades	UML.
Diagrama de comunicación	OMT
Diagrama de clases	UML y OMT. En OMT se llama Diagrama de Asociación de Clases
Diagrama de colaboración	UML
Diagrama de componentes	UML
Diagrama de flujo de datos (DFD)	OMT
Diagrama de despliegue (DPD)	UML
Diagrama de mensajes (MGD)	OMT
Diagrama de secuencia	UML y OMT. En OMT se llama diagrama de traza de eventos.
Diagrama de estado	UML y OMT. En OMT se llama diagrama de transición de estados.
Diagrama de casos de uso	UML y OMT.

~~1.3.1~~2.2.1 Fases de la metodología OMT

Con formato: Numeración y viñetas

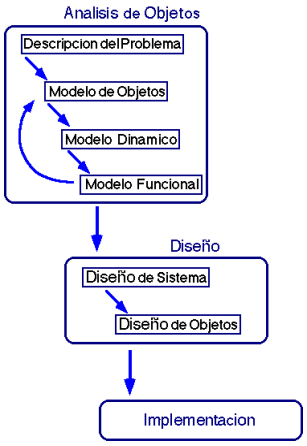


Figura 2.1~~2.1~~: Ciclo de vida OMT[3].

Los pasos para la producción de software son organizados en un ciclo de vida que consta de varias fases de desarrollo. OMT abarca sólo la primera parte de ellos.

~~1.~~Fase de análisis

Con formato: Numeración y viñetas

El objetivo de esta fase es desarrollar un modelo del funcionamiento del sistema. El modelo se expresa en términos de objetos y relaciones, el control dinámico de flujo y las transformaciones funcionales. El proceso de capturar los requerimientos y consultar con el solicitante debe ser continuo a través del análisis.

3 Tomado de: www.pisuerga.inf.ubu.es/icruzado/tfc/OMT_res.pdf

El analista construye un modelo del dominio del problema, mostrando sus propiedades más importantes. El modelo de análisis es una abstracción resumida y precisa de lo que debe de hacer el sistema deseado y no de la forma en que se hará. Los elementos del modelo deben ser conceptos del dominio de aplicación y no conceptos informáticos tales como estructuras de datos. Un buen modelo [4] debe poder ser entendido y criticado por expertos en el dominio del problema que no tengan conocimientos informáticos.

Los pasos a seguir en el análisis son:

- Contar con una descripción inicial del problema: Se concentra en entender el problema y modelar en el dominio del problema mediante un texto.
- Construir un modelo de objetos y sus relaciones. El Modelo de objetos se construye a partir de:
  - Diagrama de clases.
  - Diagrama de objetos.
  - Diagrama de casos de uso.
  - Diccionario de datos.
- Desarrollar un modelo dinámico, mediante:
  - Diagrama de secuencia.
  - Diagrama de transición de estados.
- Construir un modelo funcional, mediante:
  - Diagrama de flujo de datos.
  - Restricciones, dependencias y transformaciones.
- Verificar, iterar y refinar los tres modelos que incluya:

4 Tomado de: www.monografias.com/trabajos13/metomt/metomt.shtml

- Agregar al **modelo de objetos** operaciones clave que sean descubiertas durante la preparación del **modelo funcional**. No deben mostrarse todas las operaciones durante el análisis, sólo las más importantes.
- Verificar que las clases, asociaciones, atributos y operaciones sean consistentes y completos al nivel seleccionado de abstracción. Comparar los tres modelos con el enunciado del problema y el conocimiento relevante al dominio y probar los modelos usando varios escenarios.
- Desarrollar escenarios más detallados (incluyendo condiciones de error) como variaciones de los escenarios básicos, para cruzar los tres modelos.
- Iterar los pasos anteriores según sea necesario para completar el análisis.

El documento que genera el análisis contiene:

Documento de análisis = definición del problema + modelo de objetos + modelo dinámico + modelo funcional.

~~2.~~Fase de diseño del sistema

Con formato: Numeración y viñetas

En esta fase se selecciona la estructura de alto nivel del sistema. Se organiza el sistema en subsistemas basándose tanto en la estructura del análisis como en la arquitectura propuesta.

Los pasos que se llevan a cabo son:

- Organizar el sistema en subsistemas.

- Identificar la concurrencia inherente al problema.
- Asignar subsistemas a procesadores y tareas.
- Escoger la estrategia básica para implantar los almacenamientos de datos en términos de estructuras de datos, archivos y bases de datos.
- Identificar recursos globales y determinar seguridades para acceso.

El documento que nos proporciona el diseño del sistema es:

Documento de diseño del sistema = estructura de la arquitectura básica del sistema + decisiones estratégicas de alto nivel.

~~3.~~Fase de diseño de objetos

Con formato: Numeración y viñetas

El diseñador de objetos construye un modelo de diseño basándose en el modelo de análisis, sin entrar en detalles particulares de un lenguaje o base de datos particular, pero incorporando detalles de implementación. El diseño de objetos se centra en las estructuras de datos y algoritmos que son necesarios para implementar cada clase. OMT describe la forma en que el diseño puede ser implementado en distintos lenguajes (orientados y no orientados a objetos, bases de datos, etc.).

Su objetivo es refinar el modelo del análisis y proporcionar una base detallada para la implementación tomando en cuenta el ambiente en que se implementará.

Los pasos que se realizan en el diseño de objetos son los siguientes:

- **Obtener las operaciones para el modelo de objetos a partir de los otros modelos:**
  - Encontrar una operación para cada proceso en el **modelo funcional**.
  - Definir una operación para cada evento en el **modelo dinámico**, dependiendo de la implantación del control.
- **Diseñar los algoritmos para implantar las operaciones:**
  - Escoger los algoritmos que minimicen el costo de implementación de las operaciones.
  - Seleccionar las estructuras de datos apropiadas para los algoritmos.
  - Definir clases internas y operaciones nuevas según sea necesario.
  - Asignar las responsabilidades para las operaciones que no están asociadas claramente con una sola clase.
- **Optimizar las rutas de acceso a los datos:**
  - Agregar asociaciones redundantes para minimizar los costos de acceso y maximizar la conveniencia.
  - Reacomodar los cálculos para una mayor eficiencia.
  - Grabar valores derivados para evitar recalcular expresiones complicadas.
- **Implantar el control del software introduciendo el esquema seleccionado durante el diseño de sistemas.**
  - Ajustar la estructura de clases para incrementar la herencia.
  - Reacomodar las clases y las operaciones para incrementar la herencia.
  - Abstraer el comportamiento común de los grupos de clases.
  - Usar delegación para compartir comportamiento donde la herencia sea semánticamente inválida.

- **Diseñar la implantación de las asociaciones:**
  - Analizar las travesías de las asociaciones.
  - Implantar cada asociación como un objeto distinto o agregando atributos objeto-valor a una o ambas clases en la asociación.
- **Determinar la representación de los atributos de los objetos.**
- **Empaquetar las clases y las asociaciones en módulos.**

El documento que se genera el diseño de objetos es:

Documento de diseño de objetos = modelo de objetos detallado + modelo dinámico detallado + modelo funcional detallado.

~~4.~~Fase de implementación

Con formato: Numeración y viñetas

Las clases de objetos y relaciones desarrolladas durante el análisis de objetos se traducen finalmente a una implementación concreta, en la cual es importante mantener los principios de la ingeniería del software de forma que la correspondencia con el diseño sea directa. No tiene sentido que utilicemos AOO[5] y DOO[6] de forma que potenciemos la reutilización de código y la correspondencia entre el dominio del problema y el sistema informático, si luego perdemos todas estas ventajas con una implementación de mala calidad.

~~1.3.1~~2.2.2 Modelos de la metodología OMT

Con formato: Numeración y viñetas

- **Modelo de objetos.** Describe la estructura estática de los objetos del sistema (identidad, relaciones con otros objetos, atributos y operaciones). El modelo de objetos proporciona el entorno esencial en el cual se pueden

5 Análisis Orientado a Objetos
6 Diseño Orientado a Objetos

situar el modelo dinámico y el modelo funcional. El objetivo es capturar aquellos conceptos del mundo real que sean importantes para la aplicación. Se representará mediante:

- Diagramas de clases propuesto por UML.
- Diagramas de casos de uso propuesto por UML.
- Se realizará un diccionario de datos.

- **Modelo dinámico.** Describe los aspectos de un sistema que tratan de la temporización y secuencia de operaciones (sucesos que marcan los cambios, secuencias de sucesos, estados que definen el contexto para los sucesos) y la organización de sucesos y estados. Captura el control, aquel aspecto de un sistema que describe las secuencias de operaciones que se producen sin tener en cuenta lo que hagan las operaciones, aquello a lo que afecten o la forma en que están implementadas. Se representará gráficamente mediante:
  - Diagramas de secuencia propuesto por UML.
  - Diagramas de estados propuesto por UML.

- **Modelo funcional.** Describe las transformaciones de valores de datos (funciones, correspondencias, restricciones y dependencias funcionales) que ocurren dentro del sistema. Captura lo que hace el sistema, independientemente de cuando se haga o de la forma en que se haga. Se representa mediante:
  - Diagramas de actividad propuestos por UML.

~~1.3~~ 2.3 Lenguaje unificado de modelamiento

Con formato: Numeración y viñetas

Unified Modeling Language (UML) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar, debido a que ha sido concebido por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh, (Método de Booch, de OMT y de OOSE)

Hay que tener en cuenta que el estándar UML no define un proceso de desarrollo específico, por tanto se lo utilizará sólo como una notación. Los modelos de UML que utilizaremos en este proyecto son los siguientes:

- Diagrama de clases para graficar el modelo de objetos de OMT.
- Diagrama de casos de usos describir el modelo de objetos de OMT.
- Diagrama de estados para graficar el modelo dinámico de OMT.
- Diagrama de secuencia para graficar el modelo dinámico de OMT.
- Diagrama de actividad para graficar el modelo funcional de OMT.
- Diagrama de componentes para ilustrar la fase de diseño del sistema.

~~1.3.1~~2.3.1 Diagrama de clases

Con formato: Numeración y viñetas

Forma parte de la vista estática del sistema. En el diagrama de clases como ya hemos comentado será donde definiremos las características de cada una de las clases, interfaces, colaboraciones y relaciones de dependencia y generalización.

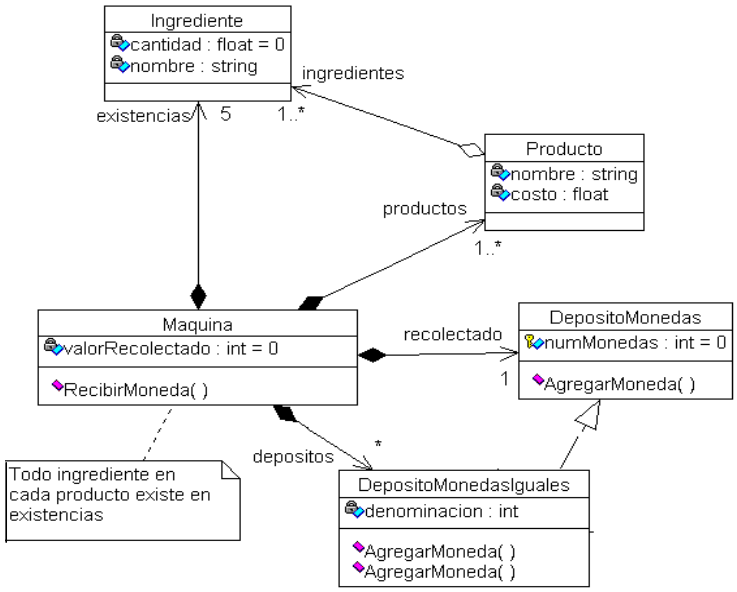


Figura 2.2~~2.2~~: Diagrama de clases[7].

Notación

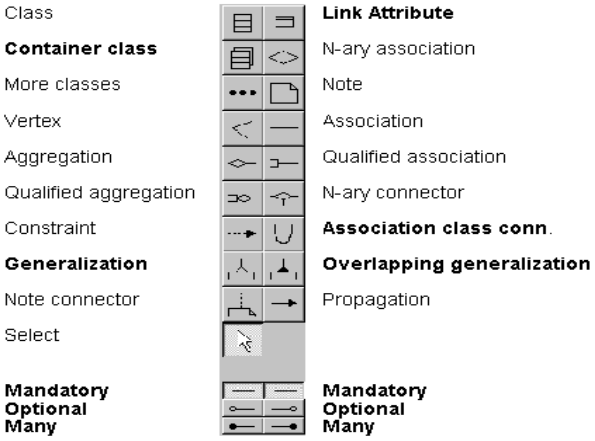


Figura 2.3~~2.3~~: Notación diagrama de clases[8].

7 Tomado de: www.ifi.uio.no/in219/verktoy/doc/html/doc/user/mg/dgmsomt.html#82494
8 Tomado de : www.ifi.uio.no/in219/verktoy/doc/html/doc/user/mg/dgmsomt1.html#1037118

~~1.3.2~~2.3.2 Diagrama de casos de uso.

Con formato: Numeración y viñetas

Se emplean para visualizar el comportamiento de una parte del sistema o de una sola clase. El diagrama de uso especifica como deben comportarse y no como están implementadas las partes que define. Por ello es un buen sistema de documentar partes del código que deban ser reutilizables por otros desarrolladores. El diagrama también puede ser utilizado para que los expertos de dominio se comuniquen con los informáticos sin llegar a niveles de complejidad. Un caso de uso especifica un requerimiento funcional, es decir, indica: Esta parte debe hacer esto, cuando pase esto. Se definen dos tipos de modelamiento, el modelo de casos de uso de contexto y el modelo de casos de uso de requisitos.

Notación

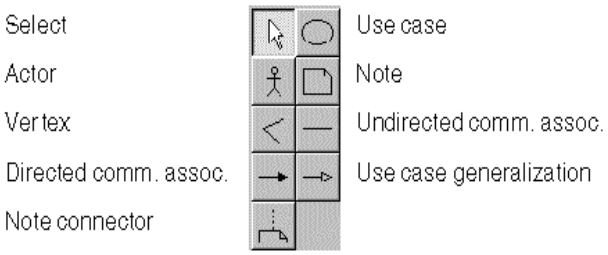


Figura 2.4~~2.4~~: Notación diagrama de casos de uso.[9]

Modelado del contexto

Se debe modelar la relación del sistema con los elementos externos, ya que son estos elementos los que forman el contexto del sistema.

Los pasos a seguir son:

- Identificar los actores que interactúan con el sistema.

9 Tomado de: www.ifi.uio.no/in219/verktoy/doc/html/doc/user/mg/dgmsuml8.html#77545

- Organizar a los actores.
- Especificar sus vías de comunicación con el sistema.

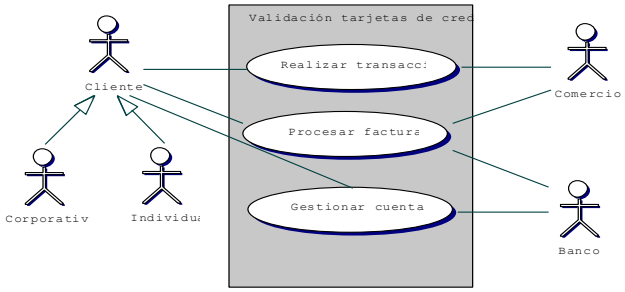


Figura 2.5~~2.5~~: Modelo de contexto[10].

Modelado de requisitos

Los requisitos establecen un contrato entre el sistema y su exterior, definen lo que se espera que realice el sistema, sin definir su funcionamiento interno. Es el paso siguiente al modelado del contexto, no indica relaciones entre autores, solo indica cuales son las funcionalidades (requisitos) del sistema. Se incorporan los casos de uso necesarios que no son visibles desde los usuarios del sistema.

Para modelar los requisitos es recomendable:

- Establecer su contexto, podemos usar un diagrama de casos de uso.
- Identificar las necesidades de los elementos del contexto (Actores).
- Nombrar esas necesidades, y darles forma de caso de uso.
- Identificar que casos de uso pueden ser especializaciones de otros, o buscar especializaciones comunes para los casos de uso ya encontrados.

10 Tomado de: www.programacion.com/tutorial/uml/3/#uml_uso_contexto

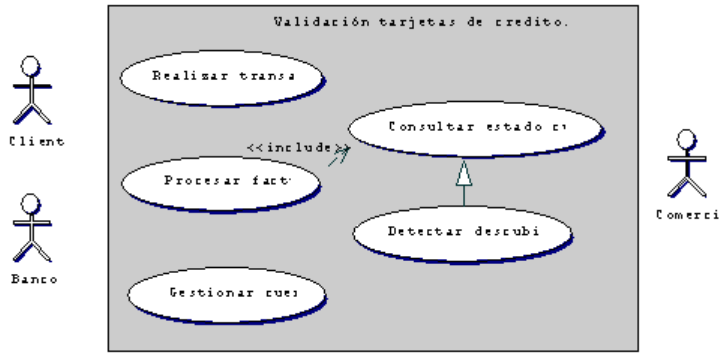


Figura 2.6~~2.6~~: Modelo de requisitos[11].

~~1.3.3~~2.3.3 Diagrama de estados

Con formato: Numeración y viñetas

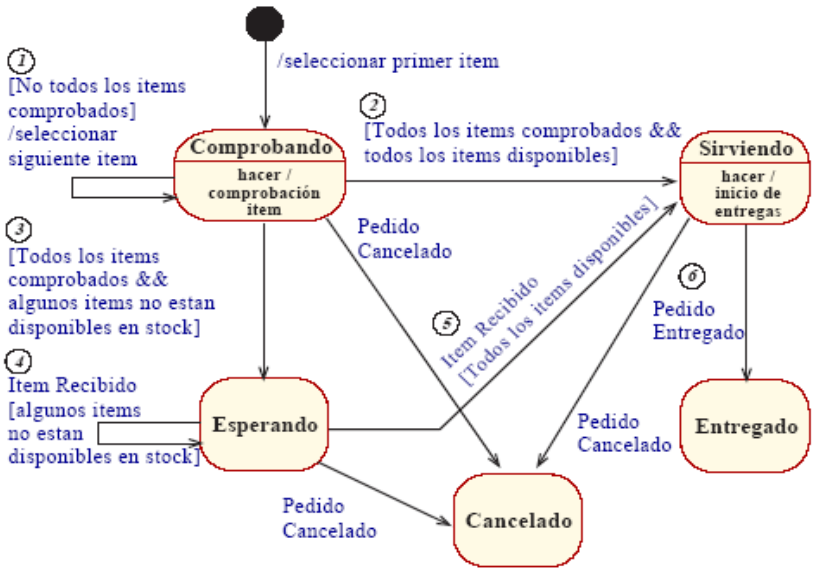


Figura 2.7:~~2.7~~ Diagrama de estados de una central telefónica[12].

Los objetos van pasando por distintos estados a medida se ven influenciados por estímulos externos. El diagrama de estados asigna estos

11 Tomado de: www.programacion.com/tutorial/uml/3/#uml_uso_req
12 Tomado de: ww.exa.unne.edu.ar/depar/areas/informatica/anasistem1/public_ht /Temas_08.pdf

estados, así como los eventos de activación que hacen que un objeto se encuentre en un estado determinado.

Notación

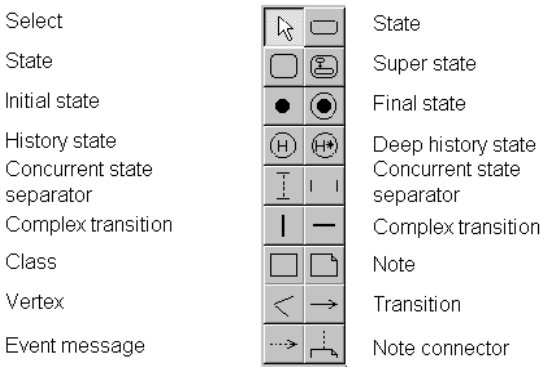


Figura 2.8~~2.8~~: Notación diagrama de estados[13].

~~1.3.4~~2.3.4 Diagrama secuencia.

Con formato: Numeración y viñetas

El diagrama de secuencia forma parte del modelado dinámico del sistema. Se modelan el orden de las llamadas entre clases desde un punto concreto del sistema. Es útil para observar la vida de los objetos en sistema, identificar llamadas a realizar o posibles errores del modelado estático, que imposibiliten el flujo de información o de llamadas entre los componentes del sistema.

Es imposible representar en un solo diagrama de secuencia todas las secuencias posibles del sistema. El diagrama se forma con los objetos que forman parte de la secuencia, estos se sitúan en la parte superior de la pantalla, normalmente en la izquierda se sitúa al que inicia la acción. De estos objetos sale

13 Tomado de: www.ifi.uio.no/in219/verktoy/doc/html/doc/user/mg/dgmsomt6.html#1040899

una línea que indica su vida en el sistema. Esta línea simple se convierte en una línea gruesa cuando representa que el objeto está activo.

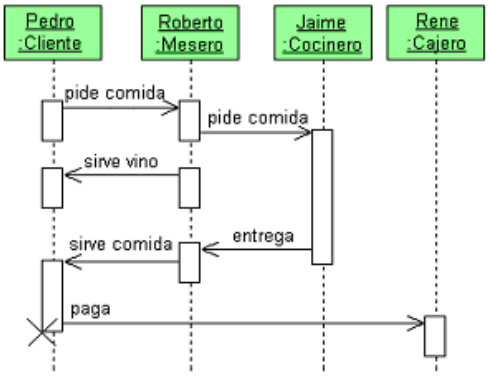


Figura 2.9~~2.9~~: Diagrama de secuencia[14].

Notación

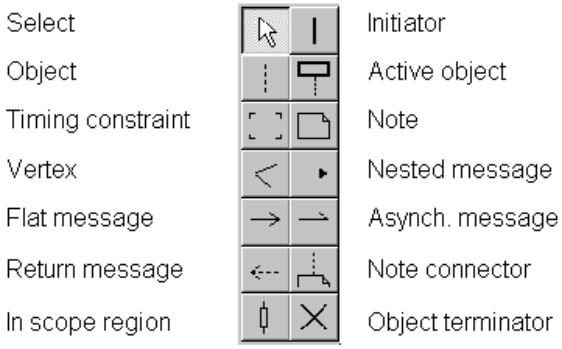


Figura 2.10~~2.10~~: Notación diagrama de secuencia[15].

14 Tomado de: www.ifi.uio.no/in219/verktoy/doc/html/doc/user/mg/dgmsomt5.html#1037391
15 Tomado de: www.ifi.uio.no/in219/verktoy/doc/html/doc/user/mg/dgmsomt5.html#1037392

~~1.3.5~~2.3.5 Diagrama de actividades

Con formato: Numeración y viñetas

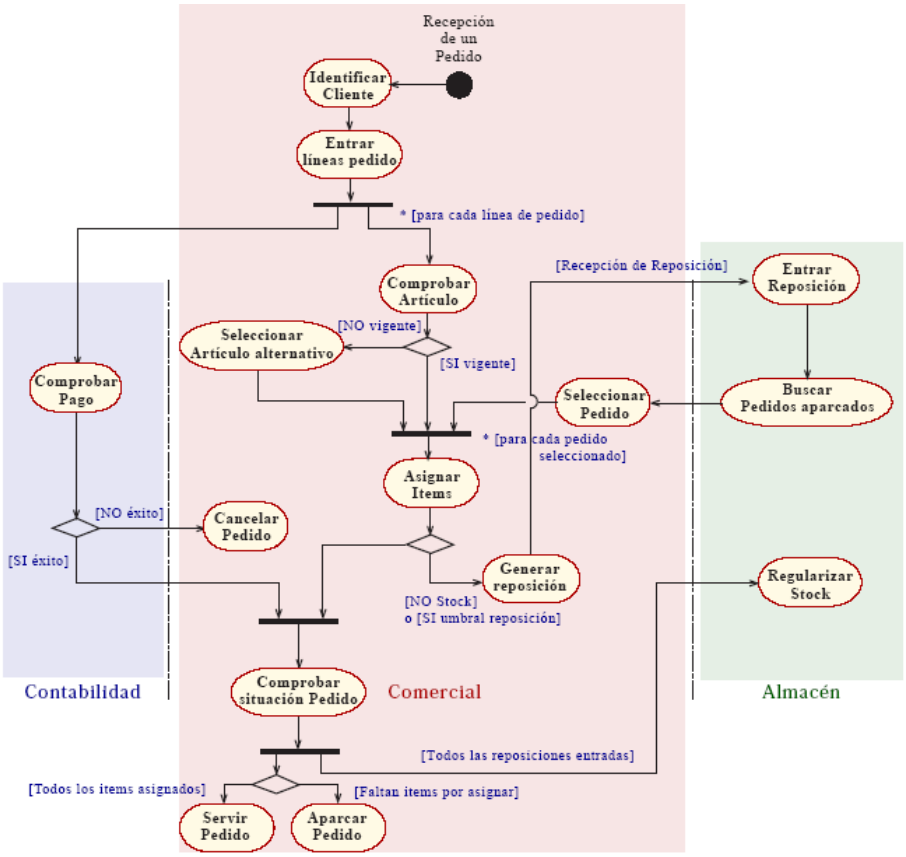


Figura 2.11~~2.11~~: Diagrama de actividades[16].

Los diagramas de actividades muestran la lógica que tiene lugar como respuesta a las acciones generadas internamente. Este tipo de diagrama está relacionado con una clase o caso de uso específico y muestra los pasos que se deben realizar para llevar a cabo una operación determinada.

16 Tomado de: www.microsoft.com/spanish/Msdn/ articulos/archivo/230801/voices/modelsoftware

Notación

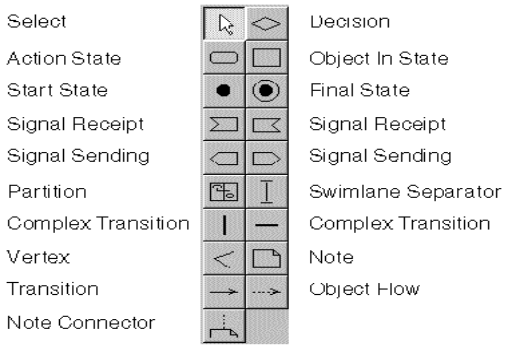


Figura 2.12~~2.12~~ Notación diagrama de actividades[17].

~~1.4~~ 2.4 SQL estándar

Con formato: Numeración y viñetas

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado. Está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

~~1.4.1~~2.4.1 Comandos

Con formato: Numeración y viñetas

Existen dos tipos de comandos SQL:

Data Definition Language (DDL) que permiten crear y definir nuevas bases de datos, campos e índices. Data Manipulation Language (DML) que permiten ordenar, filtrar y extraer datos de la base de datos.

Tabla 2.2: Comandos.

Comandos DDL	
CREATE	Utilizado para crear nuevas tablas, campos e índices.
DROP	Empleado para eliminar tablas e índices.
ALTER	Modifica las tablas agregando campos o cambiando su definición.

17 Tomado de: www.ifi.uio.no/in219/verktoy/doc/html/doc/user/mg/dgmsuml1.html#78096

Comandos DML	
SELECT	Consulta registros que satisfagan un criterio determinado.
INSERT	Carga lotes de datos en la base de datos en una única operación.
UPDATE	Modifica los valores de los campos y registros especificados.
DELETE	Utilizado para eliminar registros de una tabla de una base de datos.

~~1.4.2~~2.4.2 Cláusulas

Con formato: Numeración y viñetas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Tabla 2.3: Cláusulas.

Cláusula	Descripción
FROM	Especifica la tabla de la cual se van a seleccionar los registros.
WHERE	Especifica las condiciones para seleccionar un registro.
GROUP BY	Separa los registros seleccionados en grupos específicos.
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo.
ORDER BY	Ordena los registros de acuerdo con un orden específico.

~~1.4.3~~2.4.3 Operadores lógicos

Con formato: Numeración y viñetas

Los operadores lógicos sirven para combinar condiciones.

Tabla 2.4: Operadores lógicos.

Operador	Uso
AND	Es el "y" lógico. Sólo si ambas son ciertas, devuelve verdadero.
OR	Es el "o" lógico. Si alguna de las dos es cierta, devuelve verdadero.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

~~1.4.4~~2.4.4 Operadores de comparación

Con formato: Numeración y viñetas

Los operadores de comparación son operadores en su mayoría binarios que nos permiten comparar variables devolviendo un valor bolean, 1 (TRUE) si se cumple la condición y 0 (FALSE) en el caso contrario.

Tabla 2.5: Operadores de comparación.

Operador	Uso
<	Menor que.
>	Mayor que.
<>	Distinto de.
<=	Menor o igual que.
>=	Mayor o igual que.
=	Igual que.
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo.
IN	Utilizado para especificar registros de una base de datos.

~~1.4.5~~2.4.5 Funciones de agregado

Con formato: Numeración y viñetas

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Tabla 2.6: Funciones de agregado.

Función	Descripción
AVG	Calcula el promedio de los valores de un campo determinado.
COUNT	Utilizada para devolver el número de registros de la selección.
SUM	Devuelva la suma de todos los valores de un campo determinado.
MAX	Utilizada para devolver el valor más alto de un campo especificado.
MIN	Utilizada para devolver el valor más bajo de un campo especificado.

~~1.4.6~~2.4.6 SQL no estándar

Con formato: Numeración y viñetas

INTERSECT y MINUS son como la sentencia UNION, excepto que INTERSECT produce filas que aparecen en ambas consultas, y MINUS introduce filas que resultan de la primera consulta, pero no de la segunda. Generación de construcciones de informe: la cláusula COMPUTE es puesta al final de una

consulta para poner el resultado en una función agregada al final del listado, como COMPUTE SUM(PRECIO)

Además, algunos DBMS permiten usar más funciones en listas Select, excepto que estas funciones (algunas funciones de carácter permiten resultados de múltiples filas) vayan a ser usadas con un valor individual (no grupos), en consultas de simples filas. Las funciones deben ser usadas sólo con tipos de datos apropiados. Aquí hay algunas funciones matemáticas:

Tabla 2.7: Funciones numéricas.

ABS(X)	Valor absoluto.
CEIL(X)	X es un valor decimal que será redondeado hacia arriba.
FLOOR(X)	X es un valor decimal que será redondeado hacia abajo.
GREATEST(X,Y)	Devuelve el más grande de los dos valores.
LEAST(X,Y)	Devuelve el más pequeño de los dos valores.
MOD(X,Y)	Devuelve el resto de X / Y.
POWER(X,Y)	Devuelve X elevado a Y.
ROUND(X,Y)	Redondea X a Y lugares decimales. Si se omite Y, X se redondea al entero más próximo.
SIGN(X)	Devuelve menos sí X<0, sino un más.
SQRT(X)	Devuelve la raíz cuadrada de X.

Tabla 2.8: Funciones de caracteres.

LEFT(<string,X)	Devuelve los X caracteres a la izquierda de la cadena.
RIGHT(<string,X)	Devuelve los X caracteres más a la derecha de la cadena.
UPPER(<string)	Convierte la cadena a mayúsculas.
LOWER(<string)	Convierte la cadena a minúsculas.
INITCAP(<string)	Convierte el primer caracter de la cadena a mayúscula.
LENGTH(<string)	Devuelve el número de caracteres de cadena.
<string||<string	Concatena dos cadenas de texto.
LPAD(<string,X,'*')	Rellena la cadena por la izquierda con el * (o el caracter que haya entre las comillas).

RPAD(<string,X,'*')	Rellena la cadena por la derecha con el * (o con el caracter que haya entre las comillas).
SUBSTR(<string,X,Y)	Extrae Y letras comenzando en la posición X.
NVL(<column,<value)	Cualquier Null de la <column será sustituido por lo que haya en <value. Si el valor de la columna no el NULL, NVL no tiene efecto.

~~1.5 -~~ 2.5 Comparación de sentencias SQL

Tabla 2.9: Tabla comparativa de sentencias SQL.

CASO	INFORMIX	SQL SERVER	ORACLE 9I	SYBASE	MY SQL
ltrim, rtrim, trim	ltrim, rtrim, trim	ltrim, rtrim	ltrim, rtrim, trim	ltrim, rtrim, trim	ltrim, rtrim, trim
select a + b from	No funciona	SI funciona	No funciona	SI funciona	No funciona
Select a, b from	Si funciona	Si funciona	Si funciona	Si funciona	Si funciona
Select a as uno, b as dos from	Si funciona	Si funciona	Si funciona	Si funciona	Si funciona
Select a as "uno", b as "dos"	No funciona	Si funciona	Si funciona	Si funciona	Si funciona
Select a as 'uno', b as 'dos'	No funciona	Si funciona	No funciona	Si funciona	Si funciona
Select a uno, b dos from	Si funciona	Si funciona	Si funciona	Si funciona	Si funciona
Select a "uno", b "dos" from	No funciona	Si funciona	Si funciona	Si funciona	Si funciona
Select a 'uno', b 'dos' from	No funciona	Si funciona	No funciona	Si funciona	Si funciona
Tipo de dato Date	Si tiene	En su lugar datetime	Si, en realidad es date time	Si tiene	Si tiene
Tipo de dato datetime	Si tiene	Si tiene	Si, pero el date se comporta como datetime	Si tiene	Si tiene
select * from a_tipo_activo where fecha1_tac>= date('2003-01-01');	Si funciona, pero con 'mm-dd-aaaa'	No funciona	No funciona	Si funciona	No funciona
selec * from a_tipo_activo where fecha1_tac>='2003-01-01'; siendo fecha1_tac tipo datetime	No funciona, pero si fecha1_tac es date si	Si funciona	Si funciona pero dd-mm-yyyy	Si funciona	Si funciona

select nombre into :ls_nombre from prueba where fecha1>=:var_date using sqlca; fecha 1 es datetime	Si funciona	Si funciona	Si funciona	Si funciona	Si funciona
select nombre into: ls_nombre from prueba where fecha1>=:var_date_time using sqlca; fecha 1 es date	Si funciona, no le considera a la hora, solo compara fechas	Si funciona, pero considera la hora exacta ya que SQLServer solo tiene datetime	Si funciona, pero considera la hora exacta por defecto ya que Oracle solo tiene datetime	Si funciona, se pone automáticamente 00:00 en la fecha 1	Si funciona no le hace caso a la hora, busca solo por fecha
select * from prueba where fecha_date_time>=:var_date	Si funciona	Si funciona	Si funciona	Si funciona	Si funciona
select nombre into: ls_nombre from prueba where fecha1>=:var_date_time using sqlca; fecha 1 es date	Si funciona, no le toma en cuenta a la hora	Si funciona, considera 00:00 para la hora	Si funciona, pero considera la hora exacta por defecto ya que Oracle solo tiene datetime	Si Funciona, le aumenta a la fecha la hora 00:00	Si funciona, no le considera a la hora
Tamaño máximo que puede tener un registro char	32767	8000	2000	32767	255
select fecha1 into fecha_datetime using sqlca; fecha 1 es date en las bases que se puede	Si funciona y le asigna 00:00 a la hora	Si funciona, le asigna a la hora la que estaba en la base	Si funciona, le asigna la hora que estaba en la base	Si funciona, le asigna 00:00 a la hora	Si funciona, le asigna 00:00 a la hora
select fecha2 into: fecha_date using sqlca; fecha2 es datetime	Si funciona le pone 00:00 a la hora	No funciona, numeric value out of range	No funciona, numeric value out of range	No funciona, data truncated	No funciona, rigth truncated
update prueba set cantidad =cantidad + 1 where nombre='PEDRO';	Si funciona	Si funciona	Si funciona, pero hay que tener cuidado con los tipos de datos, ya que almacena de 10 en 10	Si, funciona	Si funciona

~~1.6 -~~ 2.6 Programación orientada a objetos

Con formato: Numeración y viñetas

El principio de la orientación a objetos es la fusión de datos y procesos, tradicionalmente separada. Así, un sistema se concibe como un conjunto de objetos que se comunican mediante mensajes. Además, es una técnica para resolver los problemas de una manera análoga a los métodos utilizados en la realidad. Los mayores beneficios son:

- Mejora la calidad del software generado.
- Acorta el tiempo de desarrollo.
- Aumenta la productividad.
- Da pie a la reutilización del software generado.

La programación orientada a objetos (POO) se basa en cuatro elementos principales: abstracción, herencia, encapsulamiento y polimorfismo. A continuación se explican cada uno de ellos:

~~1.6.1~~2.6.1 Abstracción

Con formato: Numeración y viñetas

Es la capacidad[18] de un objeto de cumplir sus funciones independientemente del contexto en el que se lo utilice; o sea, un objeto "cliente" siempre expondrá sus mismas propiedades y dará los mismos resultados a través de sus eventos, sin importar el ámbito en el cual se crea[18]. Permitiendo un mejor manejo de la complejidad del problema, al permitir suprimir o posponer detalles irrelevantes en cierto momento, y concentrarse más bien, en detalles esenciales.

18 Tomado de: microsoft.com/spanish/msdn/comunidad/dce/1/entrenamiento/foxpro/1.asp

~~1.6.2~~2.6.2 Herencia

Con formato: Numeración y viñetas

En términos simples la herencia en la POO. es la capacidad que tiene los objetos para heredar todas o algunas de sus características (datos) y comportamiento (procedimientos) a sus descendientes o herederos, permitiendo así la reutilización de código lo que en la práctica se traduce en una herramienta muy potente de programación.

Esta herencia puede ser al nivel de estado (características) o protocolo (métodos y procedimientos) o ambas.

~~1.6.3~~2.6.3 Polimorfismo

Con formato: Numeración y viñetas

Se refiere a la capacidad que tiene el objeto para hacer lo adecuado después de haber recibido un mensaje, independientemente de quien lo haya enviado. Hay que tener en cuenta que el polimorfismo se mueve a través de la herencia, dicho de otra forma trata de hacer posible enviar mensajes genéricos a objetos y que estos realicen lo estipulado de acuerdo a su codificación interna, es decir, la acción varía en relación con el receptor del mensaje.

~~1.6.4~~2.6.4 Encapsulamiento

Con formato: Numeración y viñetas

Uno de los objetivos primordiales de la POO, es la creación de objetos que funcionen como entidades complejas, es decir autosuficientes, una de las reglas del encapsulamiento es que el programador nunca necesita acceder directamente a los datos de un objeto, en vez de esto se deben definir métodos dentro del objeto que gobiernen toda la manipulación de datos. El

encapsulamiento está referido, también, a los métodos de cada objeto en el sentido que su código interno debe ser transparente a la hora de activarse.

~~1.7~~ 2.7 Patrones de diseño

Con formato: Numeración y viñetas

Los Patrones de diseño indican como construir software[19], utilizar las clases y los objetos de forma conocida. Los patrones de diseño proponen una forma reutilizar la experiencia de los desarrolladores, para ello clasifica y describe formas de solucionar problemas que ocurren de forma frecuente en el desarrollo.

Por tanto está basado en la recopilación del conocimiento de los expertos en desarrollo de software. Sus características son:

- Son soluciones concretas. Proponen soluciones a problemas concretos, no son teorías genéricas.
- Son soluciones técnicas. Indican resoluciones técnicas basadas en programación orientada a objetos.
- Se utilizan en situaciones frecuentes. Ya que se basan en la experiencia acumulada la resolver problemas reiterativos.
- Favorecen la reutilización de código. Ayudan a construir software basado en la reutilización, a construir clases reutilizables.
- El uso de un patrón no se refleja en el código. Al aplicar un patrón, el código resultante no tiene por que delatar el patrón que lo inspiró.

19 Tomado de: www.ajlopez.com.

2.7.1 Clasificación

Clasificación según su propósito:

- Patrones de creación: Tratan la creación de instancias.
- Patrones estructurales: Tratan la relación entre clases, la combinación clases y la formación de estructuras de mayor complejidad.
- Patrones de comportamiento: Tratan la interacción entre clases.

Clasificación según su ámbito:

- De clase: Basados en la herencia de clases.
- De objeto: Basados en la utilización dinámica de objetos.

Patrones de creación

Los patrones de creación abstraen la forma en que se crean los objetos, de forma que permite tratar las clases a crear de forma genérica apartando la decisión de que clases crear o como crearlas, así según a donde quede desplazada dicha decisión se habla de patrones de clase (utiliza la herencia para determinar la creación de las instancias, es decir en los constructores de las clases) o patrones de objeto (es en métodos de los objetos creados donde se modifica la clase)

Tabla 2.10: Tabla de patrones de creación.

Abstract Factory	Nos da una interfaz para crear objetos de alguna familia, sin especificar la clase en concreto.
Builder	Separa la construcción de un objeto complejo, de su representación. De esa manera, el mismo proceso de construcción puede crear diferentes resultados.
Factory Method	Se define una interfaz para crear objetos, como en el Abstract Factory, pero se delega a las subclases implementar la creación en concreto.
Prototype	Mediante una instancia prototípica, conseguimos otras instancias de ese objeto.
Singleton	Nos consigue dar un solo objeto de la clase, en cualquier momento de la aplicación.

Patrones estructurales

Tratan de conseguir que cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos. Lo fundamental son las relaciones de uso entre los objetos, y estas están determinadas por las interfaces que soportan los objetos. Estudian como se relacionan los objetos en tiempo de ejecución. Sirven para diseñar las interconexiones entre los objetos.

Tabla 2.11: Tabla de patones estructurales.

Object Adapter	Permite convertir una interfaz de una clase, en otra, que es la esperada por algún cliente.
Bridge	Desacopla una abstracción de su implementación en concreto. Luego, podemos cambiar la implementación, o la abstracción, sin cambiar la otra.
Composite	Compone objetos en una estructura de árbol, donde los objetos compuestos se tratan de forma similar a los objetos simples.
Decorator	Agrega responsabilidad a un objeto dinámicamente, dándonos una alternativa a la extensión de una clase, en lugar de usar subclases.
Facade	Provee una interfaz unificada a un conjunto de funciones de un subsistema. Es una interfase de alto nivel, para facilitar el uso del subsistema.
Flyweight	Permite compartir objetos, sin repetirlos en el sistema, eficientemente.
Proxy	Provee un subrogado a otro objeto, para controlar el acceso

Patrones de comportamiento

Los patrones de comportamiento hablan de como interaccionan entre sí los objetos para conseguir ciertos resultados.

Tabla 2.12: Tabla de patrones de comportamiento.

Command	Encapsula el requerimiento a un objeto, permitiendo incluso el "undo" de la operación.
Interpreter	Construye una representación de la gramática de un lenguaje, junto con su intérprete.
Iterator	Nos da un modo de acceder a los elementos de un objeto colección o similar, sin exponer su estructura interna.

Mediator	Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.
Memento	Sin necesitar entrar en la estructura interna de un objeto, permite capturar su estado para, por ejemplo, poder restaurarlo más adelante.
Observer	Define una relación uno a muchos, entre unos objetos y otros que están interesados en sus cambios, de nuevo, sin caer en el acople entre los mismos.
State	Permite a un objeto cambiar su conducta cuando cambia su estado interno, simulando que cambia de clase.
Strategy	Define una familia de algoritmos, y los hace intercambiables.
Template Method	Define el esqueleto de una operación, cuyas operaciones más básicas, quedan delegadas en subclases.
Visitor	Nos permite recorrer una estructura (un árbol por ejemplo), aplicando una operación a cada elemento.

Antipatrones

Los patrones nos ofrecen una forma de resolver un problema típico, los antipatrones nos enseñan formas de enfrentarse a problemas con consecuencias negativas conocidas. Los antipatrones se basan en la idea de que puede resultar más fácil detectar fallos en el desarrollo del proyecto que elegir el camino correcto, o lo que es lo mismo, descartar las alternativas incorrectas nos puede ayudar a la elección de la mejor alternativa. Los antipatrones se clasifican en antipatrones de desarrollo, de arquitectura de software y de gestión de proyectos.

~~1.8~~ 2.8 Herramientas

Con formato: Numeración y viñetas

~~1.8.1~~2.8.1 El Framework de Visual Studio .NET

El Framework de .Net es una infraestructura sobre la que se reúne todo un conjunto de lenguajes y servicios que simplifican enormemente el desarrollo de aplicaciones. Mediante esta herramienta se ofrece un entorno de ejecución altamente distribuido, que permite crear aplicaciones robustas y escalables.

