ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

DISEÑO E IMPLEMENTACIÓN DE UNA CAPA DE PERSISTENCIA DE OBJETOS DE UNA BASE DE DATOS RELACIONAL COMO BLOQUE DE DATOS EN UNA RED LAN MULTICAPA, UTILIZANDO LA ARQUITECTURA MICROSOFT .NET Y EL LENGUAJE C#.

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN INFORMÁTICA MENCIÓN REDES

MAURICIO ALEJANDRO MOLINA DÍAZ
mauri_molina75@hotmail.com

DIRECTOR: ING. FRANCISCO VILLAVICENCIO
francisco.villavicencio@epn.edu.ec

QUITO, NOVIEMBRE 2010

DECLARACION

Yo, Mauricio Alejandro Molina Díaz, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento. A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Mauricio Alejandro Molina Díaz

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Mauricio Alejandro Molina Díaz, bajo mi supervisión.

Ing. Francisco Villavicencio
DIRECTOR DEL PROYECTO

AGRADECIMIENTO

Agradezco a mi director de proyecto, Ing. Francisco Villavicencio, gracias a su orientación y conocimiento he podido culminar este proyecto de la mejor manera.

DEDICATORIA

A mis padres por su apoyo incondicional en todos los aspectos de mi vida y por siempre creer en mí. A mi esposa Johanna y a mis hijas Ana Paula y Daniela por su paciencia y ayuda en la culminación de este proyecto, también a mis hermanas Myrian y Carla por sus palabras de aliento.

CONTENIDO

RESUMEN

El desarrollo de este proyecto consiste en el diseño y construcción de una capa de persistencia en software de una base de datos relacional. La base de datos relacional está implementada en SQL Server 2008 y corresponde a una institución financiera.

El mapeo de los objetos a la base de datos se realiza aplicando patrones de modelamiento, se especifican los fundamentos utilizados para el mapeo de objetos a bases de datos relacionales, en este proyecto no se hacen comparaciones entre diferentes métodos de mapeo de objetos.

Se estudian las características de la tecnología de Microsoft .NET y C#, y sus potencialidades que permitan la implementación de capas de persistencia de objetos de software de una base de datos relacional como parte de una arquitectura multicapa en una red LAN.

Para el desarrollo de la capa de persistencia se ha utilizado Microsoft Visual Studio 2010 Ultimate como entorno integrado de desarrollo y el lenguaje C# y Windows 7 como sistema operativo, el sistema del caso de estudio se probó en una red LAN.

INTRODUCCIÓN

La mayoría de las aplicaciones de negocios modernas utilizan las tecnologías de objetos como C# para construir las aplicaciones y bases de datos relacionales para almacenar los datos. Esto no quiere decir que no existen otras opciones y existen muchas aplicaciones escritas en lenguaje procedural como COBOL y muchos sistemas utilizan bases de datos XML para almacenar los datos. Sin embargo se puede decir que las tecnologías de objetos y relacionales son actualmente la norma en el desarrollo de aplicaciones.

El presente proyecto se plantea por la incompatibilidad del modelo de objetos y el modelo relacional de datos. El paradigma de objetos está basado en construir aplicaciones que contienen datos y comportamiento encapsulado, mientras que el modelo relacional está basado en almacenamiento persistente de datos.

Se necesita entonces proveer una estrategia efectiva para las necesidades de persistencia de las aplicaciones orientadas a objetos, esto es, el envío de mensajes estilo operaciones CRUD (Create, Retrieve, Update, Delete) del modelo de datos relacional.

Las bases de datos relacionales son importantes hoy en día para mantener la información, para esto se realizan básicamente operaciones CRUD para crear, recuperar, actualizar y eliminar datos. Las bases de datos pueden también utilizarse en aspectos más avanzados como el almacenamiento de objetos. Para almacenar objetos en una base de datos son necesarias técnicas para representar el esquema de objetos en el esquema relacional.

Las bases de datos pueden estar asociadas o vinculadas con otras entidades, mientras más vínculos tengan y más unidas se encuentren, son más difíciles de mantener. Las bases de datos pueden estar vinculadas al código de la aplicación, código de otras aplicaciones, código de carga de datos o migración, código de exportación de datos, esquemas de persistencia, código de pruebas, documentación e incluso el mismo esquema de datos.

Para ocultar detalles del esquema de almacenamiento persistente como las bases de datos y reducir las asociaciones con las entidades, se puede hacer uso de la encapsulación para mantener el acceso a los datos sin importar cómo se realiza internamente, así, los desarrolladores se focalizan en el negocio y no en cómo manejar la data. La encapsulación solventa también la incompatibilidad entre modelo de objetos y el modelo relacional mencionada anteriormente.

Al aplicar una estrategia de encapsulación a los datos, se realiza una abstracción del acceso y se muestra funcionalidad, pero internamente el acceso a los datos se realiza mediante lenguaje SQL estándar que consta del Lenguaje de Definición de Datos, que se encarga de la modificación; Lenguaje de Manipulación de Datos, para realizar tareas de consulta o manipulación; y Lenguaje de Control de Datos, para otorgar permisos y demás temas inherentes al control de acceso y seguridades, parte del SQL estándar son también los procedimientos almacenados y las funciones que sirven para dar cierta funcionalidad al sistema.

Los lenguajes de programación actuales más utilizados, por ejemplo el C#, son orientados a objetos, mediante objetos se desarrollan complejas aplicaciones de negocios y generación de información, por lo que es necesario hacer que esos datos persistan.

La persistencia de objetos es un método para hacer que los objetos que están en memoria persistan y puedan ser recuperados para posterior consulta y uso. Para implementar persistencia de objetos hay mecanismos como el escribir código que mapea directamente a una base de datos relacional, usar esquemas de mapeo de objetos a modelo relacional o el uso de bases de datos orientadas a objetos.

En este proyecto se utilizan patrones de modelamiento para el mapeo de objetos a una base de datos relacional. Un esquema de modelamiento simple consiste en el mapeo o representación de atributos de objetos en columnas, clases en tablas y mapeo de estructuras de herencia donde se pueden aplicar mecanismos como: Mapeo de todas las clases en una sola tabla, cada clase concreta en una tabla, cada clase en una tabla y el mapeo en una estructura genérica de datos, este

último está basado en meta-datos y soporta todas las formas anteriores de mapeo.

Para aplicar mecanismos de mapeo entre objetos y tablas, es necesario tomar en cuenta las relaciones, es decir, que las relaciones también deben mapearse de manera correcta para no perder u omitir funcionalidad, las relaciones pueden ser por multiplicidad: uno a uno, uno a muchos, muchos a muchos o por direccionalidad: unidireccionales o bidireccionales. Cada relación debe mapearse de manera distinta siguiendo una lógica definida.

Existen varios tipos de persistencia entre los cuales se tienen: Código de SQL incrustado en clases de negocio, código SQL incrustado en clases de datos separadas y una capa de persistencia que es una capa de arquitectura de software que abstrae el acceso a un mecanismo persistente de datos.

La arquitectura en capas es aceptada y recomendada para el desarrollo de aplicaciones por lo que se sugiere un modelo donde los usuarios interactúen con la capa de interfaz, el negocio se realice en la capa de negocio o de dominio y la manipulación de datos se realice en una capa de persistencia, también se tienen clases propias del sistema y clases controladoras o de proceso.

Las aplicaciones que se desarrollan actualmente no funcionan sobre computadores aislados sino que pueden ser utilizadas desde varios ordenadores en redes de computadoras, el almacenamiento y distribución de información se realiza en computadoras independientes pero interconectadas.

Una red de computadoras es un conjunto de dispositivos conectados entre sí, dentro de las redes de computadoras se tienen a las redes LAN que son redes privadas de pocos kilómetros en tamaño, se distinguen por su tamaño, tecnología y topología. Se dice que una red LAN es multicapa porque está basada en las capas y protocolos del modelo OSI que divide a la red en capas, el protocolo más utilizado es el TCP/IP que está dividido en 5 capas: Acceso a la red, Internet, Transporte y Aplicación.

El desarrollo formal de aplicaciones requiere el uso de una metodología, la metodología que se utiliza es el Proceso Unificado de Rational o RUP por sus siglas en inglés: Rational Unified Process. El RUP es un proceso dirigido por casos de uso que capturan los requisitos de un sistema, está centrado en la arquitectura, es iterativo e incremental. El RUP tiene las siguientes fases: Inicio, Elaboración, Construcción y Transición. El RUP es la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos, no es un sistema con pasos sólidamente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada proyecto.