Los principales componentes de este entorno son:

- Lenguajes de compilación.
- Biblioteca de clases de .Net.

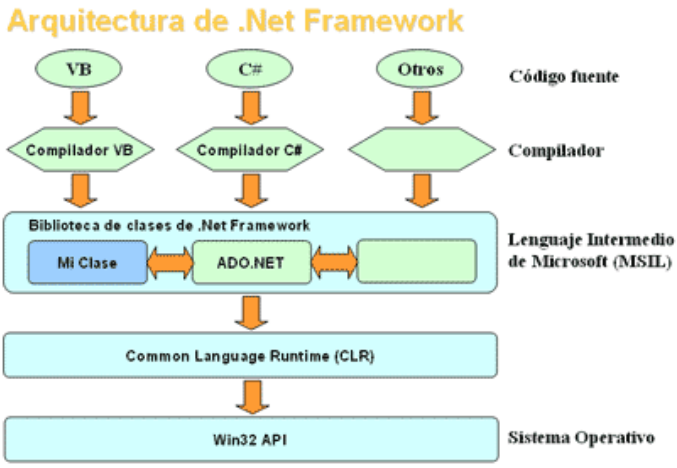


Figura 2.13~~2.13~~: Arquitectura de .Net Framework[20].

~~1.8.2~~2.8.2 Lenguajes de compilación

Con formato: Numeración y viñetas

El Framework de .Net soporta múltiples lenguajes de programación y aunque cada lenguaje tiene sus características propias, es posible desarrollar cualquier tipo de aplicación en más de 30 lenguajes adaptados a .Net.

~~1.8.3~~2.8.3 Common Language Runtime (CLR)

Con formato: Numeración y viñetas

El CLR es el verdadero núcleo del Framework de .Net, ya que es el entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes, ampliando el conjunto de servicios que ofrece el sistema operativo estándar Win32.

20 Tomado de: www.msdn.microsoft.com/netframework.

La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por .Net en un mismo código, denominado código intermedio (MSIL, Microsoft Intermediate Lenguaje). Para generar dicho código el compilador se basa en el Common Language Specification (CLS) que determina las reglas necesarias para crear código MSIL compatible con el CLR.

De esta forma, indistintamente de la herramienta de desarrollo utilizada y del lenguaje elegido, el código generado es siempre el mismo. Sin embargo, el código generado en MSIL no es código de máquina y por tanto de una herramienta denominada compilador JIT (Just-In-Time) que genera el código de máquina real.

~~1.8.4~~2.8.4 Biblioteca de clases de .Net

Con formato: Numeración y viñetas

Cuando se está programando una aplicación muchas veces se necesitan realizar acciones como manipulación de archivos, acceso a datos, conocer el estado del sistema, implementar seguridad, etc. El Framework organiza toda la funcionalidad del sistema operativo en un espacio de nombres jerárquico.

Para ello, el Framework posee un sistema de tipos universal, denominado Common Type System (CTS) que permite que el programador pueda interactuar con los tipos que se incluyen en el propio Framework (biblioteca de clases de .Net) con los creados por él mismo (clases). De esta forma se aprovechan las ventajas propias de la programación orientada a objetos, como la herencia de clases predefinidas para crear nuevas clases, o el polimorfismo de clases para actualizar o ampliar funcionalidades de clases ya existentes.

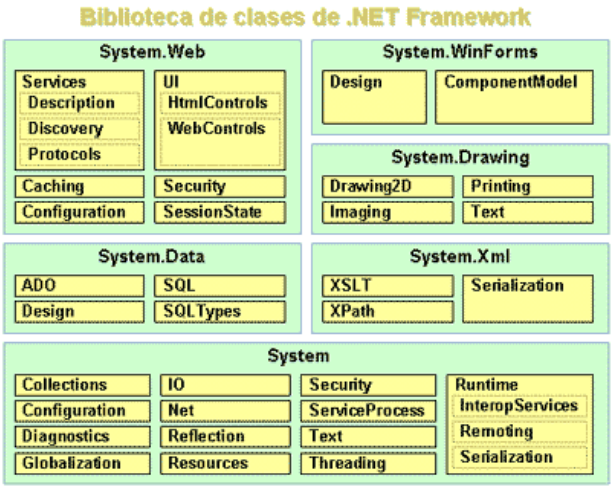


Figura 2.142.14: Biblioteca de clases de .Net Framework[21].

Para ello, el Framework posee un sistema de tipos universal, denominado Common Type System (CTS) que permite que el programador pueda interactuar con los tipos que se incluyen en el propio Framework (biblioteca de clases de .Net) con los creados por él mismo (clases). De esta forma se aprovechan las ventajas propias de la programación orientada a objetos, como la herencia de clases predefinidas para crear nuevas clases, o el polimorfismo de clases para actualizar o ampliar funcionalidades de clases ya existentes.

La biblioteca de clases de .Net Framework incluye, entre otros, tres componentes clave:

- ASP.NET para construir aplicaciones y servicios Web.
- Windows Forms para desarrollar interfaces de usuario.
- ADO.NET para conectar las aplicaciones a bases de datos.

21 Tomado de: www.msdn.microsoft.com/netframework.

La forma de organizar la biblioteca de clases de .Net dentro del código es a través de los espacios de nombres (namespaces), donde cada clase está organizada en espacios de nombres según su funcionalidad. Por ejemplo, para manejar ficheros se utiliza el espacio de nombres System.IO y si lo que se quiere es obtener información de una fuente de datos se utilizará el espacio de nombres System.Data.

~~1.8.5~~2.8.5 ADO.NET

Con formato: Numeración y viñetas

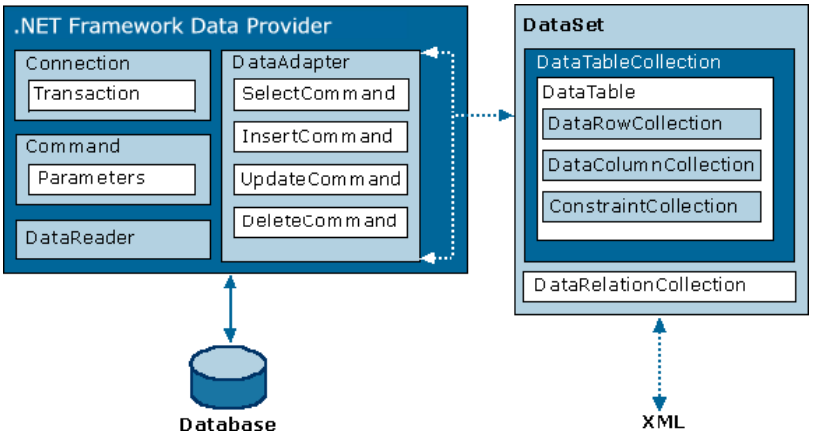


Figura 2.15~~2.15~~: Arquitectura de ADO.NET[22].

ADO.NET es el modelo de acceso a datos para aplicaciones basadas en .Net. Puede ser utilizado para acceder a gestores de datos relacionales sistemas como SQL Server 2000, Oracle, y muchas otras fuentes de datos con proveedores para OLE DB u ODBC. El diagrama anterior ilustra los componentes de la arquitectura de ADO.NET.

22 Tomado de: www.msdn.microsoft.com/library/spa/default.asp?url=/library/SPA/cpguide/html/cpconconnectingtosqlserverusingadonet.asp.

~~1.8.6~~2.8.6 Organización de los namespace para ADO.NET

Con formato: Numeración y viñetas

ADO.NET se encuentra en la biblioteca System.Data.dll, y ofrece clases en cinco namespace[23] o espacios de nombres bien diferenciados:

- **System.Data:** es el espacio de nombres primario. Dentro de este espacio de nombres se encuentra un conjunto de clases que representan, una base de datos virtual, tablas, filas, columnas, relaciones, etc. Ninguna de estas clases ofrece conexión alguna con un origen de datos, sino que simplemente representan los datos en sí mismos.
- **System.Data.Common:** entrega clases comunes entre distintos orígenes de datos, estas clases sirven de clase base para las que están contenidas en los dos espacios de nombres que vienen a continuación.
- **System.Data.OleDb:** contiene una serie de clases permiten conectarse con cualquier origen de datos e interactuar con él, al tiempo que sirven de "intermediarios" entre el origen de datos y las clases del espacio de nombres System.Data que no tienen conexión alguna con dicho origen de datos. Las clases de System.Data.OleDb usan OLEDB como tecnología subyacente.
- **System.Data.SqlClient:** contiene clases que permiten interactuar con orígenes de datos SQL Server de un modo mucho más directo que OLEDB, mejorando el rendimiento para este tipo de origen de datos. Por lo tanto, solamente se pueden utilizar para acceder a gestores de datos de SQL Server. El uso de sus clases es prácticamente equivalente al de las que se encuentran en System.Data.OleDb.

23 Provee una forma de organizar clases y otros tipos relacionados, es una agrupación lógica en lugar de una agrupación física.

- **System.Data.SqlTypes:** este espacio de nombres ofrece los tipos primitivos que usa SQL Server. Obviamente, aunque se pueden usar los tipos equivalentes del CTS, los que se incluyen en este espacio de nombres están optimizados para trabajar con SQL Server.

Dentro de la organización de los namespace cada proveedor entrega una implementación de los objetos Connection, Command, DataReader, y DataAdapter. La implementación SqlClient empieza con el prefijo "Sql" y la implementación OleDb con el prefijo "OleDb." Por ejemplo, la implementación SqlClient para el objeto Connection es SqlConnection, y para la implementación de OleDb su equivalente es OleDbConnection.

~~1.8.7~~2.8.7 Modo de trabajo de ADO.NET

Con formato: Numeración y viñetas

ADO.NET trabaja de forma desconectada lo que significa que reduce al máximo el número de "conexiones abiertas" en la base de datos, utilizando DataSets para tomar una parte de los datos y mantenerlos en memoria en forma relacional, tal cual son almacenados. Esto permite brindarle datos a la aplicación cliente sin sacrificar el rendimiento al mantener conexiones abiertas a la base de datos.

Existen objetos que trabajan en forma conectada para aplicaciones que por sus requerimientos y por la naturaleza del negocio requieren una arquitectura conectada, el objeto bandera del modelo conectado en ADO.NET se llama DataReader, cada clase implementa uno propio, SQLDataReader u OledbDataReader.

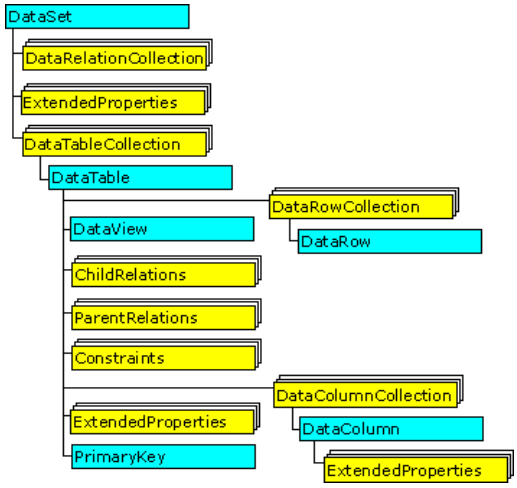


Figura 2.16~~2.16~~: Modelo de objetos del dataset[24].

Los clientes ADO.NET utilizan cursores de sólo lectura y de sólo reenvío para leer datos. Los clientes que leen y procesan datos directamente pueden utilizar el objeto DataReader de ADO.NET, que no proporciona caché para los datos devueltos. Los datos también pueden leerse en un objeto DataSet, que actúa como un caché para los datos que devuelven las consultas SQL

~~1.8.8~~2.8.8 Proveedores de datos para .NET

Con formato: Numeración y viñetas

ADO.NET utiliza objetos que proveen acceso a las fuentes de datos como Connection, Command, DataReader, y DataAdapter.

ADO.NET maneja dos tipos de proveedores: nativos y puente. Los proveedores puente permiten utilizar diseñados para tecnología de acceso a datos anteriores OLE DB y ODBC. Los proveedores nativos como SQL Server y Oracle ofrecen mejoramientos en el rendimiento porque tienen una capa menos de abstracción.

24 Tomado de: www.microsoft.com/spanish/msdn/comunidad/uni.net/default.asp

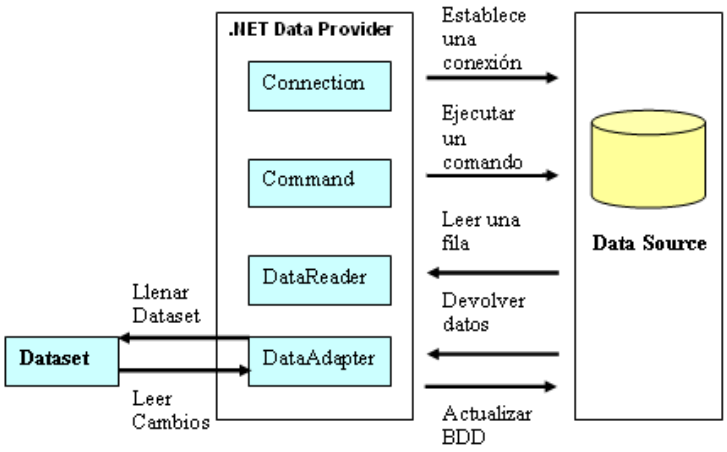


Figura 2.172.17: Proveedores de datos para .NET.

- **El proveedor de datos para SQL Server**. Este es un proveedor para gestores de datos Microsoft SQL Server 7.0 y superiores. Optimiza el acceso con SQL Server, y se comunica directamente con SQL Server utilizando el protocolo nativo de acceso a datos de SQL Server.
- **El proveedor de datos para Oracle**. El Framework de .Net proporciona un proveedor de datos para Oracle habilitando el acceso a los datos mediante el software cliente de conectividad de Oracle, el proveedor de datos soporta el cliente de Oracle desde la versión 8.1.7 y posterior.
- **El proveedor de OLE DB .NET**. Este es un manejador de proveedores para fuentes de datos OLE DB. Es menos eficiente que el proveedor nativo de SQL Server .NET, porque se llama a través de la capa OLE DB cuando se comunica con una base de datos. Para fuente de datos ODBC, se debe utilizar el proveedor de datos ODBC .NET.

~~1.8.9~~2.8.9 Reflexión

Con formato: Numeración y viñetas

Es el mecanismo por el cual se puede crear objetos dinámicamente en tiempo de ejecución, se utiliza para recuperar información sobre:

- Firmas de los métodos. Esto es la estructura de la declaración de los métodos incluida la información de los argumentos.
- Información acerca de las propiedades y miembros.
- Lista de los métodos de las clases.
- Lista de clases contenidas en módulos, si los módulos son .exe, .dll, etc.
- Lista de los tipos definidos en los assemblies.

Se puede usar Reflexión para crear una instancia de un tipo, enlazar el tipo a un objeto existente, o conseguir una referencia a un tipo dentro de un objeto existente.

~~1.8.10~~2.8.10 Jerarquía de un assembly.

Con formato: Numeración y viñetas

Los assemblies contienen módulos, los módulos contienen tipos, y los tipos contienen miembros. Para permitir al código obtener información acerca de todos los componentes de la jerarquía, Reflection nos provee de clases para referenciar los assemblies, módulos, tipos y miembros. Para recuperar la metadata a través de Reflection se debe usar en .NET el namespace System.Reflection.

El namespace System.Reflection contiene clases e interfaces que proporcionan una vista administrada de los campos, los métodos y los tipos cargados, con la posibilidad de crear e invocar tipos dinámicamente.

Los objetos que encapsulan los assemblies, módulos, tipos y miembros son instancias de las clases:

- System.Reflection.Assembly.
- System.Reflection.Module, System.Type.
- System.Reflection.MemberInfo.

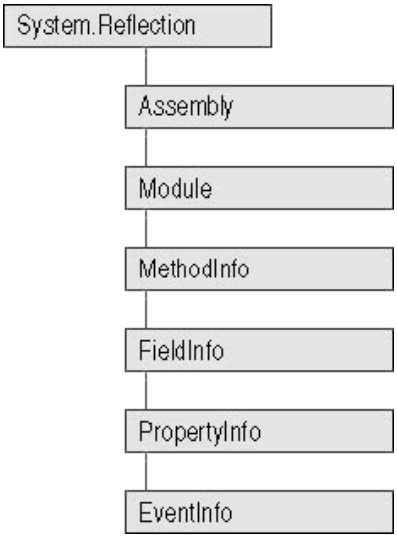


Figura 2.18~~2.17~~: Biblioteca de clases de System.Reflection. [25]

~~1.8.11~~2.8.11 Code Document Object Model (CodeDOM)

Con formato: Numeración y viñetas

CodeDOM es una librería de clases que ofrecen la posibilidad de crear una estructura de programación de espacios de nombres, objetos y construcciones de programación mediante la adición de elementos a diferentes colecciones. CodeDOM permite que se genere código fuente en varios lenguajes de programación en tiempo de ejecución, basándose en un único modelo que representa el código que se va a generar. Para representar el

25 Tomado de: http://mini-soft.net.ru/book/c_sharp/gl16/gl16.php

código fuente, los elementos de CodeDOM se vinculan entre sí para formar una estructura de datos conocida como gráfico CodeDOM, que modela la estructura de parte del código fuente.

El espacio de nombres System.CodeDom define los tipos que pueden representar la estructura lógica del código fuente, independientemente de un lenguaje de programación específico. El espacio de nombres System.CodeDom.Compiler define los tipos para generar código fuente a partir de gráficos CodeDOM y para administrar la compilación del código fuente en los lenguajes admitidos. Los proveedores de compiladores o los programadores pueden extender el conjunto de lenguajes admitidos.

El modelo de código fuente independiente del lenguaje puede ser valioso cuando un programa necesita generar código fuente para un modelo de programa en varios lenguajes o para un lenguaje de destino incierto. Por ejemplo, algunos diseñadores utilizan el CodeDOM como una interfaz de abstracción del lenguaje para generar código fuente en el lenguaje de programación correcto, si dicho lenguaje es compatible con CodeDOM. NET Framework dispone de generadores de código y compiladores de código para C#, JScript y Visual Basic.

Algunos usos habituales de CodeDOM son: Generación de código mediante plantillas: generación de código para ASP.NET, proxies de clientes de servicios Web XML, asistentes para código, diseñadores y otros

mecanismos de emisión de código. Compilación dinámica: compatibilidad para compilación de código en uno o varios lenguajes.

~~1.8~~ 2.9 Persistencia

Con formato: Numeración y viñetas

La persistencia es un tema amplio y abarca distintas áreas, como son las bases de datos, lenguajes de programación, compiladores, la orientación a objetos y la programación, que dan lugar a diferentes perspectivas y planteamientos posibles. Para muchos la persistencia es un paso sencillo, es sólo guardar y recuperar datos. Pero la experiencia, nos ha demostrado que es necesaria una visión global para entender el problema y comprender la convivencia de una solución y la persistencia de objetos.

La persistencia de los objetos [26]es necesaria en todo tipo de entorno, abarca desde los grandes sistemas a dispositivos del tamaño de una tarjeta de crédito o un moderno asistente ya que todos necesitan guardar y recuperar datos de estado.

Las aplicaciones que necesitan manejar datos que deben perdurar, plantean un escenario donde los programadores están obligados a utilizar y conocer detalladamente diferentes técnicas de acceso e interfaces de programación para cada uno de los sistemas de gestión de datos empleados.

Esto significa conocer al menos:

26 Tomado de: Robert D. Poor Hyphos, http:www.media.mit.edu/pia/Research/Hyphos

- Dos lenguajes distintos para plasmar la lógica del negocio: Front End (.Net) y el lenguaje especializado de base de datos.
- El modo de integrar ambos evitando las resistencias por la falta de correspondencia de uno y otro lenguaje; excepción hecha de los sistemas de gestión de datos basados en objetos.

Esta situación ha impulsado la construcción de servicios para almacenar y recuperar datos, servicios de persistencia que desempeñan su labor de forma transparente, uniforme e independiente del sistema plataforma de gestión de datos. Esto permitirá facilitar la tarea del programador, siendo más productiva y centrada en la lógica del negocio. La demanda de servicios de persistencia transparentes, es y ha sido fuente de gran interés para la investigación de importantes proyectos[27] y para la industria con numerosos productos[28].

Esta sección pretende dar una aproximación al problema de la persistencia y su aplicación específica para .Net mostrando:

- El problema de la persistencia estableciendo un marco de análisis sobre criterios estructurales, de organización, funcionales, tecnológicos y económicos.
- Exponer una solución al problema de convertir clases en tablas y objetos en las filas almacenadas en una base de datos relacional.
- Presentar conclusiones que ayuden a formar la opinión de quién lea este proyecto, mediante:

27 Tomado de: DR. MALCOLM ATKINSON, Pjama Project www.dcs.gla.ac.uk/pjava
28 Tomado de: Productos de persistencia: www.persistence.com/products/edgextend/index.php, www.objectmatter.com, www.chimu.com/producst/form/index.html, www.hibertante.com

a. El estudio de documentación relativa a los distintos aspectos del problema incluidos en la bibliografía.
b. Llevar a cabo un prototipo que contenga los requerimientos básicos que ilustre la forma de implementar las prácticas más comunes.
c. Elaborar unas pruebas basadas en un sencillo prototipo para confrontar las opciones de conexión que se han desarrollado.

2.9.1 Conceptos sobre persistencia de objetos

"Persistencia es la capacidad de un lenguaje de programación o entorno de desarrollo de programación para, almacenar y recuperar el estado de los objetos de forma que sobreviva a los procesos que los manipula"[29].

Esta definición indica que el programador no debería preocuparse por el mecanismo interno que hace un objeto persistente, sea este mecanismo soportado por el propio lenguaje de programación usado, o por utilidades de programación para la persistencia, como librerías, framework o compiladores.

En definitiva, el programador debería disponer de algún medio para poder convertir el estado de un objeto, a una representación adecuada sobre un soporte de información, que permitirá con posterioridad revivir o reconstruir el objeto, logrando que como programadores, no debamos preocuparnos de como esta operación es llevada a cabo.

29 Tomado de: BERTRAND MEYER, Object Oriented Software Construcction 2 Edition, Prentice Hall, 1997

~~1.8 -~~ 2.9.2 Servicio de persistencia de objetos

Con formato: Numeración y viñetas

Para presentar el concepto de servicio de persistencia asumiremos que el destino final de los datos serán los sistemas de gestión de datos.

Servicio de persistencia es un sistema o mecanismo programado para posibilitar una interfaz única para el almacenamiento, recuperación, actualización y eliminación del estado de los objetos que pueden ser persistentes en uno o más sistemas gestores de datos.

La definición hecha, considera que el sistema gestor de datos, puede ser un sistema RDBMS, un sistema OODBMS, o cualquier otro sistema; que el estado podría estar repartido sobre varios sistemas, incluso de distinto tipo; y lo más importante, que una capa de persistencia de objetos aporta los elementos necesarios para efectuar la modificación y la eliminación de los objetos persistentes, además del volcado y recuperación del estado en los siguientes gestores de datos. Y todo ello, debería ser efectuado de acuerdo a la definición de persistencia, sin necesidad de traducción explicita por parte del programador. En todos los casos, sea cual sea el tipo de gestor de datos, los servicios de persistencia de objetos, facilitan la ilusión de trabajar con un sistema de gestores de datos de objetos integrado en el lenguaje de programación, ocultando la diferencia entre el modelo de objetos del lenguaje de programación, y el modelo de datos del sistema empleado como base de datos. A pesar de todos lo dicho una capa de persistencia no es un sistema gestor datos orientado a objetos. El servicio de persistencia es un componente

esencial de todo sistema gestor de datos de objetos que resuelve otros aspectos, además de la persistencia[30].

2.9.3 Persistencia ortogonal

Dos características serán ortogonales, si el uso de una no afecta a la otra, esto es, son independientes entre sí. Programas y persistencia serán ortogonales, si la forma en la que los objetos son manipulados por estos programas, es independiente de la utilización de la persistencia, que los mismos mecanismos operan tanto sobre objetos persistentes como sobre objetos transitorios, ambas categorías serían tratadas de la misma manera con independencia de su característica de persistencia.

La persistencia de un objeto deber ser ortogonal al uso, tipo e identificación. Esto es, cualquier objeto debería poder existir el tiempo que sea preciso, ser manipulado sin tener en cuenta, si la duración de su vista, supera al proceso que los creó y su identificación no estar vinculada al sistema de tipos, como la posibilidad de dar nombres a los objetos[31].

Los beneficios que aporta la persistencia ortogonal son importantes: mayores cotas de facilidad de mantenimiento, corrección, continuidad del código y productividad de desarrollo. Se consigue:

- Menos código. Una semántica para expresar las operaciones de persistencia más simple de usar y entender. Evita la duplicidad de

30 Tomado de: M.P Atkinson, F. Bancilhon, D.J. Dewitt, K. R. Dittrich, D. Maier, and S. B. Zdonik. The object-oriented database system manifesto. SIGMOND Conference, May 1990.

31 Tomado de: M.P Atkinson, F. Bancilhon, D.J. Dewitt, K. R. Dittrich, D. Maier, and S. B. Zdonik. The object-oriented database system manifesto. SIGMOND Conference, May 1990.

código, uno preparado, para instancias transitorias y otro para instancias persistentes.

- Evitar la traducción explícita entre el estado de los objetos y su representación en base de datos, que redundan en mayor facilidad de mantenimiento y menos código también.
- Facilitar la integridad y permitir que actúe el sistema de tipos subyacentes, que automáticamente podría verificar la consistencia y la correspondencia de tipos, entre estados en base de datos y objetos en programa, la integridad no sería responsabilidad del programador.

~~1.8~~ 2.9.4 Correspondencia entre clases y datos

Con formato: Numeración y viñetas

Cuando se trabaja con sistemas gestores de datos, como gestores de datos relacionales, ficheros, gestores de datos documentales XML, etc., cuyo modelo de datos no tiene una equivalencia directa con el modelo de objetos del lenguaje de programación usado, hay una falta de correspondencia entre la base de datos y lenguaje de programación, es necesario establecer un modelo de correspondencia, que defina como una clase se convierte en datos sobre el modelo ofrecido por el sistema que albergará el estado de los objetos. A esta equivalencia entre la clase y los datos, se denomina aquí correspondencia clase – datos (object mapping). El caso particular de la correspondencia entre clases y tablas en un RDBMS, es la correspondencia objeto – registros. El proceso de mapear puede implicar cambios en ambos lados de la correspondencia, en el modelo de clases creado o modificado para asumir ciertos modelos de datos, y al contrario, el modelo de datos puede ser

diseñado o cambiado, para permitir una correspondencia más eficiente y eficaz con ciertos modelos de clases.

2.9.5 Requisitos para una capa de persistencia

Especificaremos las funcionalidades de una capa de persistencia con transparencia de datos, capacidad para guardar y recuperar concurrentemente desde diferentes sistemas gestores de datos, que sea ortogonal en la mayor medida posible, robusta, consistente, eficaz, fácil de usar y extensible desde el punto de vista de reconocidos autores como S.W. Ambler, M. Atkinson, K. Dittrich, G. Booch, J. Rumbaugh, Ivar Jacobson, cuyas referencias aparecen más adelante.

2.9.6 Requisitos estructurales

Algunas de las cualidades estructurales que debe satisfacer un buen servicio de persistencia son:

Fiabilidad. Una capa de persistencia debe ser fiable, funcionar correctamente evitando resultados no esperados o imprevisibles. La robustez se consigue gracias a un diseño que contemple el tratamiento y reacción ante situaciones anormales.

Simplicidad y sencilles. Un requisito primordial para logar una buena arquitectura es mantener la simplicidad y sencillez de la solución, evitar los artificios innecesarios

Modularidad. La solución a un problema complejo pasa por dividir el problema en partes pequeñas que contribuyen a la solución del problema. Una

buena modularidad facilita la consecución de tres propiedades: Ocultación de la información, facilidad de mantenimiento y el desarrollo en paralelo de partes.

La arquitectura desde un punto de vista general debería:

- Estructurar las responsabilidades en capas de servicios.
- Diferenciar claramente la capa de objetos de la de gestión de datos.

La separación en una capa de objetos y otra para administrar el almacenamiento, contribuye a la separación de los conceptos de orientación a objetos, de aquellos relativos a la programación de base de datos, sistemas de archivos o servicios de empresa. Entre ambas capas, se sitúan uno o más niveles de servicios que cubren parte de la funcionalidad requerida y la falta de correspondencia, entre los mundos de los mecanismos de persistencia y el mundo de los objetos. El coste de separar en capas en servicio de persistencia debe ser recuperado por un aumento en la mantenibilidad y en la facilidad de realizar ajustes en el rendimiento[32].

Encapsulamiento. Scout W. Ambler propone que idealmente solo se deberían mandar los mensajes de guardar, eliminar y recupera al objeto en cuestión ocultándose todo detalle sobre el mecanismo concreto de persistencia.

Soportar diferentes sistemas de gestión de datos. Es requisito esencial soportar, las aplicaciones de gestión con diferentes necesidades de

32 Tomado de: Scott W. Ambler desing of Robust Persistente Layer.

prestaciones y rendimiento, cubiertas por sistemas basados en archivos y en gestores relaciones[33].

Soportar diferentes arquitecturas. Las arquitecturas mayormente difundidas se clasifican de manera general en arquitecturas monolíticas, cliente / servidor en dos o más capas y aplicaciones distribuidas. Arquitecturas diferentes, con requerimientos de persistencia distintas, que implican que es necesario soportar diferentes arquitecturas. Debido a esto, el diseñador de servicios de persistencia debe seguir ciertas especificaciones que garanticen la integración con diferentes arquitecturas aunque implique un mayor coste de desarrollo [34.]

Extensibilidad. Al igual que deberíamos ser capaces de añadir nuevas clases a las aplicaciones, deberíamos ser capaces de sustituir los mecanismos de persistencia empleados. También tendría que ser posible incorporar nuevos mecanismos. De hecho con una modularidad adecuada debería ser posible extender el servicio de persistencia con nuevos mecanismos. La extensibilidad es una cualidad deseable de un buen sistema de persistencia.

Facilidad de uso. Una capa de persistencia debe presentar sus servicios de forma clara, concisa, oportuna y manejable. Ser fácil de usar contribuye a conseguir la economía de los desarrollos, mayor productividad y menor coste en horas / hombre. Utilizando características como, la flexibilidad y expresividad en la definición de las operaciones, la asignación de valores por

33 Tomado de: www.ambysoft.com/persistenceLayer.html
34 Tomado de: www.ambysoft.com/persistenceLayer.html

defecto, etc., son elementos que ayudan a conseguir una capa más fácil de usar. La facilidad de uso debería repercutir en:

- Un menor esfuerzo de diseño.
- Menor coste y esfuerzo de producción del código.
- Mejor verificabilidad
- Mejor mantenibilidad
- Facilidad de instalación y configuración

La facilidad de uso puede llegar a ser un factor crítico determinante en el éxito de un buen servicio de persistencia.

Escalabilidad y rendimientos. Se espera que una capa de persistencia se adapte a las fluctuaciones en el volumen de datos y peticiones ofreciendo el servicio más eficaz en cada caso. Esto es, una capa de persistencia debería ser escalable. Para mejorar el rendimiento, el acceso a los objetos debe ser optimizado y también el acceso a los mecanismos de persistencia, para lo cual, se debe maximizar el rendimiento de los mecanismos y minimizar el numero de peticiones a los mismos[35].

2.9.7 Requisitos funcionales

Realizar operaciones CRUD sobre objetos. Es esencial que un sistema de persistencia facilite las operaciones básicas de creación, lectura, actualización y borrado de los estados de los objetos persistentes. Es preciso

35 Tomado de: Arthur M. Keller y otros, architecting Object Applications for High Performance with Relational Databases, 1995 www.hep.net/chep95/html/papers/p59/p59.ps

facilitar la integración de claves primarias e identidad de los objetos persistentes.

Correspondencia clases – datos. Una capa de persistencia debería proporcionar los medios para expresar y establecer la correspondencia entre el modelo de clases y los esquemas de los servicios de datos. Una capa de persistencia con transparencia de datos debería producir la conversión automática entre los tipos o clases y sus correspondientes tipos, que definen su estado persistente en los depósitos de datos, la conversión debería ser efectuada sin necesidad de añadir código adicional sin la intervención del programador. La información sobre el mapeo debe ser accesible en tiempo de desarrollo de la aplicación para asegurar la integridad de los datos con los que se está trabajando.

Ortogonalidad. Una capa de persistencia debe ser lo suficientemente ortogonal como para conseguir:

- No necesitar modificar el código fuente.
- Independencia de uso.
- Transparencia de datos.
- Utilizar claves primarias para identificación y localización de objetos.

Concurrencia. La información debe ser compartida, esta es una obligación de todo servidor de base de datos que también debe reflejarse en los servicios de persistencia que usan los servicios de datos[36].

36 Tomado de: EDGAR F. CODD, The Relational Model for Database Management.: Version 2. Addison Wesley Pulishing Co. 1990.

Un objeto persistente tiene dos aspectos: uno en ejecución y otro su estado almacenado. Como consecuencia de esta dualidad, la gestión del acceso concurrente a objetos debe cubrir ambos aspectos, el objeto como entidad en ejecución y su estado almacenado en los servicios de datos. De otro modo tendríamos situaciones inconsistentes. .Net aporta un mecanismo especializado para la sincronización con la utilización del objeto dataset que tiene la capacidad de “recordar” el estado anterior de los objetos, el mismo que es verificado el momento que se accede a la base de datos para evitar errores de concurrencia.

Una capa de persistencia de objetos debe resolver el acceso concurrente a los objetos persistentes teniendo en cuenta también el acceso concurrente a su estado almacenado de forma consistente.

Cada proceso puede acceder al sistema gestor de datos, deposito de los estados de los objetos, para recuperar un mismo estado. Es necesario emplear alguna estrategia de bloqueo que garantice la consistencia. Los sistemas gestores de datos proporcionan la capacidad de compartir y concurrir a los estados guardados. Las aplicaciones utilizan las interfaces de programación de los servicios de datos para la concurrencia, basadas en las estrategias habituales de control de acceso pesimista y optimista, en el primero los datos son bloqueados para su modificación hasta finalizar el proceso que solicita la modificación, y el control optimista, no se produce el bloqueo ante nuevas modificaciones, pero puede ser analizada la consistencia por la presencia de marcas de tiempo que permiten decidir sobre el efecto de las modificaciones.

Una capa de persistencia debe aprovechar las estrategias de control de concurrencia ofrecidas por los gestores de datos con lo que interactúa y en ausencia de estas, implantar el bloqueo pesimista y opcionalmente, el bloqueo optimista.

Transacciones. El soporte de transacciones es un requisito esencial. Al menos deben ser soportadas las funcionalidades de consolidar y cancelar las modificaciones. Habrá sistemas que soporten o necesiten de anidamiento de transacciones, transacciones de larga duración, bloqueo automático o explicito, etc. Igualmente su implantación dependerá del soporte dado por el sistema gestor de datos final.

Integración de diferentes tipos de mecanismos de persistencia. Es un requisito que el soporte de los mecanismos de persistencia sea para los principales gestores de datos relacionales del mercado. También se deben soportar otros mecanismos de persistencia bajo una misma interfaz de programación. Una interfaz común permitiría unificar bajo un mismo protocolo el acceso a servicio de datos diferentes. Esto significa mayor productividad, facilidad de integración e interoperatividad.

Conexiones múltiples. Es esencial poder establecer múltiples conexiones simultáneas a cada uno de los mecanismos de persistencia soportados por el servicio de persistencia.

Integridad. Es necesario asegurar la integridad de los objetos, sus datos, sus referencias a otros objetos, y la integridad de las transacciones. Compartir y transferir datos implica la responsabilidad de asegurar la exactitud de los datos manejados, proteger datos contra modificaciones y el acceso no correctos[37]. Esta responsabilidad es preservar la integridad de los datos, que es un requisito previo para lograr la transparencia de datos.

Con formato: Numeración y viñetas

37 Tomado de: Bertrand Meyer, Object Oriented Sofware Construction " Editon, Prentice Hall, 1997

~~1~~ CAPÍTULO III

ESPECIFICACIÓN DE REQUERIMIENTOS MOTOR DE PERSISTENCIA

~~2.1 -~~ 3.1 Introducción

Con formato: Numeración y viñetas

~~2.1.1~~3.1.1 Propósito

Con formato: Numeración y viñetas

El propósito de este documento es proporcionar una lista completa y no ambigua de los requerimientos que debe cubrir el software a desarrollar. Proporcionar un soporte legal y un medio de comunicación con los usuarios del sistema, los ingenieros que dirijan este proyecto y las desarrolladoras, en el cual se reflejen los acuerdos del alcance de la aplicación práctica de la Tesis Motor de Persistencia para .Net.

~~2.1.2~~3.1.2 Ámbito del sistema

Con formato: Numeración y viñetas

Objetivo General

- Realizar el análisis, diseño, e implementación de un motor de persistencia para .Net, para simplificar las operaciones de mantenimiento (inserción, actualización, eliminación), de consulta y conexión con el gestor de datos.

Objetivos específicos

- Transparentar la inserción, actualización y eliminación de datos en el gestor de datos.

- Aplicar patrones de diseño para el desarrollo de un motor de persistencia.

~~2.1.3~~3.1.3 Alcance

Con formato: Numeración y viñetas

El motor de persistencia se codificará utilizando como plataforma de desarrollo .Net y como lenguaje C#.

Implementará interfaces para trabajar con:

- Relational Database Management Systems (RDBMS).
- Archivos planos.

Con el motor de persistencia interactúan dos tipos de usuarios, a continuación se define cada tipo:

Cualquier aplicación desarrollada utilizando .Net utiliza la funcionalidad del motor de persistencia para transparentar la comunicación entre la lógica de la aplicación y el gestor de datos.

Administrador de bases de datos (DBA) utiliza el motor de persistencia para generar entidades del negocio.

~~2.1.4~~3.1.4 Definiciones, acrónimos y abreviaturas

Con formato: Numeración y viñetas

Definiciones

Mecanismos de persistencia: Es la capacidad de un sistema de poder almacenar el estado de un elemento (objeto) de modo que sea posible recuperarlo posteriormente, desde, ó a través de múltiples destinos como gestores de datos relacionales, archivos planos, archivos XML, etc.

Acrónimos

IEEE: The Institute of Electrical and Electronics Engineers, Inc.

ERS: Especificación de Requerimientos de Software.

CORBA: Protocolo de comunicaciones utilizado en Internet.

RDBMS: Relational Database Management Systems.

XML: Extensible Markup Language.

DBA: Database Administrador.

Abreviaturas

SP: Store Procedure.

~~2.1.5~~3.1.5 Referencias

Con formato: Numeración y viñetas

Tabla 3.1: Tabla de referencias.

Ord.	Título	Número	Fecha
1	Especificación de requerimientos de software según el estándar de IEEE 830. Apuntes de la UDIS. FI. UPM	IEEE Std.830-1998	01/10/2001
2	IEEE Glosario de términos de ingeniería de software.	IEEE Std 610.12-90	1990

~~2.1.5~~3.1.6 Visión general del documento

Con formato: Numeración y viñetas

En la sección 3.2 de esta especificación se describen los factores que afectan al Motor de Persistencia para .Net y a sus requisitos. Pone al sistema en perspectiva con otros subsistemas, proporciona un sumario de las funciones

que ejecutará el software, suposiciones y dependencias que afectarán al desarrollo del motor de persistencia.

La sección 3.3 de este documento contiene en sí la ERS. Es decir, contiene los detalles que el desarrollador de software necesita saber para crear el diseño del sistema, haciendo una lista de los requisitos específicos individuales.

Cada requisito funcional se detalla en la sección 3.3.1 presentando la siguiente información:

- Introducción.
- Entrada.
- Salida.

El resto de apartados de la sección 3.3.2 listan los requisitos de interfaces externas, las limitaciones de diseño y los requisitos de mantenibilidad y seguridad.

~~2.2 -~~ 3.2 Descripción general

Con formato: Numeración y viñetas

~~2.2.1~~3.2.1 Perspectiva del producto

El motor de persistencia para .Net, nace con la necesidad de dotar a los desarrolladores de software, una forma rápida, segura y completa de comenzar sus proyectos, proporcionando gran escalabilidad y facilidad de control de los objetos del aplicativo como de los objetos de persistencia.

Mediante el motor de persistencia se podrá realizar cambios a los objetos fácilmente si se ha cambiado el medio de persistencia ya sean bases de datos, o archivos planos.

Este sistema actuará de forma dependiente con las demás capas de un sistema de software.

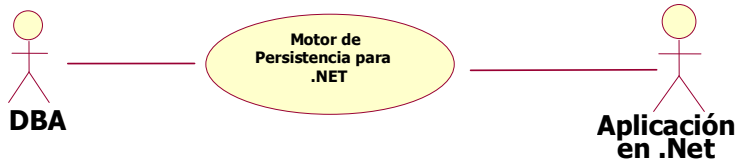


Figura 3.1~~3.1~~: Diagrama de contexto del motor de persistencia para .Net.

~~2.2.2~~3.2.2 Características de los usuarios

Con formato: Numeración y viñetas

El sistema a desarrollarse está orientado a usuarios expertos, con experiencia en desarrollo de sistemas, manejo de gestores de datos y técnicas de desarrollo e implantación de sistemas.

Con el motor de persistencia interactúan dos tipos de usuarios:

- El administrador de gestores de datos (DBA) podrá ejecutar todas las opciones del sistema y deberá conocer los siguientes temas:
  - Ambiente Windows.
  - Manejo de bases de datos.
  - Administración de sistemas.
  - Ambiente Web.

- El usuario programador utilizará la funcionalidad del motor de persistencia para trabajar con los objetos generados directamente de la base de datos.

~~2.2.3~~3.2.3 Restricciones

Con formato: Numeración y viñetas

- El software solo se puede utilizar con aplicaciones que corran sobre el sistema operativo Windows 2000 o versiones superiores
- Se requiere de licencia para desarrollo de Visual Studio .NET.
- Los usuarios del sistema deberán obligatoriamente tener conocimientos sobre análisis y diseño de sistemas.

~~2.2.4~~3.2.4 Suposiciones y dependencias

Con formato: Numeración y viñetas

- Que los usuarios tengan conocimientos de:
  - Desarrollo de software en .Net.
  - Administración de bases de datos.
  - Manejo de persistencia de objetos.
- Que el usuario de sistema tiene licencias de desarrollo y utilización del software.

~~2.3 -~~ 3.3 Requisitos específicos

Con formato: Numeración y viñetas

~~2.3.1~~3.3.1 Requisitos funcionales

Crear proveedor mecanismo de persistencia

Descripción

Instancia un objeto proveedor de datos para tener un puente entre la aplicación y un origen de datos. Recibe información sobre el assembly o ensamblado (dll) que se necesitan para crear un objeto del proveedor que se va a utilizar para realizar las operaciones de grabar, eliminar y modificar.

Entradas

- Nombre del assembly o ensamblado que contiene la clase de un proveedor de datos; ejemplo “System.Data”.
- Nombre de la clase que representa una conexión abierta a un origen de datos; ejemplo “SqlConnection”.
- Nombre de la clase que representa un conjunto de comandos de datos y una conexión de base de datos que se utilizan para rellenar un DataSet y actualizar el origen de datos; ejemplo “SqlDataAdapter”.
- Nombre de la clase que genera automáticamente, en una sola tabla, los comandos para sincronizar los cambios realizados en un DataSet con el proveedor de datos asociado; ejemplo “SqlCommandBuilder”.
- Nombre del proveedor de datos; ejemplo “System.Data.SqlClient”.

Salidas

- Devuelve una excepción cuando se detecta cualquier error en el origen de datos o en el cliente.

Obtener manejador del mecanismo de persistencia

Descripción

Obtiene un objeto proveedor que conoce cual es el mecanismo de persistencia al que va a acceder.

Entradas

- Nombre del proveedor de datos válido.

Salidas

- Devuelve un objeto proveedor.

Validar proveedor del mecanismo de persistencia

Descripción

Verifica que el proveedor de datos enviado sea válido para poder crearlo, si no es válido se envía un mensaje indicando este error.

Entradas

- Nombre del proveedor.