Las herramientas de lenguaje utilizadas para el desarrollo son el .NET y como entorno integrado de desarrollo el Microsoft Visual Studio .Net 2010 Ultimate. Se utiliza para la programación el lenguaje C# 4.0 que es un lenguaje creado desde cero y LINQ como extensión del lenguaje. Las tecnologías .NET son de las más significativas para el desarrollo actualmente y están completamente orientadas a objetos. La base de datos que se utiliza es Microsoft SQL Server 2008 que es un producto robusto de base de datos relacional que utiliza T-SQL como lenguaje de programación.

Los requerimientos para el desarrollo del caso de estudio son básicamente negociaciones de compra y venta de divisas para una entidad financiera. Como parte del análisis y diseño se tienen los siguientes artefactos: Especificación de requerimientos, Modelo de casos de uso, Especificación de casos de uso, Diagramas de actividad, Glosario de términos, Diagramas de clases, Modelo físico de datos y Diccionario de datos.

Las pruebas de la aplicación son básicamente las operaciones básicas de acceso a datos o funcionalidad CRUD, las pruebas incluyen grabación de operaciones: compra y venta, aprobación de operaciones, anulación de operaciones.

CAPÍTULO 1. MARCO TEORICO

1.1 PERSISTENCIA Y BASES DE DATOS RELACIONALES

Un aspecto importante en la mayoría de las empresas es el almacenamiento de información. En nuestra sociedad de información esto se ha convertido en un aspecto importante de negocios y gran parte de la capacidad informática del mundo está dedicada a mantener y utilizar bases de datos.

Las bases de datos están en todo tipo de negocios donde toda clase de datos como registro de ventas, datos financieros, datos de clientes, correos electrónicos, información de contacto, calificaciones y registros estudiantiles y demás, son almacenados en alguna forma de base de datos.

1.1.1 BASES DE DATOS RELACIONALES

Una base de datos relacional es un mecanismo de almacenamiento persistente que permite almacenar datos y opcionalmente implementar funcionalidad.

Las bases de datos relacionales almacenan los datos en tablas, las tablas están organizadas en columnas y cada columna almacena un tipo de dato como entero, cadena de caracteres, fechas, etc. Los datos de una sola instancia de la tabla se almacenan como una columna. Por ejemplo la tabla Empleado puede tener las columnas Cédula, Nombre y Cargo y una fila en la tabla puede ser {0704583087, “Johanna”, “Gerente”}.

Las tablas típicamente tienen claves, las claves son una o más columnas que identifican de manera única a una fila en la tabla, en el caso de la tabla Empleado, la clave puede ser la columna Cédula. Para mejorar el tiempo de acceso a una tabla de datos se define un índice en la tabla.

Un índice provee una manera rápida de buscar datos basados en una o más columnas en la tabla tal como el índice de los libros que permiten buscar información específica rápidamente. [26]

1.1.1.1 CARACTERÍSTICAS DE LAS BASES DE DATOS RELACIONALES

El uso más común de las bases de datos relacionales tiene que ver con la actualización, eliminación, lectura y actualización de los datos que contiene, a esta funcionalidad se le conoce como CRUD (Create, Retrieve, Update, Delete) que significa Crear, Recuperar, Actualizar y Eliminar respectivamente. [9]

Es posible por ejemplo, crear un empleado nuevo e insertarlo en la base de datos, puede leer un registro existente, trabajar con ese registro y después actualizar la base de datos con la nueva información, es posible también eliminar un empleado posiblemente por un cambio de empresa del mismo.

La mayoría de la interacción con la base de datos tiene que ver con operaciones CRUD y la manera más sencilla de manipular una base de datos es mediante SQL.[1]

1.1.1.2 USO AVANZADO DE LAS BASES DE DATOS RELACIONALES

Existen algunas características más avanzadas que la funcionalidad CRUD en las bases de datos relacionales, estas características son importantes y algunas muy complejas, estas características incluyen:

Almacenamiento de objetos: Para almacenar un objeto en una base de datos relacional es necesario hacerlo plano, es decir, crear una representación de datos del objeto porque las bases de datos relacionales solamente almacenan datos. Para recuperar el objeto, es necesario leer los datos desde la base de datos y crear o restaurar el objeto el basado en esos datos. Aunque el almacenar objetos

[1] SQL (Structured query Language) Lenguaje Estructurado de Consultas que sirve para definir, manipular y consultar la base de datos.