Salidas

- Devuelve una excepción cuando se detecta cualquier error en el origen de datos o en el cliente.

Obtener fuente de datos configuradas

Descripción

Obtiene una colección de las fuentes de datos que están configuradas en el archivo de configuración de la aplicación.

Entradas

- Nombre de la sección del archivo de configuración.

Salidas

- Devuelve una colección de fuentes de datos configuradas.

Validar fuente datos

Descripción

Verifica que la fuente de datos enviada sea válida para poder crearla, si no es válida se envía un mensaje indicando este error.

Entradas

- Nombre de la fuente de datos.

Salidas

- Devuelve una excepción cuando se detecta cualquier error en el origen de datos o en el cliente.

Obtener proveedores configurados

Descripción

Obtiene una colección de los proveedores que están configuradas en el archivo de configuración de la aplicación.

Entradas

- Nombre de la sección del archivo de configuración.

Salidas

- Devuelve una colección de proveedores configurados.

Crear conexión fuente datos

Descripción

Crea una instancia para conectarse a un origen de datos. Recibe información del proveedor de datos y la cadena de conexión del objeto, devuelve una instancia de la interfaz IDbConnection.

Entradas

- Instancia del objeto proveedor.
- Cadena de conexión.

Salidas

- Devuelve una excepción cuando se detecta cualquier error en el origen de datos o en el cliente.
- Instancia de la interfaz IDbConnection, que representa una conexión abierta al proveedor de datos enviado.

Realizar operaciones CRUD

Descripción

Realiza operaciones sobre los datos de una fuente de datos. Recibe información del tipo de proceso que va a ejecutar como: grabar, actualizar, eliminar, ejecutar procedimientos almacenados, argumentos de los procesos almacenados si los necesita, indicación si esta acción debe ejecutarse dentro de una transacción y devuelve el número de registros que fueron modificados con la acción que se ejecuto.

Entradas

- Puede recibir una Instancia de un objeto Connection o una cadena de conexión para crear la instancia del objeto Connection.
- Tipo de comando, como procedimiento almacenado o comando en texto que debe ejecutarse.

- Cuando el tipo de comando es procedimiento almacenado el nombre, caso contrario ningún valor.
- Cuando el tipo de comando es procedimiento almacenado los argumentos que necesita para ejecutarse, caso contrario ningún valor.
- Texto con el comando que debe ejecutarse como grabar o actualizar.
- Cuando necesita que las acciones se ejecuten dentro de una transacción se envía una instancia de la transacción actual, caso contrario ningún valor.

Salidas

- Devuelve una excepción cuando se detecta cualquier error en el origen de datos o en el cliente.
- Retorna un valor entero indicando el número de registros que fueron afectados con la acción ejecutada.

Listar

Descripción

Obtiene un conjunto de datos. Recibe información de la conexión de datos, el tipo de comando que se va a ejecutar, la consulta que se debe ejecutar, los parámetros necesarios para ejecutar la consulta y retorna el conjunto de datos seleccionados.

Entradas

- Puede recibir una instancia de un objeto Connection o una cadena de conexión para crear la instancia del objeto Connection.
- Tipo de comando, como procedimiento almacenado o comando en texto que debe ejecutarse.

- Cuando el tipo de comando es procedimiento almacenado el nombre, caso contrario ningún valor.
- Cuando la consulta o procedimiento almacenado necesita argumentos se envían, caso contrario ningún valor.
- Texto con la consulta que debe ejecutarse.
- Cuando necesita que las acciones se ejecuten dentro de una transacción se envía una instancia de la transacción actual, caso contrario ningún valor.

Salidas

- Devuelve una excepción cuando se detecta cualquier error en el origen de datos o en el cliente.
- Conjunto de datos seleccionados.

Obtener valor único

Descripción

Obtiene un valor único. Recibe información de la conexión de datos, el tipo de comando que se va a ejecutar, la consulta que se debe ejecutar o el procedimiento almacenado, los argumentos necesarios para ejecutar la consulta o el procedimiento almacenado y retorna un solo valor con el resultado de la consulta.

Entradas

- Puede recibir una Instancia de un objeto Connection o una cadena de conexión para crear la instancia del objeto Connection.
- Tipo de comando, como procedimiento almacenado o comando en texto que debe ejecutarse.

- Cuando el tipo de comando es procedimiento almacenado el nombre, caso contrario ningún valor.
- Cuando la consulta o procedimiento almacenado necesita argumentos se envían, caso contrario ningún valor.
- Texto con la consulta que debe ejecutarse.
- Cuando necesita que las acciones se ejecuten dentro de una transacción se envía una instancia de la transacción actual, caso contrario ningún valor.

Salidas

- Un objeto con un solo valor generado en el proceso.

Ejecutar mandato SQL

Descripción

Realiza operaciones sobre los datos de una fuente de datos. Recibe información del tipo de proceso que va a ejecutar como: grabar, eliminar, actualizar, ejecutar procedimientoa almacenados, argumentos de los procesos almacenados si los necesita, indicación si esta acción debe ejecutarse dentro de una transacción y devuelve el número de registros que fueron modificados con la acción que se ejecuto.

Entradas

- Puede recibir una Instancia de un objeto Connection o una cadena de conexión para crear la instancia del objeto Connection.
- Tipo de comando, como procedimiento almacenado o comando en texto que debe ejecutarse.

- Cuando el tipo de comando es procedimiento almacenado el nombre, caso contrario ningún valor.
- Cuando el tipo de comando es procedimiento almacenado los argumentos que necesita para ejecutarse, caso contrario ningún valor.
- Texto con el comando que debe ejecutarse como guardar o eliminar.
- Cuando necesita que las acciones se ejecuten dentro de una transacción se envía una instancia de la transacción actual, caso contrario ningún valor.

Salidas

- Devuelve una excepción cuando se detecta cualquier error en el origen de datos o en el cliente.
- Retorna un valor entero indicando el número de registros que fueron afectados con la acción ejecutada.

Preparar comando

Descripción

Prepara un objeto comando para ejecutar las acciones sobre un mecanismo de persistencia. Se asigna si es necesaria una conexión, transacción, tipo de comando que puede ser procedimiento almacenado, texto, etc. y parámetros para ejecutar los procedimientos almacenados si son necesarios.

Entradas

- Objeto conexión.
- Objeto transacción.
- Tipo de comando.

- El nombre del procedimiento almacenado o la sentencia SQL.
- Arreglo de parámetros para asociarlos con el comando.

Salidas

- Devuelve un comando listo para ejecutar las acciones sobre los datos.

~~2.3.2~~3.3.2 Requisitos de interfaz externa

Con formato: Numeración y viñetas

El usuario interactuará con el producto en un ambiente visual mediante una interfaz tipo escritorio.

Interfases de hardware

Los requerimientos básicos del hardware para un correcto desempeño del producto son:

Características básicas de los servidores

Procesador:	Min PENTIUM III
Memoria:	Min 64 Mb
Disco:	Min 10 Gb
Monitor:	14” .28 SVGA Color
Bus de datos:	100Mbps

Características básicas de las estaciones de trabajo

Procesador:	Min PENTIUM II
Memoria:	Min 64 Mb
Disco:	Min 6 Gb
Monitor:	14”
Bus de datos:	100Mbps

Interfases de software

Para que el producto se pueda ejecutar correctamente se necesitará del siguiente software:

En el servidor transaccional

En este servidor se ejecutarán los servidores para que puedan correr las aplicaciones:

- Servicios de componentes (COM+).
- Internet Information Server.

En el servidor de base de datos:

- SQLServer v 7.0 o superior.
- Oracle
- DB2

En las estaciones de trabajo

Para que los usuarios puedan interactuar con la aplicación las estaciones deberán tener el siguiente software:

Sistemas operativos

- Windows 2000 / 2003 Server para los servidores.
- Windows 2000 Professional / Win98 / XP para las estaciones de trabajo.

Herramientas CASE

- Rational Rose para el graficar los diagramas correspondientes al modelo estático, modelo funcional y modelo dinámico.

Lenguajes de programación

- Lenguaje C#.

Interfaces de comunicación

La comunicación se la realizará a través XML, del protocolo, SOAP para la comunicación a los Web Services. Para la comunicación del servidor de aplicaciones con la BDD se utiliza TCP/IP.

~~2.3.3~~3.3.3 Limitaciones de diseño

Con formato: Numeración y viñetas

Estándares

- El nombramiento de los datos, y todos los procesos referentes a la programación del motor de persistencia para .Net deben regirse a las normas documentadas en la sección 2.11
- El modelamiento del motor de persistencia para .Net está realizado de acuerdo a la metodología OMT y graficado con los diagramas de UML.
- El diseño de la interfaz gráfica de usuario para el prototipo, se lo realizará según el estándar que se encuentra en la sección 4.5.1

~~2.3.4~~3.3.4 Atributos del sistema

Con formato: Numeración y viñetas

Seguridad

- El motor de persistencia para .Net, no implementa mecanismos de seguridad. Durante la implementación de aplicaciones se debe considerar establecer restricciones de acceso al servidor de componentes.

Mantenibilidad

- El sistema es susceptible de ser ampliado. Por tanto deberá diseñarse fácilmente mantenible, aplicando para su desarrollo las metodologías que para ello sean precisas.

Con formato: Numeración y viñetas

~~1~~ CAPÍTULO IV

ESPECIFICACIÓN DE REQUERIMIENTOS DE LA APLICACIÓN FACTURACIÓN

~~2.5 .~~ 4.1 Introducción

Con formato: Numeración y viñetas

~~2.5.1~~4.1.1 Ámbito del sistema

Con formato: Numeración y viñetas

Objetivo General

- Realizar el prototipo de una aplicación que permita la facturación de productos en una empresa para demostrar la funcionalidad del motor de persistencia para .Net, desarrollado en esta tesis.

Objetivos específicos

- Realizar una aplicación que permita la facturación de productos a clientes previamente registrados.
- Construir una aplicación en n capas que utilice la funcionalidad del motor de persistencia.
- Acceder a la plataforma de datos a través del motor de persistencia para realizar las operaciones CRUD del prototipo planteado.

~~2.5.2~~4.1.2 Alcance

Con formato: Numeración y viñetas

El proyecto de tesis pretende realizar el desarrollo de una aplicación prototipo de facturación de productos que funcione bajo la arquitectura de n capas, utilizando el motor de persistencia para el acceso a datos.

La aplicación prototipo de facturación de productos debe utilizar al motor de persistencia para realizar la administración de entidades del negocio, y para realizar la emisión de un reporte periódico de ventas por producto.

La aplicación prototipo de facturación de productos se codificará utilizando como plataforma de desarrollo .Net, como lenguaje C#.

~~2.5.3~~4.1.3 Definiciones, acrónimos y abreviaturas

Con formato: Numeración y viñetas

Prototipo: Es una demostración consistente de un sistema.

Articulo: Cualquier producto que la empresa decida comercializar.

Unidad de Medida: Tamaños y tipos de presentación del producto.

Ubicación: La ubicación es el barrio donde reside o donde labora el cliente. El barrio es el nivel más bajo de una estructura recursiva que permite almacenar barrio, cantón, parroquia, provincia de una ubicación.

~~2.5.4~~4.1.4 Visión general del documento

Con formato: Numeración y viñetas

Cada requisito funcional se detalla en la sección 4.3.1 de este capítulo presentando la siguiente información:

- Introducción.
- Eventos principales.
- Eventos alternos.
- Eventos excepción.

- Pre – condiciones.
- Post – condiciones.

El resto de apartados de la sección 4.3.2 listan los requisitos de interfaz externas, y las limitaciones de diseño.

~~2.6 -~~ 4.2 Descripción general

Con formato: Numeración y viñetas

~~2.6.1~~4.2.1 Perspectiva del producto

La aplicación prototipo de facturación de productos se hace necesaria para demostrar la funcionalidad de motor de persistencia desarrollado para esta tesis, por lo que debe interactuar con la misma para implementar el acceso y recuperación de datos.

Deberá funcionar en un sistema de red LAN o en una red WAN. El sistema es totalmente distribuido para lo cual usa una arquitectura n capas.

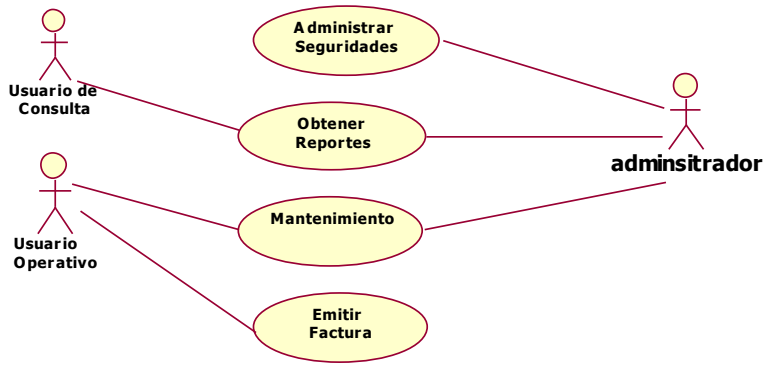


Figura 4.1~~3.2~~: Diagrama de contexto prototipo.

~~2.6.2~~4.2.2 Características de los usuarios

Con formato: Numeración y viñetas

Con el prototipo interactúan tres tipos de usuarios:

- El administrador de la aplicación que podrá ejecutar todas las opciones del sistema y deberá conocer los siguientes temas:
  - Ambiente Windows.
  - Manejo de bases de datos.
  - Administración de sistemas.
- Los usuarios operadores del sistema que pondrán acceder a las opciones de la aplicación de acuerdo a su perfil, los conocimientos del operador son los básicos del uso de computadoras.
- Los usuarios de consulta del sistema que pondrán acceder a las consultas de la aplicación de acuerdo a su perfil, los conocimientos del consultor son los básicos del uso de computadoras.

~~2.6.3~~4.2.3 Restricciones

Con formato: Numeración y viñetas

- El software solo se puede utilizar con aplicaciones que corran sobre el sistema operativo Windows 2000 o versiones superiores
- Se requiere que este instalado el Framework de Visual Studio.NET.
- El prototipo debe utiliza el motor de persistencia para el acceso a datos.

~~2.6.4~~4.2.4 Suposiciones y dependencias

Con formato: Numeración y viñetas

- Que el usuario administrador tiene conocimientos de:
  - Administración de bases de datos.
  - Manejo de persistencia de objetos.
- Que el usuario de sistema tiene licencias de desarrollo y utilización del software.

~~2.7 -~~ 4.3 Requisitos específicos

Con formato: Numeración y viñetas

~~2.7.1~~4.3.1 Requisitos funcionales

Los procesos que se encargará de realizar la aplicación prototipo de facturación de productos son especificados mediante las salidas requeridas por los usuarios y las entradas necesarias para que se realice dicho proceso.

Salidas requeridas por el usuario

a. Módulo de administración y seguridades

- Generación automática de códigos para el registro de usuarios, perfiles, opciones y acciones.
- Lista en pantalla de usuarios, perfiles, acciones, opciones, opciones por perfil, acciones por opción y perfil opción acciones registrados en el sistema.
- Datos específicos de usuarios, perfiles, acciones, opciones, opciones por perfil, acciones por opción y perfil opción acciones

b. Módulo de mantenimiento

- Generación automática de códigos para el registro de ubicaciones, artículos, unidades de medida.
- Lista en pantalla de ubicaciones, artículos, unidades de medida, formas de pago y clientes.
- Datos específicos de determinadas ubicaciones, artículos, unidades de medida, formas de pago y clientes.

c. Módulo facturación y reportes

- Generación automática de códigos para registrar facturas.
- Lista en pantalla de facturas y sus detalles.
- Datos específicos de facturas y sus detalles.
- Reporte periódico de ventas por producto.

Entradas requeridas por el usuario

Tabla 4.1: Tabla de entradas requeridas por el usuario.

A. Módulo de administración y seguridades	
Objeto	**Datos a ingresar**
Definición de Usuarios	Código de usuario.
	Descripción de usuario.
	Clave del usuario.
Definición de Perfil	Código de perfil.
	Descripción de perfil.
Definición de Opciones	Código de opción.
	Descripción de opción.
	Código padre opción.
	Descripción de objeto que los contiene.
Definición de Opciones por Perfil	Código de perfil.
	Código de opción.
Definición de Acciones por Opción	Código de opción.
	Código de acción.
Definición de Perfil Opción Acción	Código de perfil.
	Código de opción.
	Código de acciones.
B. Módulo de mantenimientos	
Mantenimiento de Ubicaciones	Código de ubicación.
	Descripción de ubicación.
	Código padre.

Mantenimiento de Artículos	Código de artículo.
	Descripción de artículo.
	Código de unidad de medida.
	Unidades disponibles del artículo.
	Valor de venta del artículo.
Mantenimiento de Clientes	Cédula del cliente.
	Nombre del cliente.
	Teléfono del cliente.
	Ubicación del domicilio.
	Ubicación del domicilio laboral.
Mantenimiento de Unidades de Medida	Código de unidad de medida.
	Descripción de unidad de medida.
Mantenimiento de Formas de Pago	Código de forma de pago.
	Descripción de forma de pago.
C. Módulo facturación y reportes	
Facturación de Artículos	Cédula del cliente.
	Fecha de emisión de la factura.
	Código de forma de pago.
	Valor total de la factura.
	Detalle de factura
	Código de factura.
	Código de ítem.
	Código del producto.
	Valor de venta del producto.
Reportes periódicos de ventas por producto.	Código de producto
	Fecha de consulta

~~2.7.2~~4.3.2 Requisitos de interfaz externa

Con formato: Numeración y viñetas

- Interfaces de usuario.
- Interfases de hardware.
- Interfases de software.
- Interfaces de comunicación.

Los requisitos de interfaz externa son los mismos y se detallan en el apartado 3.3.2, de especificación de requerimientos del motor de persistencia.

~~2.7.3~~4.3.3 Limitaciones de diseño

Con formato: Numeración y viñetas

Las limitaciones de diseño son las mismas y detallan en el apartado 3.3.2, de especificación de requerimientos del motor de persistencia.

4.3.4 Atributos del sistema

Seguridad

- Para mantener la seguridad del motor de persistencia para .Net, se han determinado el control de acceso de los usuarios operativos, de consulta y al administrador, de acuerdo a un perfil.

Mantenibilidad~~:~~

Con formato: Numeración y viñetas

- El sistema es susceptible de ser ampliado. Por tanto deberá diseñarse fácilmente mantenible, aplicando para su desarrollo las metodologías que para ello sean precisas.

CAPÍTULO V

~~5~~

~~6CAPITULO~~

~~75~~ ANÁLISIS Y DISEÑO DEL SISTEMA MOTOR DE PERSISTENCIA PARA .NET

Con formato: Numeración y viñetas

~~7.1~~5.1 Propósito

Con formato: Numeración y viñetas

El propósito de este documento es proporcionar una lista completa de los modelos.

- Moldear los requerimientos del negocio, con la tecnología disponible y diseñar los planos a partir de los cuales se construirá el sistema.
- Incluir nuevas clases para facilitar o mejorar la implementación del sistema.

~~7.2~~5.2 Fase de análisis

Con formato: Numeración y viñetas

Esta fase busca obtener la mayor claridad posible con respecto a los requerimientos y necesidades del usuario del sistema a desarrollar. La realización de ésta fase sirve para obtener un documento de requisitos.

~~7.2.1~~5.2.1 Definición del problema

Con formato: Numeración y viñetas

La capa de persistencia es un conjunto de archivos dll y utilitarios que le permiten a una aplicación acceder a información persistente. Encapsula los detalles de la implementación del acceso a datos, proporciona funcionalidad

para recuperar, actualizar y borrar datos en un almacenamiento persistente; de esta forma la capa de negocio no se preocupa de estos detalles.

La capa establece un nivel de abstracción adicional para el manejo de transacciones y conexiones, de manera que la lógica de negocio es independiente del proveedor de datos que finalmente se utilice.

~~7.2.2~~5.2.2 Modelo de objetos

Con formato: Numeración y viñetas

Describe la estructura estática de los objetos del sistema (identidad, relaciones con otros objetos, atributos y operaciones).

a. Diagrama de clases

En el diagrama de clases del motor de persistencia para .Net, se definen las características de cada una de las clases, interfaces y relaciones de dependencia y generalización. Ver Figura 5.1

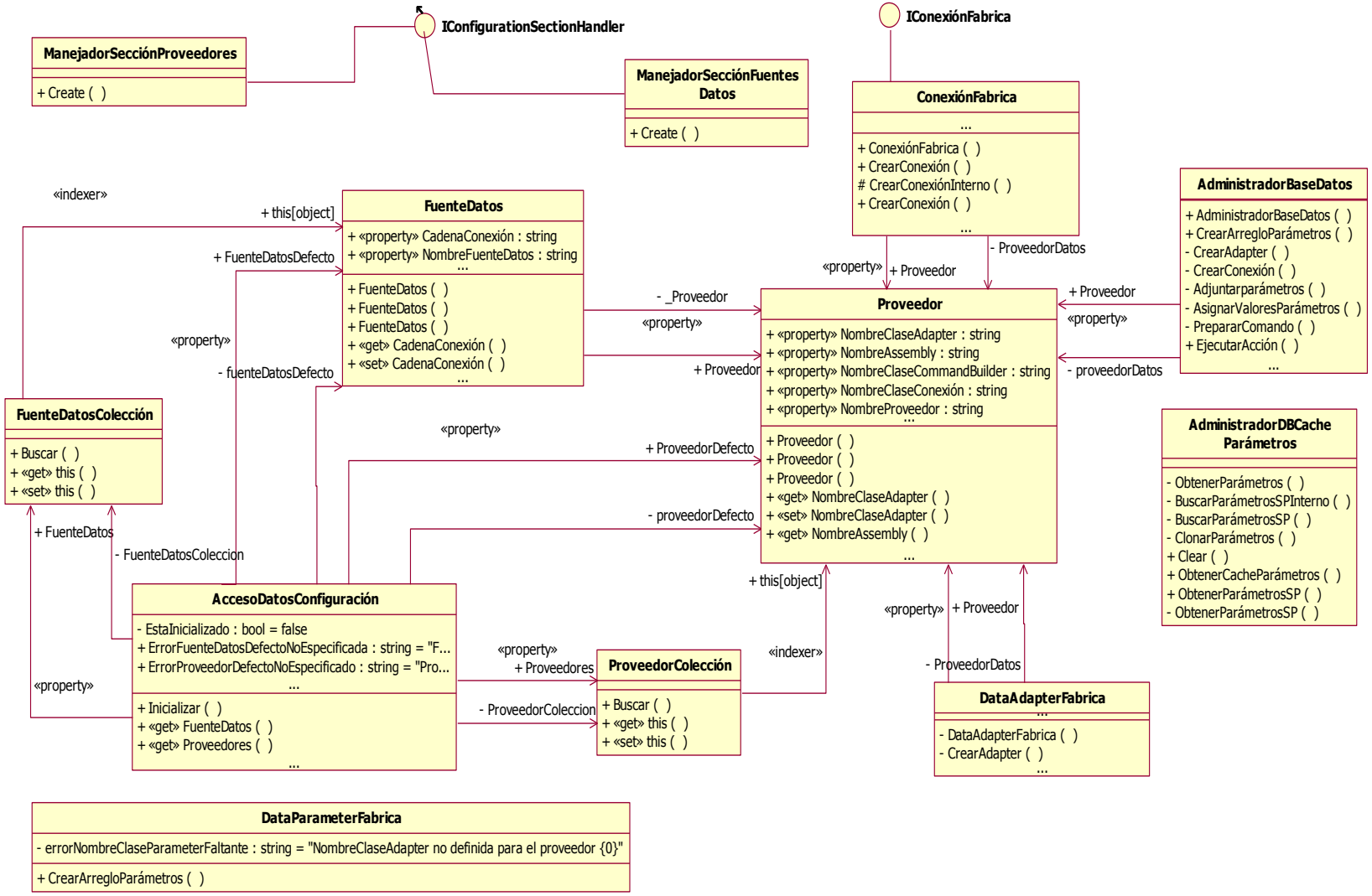


Figura 5.1~~4~~.1: Diagrama de clases del motor de persistencia para .Net.

b. Diagrama de casos de uso

En esta parte del análisis del sistema se describirán en forma general los casos de uso que se presentarán en el sistema, es decir la interacción de los usuarios con sus acciones en cada uno de los procesos que realizan. A continuación se describirán los casos de uso:

1. Manipular capa de persistencia
2. Administrar proveedor mecanismo persistencia
3. Administrar acceso a datos

1. Manipular capa de persistencia

Este diagrama muestra los principales casos de uso que se deben ejecutar para utilizar la capa de persistencia:

- Administrar proveedor mecanismo persistencia
- Administrar acceso a datos

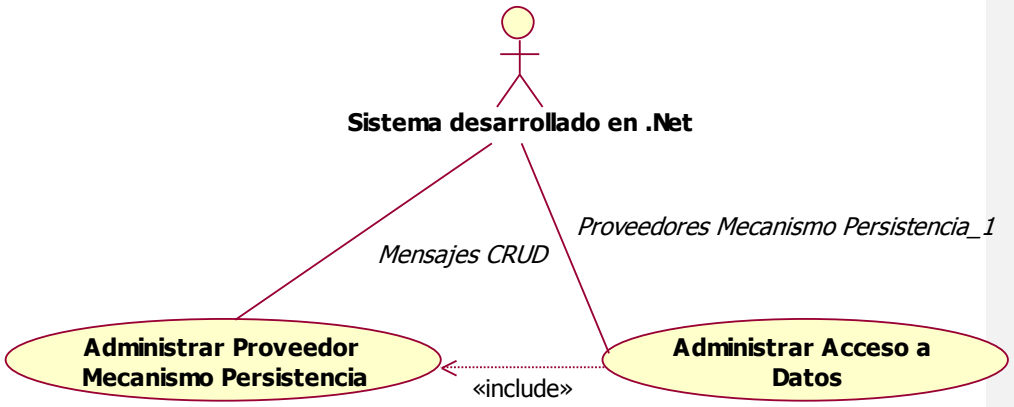


Figura 5.2~~4.2~~: Diagrama manipular capa de persistencia.

2. Administrar proveedor mecanismo persistencia

~~Administrar Proveedor Mecanismo Persistencia~~

Crear y obtener un proveedor de datos definido por el programador.

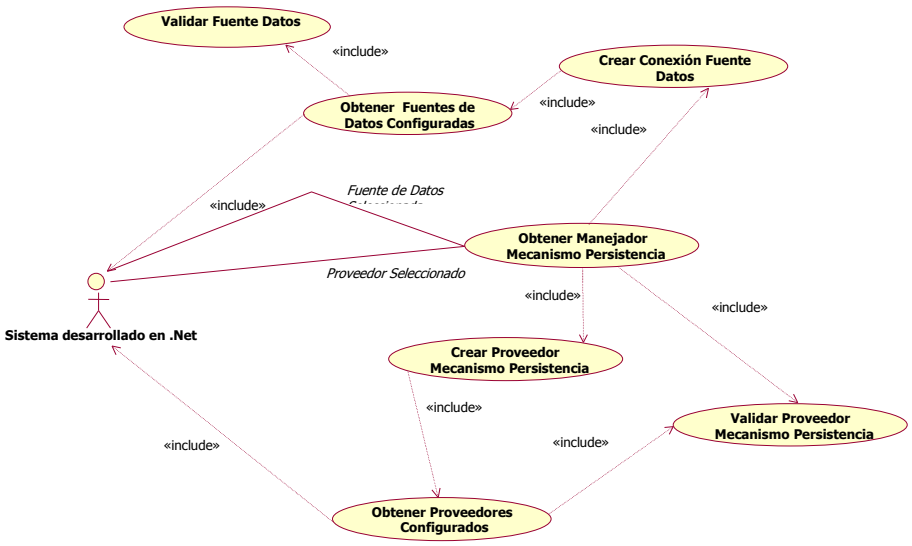


Figura 5.3~~4.3~~: Caso de uso administrar proveedor mecanismo persistencia.

CASO DE USO: Administrar proveedor origen de datos	
Eventos Principales	Para obtener un proveedor de datos se pueden dar los siguientes eventos: 1) Validar proveedor origen de datos. 2) Obtener proveedores configurados. 3) Crear proveedor origen de datos. 4) Obtener manejador origen de datos. 5) Validar fuente datos. 6) Obtener fuentes de datos configuradas. 7) Crear conexión fuente datos. El programador puede obtener un proveedor de datos de dos maneras: Para la primera forma debe seguir los eventos 1, 2, 3, 4. Para la segunda forma debe seguir los eventos 5, 6, 7, 4.
Eventos Alternos	Si no esta configurado el proveedor o la fuente de datos se busca si el proveedor necesitado ya fue creado anteriormente para obtenerlo del cache

	de proveedores.
Eventos de Excepción	Si no esta configurado el proveedor o la fuente de datos y no ha sido creado anteriormente se devuelve una excepción indicando que no está correctamente configurado el pedido para el proveedor.
Pre-Condiciones	El archivo de configuración debe estar previamente configurado y correcto.
Post-Condiciones	Se crea el proveedor de datos solicitado y se almacena en la memoria para obtenerlo cuando se lo solicite nuevamente.

CASO DE USO: Crear proveedor mecanismo de persistencia	
Eventos Principales	1) Inicializar objeto de tipo proveedor. 2) Validar parámetros. 3) Crear objetos del tipo de proveedor inicializado.
Eventos Alternos	4) Si ya esta creado, solo se obtiene el manejador del mecanismo de persistencia.
Eventos de Excepción	Devuelve una excepción cuando se detecta cualquier error en el origen de datos o en el cliente.
Pre-Condiciones	Los parámetros de configuración deben existir y ser válidos. Debe ser posible crear el proveedor con los parámetros enviados.
Post-Condiciones	Se creará una instancia del objeto Proveedor que configura los valores recibidos sobre el proveedor de datos. El proveedor de datos se almacenará en un espacio de memoria del sistema operativo.

CASO DE USO: Obtener manejador mecanismo persistencia	
Eventos Principales	Obtener un objeto de tipo proveedor 1) Inicializar objeto de tipo proveedor. 2) Validar parámetros. 3) Obtener un objeto de proveedor.
Eventos Alternos	4) Si ya esta creado, solo se retorna el manejador del mecanismo de persistencia.
Eventos de Excepción	Devuelve una excepción cuando se detecta cualquier error en el origen de datos o en el cliente.
Pre-Condiciones	Los parámetros de configuración deben existir y ser válidos. Enviar todos los parámetros para crear el proveedor.
Post-Condiciones	Se creará una instancia del objeto Proveedor que configura los valores recibidos sobre el proveedor de datos.

	El proveedor de datos se almacenará en un espacio de memoria del S.O.
CASO DE USO: Validar proveedor mecanismo de persistencia	
Eventos Principales	1) Inicializar objeto de tipo Proveedor. 2) Verificar los parámetros recibidos.
Eventos de Excepción	Devuelve una excepción cuando se detectan errores en los parámetros recibidos.
Pre-Condiciones	Los parámetros de configuración deben existir y ser válidos. Enviar todos los parámetros para crear el proveedor.
Post-Condiciones	Se emite un mensaje indicando que los parámetros con los que se inicializo el proveedor del mecanismo de persistencia son válidos y que se puede trabajar con él.

CASO DE USO: Obtener fuente de datos configuradas	
Eventos Principales	1) Inicializar sección del archivo de configuración para las fuentes de datos. 2) Leer la sección del archivo de configuración de las fuentes de datos. 3) Crear fuentes de datos configuradas.
Eventos de Excepción	Devuelve una excepción cuando se detectan errores en los parámetros leídos en el archivo de configuración.
Pre-Condiciones	Los parámetros de configuración deben existir y ser válidos. Enviar todos los parámetros para crear el proveedor.
Post-Condiciones	La colección de fuentes de datos se almacenará en un espacio de memoria del sistema operativo.

CASO DE USO: Validar fuente datos	
Eventos Principales	1) Inicializar objeto de tipo fuente de datos. 2) Verificar los parámetros configurados.
Eventos de Excepción	Devuelve una excepción cuando se detectan errores en los parámetros recibidos.
Pre-Condiciones	Los parámetros de configuración deben existir y ser válidos. Enviar todos los parámetros para crear el proveedor.
Post-Condiciones	Se emite un mensaje indicando que los parámetros con los que se inicializo la fuente de datos son válidos.

CASO DE USO: Obtener proveedores configurados	
Eventos Principales	1) Inicializar sección del archivo de configuración para los proveedores. 2) Leer la sección del archivo de configuración de los proveedores. 3) Crear proveedores de datos configurados.

Eventos de Excepción	Devuelve una excepción cuando se detectan errores en los parámetros recibidos.
Pre-Condiciones	Los parámetros de configuración deben existir y ser válidos. Enviar todos los parámetros para crear el proveedor.
Post-Condiciones	Se emite un mensaje indicando que los parámetros con los que se inicializo el proveedor son válidos. La colección de proveedores se almacenará en un espacio de memoria del sistema operativo.

Con formato: Numeración y viñetas

CASO DE USO: Crear conexión a fuente de datos	
Eventos Principales	1) Validar objeto proveedor. 2) Validar cadena de conexión. 3) Crear objeto de tipo conexión a la fuente de datos.
Eventos de Excepción	Devuelve una excepción cuando se detectan errores en los parámetros recibidos.
Pre-Condiciones	Instancias de un objeto Proveedor de datos. Definir una cadena de conexión válida.
Post-Condiciones	Se emite un mensaje indicando que los parámetros para crear la fuente de datos son válidos. Se puede interactuar con la base de datos indicada por la instancia del Proveedor de datos.

Con formato: Numeración y viñetas

3. Administrar acceso a datos

~~**Administrar Acceso a datos**~~

Realizar operaciones CRUD, obtiene datos, ejecuta procedimientos almacenados sobre un gestor de datos

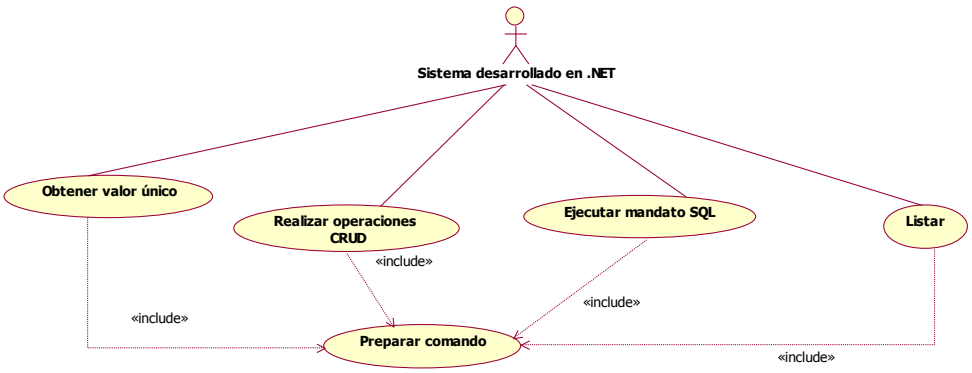


Figura 5.44.4: Caso de uso administrar acceso a datos

CASO DE USO: Administrar acceso a datos	
Eventos Principales	Para tener acceso a datos se puede utilizar transaccionalidad ó no, se pueden dar los siguientes eventos: 1) Realizar operaciones CRUD 2) Listar. 3) Obtener valor único. 4) Ejecutar Mandato SQL. 5) Preparar Comando.
Eventos Alternos	Si no esta configurado el proveedor o la fuente de datos no se puede realizar ninguno de los eventos nombrados anteriormente.
Eventos de Excepción	Si no esta configurado el proveedor o la fuente de datos y no ha sido creado anteriormente se devuelve una excepción indicando que no está correctamente configurado el pedido para el proveedor.
Pre-Condiciones	El archivo de configuración debe estar previamente configurado.
Post-Condiciones	Se crea el proveedor de datos solicitado y se almacena en la memoria para obtenerlo cuando se lo solicite nuevamente.

CASO DE USO: Realizar operaciones CRUD	
Eventos Principales	1) Actualizar información recibida 2) Eliminar información recibida
Eventos de Excepción	Devuelve una excepción cuando se detectan errores en los parámetros recibidos.
Pre-Condiciones	Validar que se haya ejecutado con éxito la Conexión al proveedor de datos.
Post-Condiciones	Se puede interactuar con la base de datos indicada por la instancia del Proveedor de datos. Los datos se actualizan en el proveedor de datos.

Con formato: Numeración y viñetas

CASO DE USO: Obtener valor único	
Eventos Principales	1) Obtener valor único
Eventos de Excepción	Devuelve una excepción cuando se detectan errores en los parámetros recibidos.
Pre-Condiciones	Validar que se haya ejecutado con éxito la Conexión al proveedor de datos. Definir una cadena de conexión válida.
Post-Condiciones	Un objeto con un solo valor generado en el proceso. Se puede realizar cualquier proceso con el valor obtenido.

Con formato: Numeración y viñetas

CASO DE USO: Ejecutar mandato SQL	
Eventos Principales	1) Ejecutar mandato SQL.
Eventos de Excepción	Devuelve una excepción cuando se detectan errores en los parámetros recibidos.
Pre-Condiciones	Validar que se haya ejecutado con éxito la conexión al proveedor de datos. Definir una cadena de conexión válida
Post-Condiciones	Retorna un valor entero indicando el número de registros que fueron afectados con la acción ejecutada. Los datos se actualizan en el proveedor de datos.

Con formato: Numeración y viñetas

CASO DE USO: Preparar comando	
Eventos Principales	1) Preparar un objeto Comando para ejecutar las acciones sobre un mecanismo de persistencia.
Eventos de Excepción	Devuelve una excepción cuando se detectan errores en los parámetros recibidos.
Pre-Condiciones	Validar que se haya ejecutado con éxito la conexión al proveedor de datos. Definir una cadena de conexión válida.
Post-Condiciones	Retorna un objeto tipo comando listo para ejecutar las acciones.

c. Diccionario de clases

El diccionario de clases correspondiente al diagrama de clases del “Motor de Persistencia para .Net”.

LISTADO DE CLASES

Namespace: CapaPersistencia.Datos

Clases que configuran el acceso a datos

- AccesoDatosConfiguración.
- FuenteDatos.
- FuenteDatosColección.
- ManejadorSecciónFuentesDatos.
- ManejadorSecciónProveedores.
- Proveedor.
- ProveedorColección.

Clases que implementan acceso a datos

- AdministradorDBCacheParámetros.
- AdministradorBaseDatos.

Clases que implementan creación de proveedor acceso a datos

- IConexiónFabrica.
- ConexiónFabrica.
- DataAdapterFabrica.
- DataParameterFabrica.

CLASES QUE CONFIGURAN EL ACCESO A DATOS

AccesoDatosConfiguración

Clase encargada de inicializar los proveedores y fuentes de datos con los que se va a trabajar. Aquí se crean algunos proveedores en código y además se leen los

del archivo de configuración. Tanto los proveedores como las fuentes de datos se almacenan en colecciones. Esta clase hace llamadas a las clases que leen el archivo de configuración, y en esas llamadas sé inicializan el proveedor por defecto y la fuente de datos por defecto.

Acceso: Público.
Clase Base: Objeto.

Atributos de la clase	
Atributos	**Descripción**
ErrorFuenteDatosDefectoNoEspecificada	Constante que guarda el mensaje de error a mostrarse cuando no se ha especificado una fuente de datos por defecto. La fuente de datos por defecto se especifica en el archivo de configuración.
ErrorProveedorDefectoNoEspecificado	Constante que guarda el mensaje de error a mostrarse cuando no se ha especificado un proveedor por defecto. El proveedor por defecto se especifica en el archivo de configuración.
ProveedorPorDefecto_Odbc	Constante que identifica al proveedor Odbc escrito en código.
ProveedorPorDefecto_OleDb	Constante que identifica al proveedor OleDb escrito en código.
ProveedorPorDefecto_Oracle	Constante que identifica al proveedor Oracle escrito en código.
ProveedorPorDefecto_SqlServ	Constante que identifica al proveedor SQLServer escrito en código.
FuenteDatosDefecto	FuenteDatos por defecto. Se especifica en el archivo de configuración y se setea en la clase 'ManejadorSecciónFuentesDatos'.
ProveedorDefecto	Proveedor por defecto. Se especifica en el archivo de configuración y sé setea en la clase 'ManejadorSecciónProveedores'.
FuentesDeDatosColección	Colección donde se almacenan todos los objetos FuenteDatos disponibles, tanto las escritas en código como las escritas en el archivo de configuración.
ProveedoresColección	Colección donde se almacenan todos los objetos Proveedor disponibles, tanto los escritos en código como los escritos en el archivo de configuración.

EstaInicializado	Campo interno de la clase que sirve como bandera para identificar si ya se ha hecho la iniciación de FuentesDatos y proveedores,
Propiedades de la clase	
Propiedades	**Descripción**
FuentesDeDatos	Propiedad que expone la colección de FuenteDatos.
Proveedores	Propiedad que expone la colección de Proveedores.
FuenteDatosPorDefecto	Propiedad que expone la FuenteDatos por defecto.
ProveedorPorDefecto	Propiedad que expone el Proveedor por defecto.
Métodos de la clase	
Métodos	**Descripción**
Inicializar	Se encarga de la creación de cuatro proveedores: OleDb, SQLServer, Oracle, Odbc. Además hace llamadas a clases que leen los proveedores y las fuentes de datos del archivo de configuración y agrega todos los proveedores y fuentes de datos a las colecciones respectivas. En las llamadas se asignan el proveedor por defecto y la FuenteDatos por defecto.

FuenteDatos

Clase con la información a partir de la cual se crearan los objetos concretos la cual es tomada de un objeto interno Proveedor, además posee información de la cadena de conexión con la cual se accede a una fuente de datos. El atributo Serializable es necesario ya que desde el archivo de configuración se pueden leer información para crear objetos FuenteDatos. Y esta información necesita ser deserializada y convertida en objetos FuenteDatos.

Acceso: Público.
Clase Base: Objeto.

Atributos de la clase	
Atributos	**Descripción**
_cadenaConexión	Cadena de conexión que indica los parámetros para conectarse a una determinada Base de Datos.

_NombreFuenteDatos	Cadena que sirve como identificador del objeto FuenteDatos.
proveedor	Contiene la información sobre el tipo de clases concretas con que ésta FuenteDatos va a trabajar, por ejemplo (OracleConnection, SQLConnection, DB2Connection, etc.)
Propiedades de la clase	
Propiedades	**Descripción**
cadenaConexión	Propiedad con la cadena de conexión a la Base de Datos.
NombreFuenteDatos	Propiedad con el identificador del objeto FuenteDatos.
ProveedorConfigurado	Tiene la información sobre los tipos de datos concretos a instanciar.
NombreProveedor	Propiedad que sirve para inicializar el campo ProveedorConfigurado. Esta propiedad es utilizada solamente en la deserializacion desde el archivo de configuración.
Métodos de la clase	
Métodos	**Descripción**
FuenteDatos	Constructor vacío, necesario para serialización, no se debe utilizar para creación de instancias.
FuenteDatos	Constructor que inicializa todos los campos de un FuenteDatos menos el nombre de FuenteDatos.
FuenteDatos	Constructor que inicializa todos los campos de un FuenteDatos.

FuenteDatosColección

La clase HybridDictionary da diferentes implementaciones para IDictionary según la cantidad de items. Cuando tenemos pocos trabaja con ListDictionary Cuando tenemos muchos trabaja con HashTable Todos los objetos a almacenar en esta colección son del tipo FuenteDatos.

Acceso: Público.

Clase Base: HybridDictionary.

Propiedades de la clase	
Propiedad	**Descripción**
This	Propiedad indexada. Sirve para acceder a los objetos de la colección directamente a través de un índice, EJ: FuenteDatosColección[2] retorna el objeto FuenteDatos

	almacenado en la tercera posición.
Métodos de la clase	
Métodos	**Descripción**
Buscar	Busca un objeto FuenteDatos dentro de la colección. Si no lo encuentra lanza una excepción, de lo contrario retorna la FuenteDatos.

ManejadorSecciónFuentesDatos

Esta clase maneja la sección *SeccionCapaPersistencia.FuentesDatos* del archivo de configuración para poder manejar secciones personalizadas implementa IConfigurationSectionHandler.

Acceso: Público.
Clase Base: Objeto.
Implementa la interfaz: IConfigurationSectionHandler.

Métodos de la clase	
Métodos	**Descripción**
Create	Este método es requerido al implementar IConfigurationSectionHandler y en este caso realiza internamente dos tareas: 1. Carga los FuentesDatos que existan en el archivo de configuración. 2. Setea el proveedor por defecto seteado en el archivo de configuración.

ManejadorSecciónProveedores

Esta clase maneja la sección *SeccionCapaPersistencia.Proveedores* del archivo de configuración para poder manejar secciones personalizadas implementa IConfigurationSectionHandler.

Acceso: Público.
Clase Base: Objeto.
Implementa la interfaz: IConfigurationSectionHandler.

Métodos de la clase	
Métodos	**Descripción**
Create	Este método es requerido al implementar IConfigurationSectionHandler y en este caso realiza internamente dos tareas: 1. Carga los proveedores que existan en el archivo de configuración. 2. Setea el proveedor por defecto seteado en el archivo de configuración.

Proveedor

Clase con la información a partir de la cual se crearan los objetos concretos. El atributo serializable es necesario ya que desde el archivo de configuración se pueden leer información para crear objetos Proveedor. Y esta información necesita ser deserializada y convertida en objetos Proveedor.

Acceso: Público.
Clase Base: Objeto.

Atributos de la clase	
Atributos	**Descripción**
nombreClaseAdapter	Información sobre cual es la clase con la que creará el objeto concreto DataAdapter que implementa la interfaz IDbDataAdapter.
nombreAssembly	Nombre del Assembly que contiene las clases para la creación de los objetos CommandBuilder, Connection y DataAdapter.
nombreClaseCommandBuilder	Información sobre cual es la clase con la que creará el objeto concreto CommandBuilder.
nombreClaseConnection	Información sobre cual es la clase con la que creará el objeto concreto Connection que implementa la interfaz IDbConnection.
nombreProveedor	Nombre que sirve como clave para identificar un objeto Proveedor determinado ya que los objetos Proveedor se almacenarán dentro de un IDictionary.
Propiedades de la clase	
Propiedades	**Descripción**

NombreClaseAdapter	Propiedad que indica cual es la clase con la que creará el objeto concreto DataAdapter.
NombreAssembly	Propiedad que indica el nombre del Assembly que contiene las clases para la creación de los objetos
NombreClaseCommandBuilder	Propiedad que indica cual es la clase con la que creará el objeto concreto CommandBuilder.
NombreClaseConnection	Propiedad que indica cual es la clase con la que creará el objeto concreto Connection.
NombreProveedor	Propiedad que indica el nombre que sirve como clave para identificar un objeto Proveedor.
Métodos de la clase	
Métodos	**Descripción**
Proveedor	Reservado para serialización y no se debe utilizar en creación de instancias.
Proveedor	Constructor que inicializa todos los campos de un Proveedor menos el nombre del Proveedor.
Proveedor	Constructor que inicializa todos los campos de un Proveedor.

ProveedorColección

La clase HybridDictionary da diferentes implementaciones para IDictionary según la cantidad de items. Cuando tenemos pocos trabaja con ListDictionary cuando tenemos muchos trabaja con HashTable, todos los objetos a almacenar en esta colección son del tipo Proveedor.

Acceso: Público

Clase Base: HybridDictionary

Propiedades de la clase	
Propiedades	**Descripción**
This	Propiedad indexada. Sirve para acceder a los objetos de la colección directamente a través de un índice, Ej: ProveedorColección[2] retorna el objeto Proveedor almacenado en la tercera posición.

Métodos de la clase	
Métodos	**Descripción**
Buscar	Busca un proveedor dentro de la colección. Si no lo encuentra lanza una excepción, de lo contrario retorna el proveedor.

CLASES QUE IMPLEMENTAN ACCESO A DATOS

AdministradorBaseDatos

Contiene funciones que permiten ejecutar acciones como INSERT, DELETE, UPDATE, para obtener un conjunto de datos o un solo valor, para asignar valores a parámetros de un comando o para añadir parámetros a un comando.

Acceso: Público.

Clase Base: Objeto.

Atributos de la clase	
Atributos	**Descripción**
Proveedor	Proveedor con la información para la creación de la conexión.
Propiedades de la clase	
Propiedades	**Descripción**
ProveedorConfigurado	Propiedad solo de lectura, expone el proveedor configurado.
Métodos de la clase	
Métodos	**Descripción**
AdministradorBaseDatos	Constructor de la clase AdministradorBaseDatos que configura un objeto Proveedor.
CrearAdapter	Método que crea un objeto concreto que implemente la interfaz IDbDataAdapter. Utiliza el método CrearAdapter de la clase DataAdapterFabrica.
CrearConnection	Método estático para la obtención de un objeto que implemente IDbConnection Para obtener la información sobre que tipo de conexión debe crear, recibe como parámetros un proveedor y una cadena de conexión.
AdjuntarParámetros	Método usado para adjuntar un arreglo de IDataParameters al objeto IDbCommand. Este método asigna un valor de Null a los parámetros que pueden ser de salida o de entrada y a los

	parámetros que no tienen valor.
AsignarValoresParámetros	Método que asigna a un arreglo de parámetros sus valores. El arreglo de parámetros y el de valores debe tener el mismo número. El orden de asignación es igual al orden que tiene el arreglo.
PrepararComando	Este método asigna, si es necesario, una conexión, transacción, tipo de comando y parámetros al comando.
EjecutarAccion(IDbTransaction, CommandType, string, IDataParameter[])	Ejecuta una acción como INSERT, DELETE, UPDATE, COUNT, MAX dentro de una transacción Método principal
EjecutarAccion(IDbTransaction, CommandType, string)	Ejecuta una acción como INSERT, DELETE, UPDATE, COUNT, MAX dentro de una transacción se envían acciones que no necesitan argumentos para ejecutarse.
EjecutarAccion(IDbConnection, CommandType, string, IDataParameter[])	Ejecuta una acción como INSERT, DELETE, UPDATE, COUNT, MAX sin transaccionalidad Método principal.
EjecutarAccion(IDbConnection, CommandType, string)	Ejecuta una acción como INSERT, DELETE, UPDATE, COUNT, MAX sin transaccionalidad se envían acciones que no necesitan argumentos para ejecutarse
EjecutarAccion(string, CommandType, string, IDataParameter[])	Ejecuta una acción como INSERT, DELETE, UPDATE, COUNT, MAX sin transaccionalidad se envían acciones que necesitan argumentos para ejecutarse. Utiliza una cadena de conexión.
EjecutarAccion(string, CommandType, string)	Ejecuta una acción como INSERT, DELETE, UPDATE, COUNT, MAX sin transaccionalidad se envían acciones que no necesitan argumentos para ejecutarse. Utiliza una cadena de conexión.
EjecutarAccion(string, string, IDataParameter[])	Ejecuta un procedimiento almacenado. Este método busca los parámetros para el procedimiento almacenado si el argumento ValoresParámetros no es null y asigna los valores basado en el orden de los parámetros.
EjecutarAccion(IDbConnection, string, IDataParameter[])	Ejecuta un procedimiento almacenado utilizado una conexión, válida. Este método busca los parámetros para el procedimiento almacenado si el argumento ValoresParámetros no es null y asigna los valores basado en el orden de los parámetros.
EjecutarAccion(IDbTransaction, string, IDataParameter[])	Ejecuta un procedimiento almacenado utilizado una transacción válida. Este método busca los parámetros para el procedimiento almacenado si el argumento ValoresParámetros no es null y asigna los valores basado en

	el orden de los parámetros.
ObtenerDatos(string, string, IDataParameter[])	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta necesita parámetros.
ObtenerDatos(IDbConnection, CommandType, string, IDataParameter[])	Obtiene un o un conjunto de datos dentro de una transacción Debe utilizarse cuando la consulta necesita parámetros.
ObtenerDatos(IDbTransaction, CommandType, string, IDataParameter[])	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta necesita parámetros.
ObtenerDatos(string, CommandType, string, IDataParameter[])	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta NO necesita parámetros.
ObtenerDatos(string, CommandType, string)	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta NO necesita parámetros
ObtenerDatos(IDbConnection, CommandType, string)	Obtiene un o un conjunto de datos dentro de una transacción. Debe utilizarse cuando la consulta NO necesita parámetros.
ObtenerDatos(IDbTransaction, CommandType, string)	Ejecuta un procedimiento almacenado, utiliza una cadena de conexión. Este método busca los parámetros para el procedimiento almacenado si el argumento Valores parámetros no es Rull y asigna los valores basado en el orden de los parámetros.
Obtener datos(IDbConnection , string, IDataParameter[])	Ejecuta un procedimiento almacenado, utiliza una conexión válida. Este método busca los parámetros para el procedimiento almacenado si el argumento ValoresParámetros no es null y asigna los valores basado en el orden de los parámetros.
ObtenerDatos(IDbTransaction, string, IDataParameter[])	Ejecuta un procedimiento almacenado, utiliza una transacción válida. Este método busca los parámetros para el procedimiento almacenado si el argumento ValoresParámetros no es null y asigna los valores basado en el orden de los parámetros.
LlenarDataTable(IDbConnectio, CommandType, e, DataTable, IDataParameter[])	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta necesita parámetros.
LlenarDataTable(IDbTransactio, CommandType, e, DataTable, IDataParameter[])	Obtiene un o un conjunto de datos dentro de una transacción. Debe utilizarse cuando la consulta necesita parámetros.
LlenarDataTable(e, CommandType, e, DataTable, IDataParameter[])	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta necesita parámetros.

LlenarDataTable(e, CommandType, e, DataTable)	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta NO necesita parámetros.
LlenarDataTable(IDbConnectio, CommandType, e, DataTable)	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta NO necesita parámetros.
LlenarDataTable(IDbTransactio, CommandType, e, DataTable)	Obtiene un o un conjunto de datos dentro de una transacción Debe utilizarse cuando la consulta NO necesita parámetros.
LlenarDataTable(e, e , DataTable, IDataParameter[])	Ejecuta un procedimiento almacenado, utiliza una cadena de conexión Este método busca los parámetros para el procedimiento almacenado si el argumento ValoresParámetros no es null y asigna los valores basado en el orden de los parámetros.
LlenarDataTable(IDbConnection, e , DataTable, IDataParameter[])	Ejecuta un procedimiento almacenado, utiliza una conexión válida. Este método busca los parámetros para el procedimiento almacenado si el argumento ValoresParámetros no es null y asigna los valores basado en el orden de los parámetros.
LlenarDataTable(IDbTransaction, e , DataTable, IDataParameter[])	Ejecuta un procedimiento almacenado, utiliza una transacción válida. Este método busca los parámetros para el procedimiento almacenado si el argumento ValoresParámetros no es null y asigna los valores basado en el orden de los parámetros.
ActualizarDataTable(IDbTransaction, CommandType, string, DataTable, IDataParameter[])	Obtiene un o un conjunto de datos dentro de una transacción. Debe utilizarse cuando la consulta necesita parámetros
ActualizarDataTable(IDbConnection, CommandType, string, DataTable, IDataParameter[])	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta necesita parámetros.
ActualizarDataTable(string, CommandType, string, DataTable, IDataParameter[])	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta necesita parámetros.
ActualizarDataTable(string, CommandType, string, DataTable)	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta NO necesita parámetros
ActualizarDataTable(IDbConnection, CommandType, string)	Obtiene un o un conjunto de datos. Debe utilizarse cuando la consulta NO necesita parámetros.
ActualizarDataTable(IDbTransaction, CommandType, string, DataTable)	Obtiene un o un conjunto de datos dentro de una transacción. Debe utilizarse cuando la consulta NO necesita parámetros
ActualizarDataTable(string, string, DataTable, IDataParameter[])	Ejecuta un procedimiento almacenado, utiliza una cadena de conexión. Este método busca los parámetros para el procedimiento almacenado si el argumento

	ValoresParámetros no es null y asigna los valores basado en el orden de los parámetros.
ActualizarDataTable(IDbConnection, string, DataTable, IDataParameter[])	Ejecuta un procedimiento almacenado, utiliza una conexión válida. Este método busca los parámetros para el procedimiento almacenado si el argumento ValoresParámetros no es null y asigna los valores basado en el orden de los parámetros.
ActualizarDataTable(IDbTransaction, string, DataTable, IDataParameter[])	Ejecuta un procedimiento almacenado, utiliza una transacción válida. Este método busca los parámetros para el procedimiento almacenado si el argumento ValoresParámetros no es null y asigna los valores basado en el orden de los parámetros.
ObtenerValor(IDbTransaction, CommandType, string, IDataParameter[])	Obtiene un solo valor como resultado de una sentencia SQL o un procedimiento almacenado que necesita parámetros Utiliza una transacción válida.
ObtenerValor(IDbConnection, CommandType, string, IDataParameter[])	Obtiene un solo valor como resultado de una sentencia SQL o un procedimiento almacenado que necesita parámetros. Utiliza una conexión válida.
ObtenerValor(string, CommandType, string, IDataParameter[])	Obtiene un solo valor como resultado de una sentencia SQL o un procedimiento almacenado que necesita parámetros. Utiliza una cadena de conexión válida para acceder a la BD.
ObtenerValor(string, CommandType, string)	Obtiene un solo valor como resultado de una sentencia SQL o un procedimiento almacenado que no necesita parámetros. Utiliza una cadena de conexión válida para acceder a la BD.
ObtenerValor(IDbConnection, CommandType, string)	Obtiene un solo valor como resultado de una sentencia SQL o un procedimiento almacenado que NO necesita parámetros. Utiliza una conexión válida.
ObtenerValor(IDbTransaction, CommandType, string)	Obtiene un solo valor como resultado de una sentencia SQL o un procedimiento almacenado que NO necesita parámetros. Utiliza una transacción válida.
ObtenerValor(string, string, IDataParameter[])	Obtiene un solo valor como resultado de un procedimiento almacenado que necesita parámetros. Utiliza una cadena de conexión válida.
ObtenerValor(IDbConnection, string, IDataParameter[])	Obtiene un solo valor como resultado de un procedimiento almacenado que necesita parámetros. Utiliza una de conexión válida.
ObtenerValor(IDbTransaction, string, IDataParameter[])	Obtiene un solo valor como resultado de una sentencia SQL o un procedimiento almacenado que necesita parámetros Utiliza una transacción válida.

AdministradorDBCacheParámetros

Contiene funciones para liberar el caché estático de los parámetros de un procedimiento almacenado y funciones para encontrar los parámetros de un procedimiento almacenado en tiempo de ejecución. Los IDataParameter se almacenan en un Hashtable para futuros usos.

Acceso: Público.

Clase Base: Objeto.

Atributos de la clase	
Atributos	**Descripción**
CacheDeParametros	Hashtable que contiene los parámetros obtenidos.
Métodos de la clase	
Métodos	**Descripción**
AdministradorDBCacheParámetros	Constructor privado para prevenir instancias de este objeto. Solo tiene métodos estáticos.
ObtenerParámetros	Obtiene los parámetros del objeto CommandBuilder.
BuscarParámetrosSPInterno	Determina en tiempo de ejecución los parámetros apropiados para un procedimiento almacenado.
BuscarParámetrosSP	Determina en tiempo de ejecución los parámetros apropiados para un procedimiento almacenado.
ClonarParámetros	Duplica los parámetros.
Clear	Limpia el caché de parámetros.
ObtenerCacheParámetros	Recupera un arreglo de parámetros del caché.
ObtenerParámetrosSP	Recupera los IDataParameters apropiados para el SP.

CLASES QUE IMPLEMENTAN CREACIÓN DE PROVEEDOR ACCESO A DATOS

IConexiónFabrica Interfase

Interfaz que define los métodos para crear objetos Connection

Acceso: Público.

Propiedades de la clase	
Propiedades	**Descripción**
AbrirEnCreación	Propiedad que indica si la conexión debe ser abierta antes de retornarla.
Métodos de la clase	
Métodos	**Descripción**
CrearConexión	Firma del método que crea un objeto concreto que implemente la interfaz IDbConnection.

ConexiónFabrica

Clase que crea los objetos Connection a partir de un Proveedor pasado como parámetro en el constructor. Cada ConexiónFabrica sabe crear un tipo específico de Connection y los ConexiónFabrica se almacenan en un diccionario para futuros usos.

Acceso: Público.
Clase Base: Objeto.
Implementa la interfaz: IConexiónFabrica.

Atributos de la clase	
Atributos	**Descripción**
tipoClaseConnection	Tipo de dato a partir del cual se creará la clase concreta Connection.
abrirEnCreacion	Indica si la conexión debe ser abierta antes de retornarla al llamador.
proveedor	Proveedor con la información para la creación de la conexión.
FabricasIDictionary	Diccionario para almacenar los objetos ConexiónFabrica que se vayan creando durante la ejecución del programa.
Propiedades de la clase	
Propiedades	**Descripción**
AbrirEnCreación	Propiedad que indica si la conexión debe ser abierta antes de retornarla al llamador.

ProveedorConfigurado	Propiedad que expone el objeto Proveedor que tiene la información sobre el objeto Connection creado.
Métodos de la clase	
Métodos	**Descripción**
ConexiónFabrica	Constructor donde se inicializa el Proveedor.
CrearConexión	Método que crea un objeto concreto que implemente la interfaz IDbConnection. Si se ha especificado AbrirEnCreación = true, la conexión es devuelta abierta.
CrearConexiónInterno	Método interno que crea por reflexión el objeto que implemente IDbConnection a partir de la información contenida en el objeto _Proveedor.
CrearConnection()	Método estático para la obtención de un objeto que implemente IDbDataAdater Para obtener la información sobre que tipo de conexión debe crear, utiliza la fuente de datos por defecto que reside en AccesoDatosConfiguración y es especificada en el archivo de configuración.
CrearConnection(Proveedor, e)	Método estático para la obtención de un objeto que implemente IDbConnection Para obtener la información sobre que tipo de conexión debe crear, recibe como parámetros un proveedor y una cadena de conexión.
CrearConnection (FuenteDatos)	Método estático para la obtención de un objeto que implemente IDbDataAdater para obtener la información sobre que tipo de conexión debe crear. Recibe como parámetro un objeto FuenteDatos.
CrearConnectionDeFuenteDatos	Método estático para la obtención de un objeto que implemente IDbDataAdater. Para obtener la información sobre que tipo de conexión debe crear, toma como parámetros un string con el nombre de una FuenteDatos, con este nombre busca una FuenteDatos en la colección AccesoDatosConfiguración.FuentesDeDatos donde están todas las FuentesDatos disponibles.
ObtenerConexiónFabrica	Método que retorna un 'ConexiónFabrica'. La primera vez que lo crea lo almacena en el diccionario _Fabricas y después no vuelve a crear una nueva instancia, lo obtiene desde _Fabricas. Para el ingreso de los 'ConexiónFabrica' al diccionario de almacenamiento requiere un (key), esta es creada usando CreateQualifiedName que genera el nombre de clase Connection.

DataAdapterFabrica

Clase que crea los objetos DataAdapter a partir de un Proveedor pasado como parámetro en el constructor. Cada DataAdapterFabrica sabe crear un tipo específico de DataAdapter y los DataAdapterFabrica se almacenan en un diccionario para futuros usos.

Acceso: Público.

Clase Base: Objeto.

Atributos de la clase	
Atributos	**Descripción**
errorNombreClaseAdapterFaltant	Constante que guarda el mensaje de error a mostrarse cuando no se haya definido el nombre de la clase del Adapter para un Proveedor.
proveedor	Proveedor con la información para la creación del Adapter.
TipoClaseAdapter	Tipo de dato a partir del cual se creará la clase del Adapter.
FabricasIDictionary	Diccionario para almacenar los objetos DataAdapterFabrica que se vayan creando durante la ejecución del programa.
Propiedades de la clase	
Propiedades	**Descripción**
ProveedorConfigurado	Propiedad que expone el objeto Proveedor que tiene la información sobre el Adapter creado.
Métodos de la clase	
Métodos	**Descripción**
DataAdapterFabrica	Constructor donde se inicializa el Proveedor.
CrearAdapter()	Método que crea un objeto concreto que implemente la interfaz IDbDataAdapter.
CrearAdapter(IDbCommand)	Método que crea un objeto que implemente la interfaz IDbDataAdapter. Pasa como parámetro el objeto comando a ejecutarse.
CrearAdapter(string,IDbConnect)	Método que crea un objeto que implemente la interfaz IDbDataAdapter. Pasa como parámetro la sentencia a ejecutarse.
CrearAdapter(Proveedor)	Método estático para la obtención de un objeto que implemente IDbDataAdater con la información del proveedor pasado como parámetro.
CrearAdapter(Proveedor,IDb	Método estático para la obtención de un objeto que implemente

Command)	IDbDataAdater con la información del proveedor y el comando pasado como parámetro.
CrearAdapter(Proveedor,string,IDbConnection)	Método estático para la obtención de un objeto que implemente IDbDataAdater con la información del proveedor, la sentencia y la conexión pasado como parámetro.
ObtenerUnDataAdapterFabrica	Método que retorna un 'DataAdapterFabrica'. La primera vez que lo crea lo almacena en el diccionario _Fabricas y después lo obtiene desde _Fabricas y no se vuelve a crear una nueva instancia. Para el ingreso de los 'DataAdapterFabrica' al diccionario de almacenamiento se necesita una clave (key), esta es creada usando CreateQualifiedName que genera el nombre completo de la clase Adapter.
CrearAdapterInterno	Crea el objeto concreto que implemente IDbDataAdapter. Lo crea por reflexion a partir de los strings con la información contenidos en el campo _Proveedor (tipo Proveedor).

DataParameterFabrica

Clase que crea los objetos DataAdapter a partir de un Proveedor pasado como parámetro en el constructor. Cada DataAdapterFabrica sabe crear un tipo específico de DataAdapter y los DataAdapterFabrica se almacenan en un diccionario para futuros usos.

Acceso: Público.

Clase Base: Objeto.

Atributos de la clase	
Atributos	**Descripción**
errorNombreClaseParameterFaltante	Constante que guarda el mensaje de error a mostrarse cuando no se haya definido el nombre de la clase del Parameter para un Proveedor.
Métodos de la clase	
Métodos	**Descripción**
CrearArregloParámetros	Método que crea un objeto concreto que implemente la interfaz IDataParameterColección.

~~7.2.3~~5.2.3 Modelo dinámico

Describe los aspectos de un sistema que tratan de la temporización y secuencia de operaciones (sucesos que marcan los cambios, secuencias de sucesos, estados que definen el contexto para los sucesos) y la organización de sucesos y estados.

a. Diagramas de secuencia propuesto por UML

Administrar proveedor mecanismo persistencia

1. Validar Proveedor mecanismo de persistencia. Diagrama de secuencia que muestra los pasos que se deben dar para verificar que el proveedor utilizado para realizar operaciones sobre el gestor de datos es válido, ver figura 5.5
2. Obtener Proveedores configurados. Diagrama de secuencia que muestra los pasos que se realizan para obtener de la colección de proveedores el objeto proveedor requerido para trabajar sobre el gestor de datos, ver figura 5.6
3. Crear Proveedor mecanismo de persistencia. Diagrama de secuencia que muestra los pasos que se realizan para construir un objeto Proveedor utilizando los parámetros configurados, ver figura 5.7
4. Obtener Manejador mecanismo persistencia. Diagrama de secuencia que muestra los pasos que se deben dar para obtener un objeto manejador del gestor de datos que pueda realizar operaciones CRUD, ver figura 5.8
5. Validar Fuente Datos. Diagrama de secuencia que muestra los pasos que se deben realizar para verificar que la fuente de datos utilizado para conectarse al gestor de datos es válida, ver figura 5.9
6. Obtener Fuente de datos configuradas. Diagrama de secuencia que muestra los pasos que se realizan para obtener de la colección de fuente de datos el

objeto fuente de datos requerido para conectarse con el gestor de datos, ver figura 5.10

7. Crear Conexión fuente datos. Diagrama de secuencia que muestra los pasos que se realizan para construir un objeto conexión a la fuente de datos utilizando los parámetros configurados, ver figura 5.11

Administrar acceso a datos

1. Realizar operaciones CRUD. Diagrama de secuencia que muestra los pasos que se deben realizar para ejecutar operaciones de actualización, eliminación, inserción sobre un gestor de datos, ver figura 5.12
2. Listar. Diagrama de secuencia que muestra los pasos que se deben realizar para obtener un listado de datos desde un gestor de datos, ver figura 5.13
3. Obtener valor único. Diagrama de secuencia que muestra los pasos que se deben realizar para obtener un valor escalar desde un gestor de datos, ver figura 5.14
4. Ejecutar Mandato SQL. Diagrama de secuencia que muestra los pasos que se deben realizar para ejecutar sentencias SQL sobre un gestor de datos, ver figura 5.15
5. Preparar Comando. Diagrama de secuencia que muestra los pasos que se deben realizar para preparar un objeto comando que permite que las operaciones que se realizan sobre el gestor de datos se realicen con características como transaccionalidad, parámetros, etc., ver figura 5.16

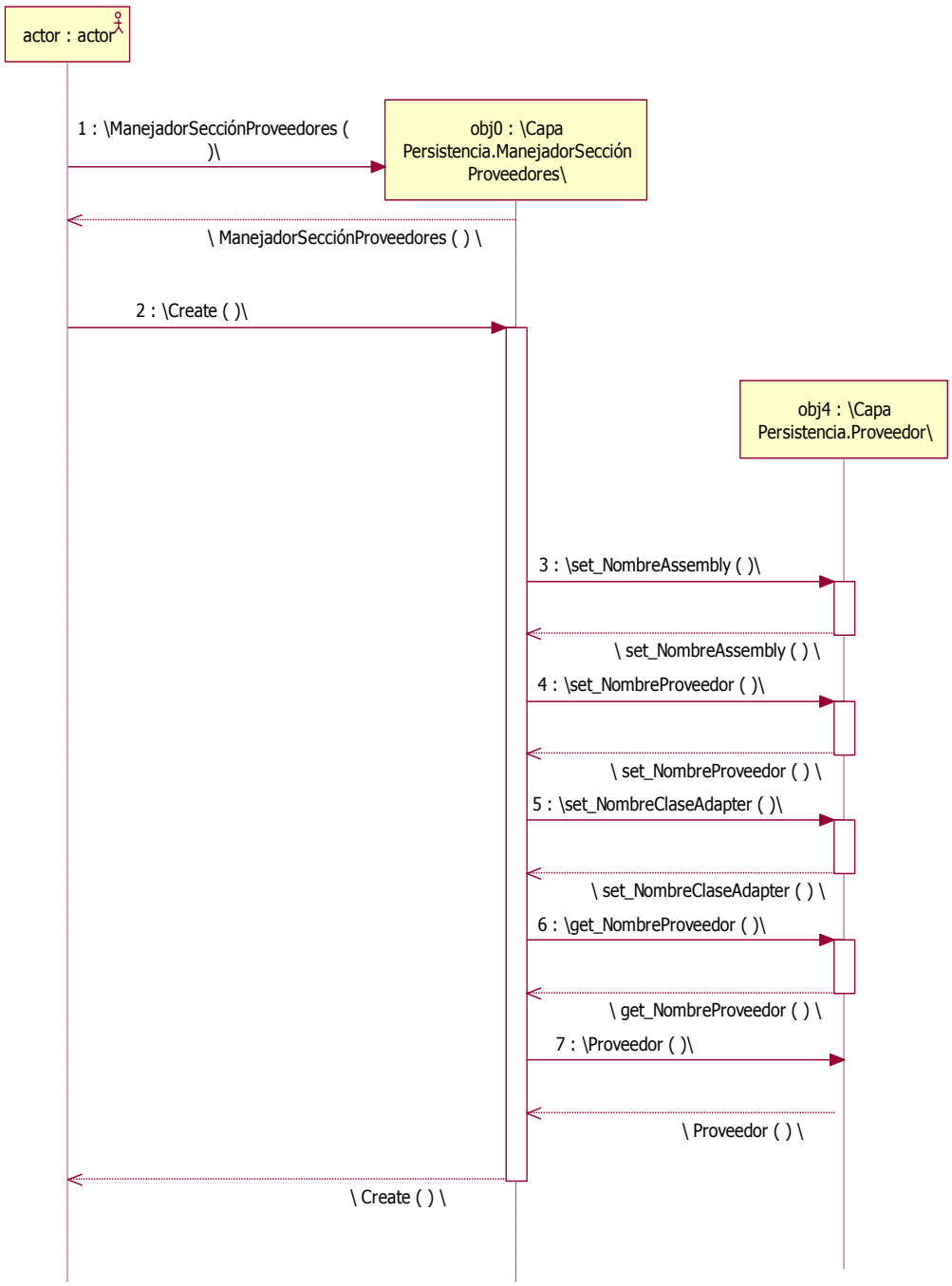


Figura 5.5: Diagrama de secuencia valida proveedor de mecanismo de persistencia

La figura muestra validación de una fuente de datos.

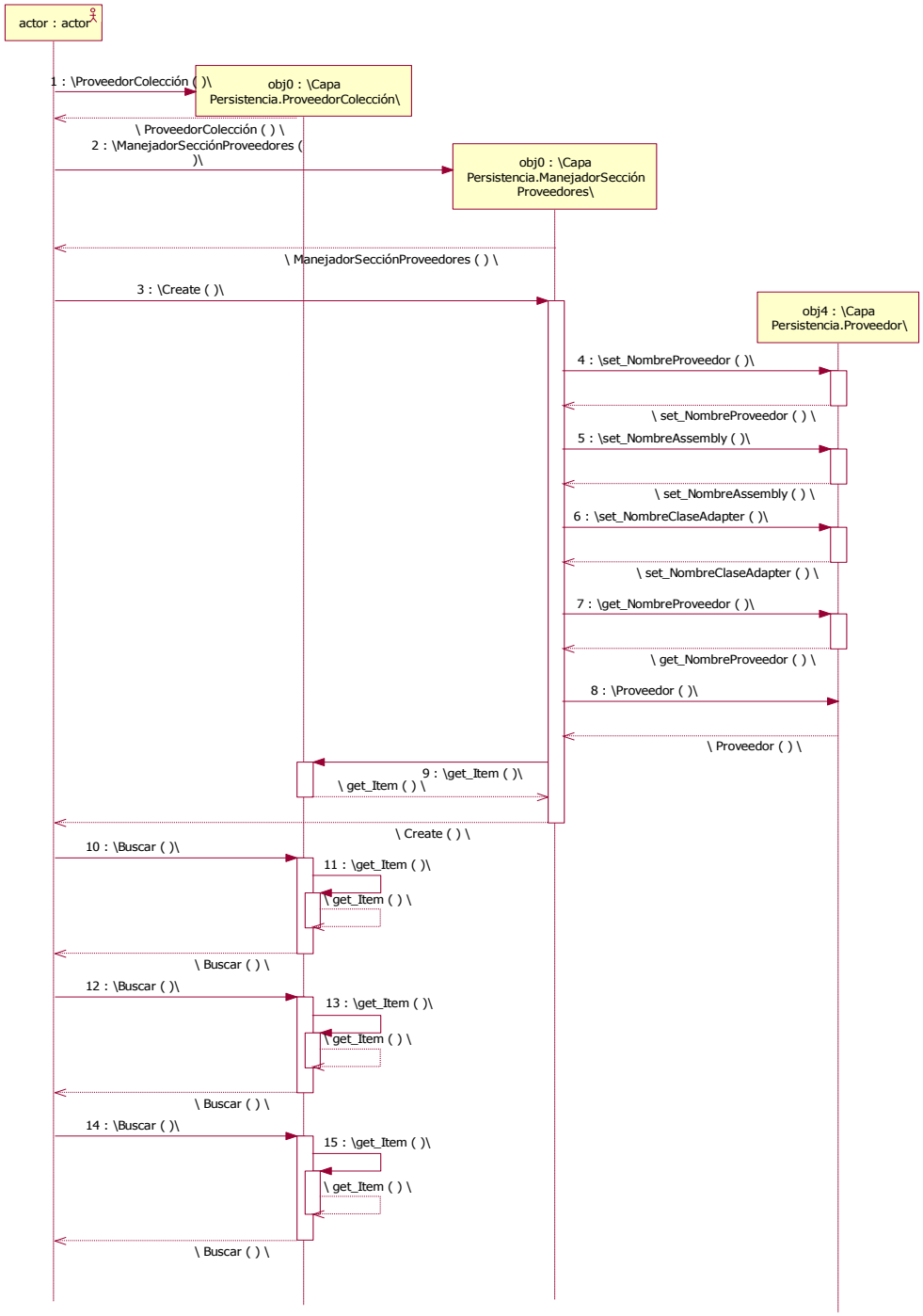


Figura 5.6: Diagrama de secuencia obtener proveedor configurado.

La figura muestra como se obtiene un proveedor de datos, configurado.

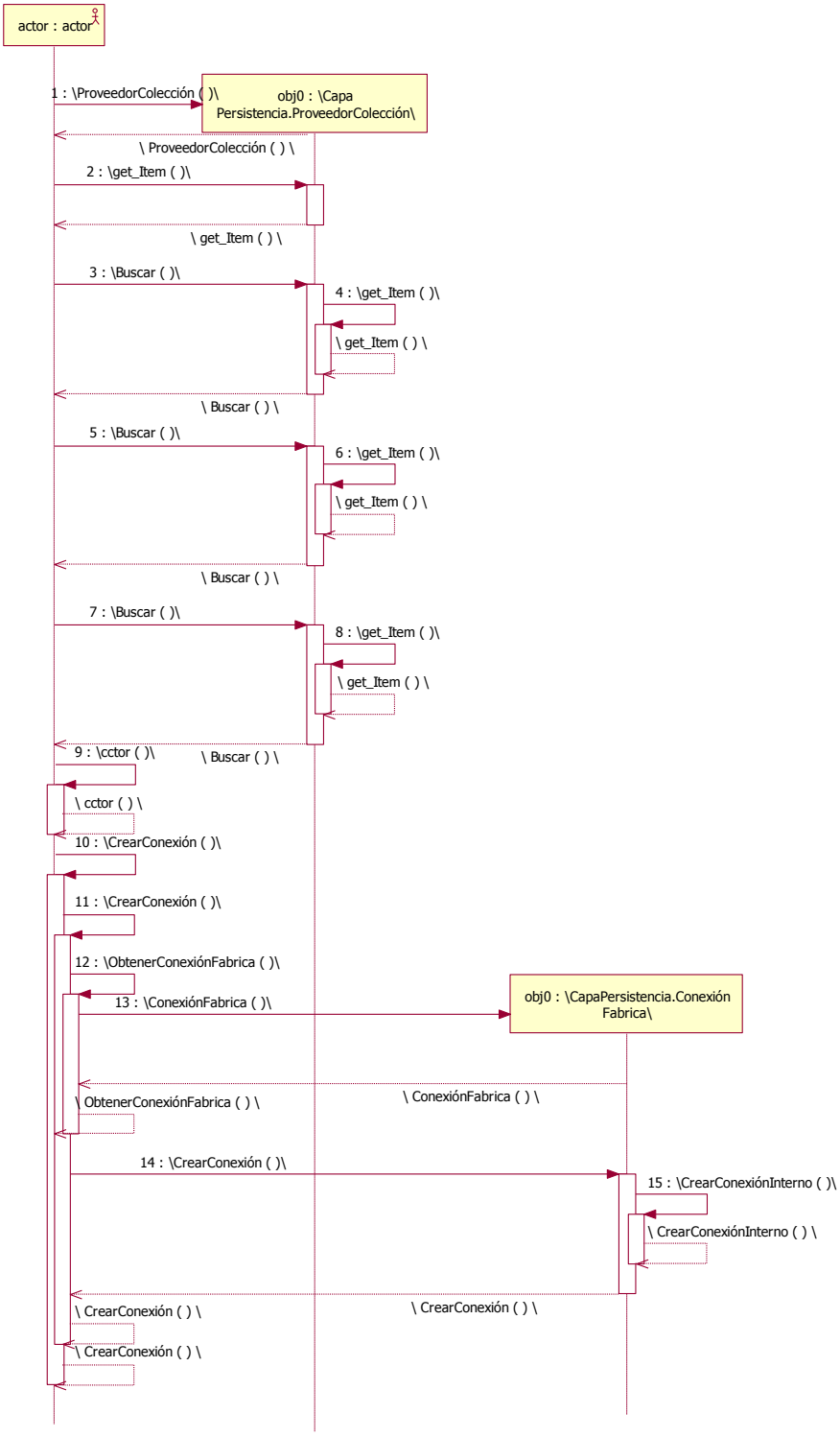


Figura 5.7: Diagrama de secuencia crear proveedor mecanismo de persistencia

La figura muestra como crear un objeto proveedor con los datos configurados.

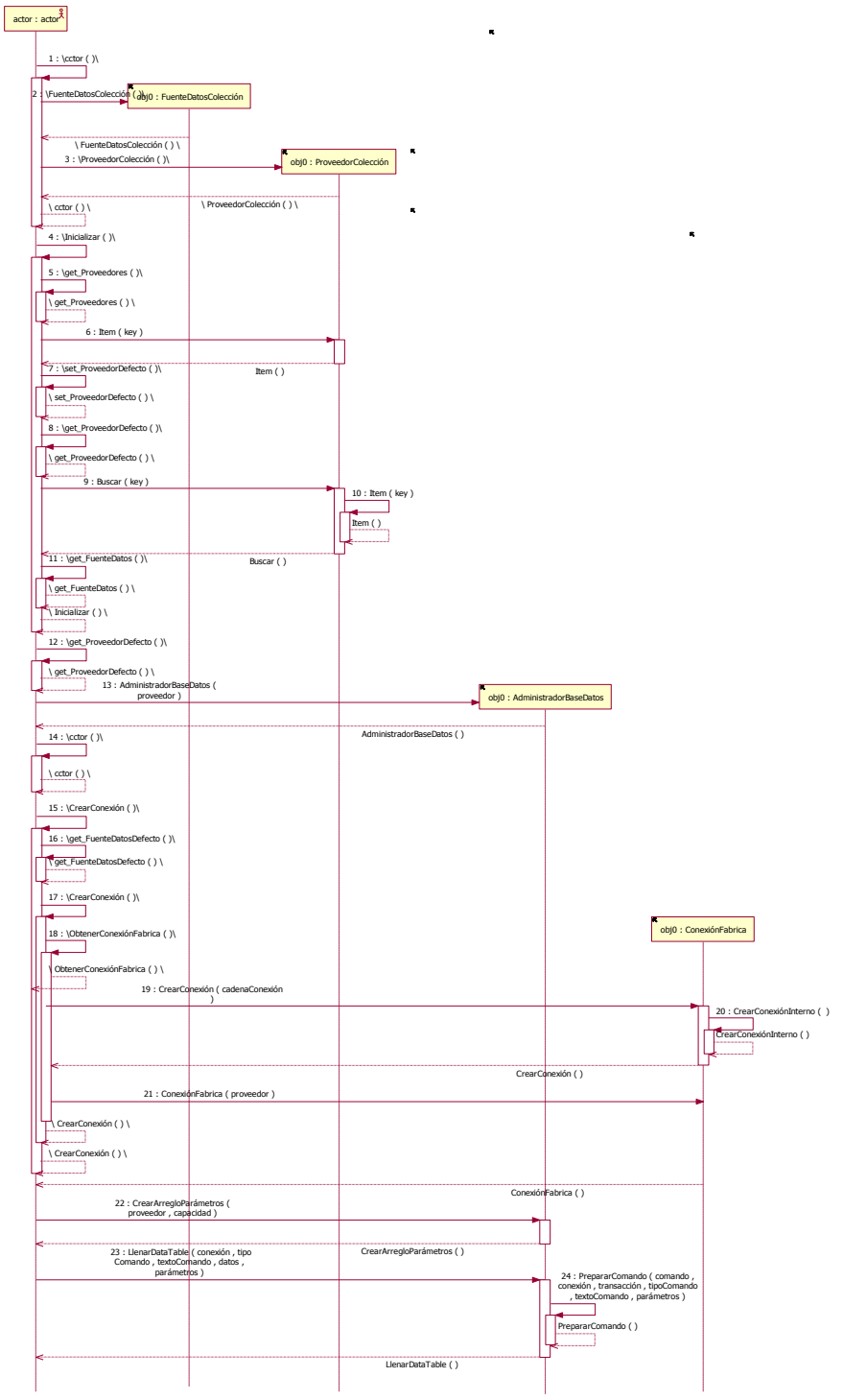


Figura 5.8: Diagrama de secuencia obtener manejador mecanismo persistencia.

La figura muestra como se obtiene un objeto manejador de datos.

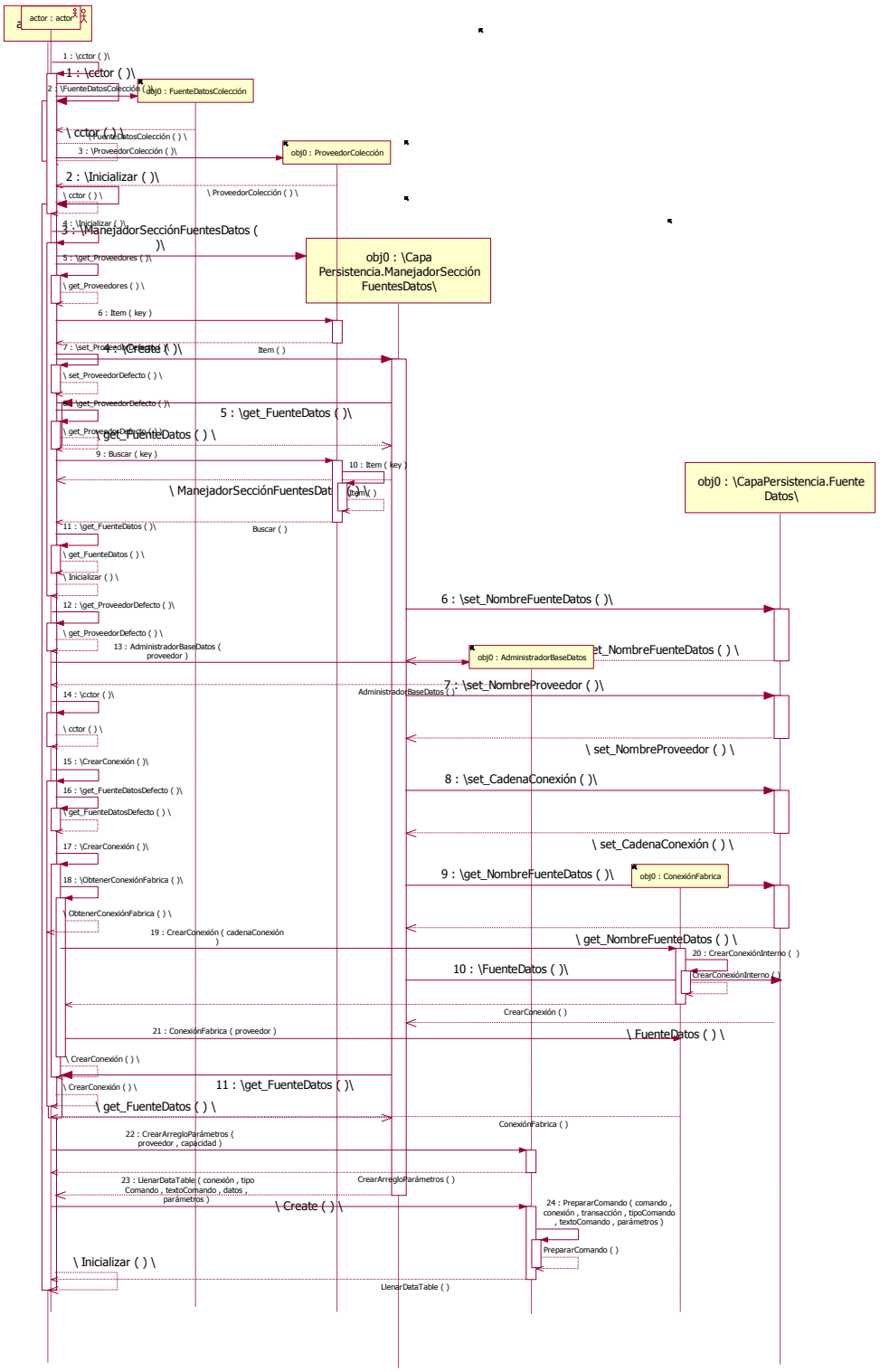


Figura 5.9: Diagrama de secuencia validar fuente de datos

La figura muestra la validación de una fuente de datos.