[26] Tore Bostrup, *Introduction to Relational Databases - Part 1: Theoretical Foundation.*

[9] Kauffman, John., *Beginning ASP.NET Databases Using VB.NET*

en una base de datos relacional parece algo simple, en la práctica se demuestra que no lo es. Esto se debe a la incompatibilidad del modelo de objetos con el modelo relacional porque la tecnología del modelo relacional y la tecnología del modelo de objetos están basadas en diferentes teorías. Para almacenar objetos de forma correcta en una base de datos relacional es necesario aprender cómo representar el esquema de objetos en el esquema de base de datos relacional.

Implementación de comportamiento en la base de datos. Se implementa comportamiento en una base de datos relacional mediante procedimientos almacenados y/o funciones que pueden ser invocados internamente en la base de datos y algunas veces por aplicaciones externas. Los procedimientos almacenados y las funciones son operaciones que corren dentro de una base de datos relacional, las diferencias entre las funciones y los procedimientos almacenados no son importantes para el presente estudio así que se hará referencia a las funciones y procedimientos como procedimientos almacenados. Generalmente los procedimientos almacenados ejecutan código SQL, manipulan los datos y luego envían una respuesta de cero o más registros o un código de respuesta o un mensaje de error de base de datos. [1]

Control de concurrencia. La concurrencia se refiere al acceso simultáneo a datos específicos, se puede considerar por ejemplo un sistema de reservación de boletos aéreos donde existe un asiento y dos personas tratando de reservar el lugar, ambas personas revisan el estado del vuelo y miran el asiento disponible, ingresan los datos requeridos y hacen clic en reservar al mismo tiempo. Si el sistema está implementado correctamente, una sola persona tomará ese asiento y a la otra se le indicará que ese asiento ya no está disponible, esto es posible gracias al control de concurrencia que debe ser implementado en al código del objeto y en la base de datos.

Control de transacción. Una transacción es una colección de acciones en la base de datos que conforman una unidad de trabajo. Las transacciones planas

---

[1] Ambler, Scott W., *Agile Database Techniques: Effective Strategies for the Agile Software Developer.*

son de tipo “todo o nada” donde todas las acciones deben ejecutarse con éxito o todas deben ser canceladas.
Una transacción anidada es una transacción donde alguna de sus acciones son transacciones. Estas transacciones anidadas no son canceladas si la transacción que las contiene es cancelada.

Implementación de integridad referencial. La integridad referencial es la certeza de que la referencia de una entidad a otra entidad es válida. Por ejemplo, si un cliente referencia una dirección, entonces esa dirección debe existir. Si la dirección es eliminada, entonces todas las referencias a ésta deben ser removidas, caso contrario el sistema no debe permitir su eliminación. La integridad referencial no es un asunto solo de la base de datos sino de todo el sistema.

1.1.1.2.1 Asociaciones o Vínculos de las bases de datos

Los vínculos de las bases de datos son una medida del grado de dependencia entre dos entidades, mientras más unidas están es mayor el cambio que se requiere en una entidad cuando se cambia la otra. Si existen más ítems asociados o unidos al esquema de la base de datos es más difícil de mantener. Los esquemas de bases de datos pueden estar unidos o vinculados a las siguientes entidades: [26]

El código de la aplicación. Cuando se cambia el esquema de la base de datos es necesario cambiar el código fuente de la aplicación que accede a la porción del esquema modificado. La Figura 1 muestra el mejor caso donde solamente el código de la aplicación está unido al esquema de la base de datos. Estas situaciones existen y son conocidas como aplicaciones “Stand Alone” aunque hoy en día estas aplicaciones son escasas.

---

[26] Tore Bostrup, *Introduction to Relational Databases - Part 1: Theoretical Foundation.*

Código fuente de otras aplicaciones. La Figura 2 describe la peor situación para las bases de datos relacionales, una amplia gama de sistemas están vinculados o unidos al esquema de la base de datos, esta situación es común con las bases de datos existentes en los negocios. Es común además encontrar otras aplicaciones diferentes asociadas al esquema de la base de datos sobre las cuales no se tiene mucho conocimiento, también es posible que sistemas que están en el Internet lean o escriban datos a la base de datos o que un usuario tenga una hoja de cálculo que importa datos desde la base de datos. Con estos casos es complicado realizar un cambio en el esquema de la base de datos ya que se afectan muchas aplicaciones propias o de terceros.