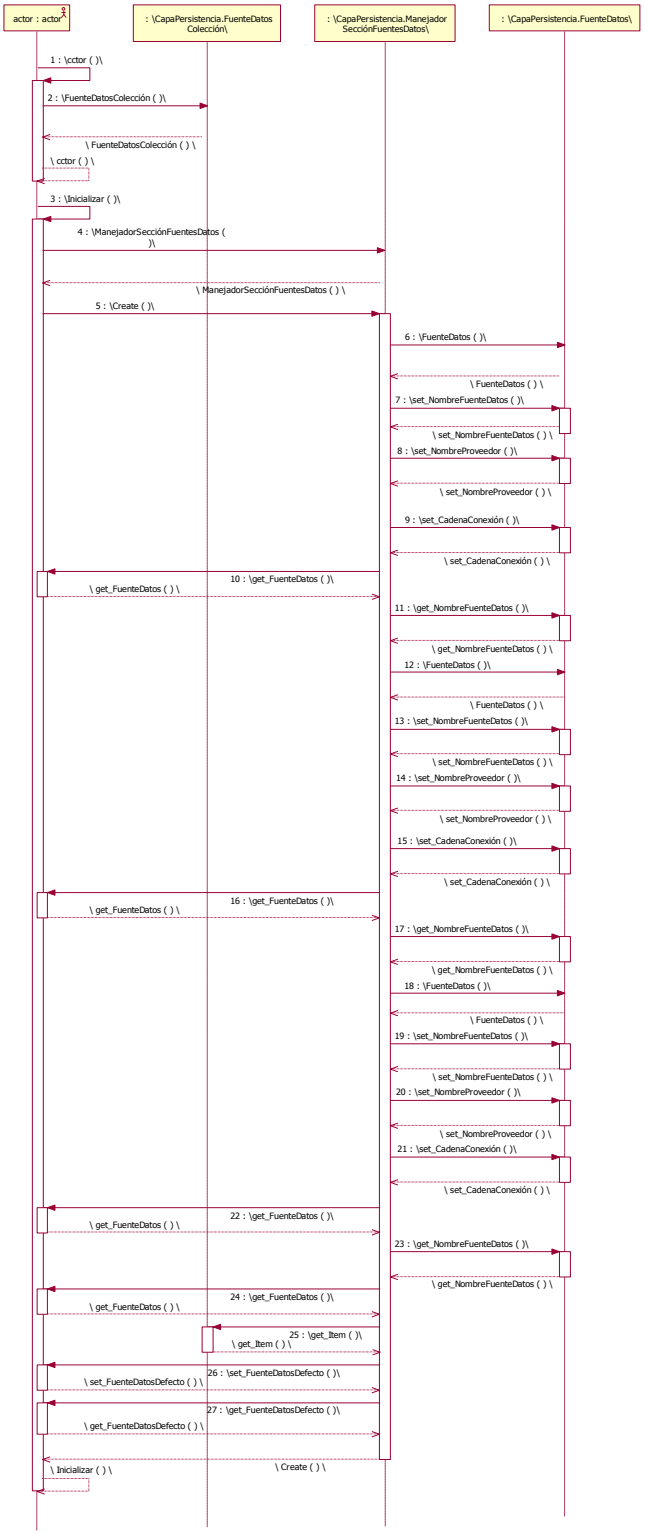


Figura 5.10: Diagrama de secuencia obtener fuente de datos configuradas.

La figura muestra como se obtiene una fuente de datos configurada.

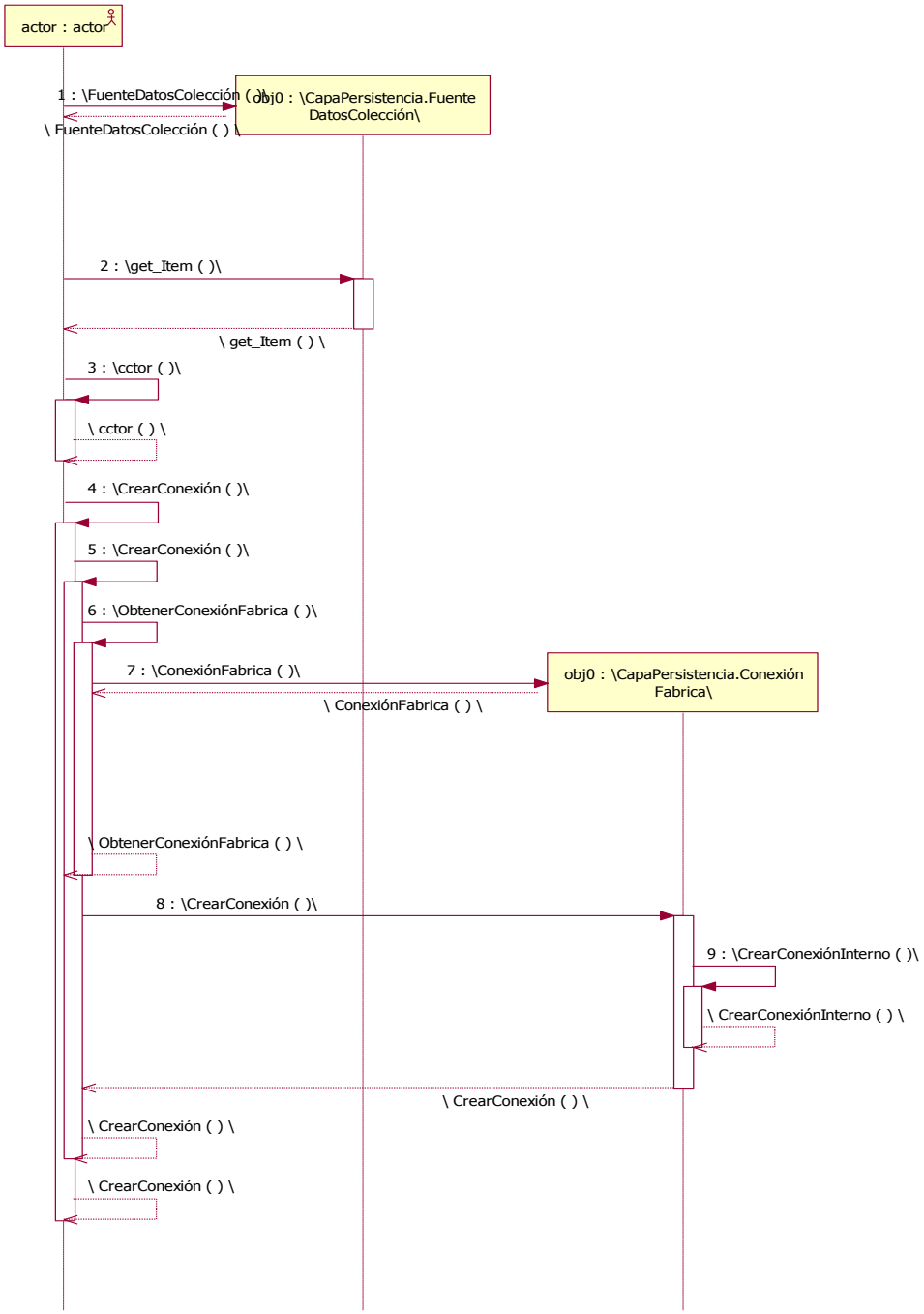


Figura 5.11: Diagrama de secuencia crear conexión fuente datos

Crear un objeto conexión con una fuente de datos por omisión.

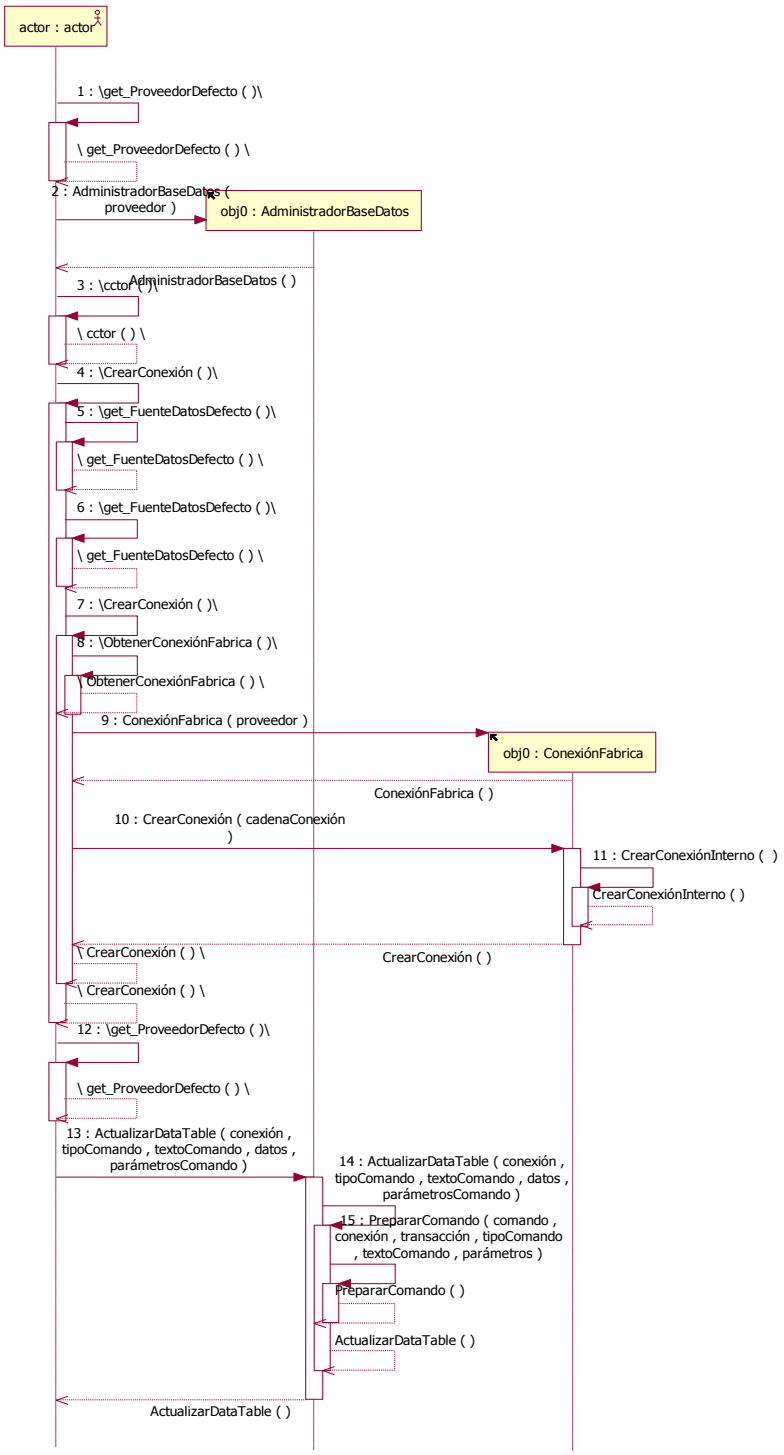


Figura 5.12: Diagrama de secuencia realizar operaciones CRUD.

La figura muestra como se realizan la operaciones CRUD.

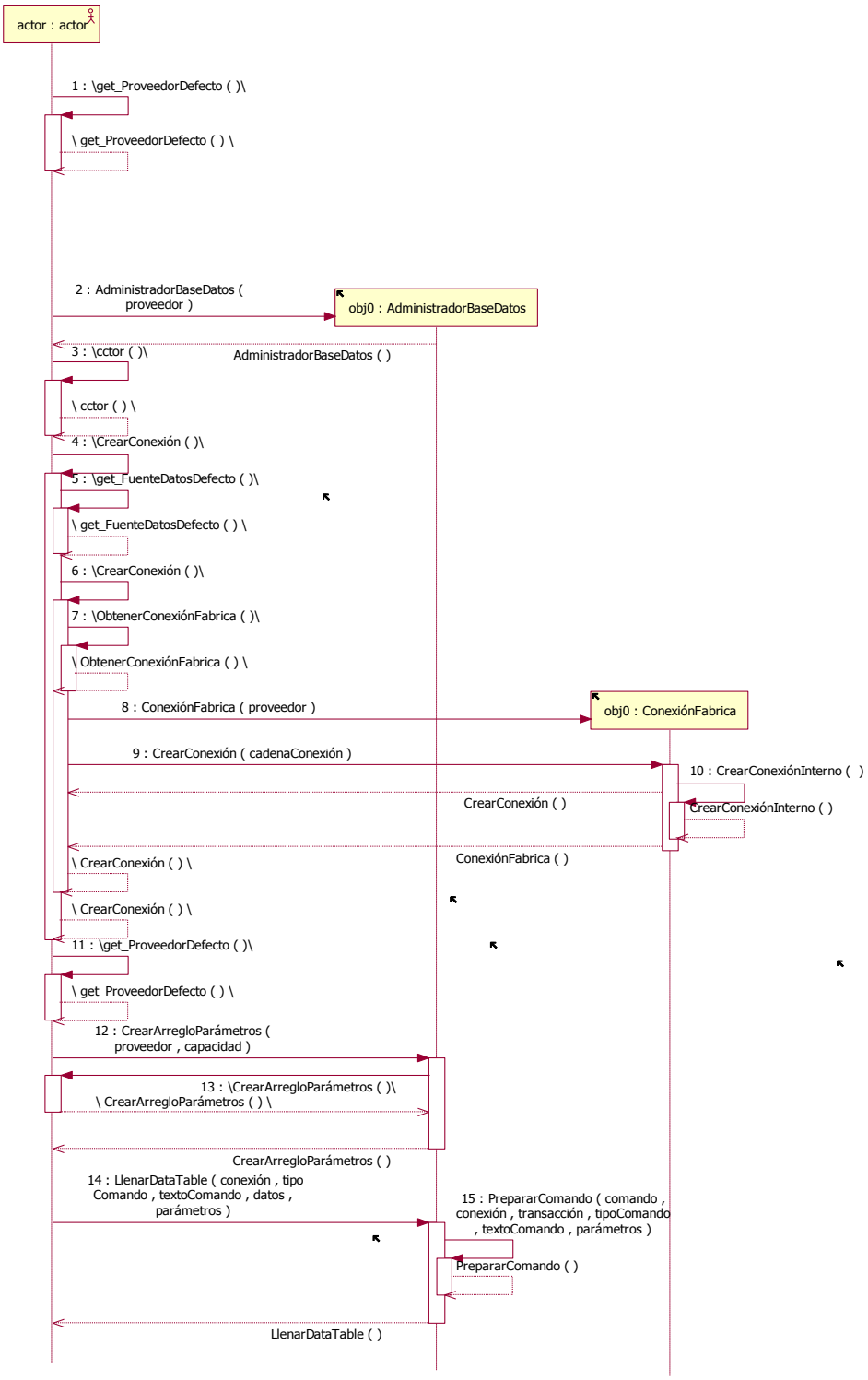


Figura 5.13: Diagrama de secuencia listar.

La figura muestra como retornar un grupo de registros.

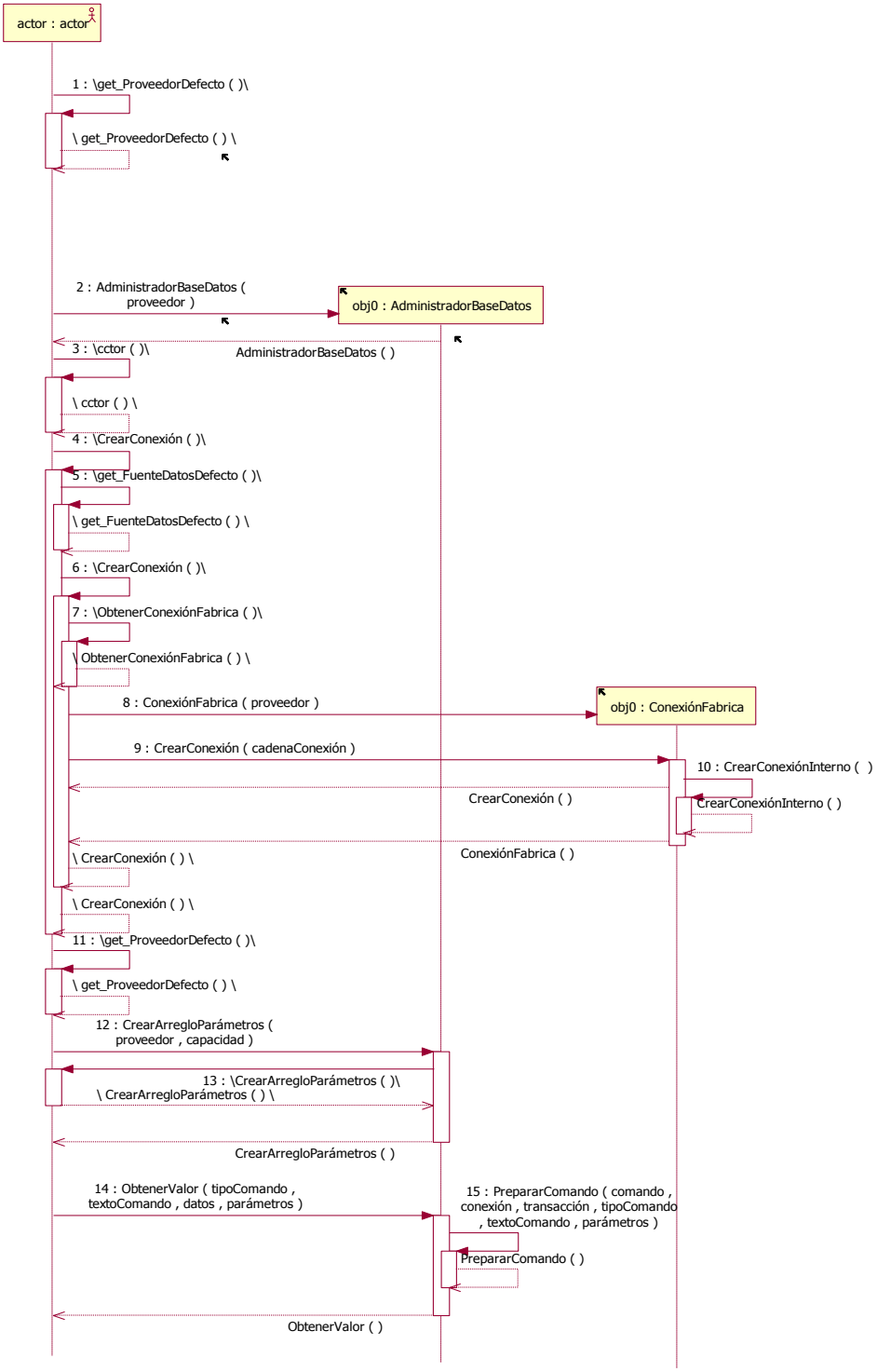


Figura 5.14: Diagrama de secuencia obtener valor único.

La figura muestra como se obtiene un solo valor ó escalar

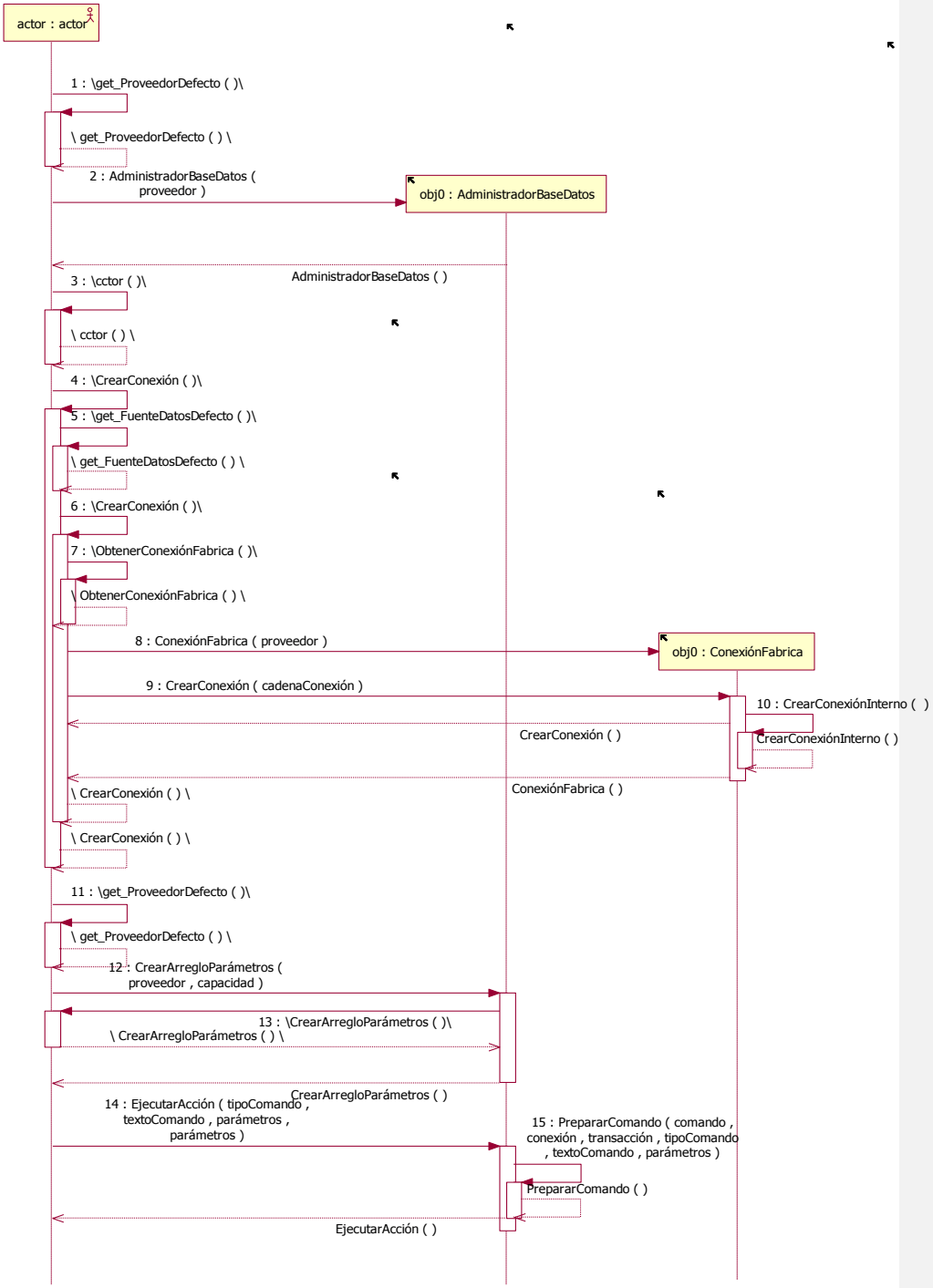


Figura 5.15: Diagrama de secuencia ejecutar mandato SQL.

La figura muestra como como se ejecucta una sentencia SQL.

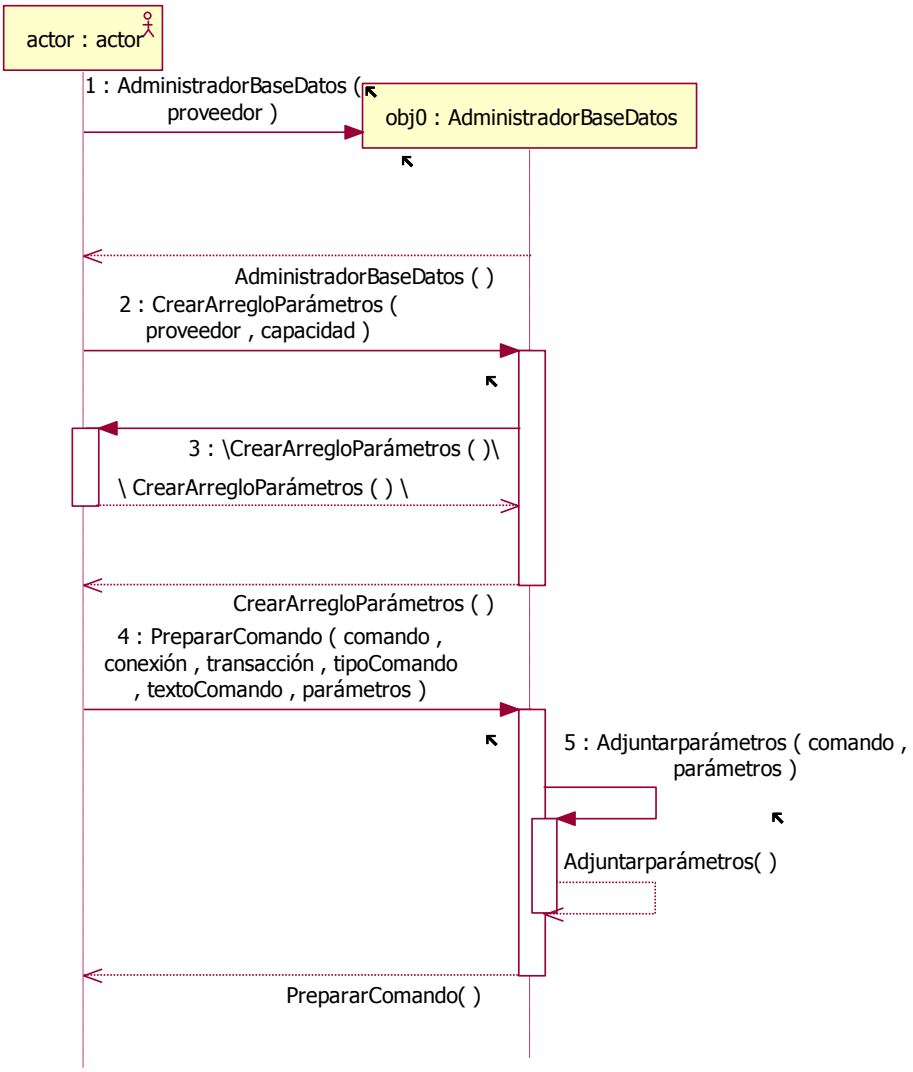


Figura 5.16: Diagrama de secuencia preparar comando.

La figura muestra como se configura un objeto comando.

~~Dinámico~~

~~7.2.4~~5.2.4 Modelo ~~Dinámico~~funcional

a. Diagramas de actividad propuesto por UML

1. Diagramas de actividad propuesto por UML.
    a. Iniciar capa de persistencia.
    b. Crear conexión.
    c. Crear DataAdapter.

Iniciar capa de persistencia

Describe las actividades que se realizan para que la capa de persistencia se configure correctamente para utilizarla dentro de una aplicación.

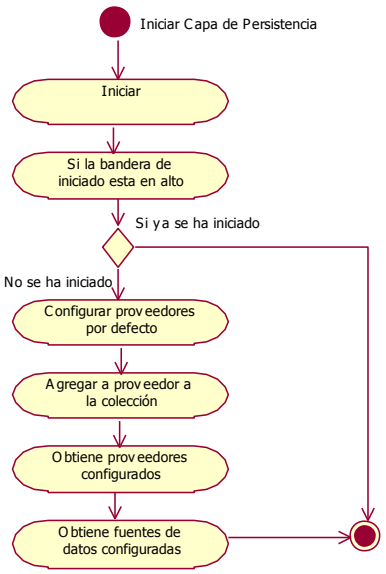


Figura 5.17~~4.7~~: Diagrama de actividades iniciar capa de persistencia.

Crear conexión

Describe las actividades que se realizan para crear un objeto de tipo dbconnection a un mecanismo de persistencia definido. Verifica si todos los parámetros necesarios son correctos para crear el objeto, si no son correctos emite un mensaje indicando que se deben configurar correctamente los parámetros necesarios.

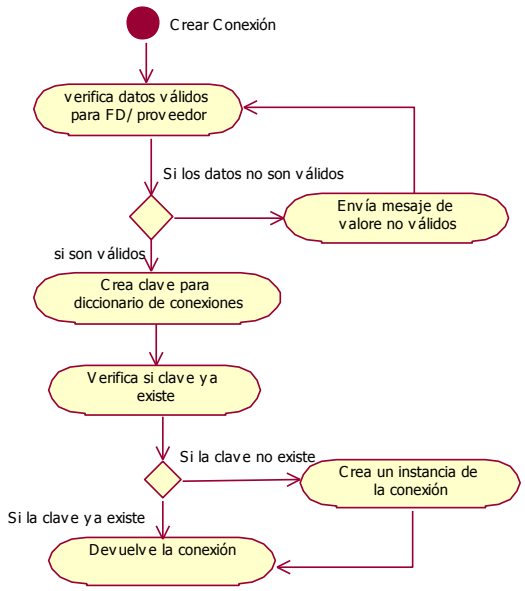


Figura 5.184.8: Diagrama de actividades crear conexión.

Crear DataAdapter

Describe las actividades que se realizan para crear un objeto de tipo DataAdapter. Verifica si todos los parámetros necesarios son correctos para crear el objeto, si no son correctos emite un mensaje indicando que se deben configurar correctamente los parámetros necesarios.

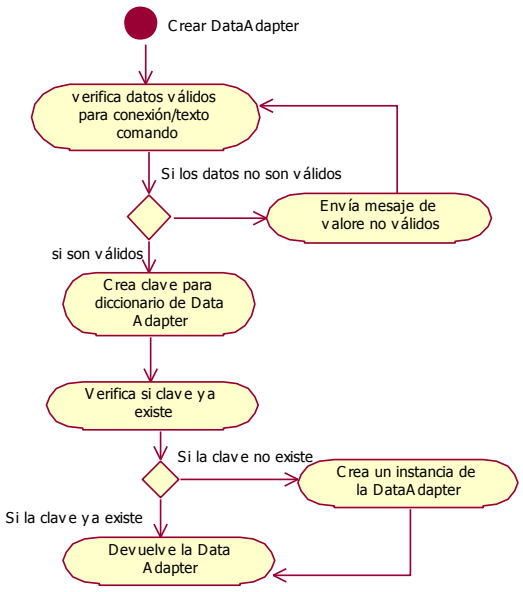


Figura 5.194.9: Diagrama de actividades DataAdapter.

~~7.3~~5.3 Fase de diseño del sistema

Con formato: Numeración y viñetas

La fase de diseño selecciona la estructura de alto nivel del sistema. Los requisitos estructurales son las cualidades y capacidades elegidas como referente para la construcción de la funcionalidad[38]. En esta sección se describirán las características estructurales que debe satisfacer un servicio de persistencia, se planteará la arquitectura y se mostrará el diagrama de componentes propuesto por UML.

A continuación se menciona tanto las características estructurales, como las funciones que debe tener una capa de persistencia, para una descripción detallada se encuentra en la sección 2.9 de esta tesis.

~~7.3.1~~5.3.1 Características estructurales

- Simplicidad y sencillez
- Modularidad
- Encapsulación
- Soportar diferentes arquitecturas
- Extensibilidad
- Facilidad de uso
- Escalabilidad y rendimientos

~~7.3.2~~5.3.2 Características funcionales

- Realizar operaciones CRUD sobre objetos
- Correspondencia clases – datos

38 Tomado de: BERTRAND MEYER, Construcción de software orientado a objetos, Prentice Hall,1997,Pag 55.

- Ortogonalidad
- Concurrencia
- Transacciones
- Integración de diferentes tipos de mecanismos de persistencia
- Conexiones múltiples
- Integridad

~~7.3.3~~5.3.3 Identificación de la concurrencia

Un objeto persistente tiene dos aspectos: uno durante la ejecución y otro en su estado almacenado. Como consecuencia de esta dualidad, la gestión del acceso concurrente a objetos debe cubrir ambos aspectos, el objeto como entidad en ejecución y su estado almacenado en los servicios de datos.

De otro modo tendríamos situaciones inconsistentes. Por ejemplo, los movimientos de una cuenta suman un saldo, pero la cuenta muestra otro distinto.

Visual Studio .Net aporta mecanismos para la sincronización a distintos niveles: de clase, de objeto, de método y bloque de sentencias, con cláusulas primitivas que controlan el acceso concurrente

~~7.3.4~~5.3.4 Arquitectura

Las arquitecturas se clasifican de manera general en arquitecturas monolíticas, cliente / servidor en dos o más capas y las distribuidas. Arquitecturas diferentes, con requerimientos de persistencia distintos, que conducen a que es necesario soportar diferentes arquitecturas. Debido a esto el diseño de servicios de persistencia deben seguir ciertas especificaciones que garanticen la

integración con diferentes arquitecturas aunque implique un mayor coste de desarrollo.

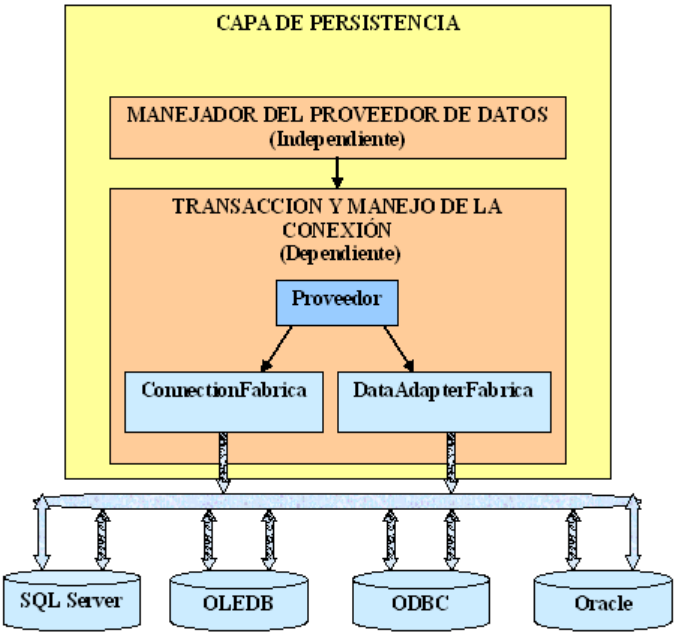


Figura 5.20~~4.13~~ Esquema de la arquitectura de acceso a datos.

Se separa en dos conjuntos de clases y métodos; el primero proporciona la interfaz de programación pública y es independiente de los proveedores de datos, el segundo es el que crea las clases dependientes de los proveedores de datos y se encargan de manejar los detalles finales del acceso a datos.

La capa de persistencia involucra un conjunto de archivos dll y utilitarios que ayudan a una aplicación a realizar operaciones CRUDs (Create, Retrieve, Update, Delete en inglés) persistentes de sus objetos desde y hacia una fuente de datos.

Implementa funcionalidad para el uso de proveedores de datos que soporten SQLServer, Oracle, OLEDB y ODBC.

a. Diagrama de componentes propuesto por UML

~~1. Componentes~~

~~a. Capa de Persistencia~~

~~b. Test~~

~~**Componentes**~~

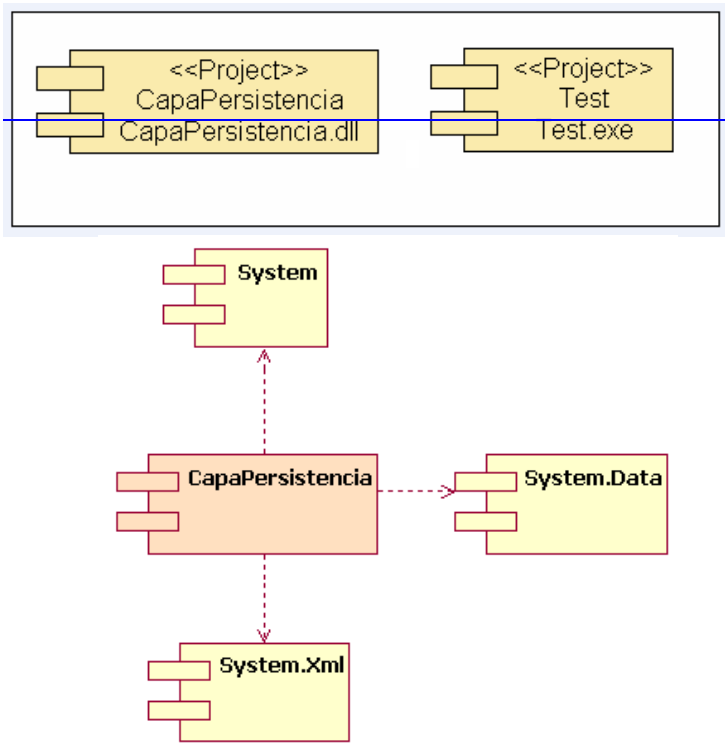


Figura 5.21: Diagrama de componentes motor de persistencia para .Net.

La figura muestra como la capa de persistencia interactúa con un conjunto de archivos dll y utilitarios que ayudan a una aplicación a realizar operaciones CRUDs (Create, Retrieve, Update, Delete en inglés) persistentes de sus objetos desde y hacia una fuente de datos.

~~7.4~~5.4 Fase de diseño de objetos

Con formato: Numeración y viñetas

En esta fase se van a desarrollar los métodos y propiedades de las clases definidas en la fase de análisis.

En la figura 5.22 se muestra el modelo de objetos, incluyendo el diseño de los métodos y propiedades de cada clase.

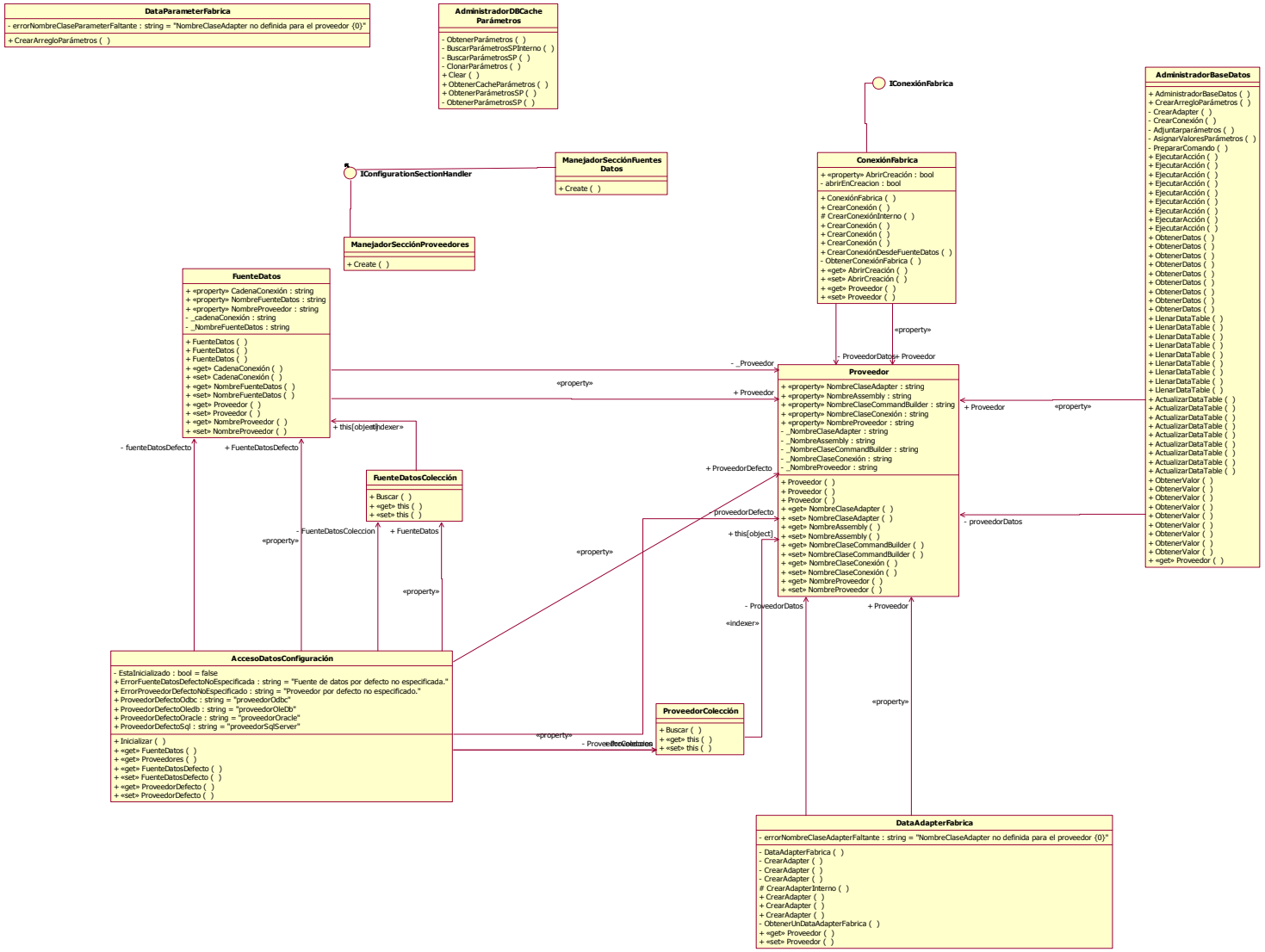

DataParameterFabrica
- errorNombreClaseParameterFaltante : string = "NombreClaseAdapter no definida para el proveedor {0}"
+ CrearArregloParámetros ( )
AdministradorDBCache Parámetros
- ObtenerParámetros ( )
- BuscarParámetrosSPInterno ( )
- BuscarParámetrosSP ( )
- ClonarParámetros ( )
+ Clear ( )
+ ObtenerCacheParámetros ( )
+ ObtenerParámetrosSP ( )
- ObtenerParámetrosSP ( )
IConexiónFabrica
AdministradorBaseDatos
+ AdministradorBaseDatos ( )
+ CrearArregloParámetros ( )
- CrearAdapter ( )
- CrearConexión ( )
- Adjuntarparámetros ( )
- AsignarValoresParámetros ( )
- PrepararComando ( )
+ EjecutarAcción ( )
+ EjecutarAcción ( )
+ EjecutarAcción ( )
+ EjecutarAcción ( )
+ EjecutarAcción ( )
+ EjecutarAcción ( )
+ EjecutarAcción ( )
+ EjecutarAcción ( )
+ EjecutarAcción ( )
+ ObtenerDatos ( )
+ ObtenerDatos ( )
+ ObtenerDatos ( )
+ ObtenerDatos ( )
+ ObtenerDatos ( )
+ ObtenerDatos ( )
+ ObtenerDatos ( )
+ ObtenerDatos ( )
+ ObtenerDatos ( )
+ LlenarDataTable ( )
+ LlenarDataTable ( )
+ LlenarDataTable ( )
+ LlenarDataTable ( )
+ LlenarDataTable ( )
+ LlenarDataTable ( )
+ LlenarDataTable ( )
+ LlenarDataTable ( )
+ ActualizarDataTable ( )
+ ActualizarDataTable ( )
+ ActualizarDataTable ( )
+ ActualizarDataTable ( )
+ ActualizarDataTable ( )
+ ActualizarDataTable ( )
+ ActualizarDataTable ( )
+ ActualizarDataTable ( )
+ ObtenerValor ( )
+ ObtenerValor ( )
+ ObtenerValor ( )
+ ObtenerValor ( )
+ ObtenerValor ( )
+ ObtenerValor ( )
+ ObtenerValor ( )
+ ObtenerValor ( )
+ ObtenerValor ( )
+ «get» Proveedor ( )
ManejadorSecciónFuentes Datos
+ Create ( )
IConfigurationSectionHandler
ManejadorSecciónProveedores
+ Create ( )
ConexiónFabrica
+ «property» AbrirCreación : bool
- abrirEnCreacion : bool
+ ConexiónFabrica ( )
+ CrearConexión ( )
# CrearConexiónInterno ( )
+ CrearConexión ( )
+ CrearConexión ( )
+ CrearConexión ( )
+ CrearConexiónDesdeFuenteDatos ( )
- ObtenerConexiónFabrica ( )
+ «get» AbrirCreación ( )
+ «set» AbrirCreación ( )
+ «get» Proveedor ( )
+ «set» Proveedor ( )
FuenteDatos
+ «property» CadenaConexión : string
+ «property» NombreFuenteDatos : string
+ «property» NombreProveedor : string
- _cadenaConexión : string
- _NombreFuenteDatos : string
+ FuenteDatos ( )
+ FuenteDatos ( )
+ FuenteDatos ( )
+ «get» CadenaConexión ( )
+ «set» CadenaConexión ( )
+ «get» NombreFuenteDatos ( )
+ «set» NombreFuenteDatos ( )
+ «get» Proveedor ( )
+ «set» Proveedor ( )
+ «get» NombreProveedor ( )
+ «set» NombreProveedor ( )
Proveedor
+ «property» NombreClaseAdapter : string
+ «property» NombreAssembly : string
+ «property» NombreClaseCommandBuilder : string
+ «property» NombreClaseConexión : string
+ «property» NombreProveedor : string
- _NombreClaseAdapter : string
- _NombreAssembly : string
- _NombreClaseCommandBuilder : string
- _NombreClaseConexión : string
- _NombreProveedor : string
+ Proveedor ( )
+ Proveedor ( )
+ Proveedor ( )
+ «get» NombreClaseAdapter ( )
+ «set» NombreClaseAdapter ( )
+ «get» NombreAssembly ( )
+ «set» NombreAssembly ( )
+ «get» NombreClaseCommandBuilder ( )
+ «set» NombreClaseCommandBuilder ( )
+ «get» NombreClaseConexión ( )
+ «set» NombreClaseConexión ( )
+ «get» NombreProveedor ( )
+ «set» NombreProveedor ( )
FuenteDatosColección
+ Buscar ( )
+ «get» this ( )
+ «set» this ( )
AccesoDatosConfiguración
- EstaInicializado : bool = false
+ ErrorFuenteDatosDefectoNoEspecificada : string = "Fuente de datos por defecto no especificada."
+ ErrorProveedorDefectoNoEspecificado : string = "Proveedor por defecto no especificado."
+ ProveedorDefectoOdbc : string = "proveedorOdbc"
+ ProveedorDefectoOledb : string = "proveedorOleDb"
+ ProveedorDefectoOracle : string = "proveedorOracle"
+ ProveedorDefectoSql : string = "proveedorSqlServer"
+ Inicializar ( )
+ «get» FuenteDatos ( )
+ «get» Proveedores ( )
+ «get» FuenteDatosDefecto ( )
+ «set» FuenteDatosDefecto ( )
+ «get» ProveedorDefecto ( )
+ «set» ProveedorDefecto ( )
ProveedorColección
+ Buscar ( )
+ «get» this ( )
+ «set» this ( )
DataAdapterFabrica
- errorNombreClaseAdapterFaltante : string = "NombreClaseAdapter no definida para el proveedor {0}"
- DataAdapterFabrica ( )
- CrearAdapter ( )
- CrearAdapter ( )
- CrearAdapter ( )
CrearAdapterInterno ( )
+ CrearAdapter ( )
+ CrearAdapter ( )
+ CrearAdapter ( )
- ObtenerUnDataAdapterFabrica ( )
+ «get» Proveedor ( )
+ «set» Proveedor ( )
«property»
- _Proveedor
- ProveedorDatos
+ Proveedor
+ this[object]
«indexer»
- fuenteDatosDefecto
+ FuenteDatosDefecto
- FuenteDatosColeccion
+ FuenteDatos
+ ProveedorDefecto
- proveedorDatos

Figura 5.22: Diagrama de clases detallado del motor de persistencia para .Net

~~7.5~~5.5 Fase de implementación

Con formato: Numeración y viñetas

La fase de implementación constituye la elección de una herramienta de programación que permita el manejo y administración de un motor de base de datos. La misma debe proporcionar en una interfaz amigable la información necesaria para cumplir con los objetivos propuestos.