Código de carga de datos. La carga de datos desde otras fuentes como por ejemplo archivos planos que son enviados desde instituciones gubernamentales pueden estar asociados al esquema de la base de datos.

Código de exportación de datos. Pueden existir programas o "scripts"[2] que realizan exportación de datos desde el esquema de base de datos hacia otras bases de datos mediante archivos XML, archivos planos o simplemente pasando los datos de un esquema a otro.

Esquemas o capas de persistencia. Los esquemas de persistencia encapsulan la lógica para representar fuentes de almacenamiento persistente en clases de código de la aplicación. Cuando se modifica el esquema de la base de datos, es necesario actualizar el código del esquema de persistencia o los metadatos que describen el esquema.

Asociación en el mismo esquema. Existen vínculos dentro del mismo esquema de la base de datos. Una simple columna está asociada a un procedimiento almacenado que hace referencia a la misma, a otras tablas que tienen esa

[2] Script es un conjunto de instrucciones que permiten la automatización de tareas creando utilidades pequeñas.

columna como clave foránea, una vista que referencia la columna y así por el estilo. Un cambio simple puede resultar en muchos cambios en la base de datos.

Scripts de migración o creación de datos. Los cambios al esquema de la base de datos requieren que se modifiquen los scripts de migración o creación asociados.

Código de pruebas. El código de pruebas incluye código que pone a la base de datos en un estado conocido, se realizan transacciones y se comparan con resultados esperados. Los cambios en el esquema de la base de datos hacen necesario el cambio de este código de pruebas que puede ser de pruebas unitarias.

Documentación. Parte de la documentación importante de un sistema pertenece al esquema físico de la base de datos, incluyendo el modelo físico de datos y los meta-datos que describen el esquema. Cuando se cambia el esquema de la base de datos, es necesario cambiar la documentación asociada.

Como se ha explicado, las asociaciones de entidades con el esquema de la base de datos son un asunto importante que generalmente se ignora en la teoría de diseño de bases de datos y normalización y que para el caso del presente estudio se maneja mediante una capa de persistencia y el mapeo de objetos.

La tecnología tiene fortalezas y debilidades y las bases de datos relacionales no son la excepción, afortunadamente existen técnicas que son útiles para mantener el código y el esquema de la base de datos y disminuir las complicaciones que se puedan presentar en casos como el de la figura 2. Una de las técnicas más útiles es la encapsulación.

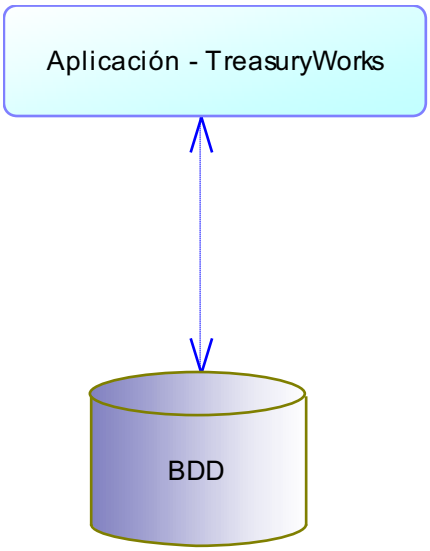


Figura 1. El mejor escenario de asociaciones con base de datos.