~~7.5.1~~5.5.1 Nomenclatura de programación[39]

Con formato: Numeración y viñetas

Estilos de mayúsculas

Tabla 5.1: Tabla de convenciones de mayúsculas para nomenclatura.

Tipo Mayúsculas	Descripción	Ejemplo
Mayúsculas y minúsculas Pascal	La primera letra de las palabras concatenadas en mayúsculas	BackColor
Mayúsculas y minúsculas Camel	La primera letra en minúscula y la primera letra de las siguientes palabras en mayúscula	backColor
Mayúsculas	Todas las letras del identificador van en mayúsculas.	System.IO System.Web.UI

Para evitar confusiones y garantizar la interoperación entre lenguajes, siga las siguientes recomendaciones:

- No utilice nombres que requieran distinción entre mayúsculas y minúsculas.
- No utilice abreviaturas ni contracciones.
- Solo utilice acrónimos que estén aceptados en el campo de la informática. Cuando sea necesario, utilice acrónimos conocidos para reemplazar nombres largos. Ej: UI, para interfaz de usuario.

39 Tomado de: www.microsoft.com/spanish/msdn/arquitectura/DesignGuidelines/Guidelines2.asp

- No utilice palabras reservadas o nombres utilizados habitualmente en .NET. Framework, como System, Collections, Forms o UI.

Instrucciones de nomenclatura

Son reglas generales de nomenclatura:

- Utilice el estilo de mayúsculas y minúsculas Pascal.
- Utilice las abreviaturas con moderación.
- No utilice el caracter de subrayado (_).

Espacios de nombres

- En los espacios de nombres se utiliza el nombre de la compañía seguido del nombre de la tecnología y, opcionalmente, la característica y el diseño como se muestra a continuación. Ej: Microsoft.Media.Design.
- Separar los componentes lógicos con puntos, Ej.: Microsoft.Office.PowerPoint.
- No utilice el mismo nombre para un espacio de nombres y para una clase, ni tampoco para un espacio de nombres y para un ensamblado.

Nomenclatura de clases

- No utilice un prefijo de tipo, como C para clase, en un nombre de clase.
- Utilice un sustantivo o un sintagma nominal (**CustomAttributeProvider**) para dar nombre a una clase.
- No utilice el caracter de subrayado (_).
- Es correcto que nombre de clase que comience con la letra I, aunque la clase no sea una interfaz.

- Cuando sea apropiado, utilice una palabra compuesta en el nombre de una clase derivada. La segunda parte del nombre de la clase derivada debe ser el nombre de la clase base. Ej.: ApplicationException

Nomenclatura de interfaces

- Asigne nombres a interfaces utilizando sustantivos, sintagmas nominales o adjetivos que describan su comportamiento. Ej:
  Interfaz IComponent se utiliza un sustantivo descriptivo.
  ICustomAttributeProvider se utiliza un sintagma nominal.
  IPersistable se utiliza un adjetivo.
- Incluya un prefijo con la letra I en los nombres de interfaces para indicar que el tipo es una interfaz.
- Utilice nombres similares cuando defina un par clase / interfaz, donde la clase es una implementación estándar de la interfaz. Los nombres deben ser distintos sólo en el prefijo I del nombre de la interfaz.

Nomenclatura de atributos

Deberá agregar siempre el sufijo **Attribute** a las clases de atributos personalizados Ej.: Public Class ObsoleteAttribute.

Nomenclatura de campos estáticos

- Utilice sustantivos, sintagmas nominales o abreviaturas de nombres al asignar nombres a campos estáticos.
- Se recomienda utilizar propiedades estáticas en lugar de campos estáticos públicos cada vez que sea posible.

~~1~~ CAPÍTULO VI

~~86~~ ANÁLISIS Y DISEÑO PROTOTIPO DE UNA APLICACIÓN DE FACTURACIÓN

~~8.1~~6.1 Fase de análisis

~~8.1.1~~6.1.1 Definición del problema

El prototipo de una aplicación de facturación se va a utilizar para demostrar la funcionalidad del motor de persistencia para .Net desarrollado en esta tesis.

El prototipo debe implementar las siguientes opciones básicas de un sistema real de facturación y venta de productos:

- Formulario simple de mantenimiento de datos Ej: Mantenimiento de Artículos.
- Formulario maestro detalle de mantenimiento de datos Ej: Registro de Factura y Detalle de Factura.
- Formulario recursivo de mantenimiento de datos Ej: Mantenimiento de Ubicación.
- Formulario de presentación de reportes Ej: Reporte periódico de ventas por producto.
- Asignación de privilegios y seguridades por perfil.

~~8.1.2~~6.1.2 Modelo de objetos

Con formato: Numeración y viñetas

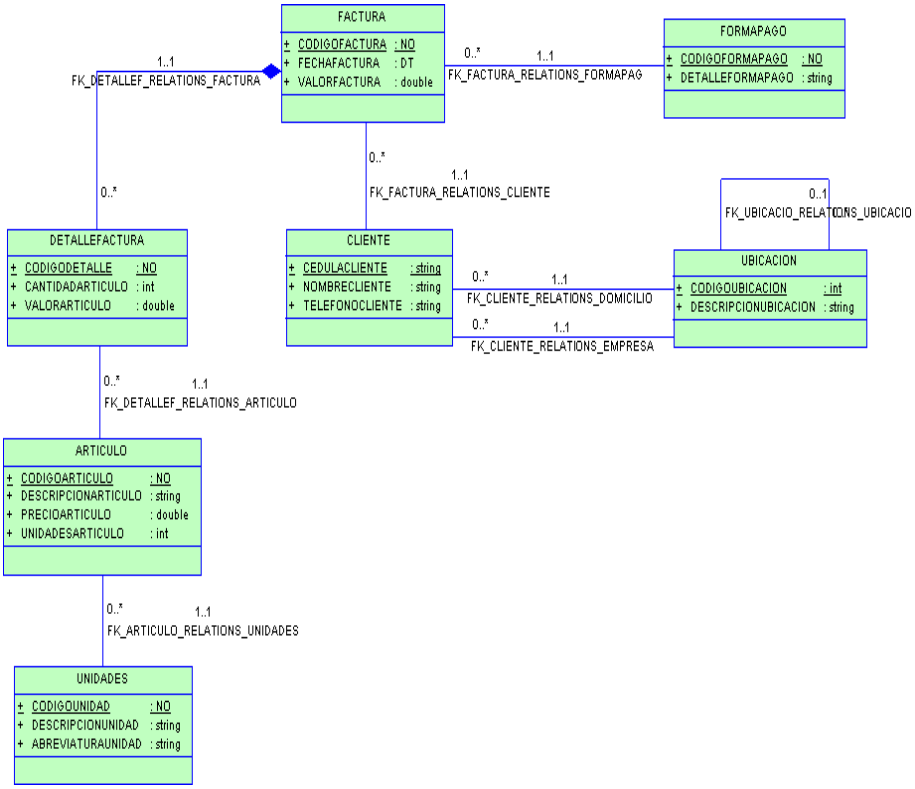


Figura 6.1:~~4.14~~ Diagrama de objetos Ebuy.

La figura muestra el modelo de objetos y sus relaciones propuesto para la implementación del prototipo de una aplicación de facturación.

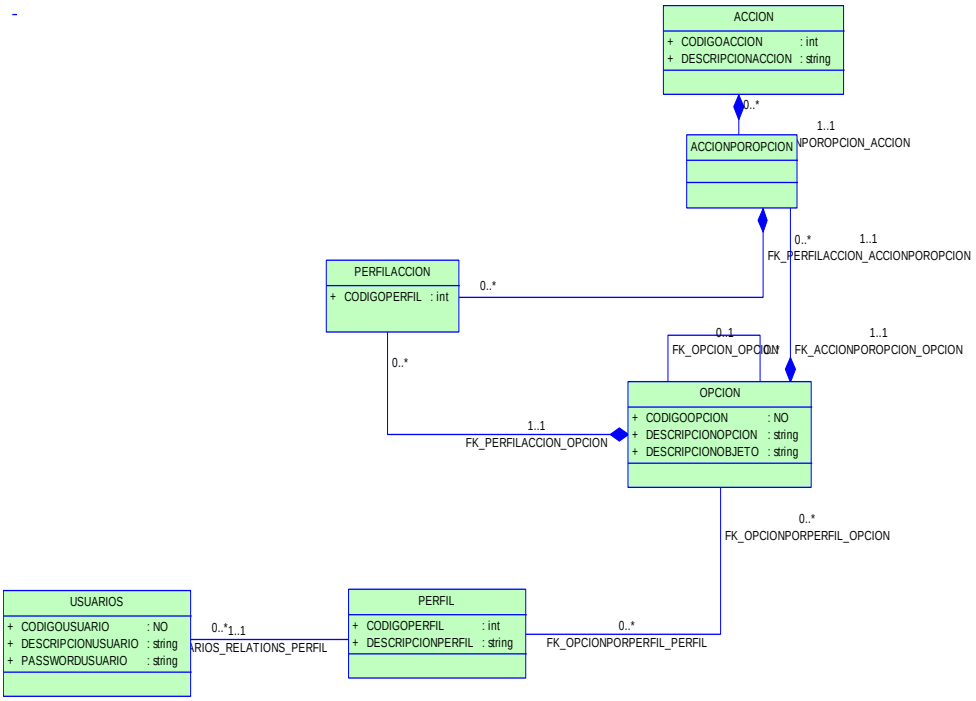


Figura 6.2~~4.15~~: Diagrama de objetos seguridades.

La figura muestra el modelo de objetos y sus relaciones propuesto para la implementación de seguridades de una aplicación en general.

~~c.~~a. Diagrama de casos de uso

Con formato: Numeración y viñetas

A continuación se detallan los casos de uso con los actores respectivos.

Administrador~~:~~

Tiene acceso a todas las opciones de menú y se encarga de administrar la base de datos, de realizar mantenimiento de las entidades de negocio y de asignar permisos de acceso a los usuarios según la jerarquía establecida en el modulo de seguridades.

Usuario operativo~~:~~

Se encarga de la facturación de productos y tiene además permiso para el registro de nuevos clientes.

Usuario consulta~~:~~

Tiene acceso únicamente a los reportes del sistema.

Casos de Uso:

1. Diagrama de contexto.
2. Mantenimiento de entidades.
3. Emitir factura.
4. Administrar seguridades.
5. Obtener reportes.

Diagrama de contexto del prototipo de una aplicación de facturación

~~Diagrama de contexto del Prototipo de una Aplicación de Facturación~~

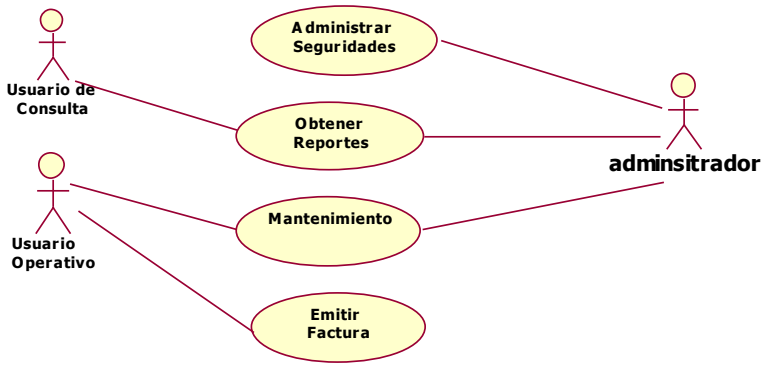


Figura 6.3~~4.16~~: Diagrama de contexto.

~~**CASO DE USO:**~~

Mantenimiento cliente

~~Descripción:~~ Permite registrar los datos de un cliente, para poder emitirle una factura por la compra de productos.

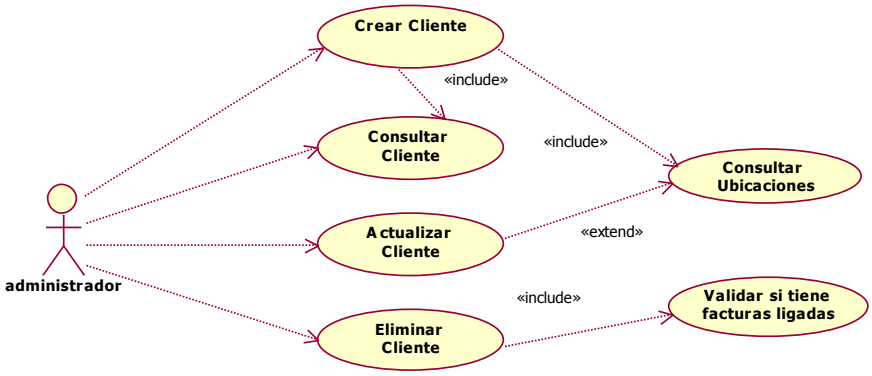


Figura 6.4~~4.17~~ Caso de uso mantenimiento cliente.

CASO DE USO: Mantenimiento cliente	
Eventos principales	4) Indica la acción que desea realizar: crear nuevo, modificar, eliminar, salir. 5) Si el usuario escoge las acciones: ▪ **Crear nuevo**: Se ejecuta los pasos: 1, 2, 3, 4, 6. ▪ **Modificar**: Se ejecuta los pasos:2, 3, 4. ▪ **Eliminar**: Se ejecuta los pasos: 2, 3, 5. ▪ **Salir**: Termina el caso de uso. 6) Al escoger la opción de nuevo se presenta una línea vacía, para el ingreso de clientes, se presenta además la lista de ubicaciones geográficas disponibles. 7) El usuario ingresa una cédula. 8) El sistema despliega los datos existentes de esa cédula. 9) El usuario ingresa o modifica los demás datos del cliente. 10) El usuario elige la opción de eliminar al cliente. El sistema, valida el ingreso de datos obligatorios del cliente (cedula, nombre, ubicación del domicilio y de la empresa).
Eventos alternos	~~4)~~11) Si la cedula ya existe se despliega un mensaje (E - 1) Si los datos son correctos se graba la información.
Eventos de excepción	E-1: Si la cedula ya existe, se despliegan los datos.
Pre-condiciones	▪ Deben existir ubicaciones geográficas previamente. ▪ El operador debe haber completado su login.
Post-condiciones	▪ Se graba los datos del cliente y se puede facturar productos al cliente.

~~CASO DE USO:~~ Ingresar artículo

~~- Descripción:~~ Permite registrar los datos de un artículo, la presentación existentes y las unidades disponibles. El Caso de uso Mantenimiento Artículo aplica para las siguientes entidades: Unidad de Medida, forma de pago, ubicaciones, acciones, opciones, perfiles.

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

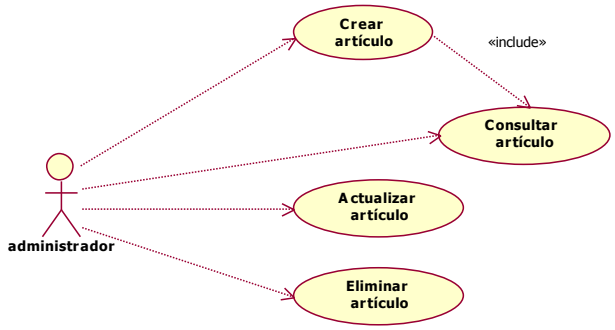


Figura 6.5~~4.18~~: Caso de uso mantenimiento artículo.

CASO DE USO: Mantenimiento artículo	
Eventos principales	1) Indica la acción que desea realizar: crear nuevo, modificar, eliminar, salir. 2) Si el usuario escoge las acciones. ▪ **Crear nuevo: Se ejecuta los pasos: 1, 2, 3, 5.** ▪ **Modificar: Se ejecuta los pasos:2, 3, 5.** ▪ **Eliminar: Se ejecuta los pasos: 2, 4, 6.** ▪ **Salir: Termina el caso de uso.** 3) Al escoger la opción de nuevo se presenta un registro vacío, para el ingreso de artículos, se presenta además la lista de unidades de medida disponibles. 4) El sistema despliega los datos existentes de artículos. 5) El usuario ingresa o modifica las unidades existentes, el precio de venta y los demás datos del artículo. 6) El usuario elige la opción de eliminar el artículo. El sistema valida que todos los datos obligatorios del artículos haya sido ingresados (nombre, unidad de medida, unidades disponibles, precio de venta).
Eventos alternos	7) Se valida que los datos sean válidos (E - 1). Si los datos son correctos se graba la información.
Eventos de excepción	E-1 Si los datos no son válidos se despliega un mensaje.
Pre-condiciones	▪ Deben existir unidades de medida previamente. ▪ El operador debe haber completado su login.
Post-condiciones	▪ Se graba los datos de artículos y se puede facturar.

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

~~CASO DE USO:~~ Administración de seguridades

~~Descripción:~~ Permite registrar los datos de un cliente, para poder emitirle una factura por la compra de productos.

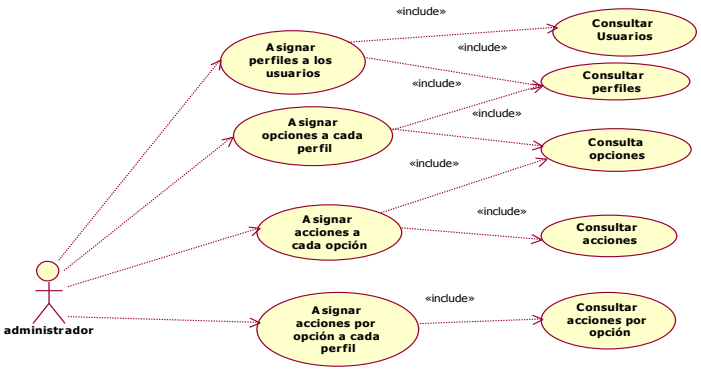


Figura 6.6~~4.19~~: Casos de uso administración de seguridades.

CASO DE USO: Administración de seguridades	
Eventos principales	1) Asignar perfiles a cada usuario. 2) Asignar acciones a cada opción. 3) Asignar opciones a cada perfil. 4) Asignar acciones por opción a cada perfil.
Eventos alternos	~~1)~~5)Se valida que los datos sean válidos (E - 1). Si los datos son correctos se graba la información.
Eventos de excepción	E-1 Si los datos no son válidos se despliega un mensaje.
Pre-condiciones	▪ Deben existir acciones, opciones, usuarios, perfiles. ▪ El operador debe haber completado su login.
Post-condiciones	Se graba los datos de permisos y se puede acceder a la aplicación.

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

~~CASO DE USO:~~ Emitir factura

~~Descripción:~~ Permite emitir una factura por compra de productos.

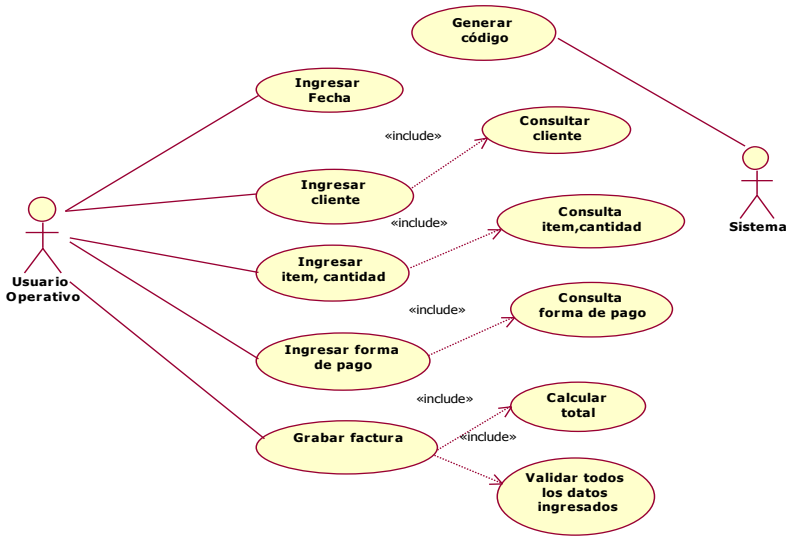


Figura 6.7~~4.20~~: Caso de uso emitir factura.

CASO DE USO: Emitir factura

~~Eventos Principales~~

~~1)Indica la acción que desea realizar: crear nuevo, consulta, salir~~

~~2)Si el usuario escoge las acciones~~

~~•Crear nuevo: Se ejecuta los pasos: 1,2,3,4,5,6~~

~~•Consulta: Se ejecuta los pasos:2,3~~

Con formato: Numeración y viñetas

	~~•**Salir: Termina el caso de uso**~~ ~~3)El sistema genera un número secuencial de factura, con la fecha actual y la lista de clientes, las formas de pago, los productos y las presentaciones existentes en la base.~~ ~~4)El usuario elige un cliente, un forma de pago~~ ~~5)El usuario comienza a escoger los productos que desea ingresar en la factura con la presentación y la cantidad que desea.~~ ~~6)El sistema comprueba cada entrada de productos para verificar la disponibilidad de la cantidad de unidades.~~ ~~El sistema valida que todos los datos haya sido ingresados y al menos un producto seleccionado (E1)~~
Eventos principales	1) Indica la acción que desea realizar: crear nuevo, salir. 2) Si el usuario escoge las acciones: ▪ **Crear nuevo: Se ejecuta los pasos: 2, 3, 4, 5, 6.** ▪ **Salir: Termina el caso de uso.** 3) El sistema genera un número secuencial de factura, con la fecha actual y la lista de clientes, las formas de pago, los productos y las presentaciones existentes en la base. 4) El usuario elige un cliente, una forma de pago. 5) El usuario comienza a escoger los productos que desea ingresar en la factura con la presentación y la cantidad que desea. 6) El sistema comprueba cada entrada de productos para verificar la disponibilidad de la cantidad de unidades. 7) El sistema valida que todos los datos hayan sido ingresados y al menos un producto seleccionado (E1).
Eventos alternos	~~1)~~8)Si no existen suficiente cantidad de unidades se despliega un mensaje (E - 2). ~~2)~~9)Si existen suficientes unidades el sistema deja seleccionar el producto al usuario. Si se han llenado los datos correctamente se genera la factura.
Eventos de excepción	E-1 Si el usuario no ha seleccionado los datos obligatorios (cliente, forma de pago y al menos un producto), se genera un mensaje de error. E-2 Si no existe la cantidad suficiente de unidades el usuario puede volver a digitar el valor.
Pre-condiciones	▪ El cliente debe estar existir en la base de datos. ▪ Deben existir formas de pago en la base de datos. ▪ Deben existir productos en la base de datos. ▪ El operador debe haber completado su login.
Post-condiciones	▪ Se graba la factura y se entrega al cliente.

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

b. Diccionario de Datos

Con formato: Numeración y viñetas

A continuación se detallan el listado de clases.

LISTADO DE CLASES

Entidades del Negocio

Dsarticulo.
Dscliente.
Dsdetallefactura.
Dsfactura.
Dsfacturas.
Dsformapago.
Dsubicacion.
Dsunidades.

Entidades de Seguridades

Dsaccion.
Dsaccionporopcion.
Dsopcion.
Dsopcionporperfil.
Dsperfil.
Dsperfilaccion.
Dsusuarios.
Dsvistaopcionporusuario.

Lógica del Negocio

Artículos.
Clientes.
Facturas.
FormasPago.
Inicializar.
Ubicaciones.

Unidades.

Servicios Web

ServicioArticulo.

ServicioClientes.

ServicioFacturas.

ServicioFormaPago.

ServicioUbicacion.

ServicioUnidades.

Entidades del Negocio

Dsarticulo

Objeto de tipo dataset que contiene los datos de los artículos.

Atributos de la clase	
Atributos	**Descripción**
CODIGOARTICULO	Código del artículo.
CODIGOUNIDAD	Código de las unidades.
DESCRIPCIONARTICULO	Descripción o nombre del artículo.
PRECIOARTICULO	Precio de venta de artículo.
UNIDADESARTICULO	Unidades disponibles.

Dscliente

Objeto de tipo dataset que contiene los datos de los clientes.

Atributos de la clase	
Atributos	**Descripción**
CEDULACLIENTE	Cédula del cliente.
CODIGOUBICACIONCLIENTE	Código de ubicación del domicilio.
CODIGOUBICACIONEMPRESA	Código de ubicación de la empresa.
NOMBRECLIENTE	Nombre del cliente.
TELEFONOCLIENTE	Teléfono domicilio del cliente.

Dsdetallefactura

Objeto de tipo dataset que contiene los datos del detalle de una factura.

Atributos de la clase	
Atributos	**Descripción**
CODIGOFACTURA	Código de la factura.
CODIGODETALLE	Código del número de ítem de una factura.
CODIGOARTICULO	Código de artículo vendido.
CANTIDADARTICULO	Cantidad del artículo vendido.
VALORARTICULO	Valor de venta del artículo.

Dsdetallefactura

Objeto de tipo dataset que contiene los datos del detalle de una factura.

Atributos de la clase	
Atributos	**Descripción**
CODIGOFACTURA	Código de la factura.
CODIGODETALLE	Código del número de ítem de una factura.
CODIGOARTICULO	Código de artículo vendido.
CANTIDADARTICULO	Cantidad del artículo vendido.
VALORARTICULO	Valor de venta del artículo.

Dsfactura

Objeto de tipo dataset que contiene los datos de una factura.

Atributos de la clase	
Atributos	**Descripción**
CODIGOFACTURA	Código de la factura.
CEDULACLIENTE	Cédula del cliente.

CODIGOFORMAPAGO	Código de la forma de pago.
FECHAFACTUA	Fecha de facturación.

Dsformapago

Objeto de tipo dataset que contiene los datos de forma de pago.

Los siguientes dataset tienen la misma estructura:

- Ubicaciones.
- Unidades.
- Acciones.
- Opciones.
- Perfiles.
- Usuarios.

Atributos de la clase	
Atributos	**Descripción**
CODIGOFORMAPAGO	Código de la forma de pago de la factura.
DESCRIPCIONFORMAPAGO	Descripción de la forma de pago de la factura.

Dsaccionporopcion

Objeto de tipo dataset que contiene los datos acciones por opción.

Atributos de la clase	
Atributos	**Descripción**
CODIGOOPCION	Código de la opción.
CODIGOACCION	Código de la acción.
SELECCIONADO	Bandera indicador de activado.

Dsopcionporperfil

Objeto de tipo dataset que contiene los datos opción por perfil.

Atributos de la clase	
Atributos	**Descripción**
CODIGOOPCION	Código de la opción.

DESCRIPCIONOPCION	Descripción de la opción.
CODIGOCONTENEDOR	Código del contenedor.
DESCRIPCIONOBJETO	Descripción del objeto Ej: Menú insertar.

Dsperfilaccion

Objeto de tipo dataset que contiene los datos perfil por acción.

Atributos de la clase	
Atributos	**Descripción**
CODIGOPERFIL	Código del perfil.
CODIGOACCION	Código de la acción.
CODIGOOPCION	Código de la opción.

Lógica del Negocio

Articulo

Clase que contiene los atributos de los artículos y contiene los métodos y operaciones para acceder a ellos.

Acceso: Público.

Clase Base: ServicedComponent.

Métodos de la clase	
Métodos	**Descripción**
ObtenerArticulos	Obtiene y retorna un dataset de artículos.
grabarArticulos (DSARTICULO)	Graba las modificaciones, eliminaciones, inserciones que se realizaron sobre el dataset de los artículos.

Aplica para las siguientes clases:

- Clientes.
- Formas de Pago.
- Unidades.

Facturas

Clase que contiene los atributos de facturas y contiene los métodos y operaciones para acceder a ellos.

Acceso: Público.

Clase Base: ServicedComponent.

Métodos de la clase	
Métodos	**Descripción**
ObtenerFactura	Obtiene y retorna un dataset de facturas según el código de factura enviada.
ObtenerFacturas	Obtiene y retorna un dataset de facturas.

Servicios Web

ServicioArticulo

Utiliza un componente de tipo artículo para exponer los métodos públicos en el web. Todos los servicios actúan de la misma forma que este componente

Acceso: Público.

Clase Base: WebService.

Métodos de la clase	
Métodos	**Descripción**
InitializeComponent	Inicializa el componente.
ObtenerArticulos	Utiliza el método ObtenerArticulos del componte.
GrabarArticulos	Utiliza el método GrabarArticulos del componte.

6.1.3 Modelo funcional

~~**b.Modelo Funcional**~~

~~**c.Diagramas de Actividades propuesto por UML**~~

a. **Diagramas de actividades propuesto por UML**

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

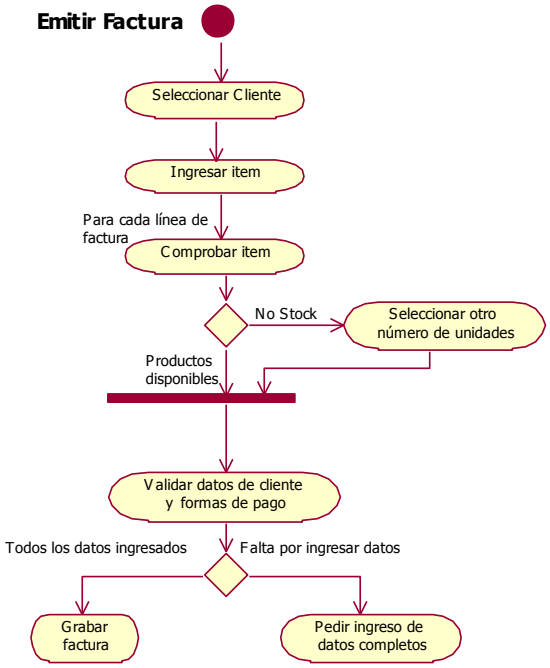


Figura 6.8~~4.21~~: Diagrama de actividad emitir factura.

Para emitir una factura, el usuario debe seleccionar un cliente e ingresar al menos un producto, con un número de unidades diferente de 0. El sistema debe validad que se escoja también la forma de pago.

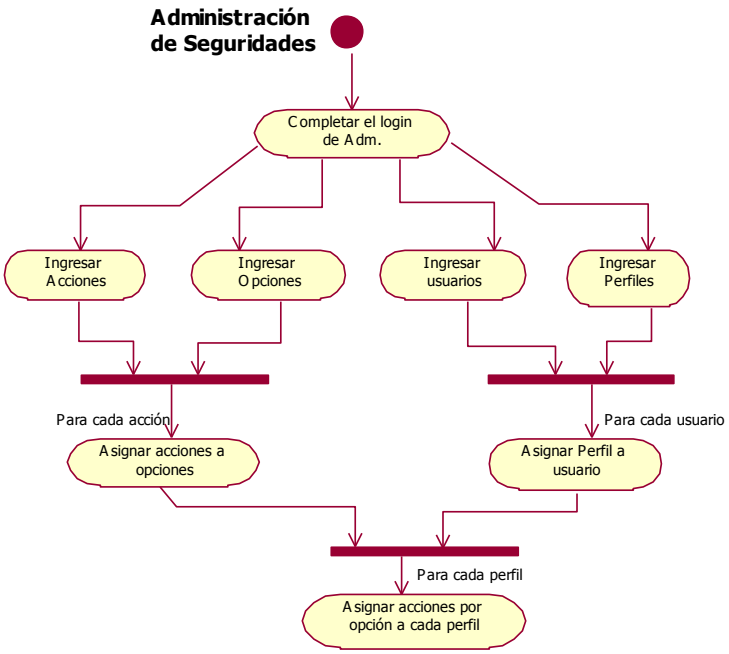


Figura 6.94.22: Diagrama de actividades administrar seguridades.

La administración de seguridades, permite ingresar acciones permitidas sobre las opciones del menú, dependiendo del perfil al que pertenezca un usuario.

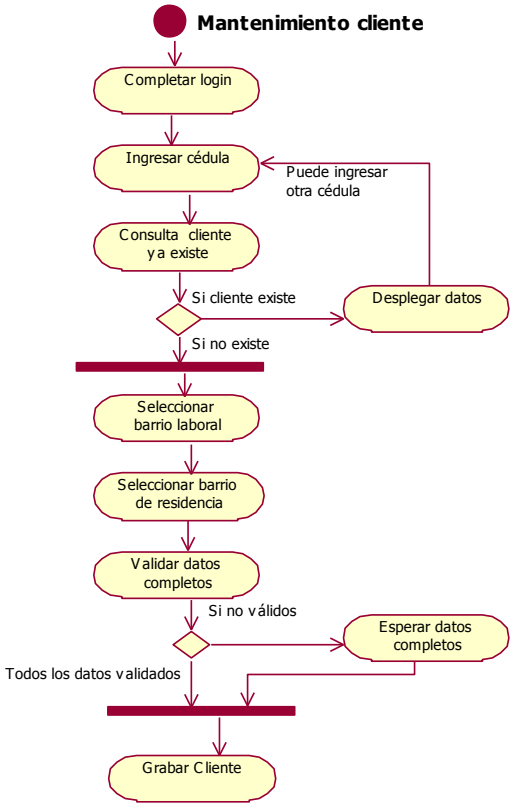


Figura 6.104~~.23~~: Diagrama de actividades mantenimiento clientes.

El mantenimiento de una clase permite realizar las acciones básicas: inserción, modificación y eliminación, para cada clase se valida los datos obligatorios y el tipo de dato que se ingresa, en el caso particular de clientes se valida que se escoja la ubicación laboral y la ubicación de residencia.

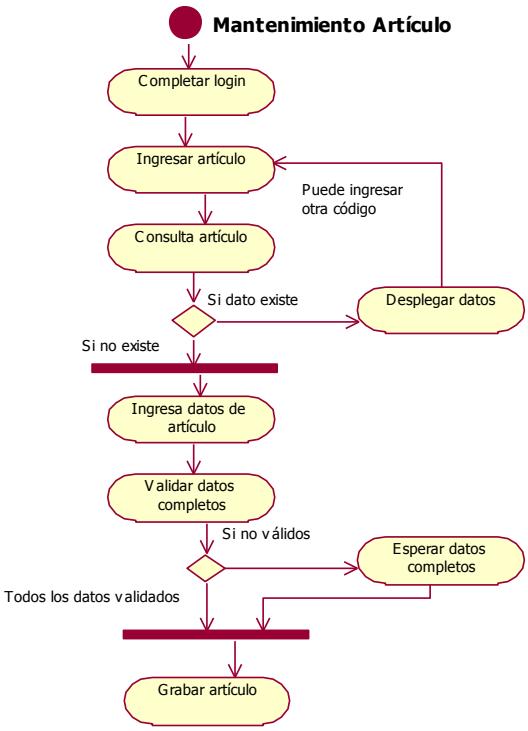


Figura 6.114.24: Diagrama actividades mantenimiento artículo.

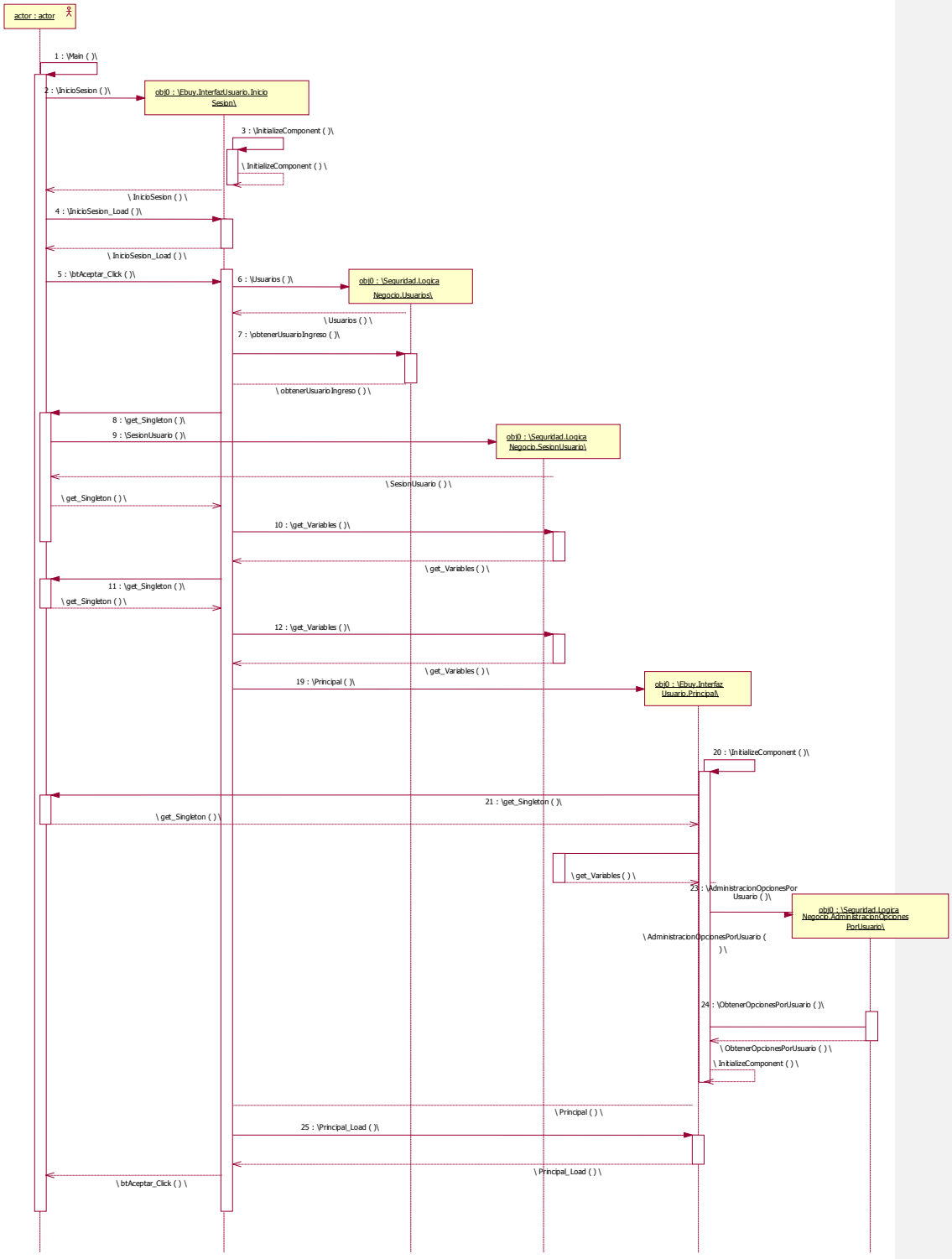
La figura muestra el conjunto de actividades que se debe realizar par realizar el mantenimiento de una clase. Este diagrama aplica para todas las clases simples del prototipo.

Con formato: Numeración y viñetas

~~8.1.3~~6.1.4 Modelo dinámico

a. Diagramas de secuencia propuesto por UML

1. Inicio de sesión.
2. Mantenimiento de perfiles (aplica para todos mantenimientos, clases simples).
4. ~~3.~~ Servicio artículos (aplica para todos servicios). **Con formato:** Numeración y viñetas
4. Mantenimiento opciones por perfil (aplica para todos mantenimientos clases maestro detalle).
5. ~~3.~~ Insertar factura. **Con formato:** Numeración y viñetas
6. Grabar factura.
7. Reporte factura



~~7.~~Figura 6.12: Diagrama de secuencia inicio de sesión.

Con formato: Numeración y viñetas

La figura muestra como se inicializa la capa de persistencia.

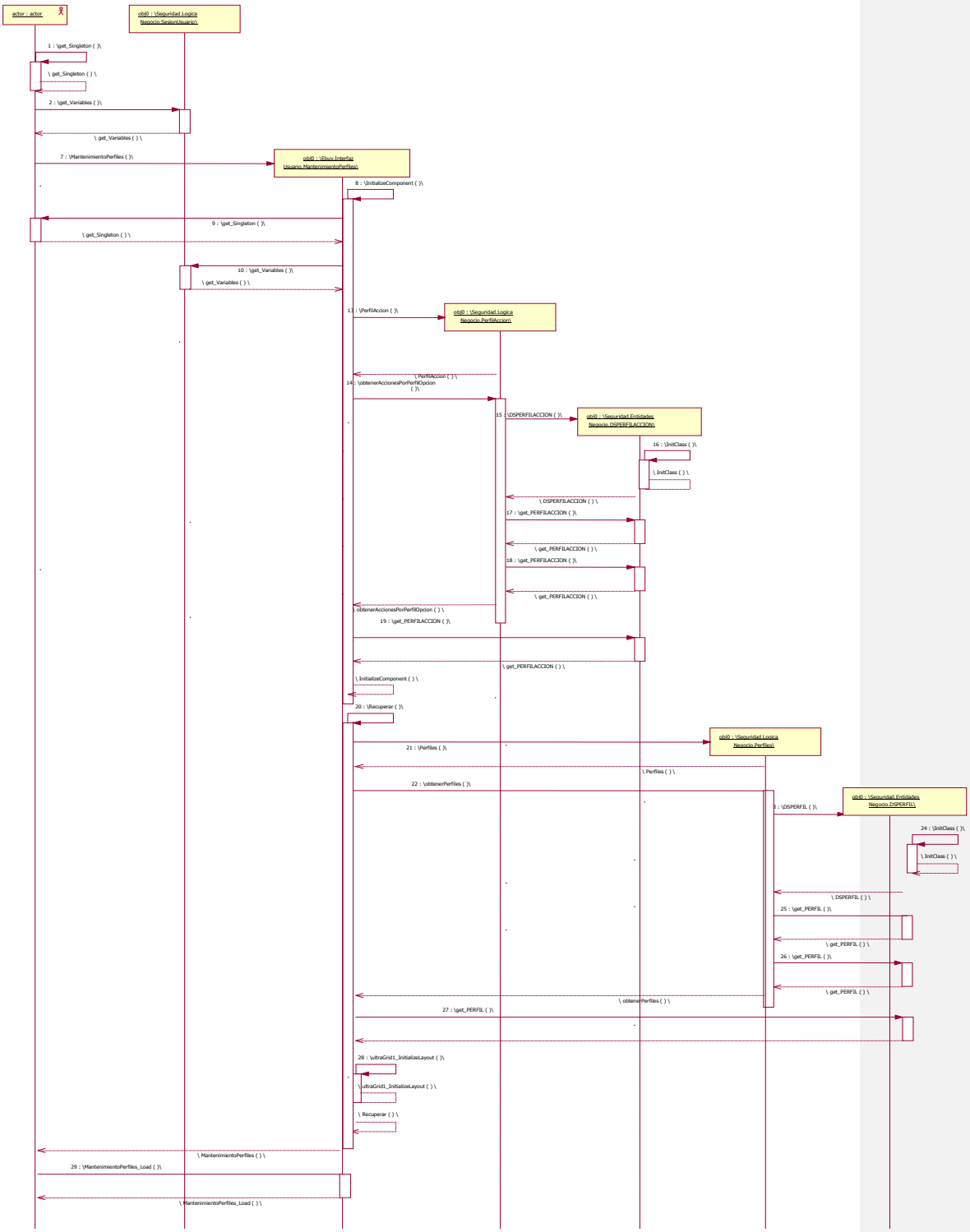
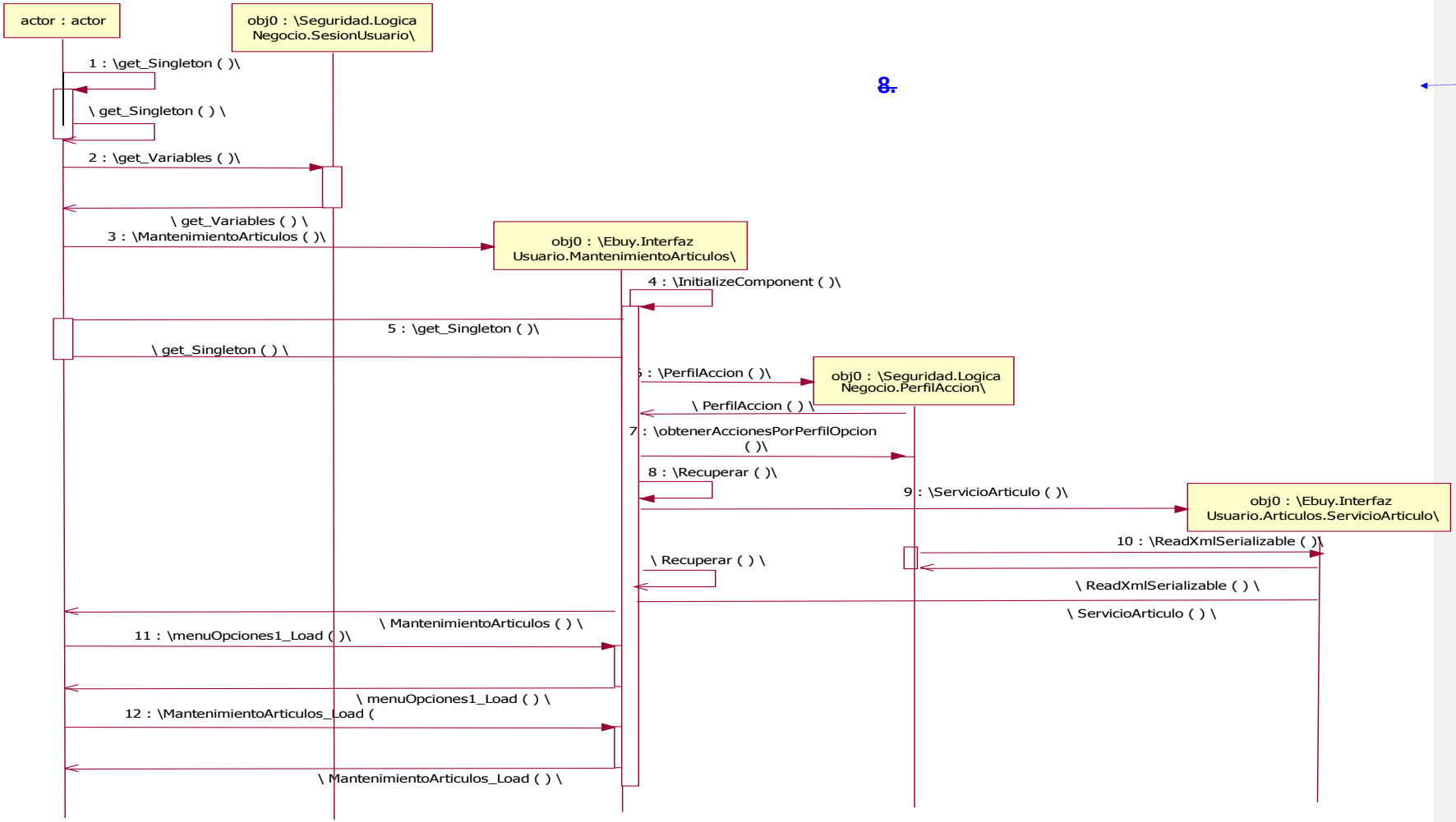


Figura 6.13: Diagrama de secuencia mantenimiento de perfiles.

La figura muestra el mantenimiento de una tabla sencilla.



~~8.~~Figura 6.14: Diagrama de secuencia servicio artículos.

La figura muestra la llamada a un servico web.

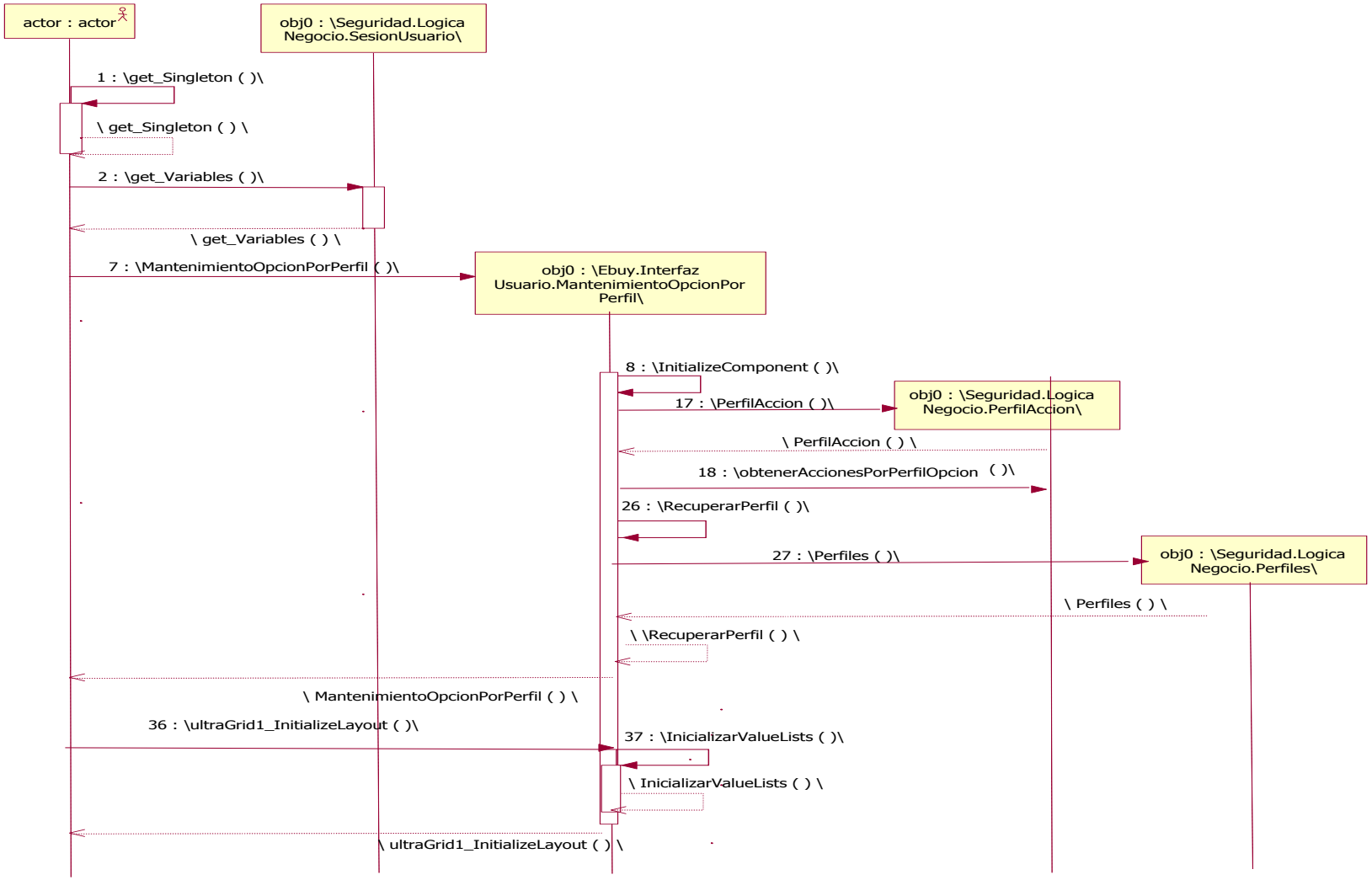
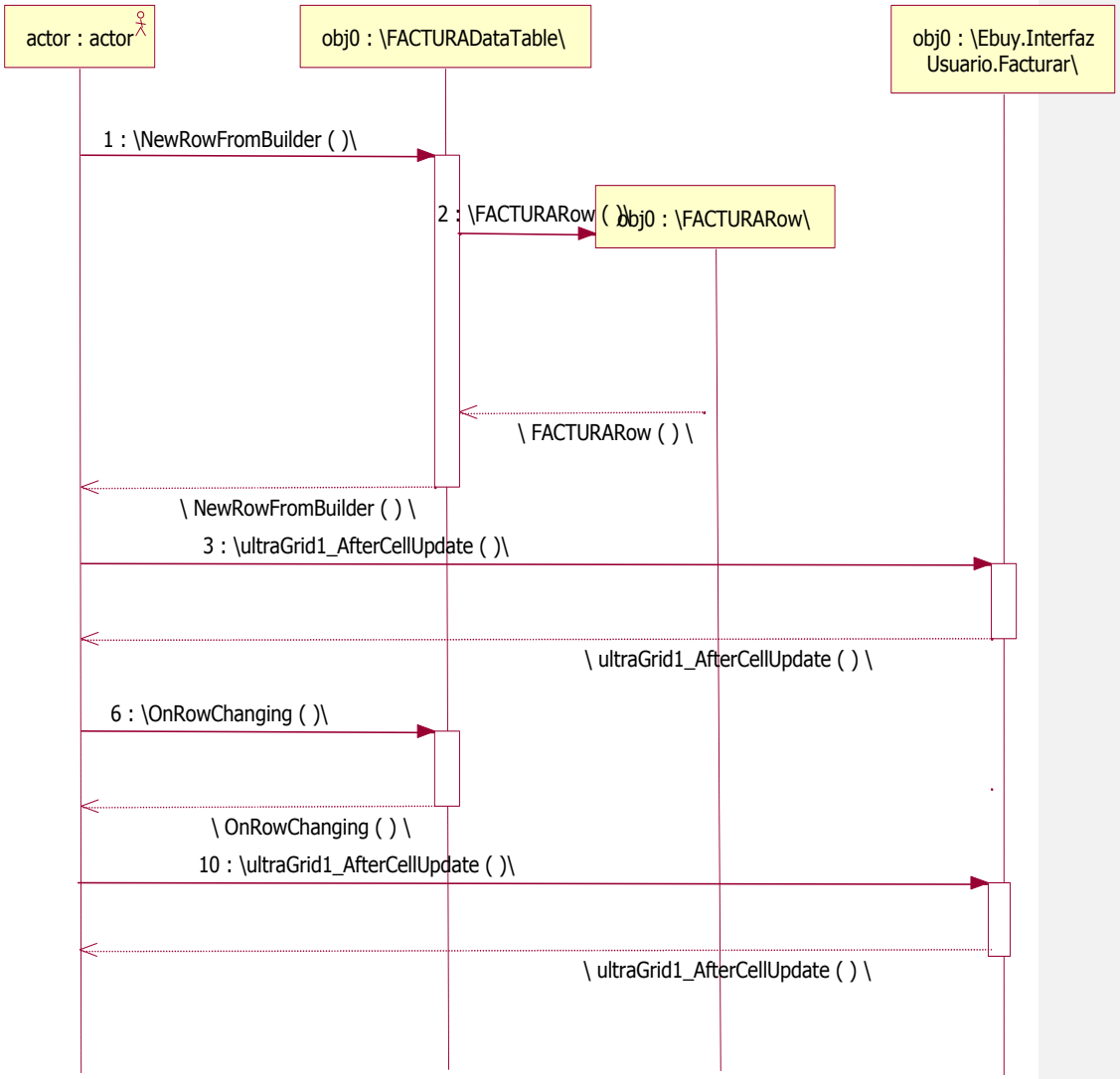


Figura 6.15: Diagrama de secuencia mantenimiento opciones por perfil.

La figura muestra el mantenimiento de una tabal maestro detalle.

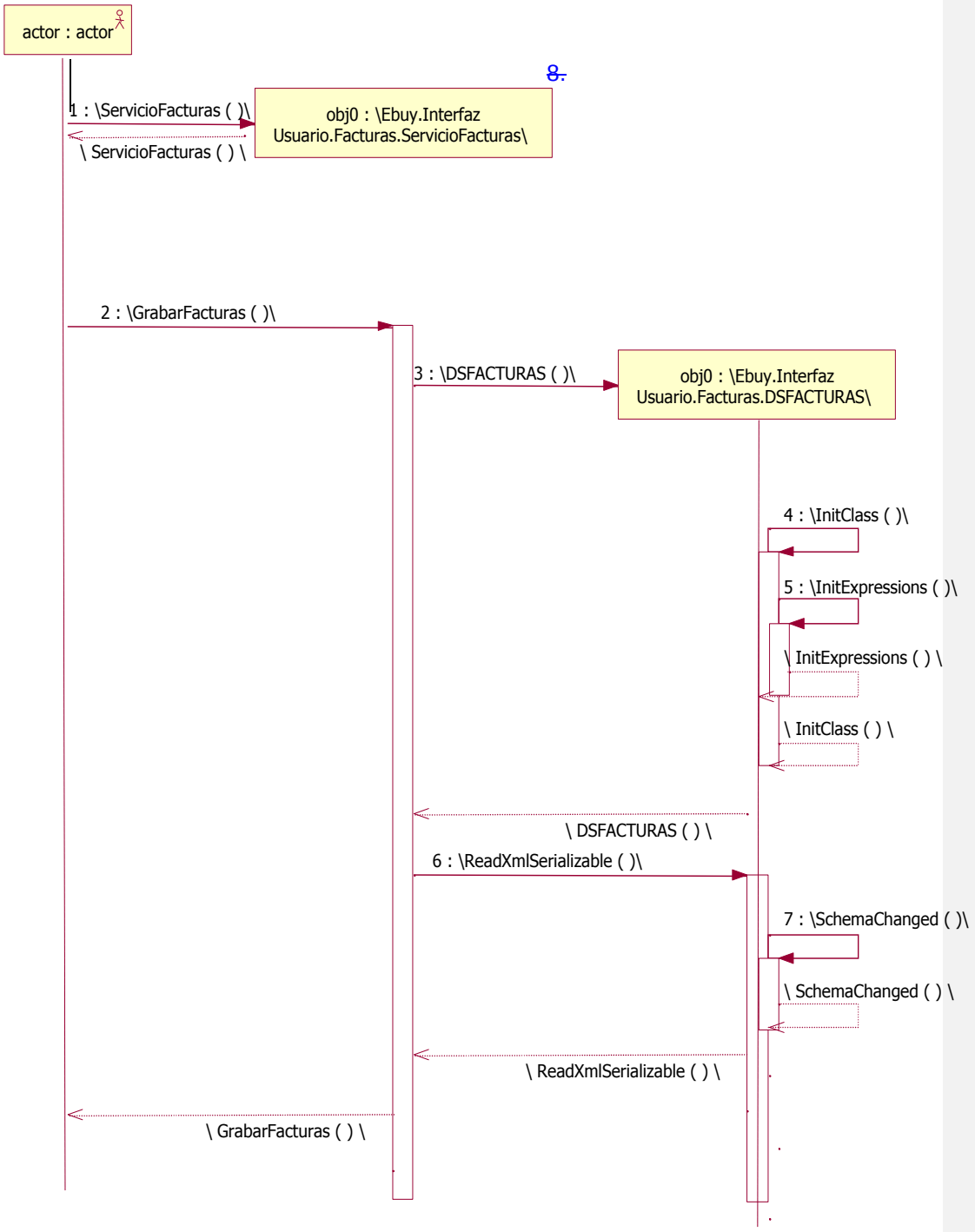


~~3.~~Figura 6.16: Diagrama de secuencia insertar factura.

Con formato: Numeración y viñetas

La figura muestra como la creación de una factura.

Con formato: Numeración y viñetas



~~8.~~Figura 6.17: Diagrama de secuencia grabar factura.

Con formato: Numeración y viñetas

La figura muestra como se graba una factura.

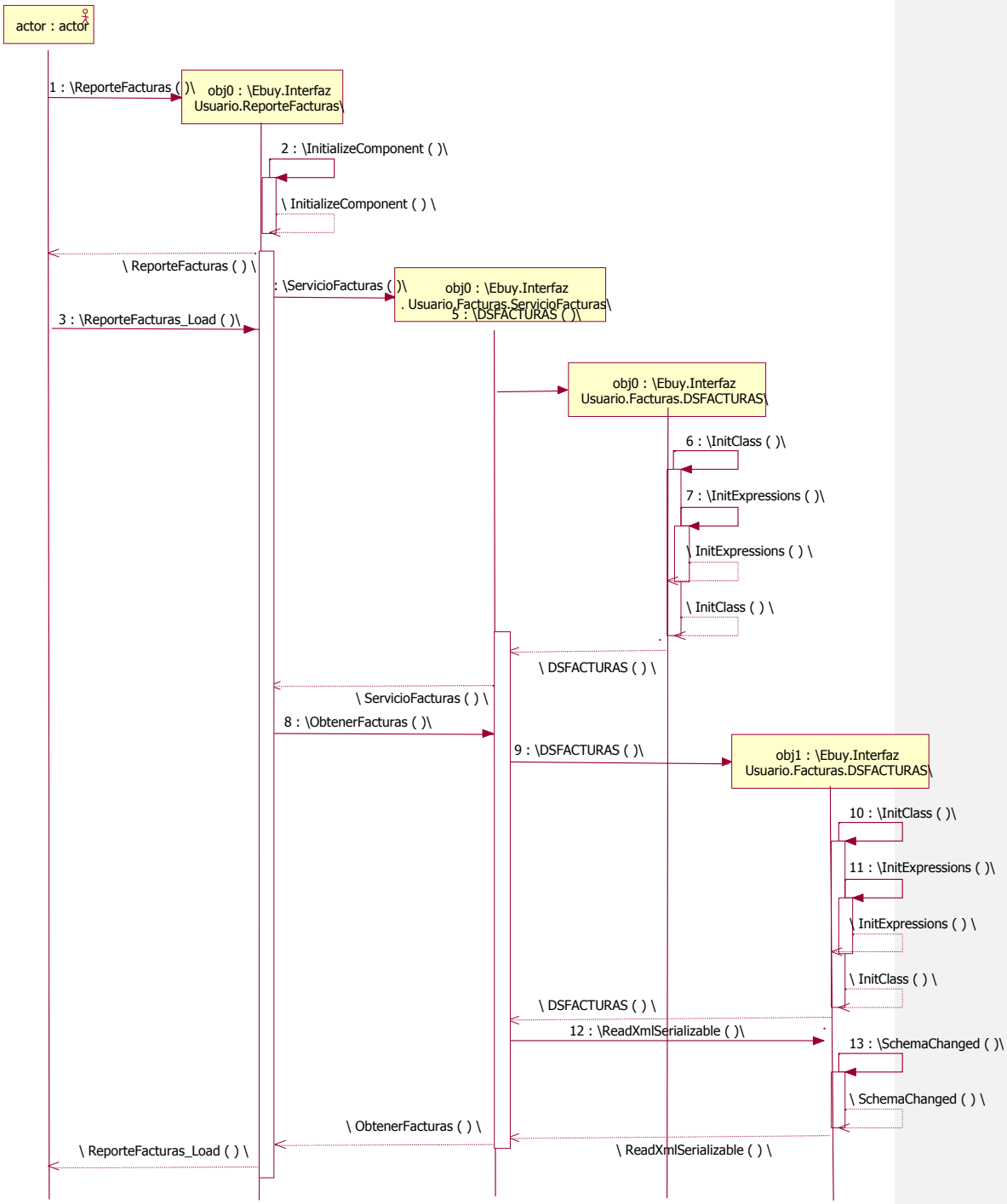


Figura 6.18: Diagrama de secuencia reporte factura.

Objeto usuario.

La figura muestra como obtien datos para un reporte.

Estados del objeto Usuario con sus descripciones.

Estado	Descripción
Usuario Almacenado	Si la información es correcta se almacenan los datos del usuario.
Usuario Modificado	Si la información registrada en la base de datos es incorrecta se procede a actualizar la información de usuario.
Usuario Eliminado	Si la información existente en la base de datos es obsoleta se puede eliminar el registro.
Usuario Consultado	Despliega la información del registro seleccionado en la pantalla.

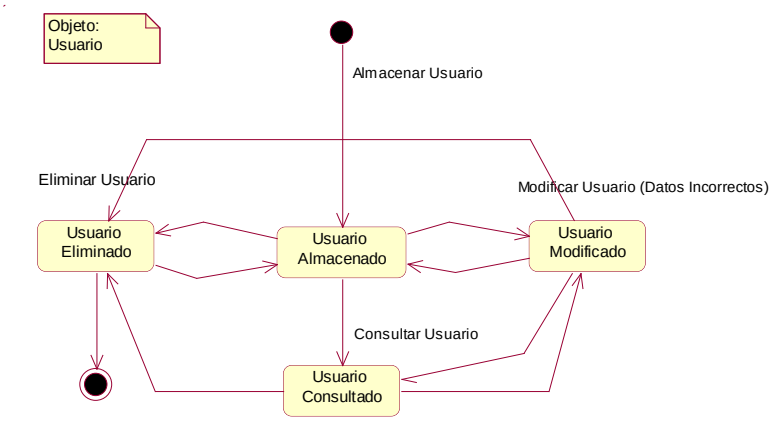


Figura 6.194.25: Diagrama de estados del objeto usuario.

El diagrama de estados del objeto “Usuario” es aplicable para diagramas de estado de los siguientes objetos:

- Cliente.
- Artículo.
- Forma de pago.
- Ubicación.

Con formato: Numeración y viñetas

- Unidad de medida.
- Perfil.
- Acción.
- Opción.
- Cliente.

Objeto Factura

Estados del objeto Factura con sus respectivas descripciones.

Estado	Descripción
Factura Grabada	Si la información es correcta se almacenan los datos de la factura.
Factura Esperando	Si no hay unidades suficientes espera a que se corrijan los datos.
Factura Consultada	Despliega la información del registro seleccionado en la pantalla.

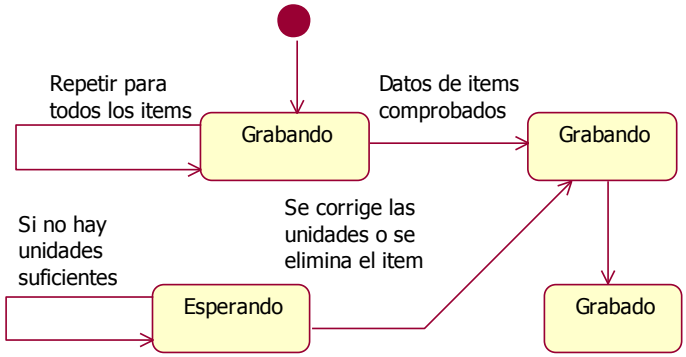


Figura 6.20~~4.26~~: Diagrama de estados del objeto factura.

~~8.1.4~~6.1.5 Modelo de datos (análisis estructurado)

Con formato: Numeración y viñetas

a. Modelo lógico

Con formato: Numeración y viñetas

~~Modelo Lógico~~

Este modelo se encarga de desplegar todas las entidades con sus respectivas relaciones y cardinalidad, así como sus tablas de rupturas en el caso de existir relaciones de muchos a muchos.

El modelo lógico del prototipo de una aplicación de facturación se lo puede ver en la Figura 6.21 El modelo lógico de la implementación de seguridades se lo puede ver en la figura 6.22.

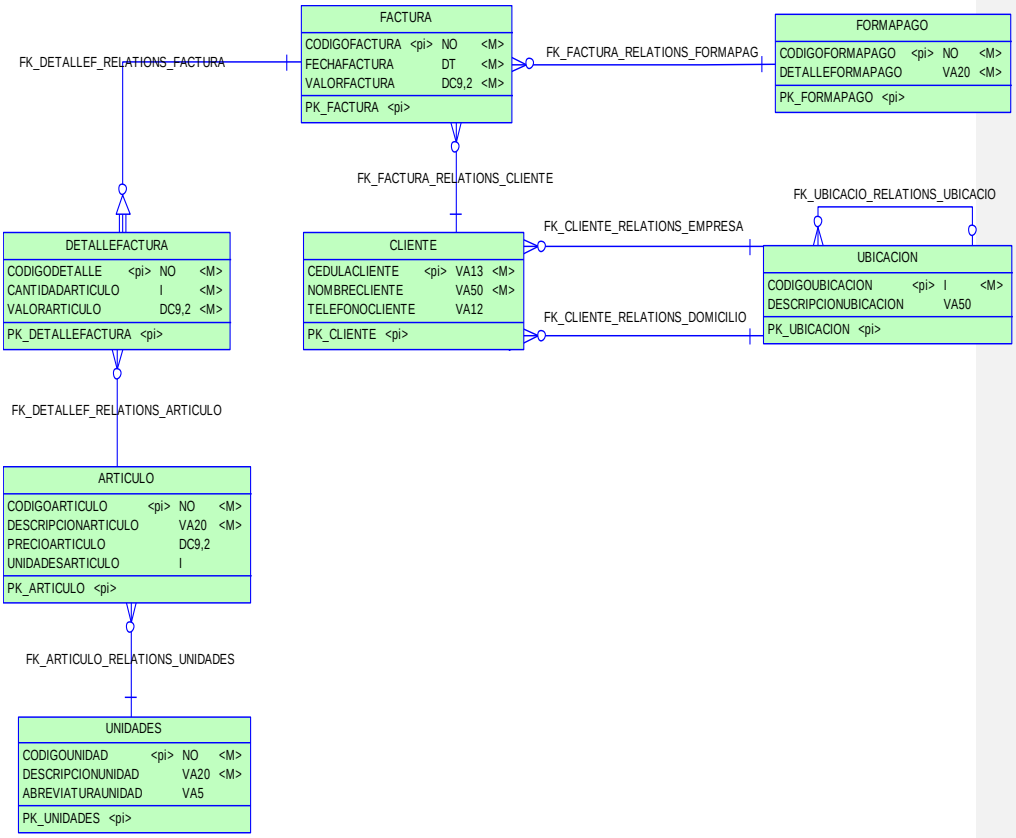


Figura 6.21~~4.27~~: Diagrama lógico entidad relación Ebuy.

La figura muestra el diagrama lógico del prototipo de una aplicación de facturación.

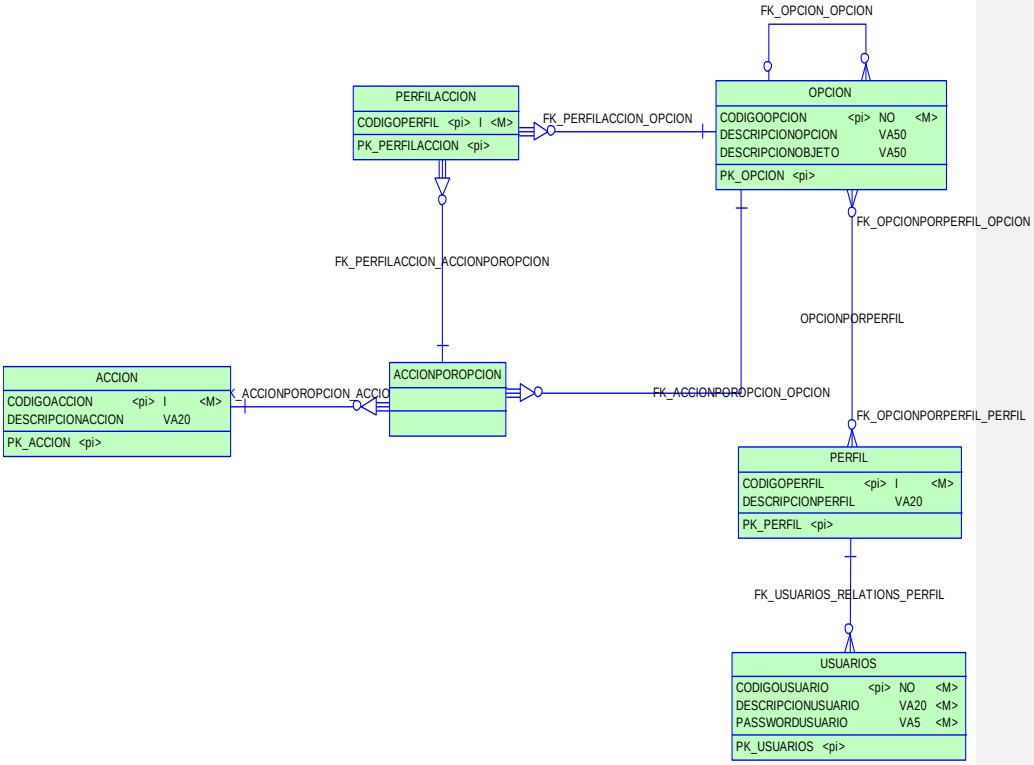


Figura 6.224.28: Diagrama lógico entidad relación seguridades.

La figura muestra el diagrama lógico de una aplicación de seguridades y permisos.

~~a.~~b. Modelo físico

Este modelo a diferencia del modelo lógico se encarga de mostrar todas las entidades que intervendrán en el sistema pero con sus respectivas claves primarias, clave foráneas y tipo de datos que posee cada atributo de la entidad. A partir de éste modelo de obtiene el script de base de datos para el sistema.

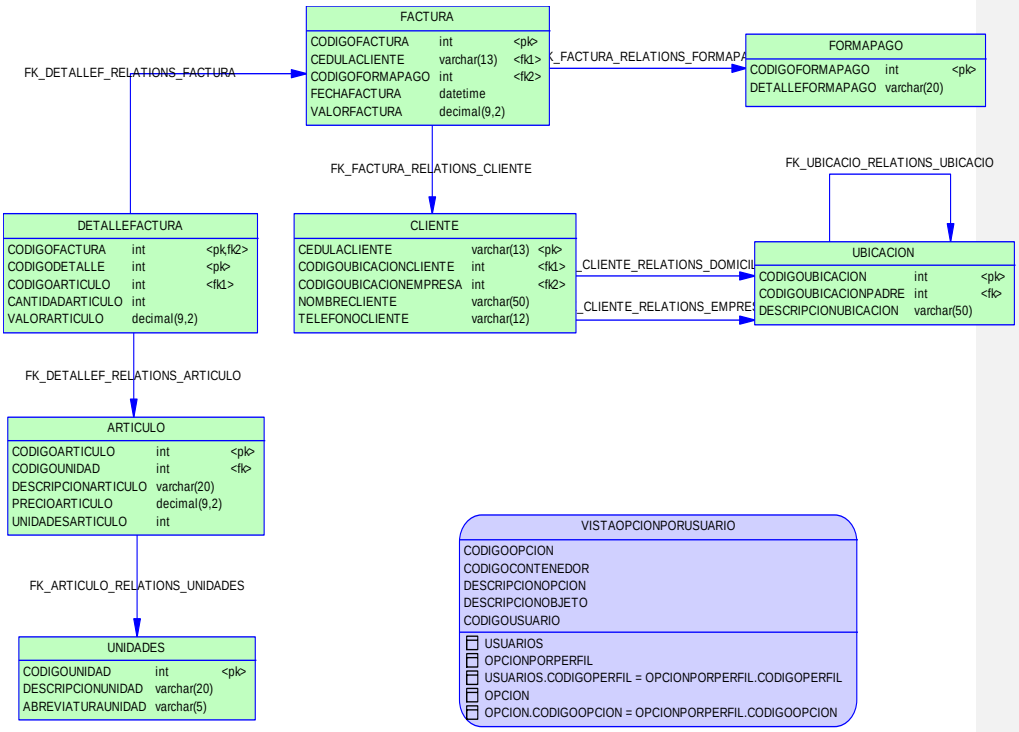


Figura 6.23~~4.29~~: Diagrama físico entidad relación Ebuy.

La figura muestra la implementación de estructura física de los objetos en una base de datos de tipo relacional.

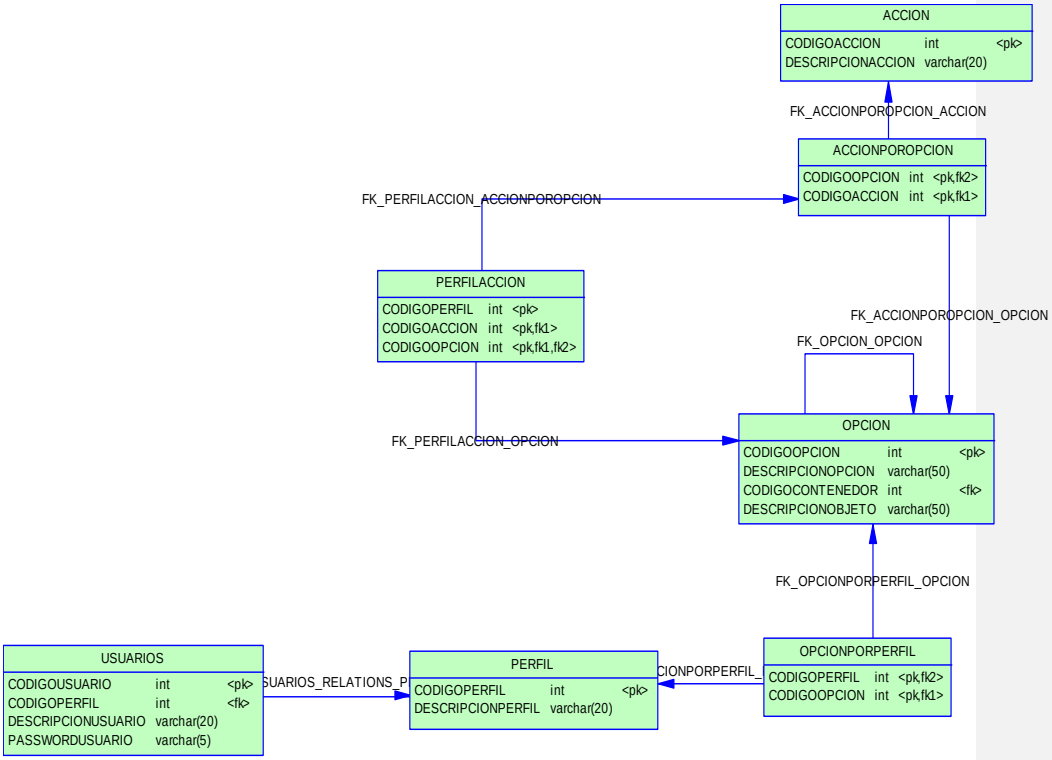


Figura 6.24~~4.30~~: Diagrama físico entidad relación seguridades.

La figura muestra la implementación de estructura física de los objetos en una base de datos de tipo relacional.

~~b.~~c. Diccionario de datos

Lista de tablas

- ACCION.
- ACCIONPOROPCION.
- OPCION.
- OPCIONPORPERFIL.
- PERFIL.
- PERFILACCION.
- USUARIOS.
- CLIENTE.
- UBICACIÓN.
- ARTICULO.
- FORMAPAGO.
- UNIDADES.
- FACTURA.
- DETALLEFACTURA.

Tabla: Usuarios.

Descripción: Lista de usuarios registrados en el sistema.

Nombre	Tipo	I	M	Descripción
CODIGOUSUARIO	NO	Si	Si	Código del usuario.
DESCRIPCIONUSUARIO	VA20	No	Si	Descripción del usuario.
PASSWORDUSUARIO	VA5	No	Si	Password del usuario.

Tabla: Perfil.

Descripción: Contiene la lista de perfiles definidos para el sistema.

Nombre	Tipo	I	M	Descripción
CODIGOPERFIL	I	Si	Si	Código del perfil.
DESCRIPCIONPERFIL	VA20	No	No	Descripción del perfil.

Tabla: Acción por Opción.

Descripción: Acciones permitidas en una Opción Ej: Eliminar.

Nombre	Tipo	I	M	Descripción
CODIGOACCION	I	Si	Si	Código de la acción.
CODIGOOPCION	I	Si	Si	Código de la opción.

Tabla: Acción por Opción .

Descripción: Acciones permitidas en una opción Ej: Eliminar.

Tabla: Opción por perfil.

Descripción: Opciones permitidas a un perfil Ej: Mantenimiento a Administrador.

Nombre	Tipo	I	M	Descripción
CODIGOOPCION	I	Si	Si	Código de la opción.
CODIGOPERFIL	I	Si	Si	Código del perfil.

Tabla: Perfil Acción.

Descripción: Acciones permitidas sobre una opción a un perfil Ej: Eliminar en la opción Mantenimiento al perfil Administrador.

Nombre	Tipo	I	M	Descripción
CODIGOPERFIL	I	Si	Si	Código del perfil.
CODIGOACCION	I	Si	Si	Código de la acción.

Tabla: Cliente.

Descripción: Contiene la acciones permitidas sobre una entidad.

Nombre	Tipo	I	M	Descripción
CEDULACLIENTE	VA13	Si	Si	Cédula del cliente.
NOMBRECLIENTE	VA50	No	Si	Nombre del cliente.
TELEFONOCLIENTE	VA12	No	No	Teléfono del domicilio.

Tabla: Forma de Pago.

Descripción: Contiene la acciones permitidas sobre una entidad.

Nombre	Tipo	I	M	Descripción
CODIGOFORMAPAGO	NO	Si	Si	Código de forma de pago.
DETALLEFORMAPAGO	VA20	No	Si	Descripción forma de pago.

Tabla: Ubicación.

Descripción: Contiene la acciones permitidas sobre una entidad.

Nombre	Tipo	I	M	Descripción
CODIGOUBICACION	I	Si	Si	Código de la ubicación.
CODIGOPADRE	I	Si	Si	Código de la ubicación padre.
DESCRIPCIONUBICACION	VA50	No	No	Descripción ubicación.

Tabla: Artículo.

Descripción: Contiene la acciones permitidas sobre una entidad.

Nombre	Tipo	I	M	Descripción
CODIGOACCION	Integer	Si	Si	Código de analista que.

CODIGOUNIDAD	NO	Si	Si	Código de unidad de medida.
ESCRIPCIONACCION	VA 20	No	No	Nombre de los analistas.

Tabla: Unidades.

Descripción: Contiene la acciones permitidas sobre una entidad.

Nombre	Tipo	I	M	Descripción
CODIGOUNIDAD	NO	Si	Si	Código de unidad de medida.
DESCRIPCIONUNIDAD	VA20	No	Si	Descripción de U. de medida.
ABREVIATURAUNIDAD	VA5	No	No	Abreviatura U. de medida.

Tabla: Factura.

Descripción: Contiene la acciones permitidas sobre una entidad.

Nombre	Tipo	I	M	Descripción
CODIGOFACTURA	NO	Si	Si	Código de factura.
CODIGOFORMAPAGO	NO	Si	Si	Código de forma de pago.
CEDULACLIENTE	VA13	Si	Si	Cédula del cliente.
FECHAFACTURA	DT	No	Si	Fecha de factura.
VALORFACTURA	DC9,2	No	Si	Valor total de Factura.

Tabla: Detalle Factura.

Descripción: Contiene la acciones permitidas sobre una entidad.

Nombre	Tipo	I	M	Descripción
CODIGOFACTURA	NO	Si	Si	Código de factura.
CODIGODETALLE	NO	Si	Si	Código de ítem.
CANTIDADARTICULO	I	No	Si	Código del artículo.
VALORARTICULO	DC9,2	No	Si	Valor del artículo.

Tabla: Opción.

Descripción: Contiene las acciones permitidas sobre una entidad Ej:

Mantenimiento.

Nombre	Tipo	I	M	Descripción
CODIGOOPCION	I	Si	Si	Código de la opción.
DESCRIPCIONOPCION	VA50	No	No	Descripción de la opción.
CODIGOCONTENEDOR	I	No	No	Código Padre de la opción.
DESCRIPCIONOBJETO	VA50	No	No	Descripción del objeto.