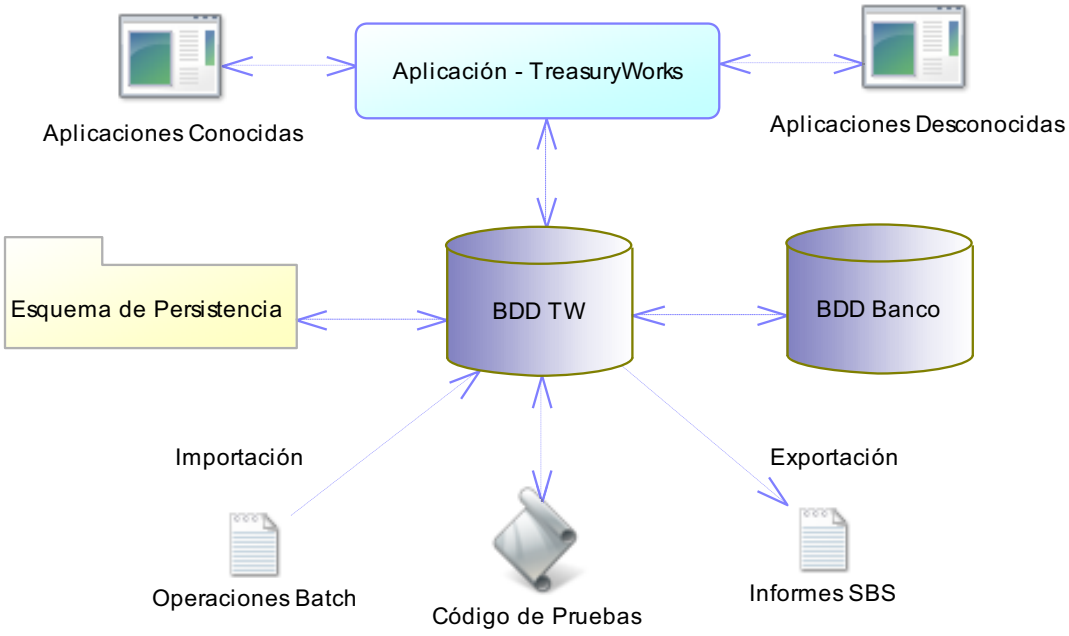


Figura 2. El peor escenario de asociaciones con bases de datos relacionales.

1.1.1.2.2 Encapsulación

La encapsulación es un esquema de diseño que se encarga de cómo la funcionalidad está integrada en un sistema. No es necesario saber cómo está implementado un objeto para usarlo. El resultado de la encapsulación es que se puede desarrollar cualquier componente u objeto y luego cambiar la implementación sin afectar otros componentes en el sistema (siempre y cuando no se afecte la interfaz con la que interactúan los componentes).

La encapsulación es como una “caja negra” donde se define qué procesos o acciones se van a realizar pero no se especifica cómo se desarrollan o cómo se logran los resultados. Por ejemplo considérese un cajero automático donde se realizan transacciones, no importa qué sistema operativo usa, ni qué base de datos tiene ni qué procesador utiliza el computador del banco, lo importante es que se pueden realizar transacciones y se almacena la información de la cuenta bancaria. [22]

Encapsulando el acceso a la base de datos mediante algo simple como objetos de datos o algo más complejo como un mecanismo de persistencia se pueden reducir las asociaciones o uniones de objetos o componentes con la base de datos. La encapsulación del acceso a la base de datos esconde los detalles del esquema de la base de datos a los desarrolladores pero mantiene el acceso a la base. Los desarrolladores del esquema de encapsulación son responsables del mantenimiento del mismo.

La mejor ventaja de encapsular el acceso a la base de datos es que habilita a los desarrolladores de aplicaciones focalizar sus esfuerzos en el desarrollo del negocio y no en la forma de acceder o mantener los datos. Por ejemplo si se tienen objetos simples de acceso a datos que implementan código SQL para acceder al esquema de base de datos. Los desarrolladores de la aplicación trabajarán con las clases de acceso a los datos y no directamente con el esquema, esto facilita el cambio del esquema de la base de datos y la

[22] Ambler, Scott W., *Encapsulating Database Access: An Agile "Best" Practice.*

actualización de las clases de acceso sin modificar el código del negocio de la aplicación.

La figura 3 describe el concepto de encapsular el acceso a la base de datos, se indica cómo cambia el caso de la figura 1 y la figura 2 cuando se utiliza la encapsulación. En el mejor caso la aplicación interactúa con las clases de acceso a los datos en lugar de la base directamente. La ventaja es que todo el código relacionado con el acceso a los datos está en un solo lugar haciéndolo fácil de modificar cuando cambia el esquema de la base de datos. El código del negocio de la aplicación aún está asociado con las clases de acceso a datos pero es necesario modificarlo solo si cambia la interfaz de acceso con las clases. Con una estrategia de encapsulación de acceso a la base de datos los programadores se preocupan del código fuente y no del código fuente más código SQL de la base de datos.