Con formato: Numeración y viñetas

~~8.2~~6.2 Fase de diseño del sistema

El diseño del sistema es la estrategia de alto nivel para resolver problemas y construir una solución. Este incluye decisiones acerca de la organización del sistema en subsistemas, la asignación de subsistemas a componentes hardware y software, y decisiones fundamentales conceptuales y de política que son las que constituyen un marco de trabajo para el diseño detallado.

a. Diagrama de componentes propuesto por UML

Con formato: Numeración y viñetas

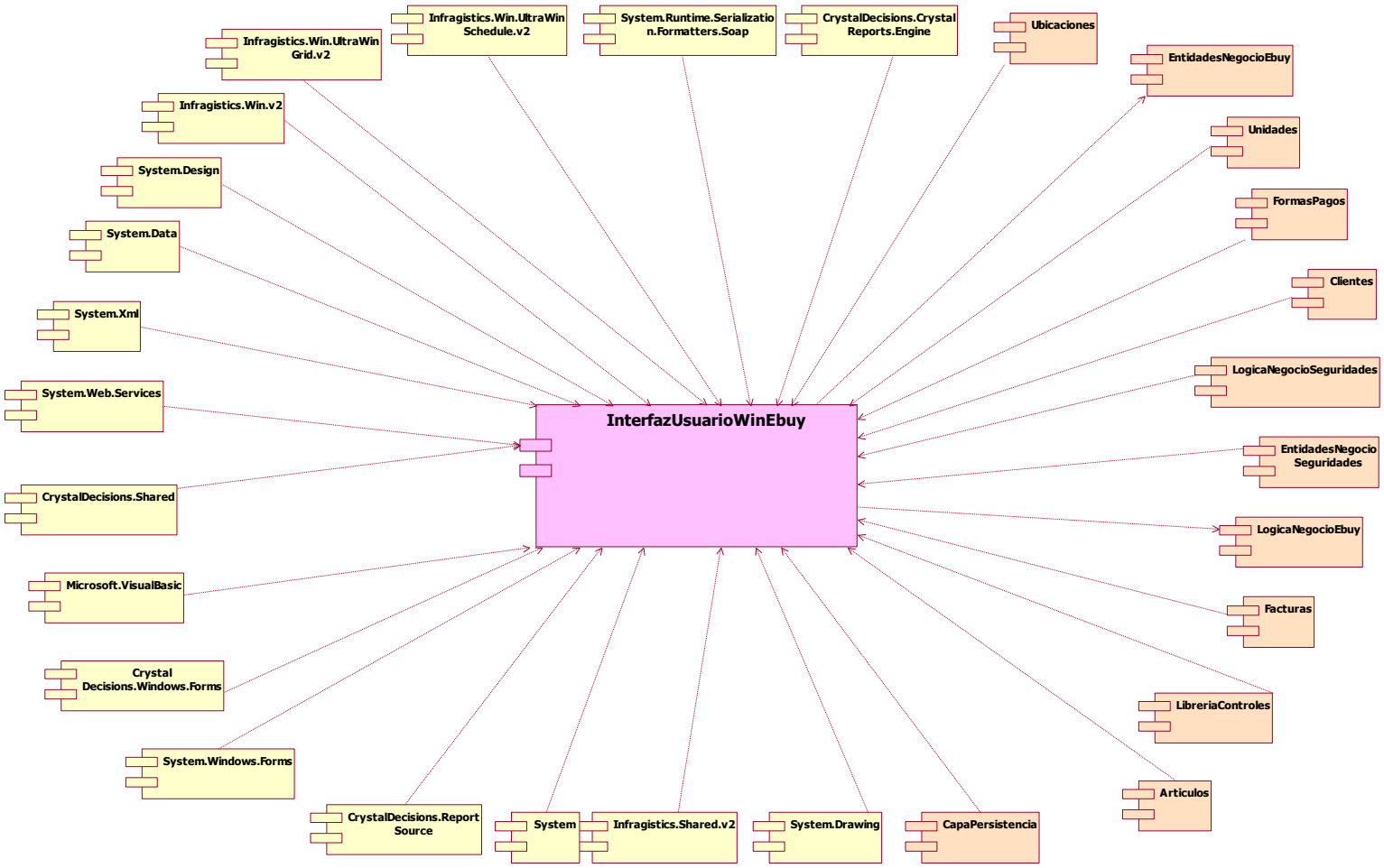
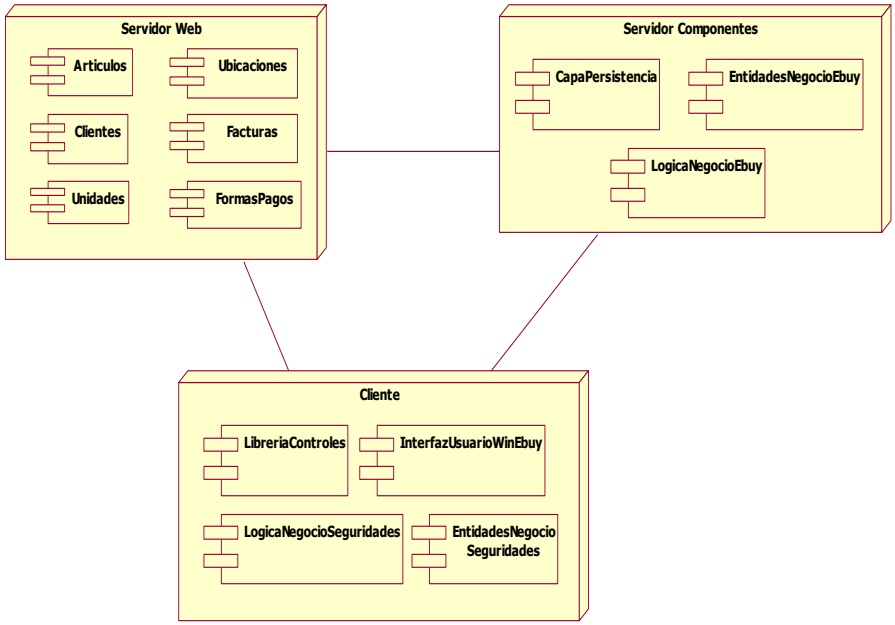


Figura 6.25: Diagrama de componentes prototipo de la aplicación

Este diagrama muestra en el lado izquierdo los componentes del Framework y a la derecha los componentes del prototipo que interactúan para lograr la funcionalidad del sistema.

b. Diagrama de despliegue

Con formato: Numeración y viñetas



~~3.7.1~~ Figura 6.26: Diagrama de despliegue para el prototipo de la aplicación

Con formato: Numeración y viñetas

El diagrama de despliegue muestra las relaciones físicas entre los componentes hardware y software en el prototipo de una aplicación de facturación.

~~El diseño del sistema es la estrategia de alto nivel para resolver problemas y construir una solución. Éste incluye decisiones acerca de la organización del sistema en subsistemas, la asignación de subsistemas a componentes hardware y software, y decisiones fundamentales conceptuales y de política que son las que constituyen un marco de trabajo para el diseño detallado~~

~~8.2.1~~6.2.1 Definición de subsistema

Con formato: Numeración y viñetas

El prototipo de una aplicación de facturación contiene los siguientes módulos:

- Manejo de seguridades.
- Mantenimiento de entidades.
- Facturación.
- Reportes.

~~8.2.2~~6.2.2 Identificación de la concurrencia

Con formato: Numeración y viñetas

Visual Studio .Net aporta mecanismos para la sincronización a distintos niveles: de clase, de objeto, de método y bloque de sentencias, con cláusulas primitivas que controlan el acceso concurrente

~~8.2.3~~6.2.3 Asignación de procesadores

Con formato: Numeración y viñetas

Cada subsistema concurrente debe ser asociado a una unidad de hardware, bien a un procesador de propósito general o a una unidad funcional especializada.

Debido a que el prototipo va ha manejar solo datos de prueba los recursos de los servidores destinados para las pruebas son suficientes para su funcionamiento.

Todos los módulos del prototipo van a ejecutarse en un solo procesador.

~~8.2.4~~6.2.4 Almacenamiento de datos

Con formato: Numeración y viñetas

Las bases de datos, van a ser administradas mediante sistemas de gestión de gestores de datos de tipo relacional, para el desarrollo de ésta tesis se realizará una demostración de las siguientes bases: SQL SERVER 2000, ORACLE, SYBASE.

~~8.2.5~~6.2.5 Arquitectura

Con formato: Numeración y viñetas

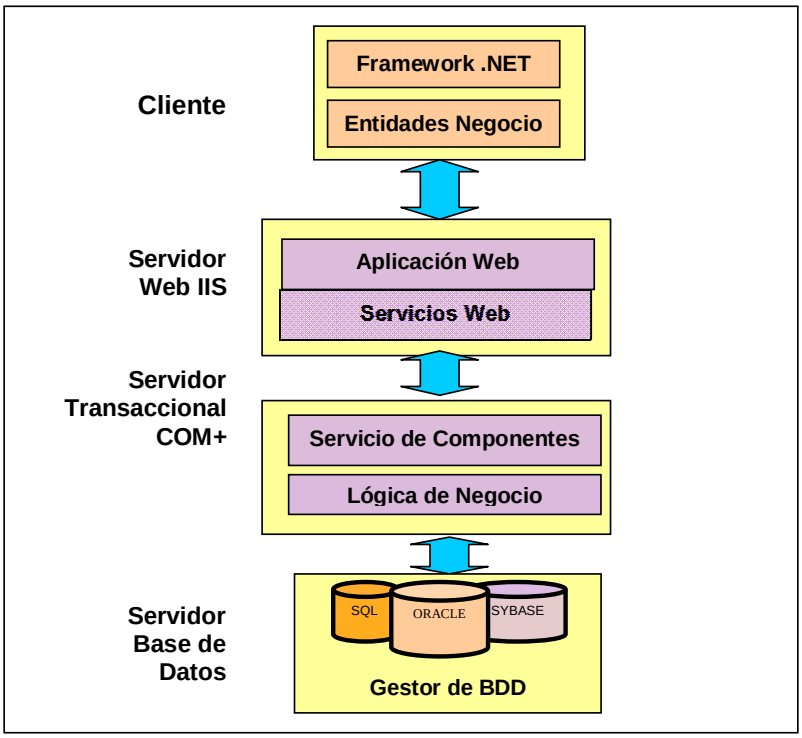


Figura 6.27:~~4.31~~ Esquema de la arquitectura del prototipo.

Con formato: Numeración y viñetas

6.3 Fase de diseño de objetos

~~**8.3Fase de diseño de objetos**~~

Con formato: Numeración y viñetas

En esta fase se van a desarrollar los métodos y propiedades de las clases definidas en la fase de análisis.

a. Diagrama de clases del prototipo

Con formato: Numeración y viñetas

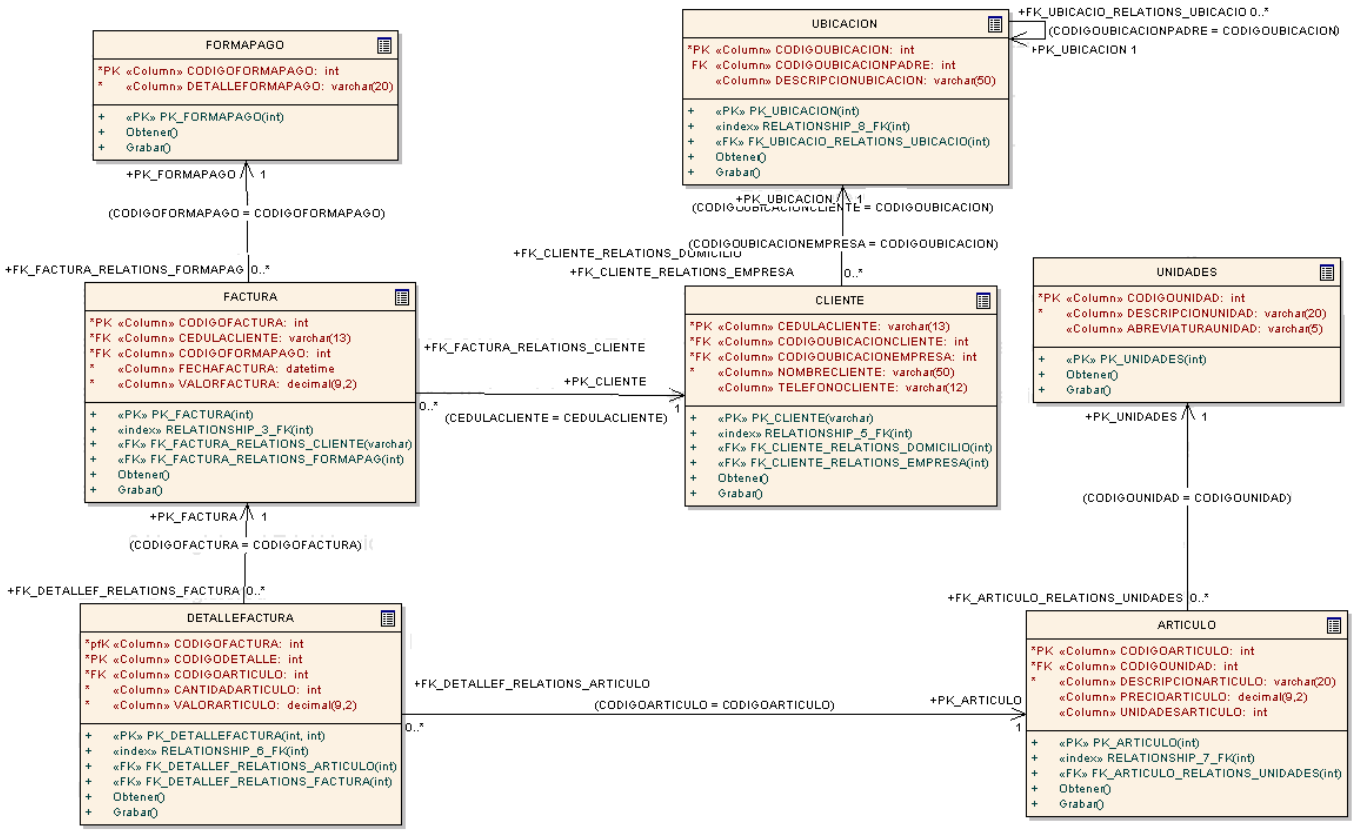


Figura 6.28:~~4.31~~ Diagrama de clases y métodos Ebuy.

La figura muestra el conjunto de clases y las relaciones que entre si, para el prototipo de una aplicación de facturación.

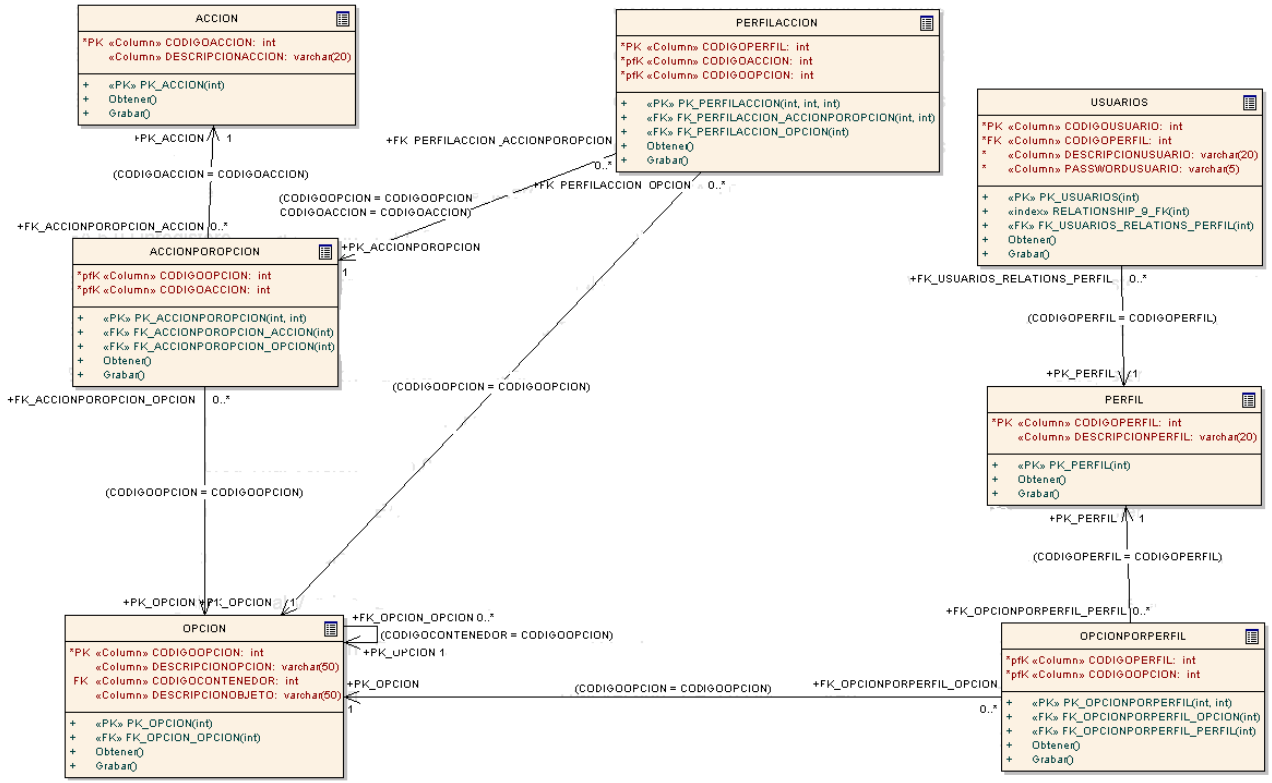


Figura 6.29:~~4.31~~ Diagrama de clases seguridades.

La figura muestra el conjunto de clases y las relaciones que entre sí, para la implementación de seguridades.

~~8.4~~6.4 Fase de implementación

~~8.4.1~~6.4.1 Nomenclatura para el diseño para la base de datos

Formato para el nombre de tablas:

- Se debe utilizar palabras completas.
- Se debe utilizar letras mayúsculas para los nombres de las tablas.
- Los campos deben indicar si es un código, una descripción o contener palabras que describan el contenido del campo.

Nombre de la entidad:	ARTICULO.
Nombre de los atributos:	CODIGOARTICULO.
	DESCRIPCIONARTICULO.
	UNIDADESDISPONIBLES.

1 CAPÍTULO VII

97 IMPLEMENTACIÓN Y PRUEBAS

4

Con formato: Numeración y viñetas

9.17.1 Propósito

- Clasificar los macro procesos que se desarrollan en el sistema para el correcto funcionamiento del mismo.
- Definir los resultados obtenidos en la finalización del proyecto.

9.27.2 Implementación

Para utilizar la capa de persistencia se deben seguir los siguientes pasos:

- Crear el archivo de configuración de la aplicación según el anexo.
- En la función principal del programa ejecutar la función inicializar de la clase AccesoDatosConfiguración, ejemplo:

```
[STAThread]
static void Main()
{
AccesoDatosConfiguración.Inicializar();
Application.Run(new InicioSesion());
}
```

- Para crear objetos de tipo conexión con la base de datos se debe ejecutar el método CrearConexión de la clase ConexiónFabrica, ejemplo:

```
IDbConnection conexión = ConexiónFabrica.CrearConexión ()
```

- Para manipular los datos se debe crear un objeto de la clase AdministradorBaseDatos, esta clase contiene funciones que permiten realizar operaciones CRUD sobre los datos.

    a. Ejemplo para obtener datos:

```
public DSARTICULO ObtenerArticulos()
{
        AdministradorBaseDatos admBD;
        DSARTICULO datos = new DSARTICULO();
        try
        {
                admBD = new AdministradorBaseDatos (
        AccesoDatosConfiguración.ProveedorDefecto);
                using (IDbConnection conexión = ConexiónFabrica.CrearConexión ())
                {
                string sentenciaSQL = String.Format ("SELECT * FROM {0}",
                        datos.ARTICULO.TableName );

                admBD.LlenarDataTable (conexión,CommandType.Text,sentenciaSQL,
                datos.ARTICULO);

                sentenciaSQL= string.Format("SELECT * FROM {0}",
                datos.UNIDADES.TableName );

                admBD.LlenarDataTable(conexión, CommandType.Text, sentenciaSQL,
                datos.UNIDADES);
                }
        }
        catch (Exception e1)
        {
                throw new Exception ("ObtenerArticulos", e1);
        }
                return datos;
}
```

    b. Ejemplo para grabar datos:

```
public DSARTICULO grabarArticulos( DSARTICULO datos)
{
        AdministradorBaseDatos admBD;
        try
        {
                admBD = new AdministradorBaseDatos(
        AccesoDatosConfiguración.ProveedorDefecto );
                using (IDbConnection conexión =  ConexiónFabrica.CrearConexión() )
                {
                string sentenciaSQL = string.Format ("SELECT * FROM {0}",
        datos.ARTICULO.TableName );
                admBD.ActualizarDataTable (conexión,CommandType.Text ,sentenciaSQL,
        datos.ARTICULO);
        }
        }
        catch(Exception e1)
        {
        throw new Exception ("GrabarArticulo",e1);
        }
        return datos;
}
```

~~9.3~~7.3 Alcance de las pruebas

Con formato: Numeración y viñetas

Los aspectos aplicados en la planificación y ejecución de pruebas son las siguientes:

- Utilizar diferentes tipos de pruebas con el fin de poder comprobar la funcionalidad total del sistema.
- Identificar los procesos críticos del sistema que son más propensos a fallas.

~~9.4~~7.4 Ambiente de Pruebas

Con formato: Numeración y viñetas

El ambiente de pruebas lo conformaron una pequeña red en donde se conectaron 4 computadoras en forma estrella, 2 de ellas actuaron como cliente y 2 como servidores.

Además de lo descrito anteriormente fue necesario instalar, el software de base, el motor de persistencia y la aplicación prototipo a todas las computadoras que intervendrían en el proceso de pruebas. Los requerimientos en hardware, software y comunicaciones serán descritos en “Recursos utilizados para ejecutar las pruebas”.

~~9.5~~7.5 Recursos de pruebas

Con formato: Numeración y viñetas

A continuación se enumera los recursos utilizados en las pruebas.

Recursos Hardware:

- 4 computadoras

Requerimientos básicos para Pc clientes

Procesador:	Min PENTIUM II.
Memoria:	Min 64 Mb.
Disco:	Min 6 Gb.
Monitor:	14”.
Bus de datos:	100Mbps.

Requerimientos básicos para servidores

Procesador:	Min PENTIUM III.
Memoria:	Min 256 Mb.
Disco:	Min 10 Gb.
Monitor:	14” .28 SVGA Color.
Bus de datos:	100Mbps.

Requerimientos Software:

PC clientes:

- Microsoft .Net framework 1.1.
- Internet Explorer v6.0 o superior.

Servidor de base de datos

- Microsoft .Net framework 1.1.
- SQL Server 2000.
- Oracle.
- IBM BD2.
- Sybase.

- Manejador de proveedores de datos para cada motor de BDD.

Requerimientos comunicaciones:

- Equipos conectados en red usando el protocolo de comunicaciones TCP/IP.
- Puerto que reciba las conexiones hacia el servidor.
- IP privada.

~~9.6~~7.6 Pruebas

Con formato: Numeración y viñetas

~~9.6.1~~7.6.1 Pruebas de Estándares

Estas pruebas consisten en verificar que se cumplan los estándares tanto en el diseño como en el desarrollo el sistema.

Estándares de diseño:

- Descripción de las clases.
- Descripción de atributos de las clases.

Parámetros:

- Espacios de nombres.
- Nomenclatura de clases.
- Nomenclatura de interfaces.
- Nomenclatura de atributos.
- Nomenclatura de campos estáticos.
- Nomenclatura de parámetros.
- Nomenclatura de métodos.

Estándares de desarrollo:

- Descripción de las clases.
- Descripción de atributos de las clases.
- Descripción de menús.

Parámetros:

- Formato de ingreso de datos.
- Formato de mensajes.
- Formato de etiquetas.

Todos los estándares antes mencionados fueron proporcionados por la empresa auspiciante y su verificación en el cumplimiento de las mismas fue minuciosamente comprobada.

~~9.6.2~~7.6.2 Resultado de las pruebas de estándares

Con formato: Numeración y viñetas

La verificación de los estándares lo realizó una persona designada por la empresa auspiciante, de acuerdo a la guía de pruebas realizada por las desarrolladoras.

En este tipo de pruebas no se encontró errores significativos, en el motor de persistencia, ya que la empresa auspiciante realiza un control temprano de los estándares. Para la parte del prototipo se realizaron las verificaciones respectivas y se corrigió, los nombres que no estaban de acuerdo a la nomenclatura. Posteriormente se pidió a la misma persona designada por la empresa

auspiciante que realizara la misma tarea para el prototipo. Los resultados fueron satisfactorios. Para mayor detalle referirse al documento: Guía de pruebas.

~~9.6.3~~7.6.3 Pruebas de Unitarias

Con formato: Numeración y viñetas

Las pruebas unitarias fueron realizadas con previa planificación ya que del correcto funcionamiento de cada una de ellas depende el funcionamiento integral de la CAPA DE PERSISTENCIA.

A continuación se detallan los aspectos que fueron tomados en cuenta:

Generales

- Conexión al servidor de base de datos.
- Acceso concurrente a las tablas.
- Correcto funcionamiento de las operaciones de inserción, modificación, eliminación y de búsquedas.
- Mensajes de error.

Modulo de seguridades

- Correcto almacenamiento de datos de tipos de usuarios
- Correcto almacenamiento de datos de usuarios
- Correcta asignación de permisos

Modulo de administración

- Correcto almacenamiento de ubicaciones (recursivas)
- Consistencia de datos en la visualización de reportes.

~~9.6.4~~7.6.4 Resultaltado de las pruebas de unitarias

Con formato: Numeración y viñetas

Generales

Las pruebas generales estaban orientadas para comprobar el correcto funcionamiento del motor de persistencia y su integración con la aplicación que la utiliza.

Se realizaron las pruebas básicas de conexión con los 3 tipos de servidores con los que se realiza el prototipo: SQL Server 2000, Oracle, DB2, con los parámetros necesarios para establecer la conexión: cadena de conexión, Proveedro de datos y usuario.

La observación más importante se encontró al realizar la conexión con DB2, ya que el usuario solicitante del servicio en este caso el usuario ASP.NET no estaba en el grupo de DB2USERS, y no permitía atender la solicitud.

Modulo de seguridades

En estas pruebas se dio prioridad a la asignación de permisos y a la verificación de que estos permisos se cumplen en cada perfil, además se solicitó verificar si había algún problema al ingresar caracteres especiales en los campos de descripción de los objetos de este módulo.

Modulo de administración

Este módulo prestaba tres elementos muy importantes que debían verificarse:

Objetos maestros detalle Se verificó que el ingreso de facturas era consistente.

Objeto recursivo Se verificó que la clase Ubicaciones tenga la funcionalidad recursiva, tanto al leer datos, como al almacenar.

Reportes Se verificó que los reportes obtenían los datos adecuados de acuerdo a los parámetros ingresados

~~9.6.5~~7.6.5 Pruebas de intregración

Las pruebas de integración, permiten validar la integridad de los datos, que consiste en la eliminación de registros que tienen datos relacionado en otras tablas y la inserción de valores que no constan la tabla catalogo o independientes.

- Integridad de datos.

~~9.6.6~~7.6.6 Resultaltado de las pruebas integración

Con formato: Numeración y viñetas

Estas pruebas verificaron la consistencia de la información tanto al eliminar como al almacenar datos. Estas pruebas fueron aplicadas a las clases del prototipo ya que el motor de persistencia es el ejecutor de estás acciones.

Para esta validación se planteó los siguientes listados:

- Listado de objetos a probar.
- Listado de acciones a ejecutar.
- Listado de Mensajes esperados.

Con estos listados se fueron realizando las pruebas y al final se añadió una columna para registrar el resultado obtenido, con cada uno de los servidores de datos.

~~9.6.7~~7.6.7 Pruebas de validación

Validación de formatos

- Ingreso de datos tipo fecha.
- Validación de fechas válidas.
- Ingreso de datos tipo caracteres.
- Ingreso de datos tipo numérico.
- Ingreso de caracteres nulos.
- Ingreso de caracteres en blanco.

Validación de acceso

- Acceso de usuarios no registrados en el sistema o usuarios que intentan ingresar con claves incorrectas.
- Acceso de usuarios que están inicializados (logeados) en el sistema.
- Duplicidad de nombres y claves de usuarios.
- Restricciones de acceso (permisos) a los usuarios que se encuentran registrados en el sistema.

~~9.6.8~~7.6.8 Resultado de las pruebas de validación

El resultado de las pruebas de validación, fueron las esperadas ya que anteriormente se había documentado en la sección 2.5, las diferencias de formato que existen para cada uno de los servidores de datos.

~~9.6.9~~7.6.9 Validación de funcionalidad mediante casos de uso

Nos permiten validar si el software desarrollado cumple con las funciones descritas en sus casos de uso.

Cada caso a probar describe tres escenarios de uso:

- Escenario correspondiente al curso normal del caso de uso.

- Escenario que incluyen al menos un curso opcional del caso de uso.
- Escenario que produce al menos una excepción.

~~9.6.10~~7.6.10 Resultado validación de funcionalidad mediante casos de uso

Las pruebas mediante casos de uso se realizaron para probar al menos una vez cada uno de los caminos posibles que va ha realizar el usuario.

El escenario normal lista secuencialmente los pasos del camino normal. El escenario opcional indica el número de paso opcional y supone que los pasos anteriores y posteriores deben ser ejecutados.

Se planteó así mismo el listado de los casos de uso y las acciones que se van a probar en cada uno. Para mayor detalle referirse al documento: Guía de pruebas.

CONCLUSIONES

- Se realizó el análisis, diseño, e implantación de un Motor de Persistencia para el gestor de datos, utilizando C# como lenguaje de desarrollo que realiza de manera transparente las operaciones de mantenimiento (Inserción, actualización, eliminación), consulta y conexión sin importar cual es la conexión que se está utilizando o el gestor de datos

- Se realizó el análisis, diseño, e implantación de un prototipo de una aplicación de facturación de productos en n capas para demostrar la funcionalidad del Motor de Persistencia para .Net, utilizando C#, Windows forms y Web Services como plataforma de desarrollo.

- Al utilizar el Motor de Persistencia para .Net la aplicación prototipo se pudo conectar con los gestores de datos DB2, SQL Server 2000 sin tener que cambiar el código desarrollado, enviando como parámetros la cadena de conexión y la información de configuración de los proveedores de datos.

- La utilización de patrones de creación aceleró el desarrollo del Motor de persistencia para .Net. El patrón de creación utilizado fue Abstract Factory que permite crear de forma genérica las clases, apartando la decisión de como crearlas, gracias a esto podemos crear una serie de objetos que permiten conectarse con cualquier origen de datos e interactuar con él.

- El Motor de Persistencia para .Net es adaptable a distintas arquitecturas de aplicación, monolíticas locales, cliente servidor, desplegadas en un servidor Web, en servidor de aplicaciones.

- Después de realizar las pruebas se determinó que los errores más frecuentes se deben a aspectos de configuración. Es necesario que el usuario ASP.NET este incluido como usuario en el gestor de datos correspondiente y que tenga permisos para realizar operaciones CRUD sobre los objetos de la base. Y es necesario que el archivo de configuración de la aplicación tenga los datos de información del proveedor y fuente de datos para estar correctamente configurado y que funcione la aplicación.

RECOMENDACIONES

- Se recomienda la adopción del Motor de Persistencia para .Net, en la producción de aplicaciones que trabajen sobre una base de datos relacional u otra forma de almacenamiento, especialmente si emplean sistemas gestores de datos de categorías distintas. En todos los casos se va a obtener un código con menos líneas, más fácil de mantener, más correcto y productivo.

- Antes de probar la funcionalidad de la capa de persistencia con una aplicación, hay que verificar que los aspectos de configuraciones, servicios, permisos sobre las bases para el usuario ASP.NET, estén habilitados y en correcto funcionamiento.

- Se debe capacitar al personal que va a utilizar el Motor de Persistencia para .Net sobre el negocio al que se refiere la aplicación para producir programas correctos, ya que la capa de persistencia no va a permitir producir programas correctos si el personal, no tiene la formación y preparación adecuadas.

- Se recomienda que para aumentar el rendimiento de las aplicaciones se debe trabajar con proveedores administrados para cada uno de los gestores de datos existentes para .Net, también se puede implementar el proveedor de datos ODBC que puede utilizarse para funcionar con todos los gestores de datos que cumplan las especificaciones ODBC.

- Se deben escribir sentencias SQL utilizando ANSI SQL para que todos los gestores de datos las puedan interpretar, si no es posible escribir sentencias SQL con ANSI SQL se deben escribir sentencias SQL específicas para que sean interpretadas por el gestor de datos que le corresponde.

- Se recomienda que se ejecuten las sentencias SQL con una cuenta con privilegios mínimos.

- Para evitar un ataque contra un gestor de datos que entiende SQL se debe indicar a los programadores que para escribir sentencias SQL, llamar procedimientos almacenados se utilicen parámetros.

- Para evitar la inyección de SQL se debe filtrar las entradas de caracteres SQL del usuario en la aplicación desarrollada, además evitar la entrada no válida en los campos de la aplicación cliente limitando el tamaño y el tipo de entrada.

- Se recomienda a los programadores no exponer al usuario final los errores SQL mostrados por el gestor de la base de datos.

BIBLIOGRAFÍA

- PRESSMAN ROGER, Ingeniería de Software un enfoque práctico, 4ta. Edición.
- M. ATKINSON R.MORRISON, Orthoganally Persistent Object Systems. The VLDB.
- M.P ATKINSON, F. BANCILHON, D.J. DEWITT, K. R. DITTRICH, D. MAIER, AND S. B. ZDONIK. The object-oriented database system manifesto. SIGMOND Conference, May 1990.
- EDGAR F. CODD, The Relational Model for Database Management. Version 2. Addison Wesley Pulishing Co. 1990.
- BERTRAND MEYER, Construcción de software orientado a objetos, Prentice Hall,1997
- ARTHUR M. KELLER y otros, architecting Object Applications for High Performance with Relational Databases, 1995

DOCUMENTOS EN LA RED

- ROBERT D. POOR HYPHOS, ww.media.mit.edu/pia/Research/Hyphos
- DR. MALCOLM ATKINSON, Pjama Project www.dcs.gla.ac.uk/pjava/
- Telefonia global www.iridium.com www.objectivity.com/NewsEvents/Releases/archived_releases/1999/MotorolaDeliveresIridium.html
- Babar database www.slac.stanford.edu/BFROOT/www/Pulbic/Computing/Databases/index.ht
- SCOTT W. AMBLER desing of Robust Persistente Layer. www.ambysoft.com/persistenceLayer.html

- www.hep.net/chep95/html/papers/p59/p59.ps
- www.ifi.uio.no
- www.monografias.com/trabajos13/metomt/metomt.shtml
- www.exa.unne.edu.ar/depar/areas/informatica/anasistem1/public_ht /Temas_08.pdf
- www.microsoft.com/spanish/Msdn/articulos/archivo/230801/voices/modelsoftware
- www.microsoft.com/spanish/msdn/comunidad/dce/1/entrenamiento/foxpro/1.asp
- www.ajlopez.com.
- www.msdn.microsoft.com/netframework.
- www.msdn.microsoft.com/library/spa/default.asp?url=/library/SPA/cpguide/html/cpconconnectingtosqlserverusingadonet.asp.
- www.microsoft.com/spanish/msdn/comunidad/uni.net/default.asp
- www.microsoft.com/spanish/msdn/arquitectura/DesignGuidelines/Guidelines2.asp
- Productos de persistencia
    - www.persistence.com/products/edgextend/index.php,
    - www.objectmatter.com
    - www.chimu.com/producst/form/index.html,
    - www.hibernate.org

GLOSARIO

RDBMS

Relational Data Base Management Systems.

OODBMS

Object Oriented Data Base Managment Systems.

CORBA

Common Object Request Broker Architecture.

COM()

Inteface estándar para objetos (no importa como están programados)

COM+

Extensión de COM en el que se añade un modelo para la programación de objetos.

API

Application Programming Interface.

CRUD

Create, retrieve, update,delete.

XML

Extensible markup language.

Assembly

Los ensamblados son ficheros con forma de EXE o DLL que contienen toda la funcionalidad de la aplicación de forma encapsulada. Assembly es la unidad mínima de ejecución para el .Net Reconoce una serie de *directivas* o *meta instrucciones* que indican ciertos parámetros de funcionamiento del ensamblado.

Reflexión

Permite acceder a los campos y métodos de una clase, y también acceder a campos y métodos de un objeto concreto. Se puede recuperar la información de los metadatos de los *assemblys* cargados.

Web-Services

Es considerado cualquier servicio accesible vía Internet que se encuentre basado en el protocolo XML / SOAP permitiendo una invocación independiente de la plataforma o lenguaje en el que se encuentre escrito.

Framework .Net

Es el ambiente en el cual es posible desarrollar cualquier aplicación .Net, incluye las clases base de .Net (*Foundation Class Library*), un compilador de C#, documentación, el ".Net Runtime" encargado de ejecutar/interpretar componentes. Net y otras herramientas más.

Metodología

- Es un enfoque particular, fundado en ciertos principios generales, de orden filosófico; es un modo de comprender la realidad.

Método

- Es un conjunto de técnicas, herramientas y tareas que, de acuerdo a un enfoque metodológico, se aplican para la resolución de un problema.

Metadatos

- Son un conjunto de datos organizados en forma de tablas que almacenan información sobre los tipos definidos en el módulo, los miembros de éstos y sobre cuales son los tipos externos al módulo a los que son referenciados en el mismo.

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

ANEXOS

ARCHIVO DE CONFIGURACIÓN DE LA APLICACIÓN

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
	<configSections>
		<!-- name: Nombre de la seccion personalizada.
		    type: Clase encargada de leer la seccion dentro del proyecto -->

		<section name="SeccionCapaPersistencia.Proveedores" type="CapaPersistencia.ManejadorSecciónProveedores, CapaPersistencia"/>
		<section name="SeccionCapaPersistencia.FuentesDatos" type="CapaPersistencia.ManejadorSecciónFuentesDatos, CapaPersistencia"/>

	</configSections>
	<!-- El atributo proveedorDefecto se setea en el programa la propiedad ProveedorDefecto de la
	     clase AccesoDatosConfiguración  el valor (proveedorSqlServer) debe ser uno de los nombres  de los proveedores existentes ya en la misma clase.  -->

		<SeccionCapaPersistencia.Proveedores proveedorDefecto="proveedorSqlServer">

			<!-- Los nombres de los atributos (nombreProveedor, nombreAssembly, ...)
			para la deserializacion se definen en las propiedades de la clase Proveedor. Con esa  información
			el deserializador sabe a que propiedad poner los valores. -->
			<Proveedor nombreProveedor="ProveedorDB2" nombreAssembly="IBM.Data.DB2" nombreClaseConexión="IBM.Data.DB2.DB2Connection"
				nombreClaseAdapter="IBM.Data.DB2.DB2DataAdapter" nombreClaseCommandBuilder="IBM.Data.DB2.DB2CommandBuilder"/>
			<Proveedor nombreProveedor="ProveedorSQL" nombreAssembly="IBM.Data.DB2" nombreClaseConexión="IBM.Data.DB2.DB2Connection"
				nombreClaseAdapter="IBM.Data.DB2.DB2DataAdapter" nombreClaseCommandBuilder="IBM.Data.DB2.DB2CommandBuilder"/>
		</SeccionCapaPersistencia.Proveedores>

		<SeccionCapaPersistencia.FuentesDatos fuenteDatosDefecto="Fuente2">
			<FuenteDatos nombreFuenteDatos="Fuente1" nombreProveedor="Proveedor1" cadenaConexión="User Id=tiger;Password=scott;Data Source=MaBase;"/>
			<FuenteDatos nombreFuenteDatos="Fuente2" nombreProveedor="proveedorSqlServer" cadenaConexión="workstation id=SERVER;packet size=4096;user
				id=sa;integrated security=SSPI;data source='.';persist security info=False;initial catalog=Ebuy"/>
			<FuenteDatos nombreFuenteDatos="Fuente3" nombreProveedor="ProveedorDB2" cadenaConexión="database=EBUY;Connect Timeout=30;user
				Id=Administrador;pwd=server"/>
		</SeccionCapaPersistencia.FuentesDatos>
</configuration>
```

MERCY GEOVANNA PINARGOTE ALARCÓN

La Pradera N30 – 258 y Mariano Aguilera
Teléfonos: 593-2-2557 691 / 593-2-2557 692
mercy.pinargote@logicstudio.net
Quito–Ecuador

Educación:

- 2005. Escuela Politécnica del Ejército. Título obtenido: Ingeniera en Sistemas.

Cursos y Seminarios:

- 2000. Mantenimiento de computadoras Escuela Politécnica del Ejército.
- 2001. Microsoft. Visual Studio.Net.
- 2002. Talleres ASP.NET, dictados en Objeq por el Microsoft Regional Director de Ecuador.
- 2002. Talleres C# (Básico y Avanzado), dictados en Objeq por el Microsoft Regional Director de Ecuador.

Experiencia:

- Septiembre 2005 - Actual. LogicStudio - Consultor Asociado.
- Enero 2002 – Febrero 2005. Seriva Inc.- Developer Senior.
- Enero 1999 – Diciembre 2001. Areasistemas - Desarrollador de Sistemas.

Proyectos realizados

- 2005 – Actual. Logic Studio. Consultor asociado a cargo de implementar el sistema para control de las inspecciones fitosanitarias del banano para el SESA.
  Tecnologías utilizadas: Windows Mobile 5.0, Pocket PC, SQL Server 2005, Reporting Services 2005, ASP.NET 2.0, Replication management, ActiveSync 4.0, C#.

- 2002 – 2005 Seriva. Developer Senior a cargo del desarrollo de los módulos de: Interbancarios, Repos, Forwards, Spot para el sistema de Tesorería “Seriva.Net”.
  Tecnologías utilizadas: SQL Server 2000, ASP.NET, Windows Forms, JavaScript, Crystal Reports, Web Services, C#.

- 1999 – 2001 Areasistemas. Desarrollador de Sistemas a cargo de varios proyectos:
    - Solca: sistema para el registro nacional de tumores.
    - Solca: Sistema de patología.
    - Fundación natura: Sistema de control de la opacidad para gasolina y diesel.
    - Ministerio del ambiente: Sistema para el control de emisiones de fuentes móviles.

  Tecnologías utilizadas: Visual Basic 6.0, Power Builder, Oracle, SQL Server.

Idiomas:

- Español: Idioma natal.
- Inglés: Conocimiento avanzado.

SANDRA MARIBEL PILACUAN ALEGRIA

El Trigal, calle principal No.192
Teléfono: 593-2-3202230
mpilacuan@unibanco.fin.ec
Quito–Ecuador

Educación:

- 2005. Escuela Politécnica del Ejército. Título obtenido: Ingeniera en Sistemas.

Cursos y Seminarios:

- 2000. Mantenimiento de computadoras Escuela Politécnica del Ejército.
- 2001. Microsoft. Visual Studio.Net.
- 2004.Curso de Herramientas COGNOS Power Play Enterprise Server, Metrics, Arquitect
- 2004. Metodología Microsoft Operations Framework, referente a dirección y ejecución de proyectos tecnológicos.
- 2005. Seminario Basilea II, Riesgos de Crédito y Ciclo del Cliente.

Experiencia:

- Febrero2004 - Actual. Unibanco - Oficial de Sistemas de Información.
- Mayo 2003-Febrero2004. Produbanco.- Analista Señor.
- Agosto 2002 – Mayo 2003. OEA – UPD.- Asistencia Técnica al TSE.
- Enero 2002 – Julio 2002. Seriva Inc.- Analista de Sistemas.
- Enero 1999 – Diciembre 2001. Areasistemas - Analista de Sistemas.
- Enero 1997 – Diciembre 1998. ESPE – Prácticas en los Laboratorios.

Proyectos realizados

- 2004- Actual Unibanco. Oficial de sistemas de información, equipo de datawarehuse. Coordinación del área de sistemas del Dpto. de Riesgos: diseño de base de datos, planificación de Tareas y cronogramas del grupo de trabajo.
  Desarrollo de Sistemas de Información Gerencial.
  Reingeniería del Datawahouse existente.
  Extracción de información destinada al Scoring de Cobranzas, Scoring de asignación de cupos de crédito para el producto TCF.
  Administración de Base de Datos, manejo de herramientas ETL (Extract, Transform & Load). Actualmente el área está realizando la planificación y el levantamiento de información del proyecto PLANING, para presupuestación del Dpto, de Control Financiero.
  Tecnologías utilizadas: Windows, Sql Server 2000
  Herramientas: Analysis Services, Power Play, Cognos Metrics

- 2004 Produbanco. Analista senior equipo de datawarehuse. Participación en el desarrollo, implementación e implantación de 15 modelos de Datawarehouse para Produbanco, modelos (Clientes, Productos, Cuentas, Portafolios, Balances, Líneas de Crédito, Análisis de Riesgo), bajo el proyecto Prometeus.
  Tecnologías utilizadas: Windows, Sql Server 2000
  Herramientas: Analysis Services, Power Play, Cognos Metrics

- 2003 OEA-UPD. Asistencia técnica brindada al TSE.
  Tecnologías utilizadas: Windows, ORACLE, Visual Basic 6.0

- 2002 Seriva. Analista de sistemas: implementación del módulo de cotizaciones para el sistema de Tesorería “Seriva.Net”.
  Tecnologías utilizadas: SQL Server 2000, Jaguar, Power Builder 8.0 , Crystal Reports.

- 1999 – 2001 Areasistemas. Analista de sistemas a cargo de varios proyectos:
  - Cepam: Sistema de Admistración y seguimiento a usuarias.
  - Fundación natura: Sistema de control de la opacidad para gasolina y diesel.
  - Ministerio del ambiente: Sistema para el control de emisiones de fuentes móviles.

  Tecnologías utilizadas: Visual Basic 6.0, Power Builder, Oracle, SQL Server.

Idiomas:

- Inglés: suficiencia.

HOJA DE LEGALIZACION DE FIRMAS

ELABORADA POR:

_____

Sandra Maribel Pilacuan Alegría

_____

Mercy Geovanna Pinargote Alarcón

DECANO DE LA FACULTAD DE INGENIERIA

_____

Tcrn. de E.M. Marco Quintana C.

Lugar y fecha: SANGOLQUÍ, 30 DE NOVIEMBRE DE 2005