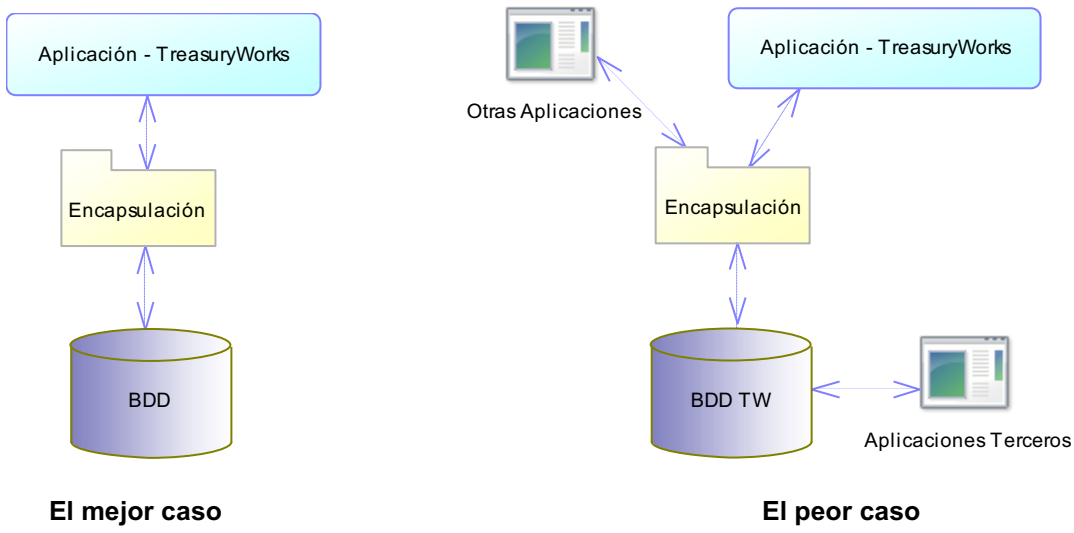


Figura 3. El mejor y peor escenario de asociaciones con la base de datos usando encapsulación.

En el peor caso en la figura 3 no todo es perfecto, aunque es posible que todas las aplicaciones puedan tomar ventaja de la encapsulación, en realidad, solamente una parte puede beneficiarse. La incompatibilidad de las plataformas

es un problema, puede ser que objetos de acceso a los datos están escritos en C# y otras aplicaciones antiguas están generadas en un lenguaje que no puede acceder fácilmente a C#. Probablemente es necesario rehacer parte de las aplicaciones antiguas para que se beneficien del uso de la encapsulación desarrollada, es posible también que ya exista encapsulación por lo que se debe considerar reutilizar lo existente en lugar de crear una nueva.
Es preciso a veces acceder directamente a la base de datos por ejemplo para hacer una carga masiva desde un archivo, el punto es que no todo puede beneficiarse de la encapsulación y es importante diferenciar qué se puede cambiar y qué se debe mantener.

Aún así una técnica de encapsulación es un beneficio para reducir costos de desarrollo y mantenimiento. Otra ventaja de encapsular el acceso a la base de datos es que se tiene un lugar común, a parte de la base de datos, donde implementar reglas de negocio orientadas a los datos.

1.1.1.3 INCOMPATIBILIDAD DEL MODELO DE OBJETOS Y EL MODELO RELACIONAL

Una realidad al construir aplicaciones orientadas a objetos es que la única manera de hacer persistir los objetos es mediante las bases de datos relacionales. Debido a la llamada incompatibilidad Modelo de Objetos – Modelo Relacional (esto es, el deficiente mapeo de la tecnología de objetos que utiliza localización de memoria para identificación de objetos, sobre bases de datos relacionales que utilizan claves primarias para la identificación de objetos) ambos tienen una relación de funcionamiento complejo. Las complejidades técnicas para que ambos modelos trabajen tienen que ver con la división objeto – datos. [25]

La división objeto – datos se refiere específicamente a las dificultades de los desarrolladores al trabajar con técnicas de orientación a objetos y técnicas de orientación a datos, la división objeto – datos incluye a los desarrolladores de objetos que reclaman que la tecnología de datos no se puede usar para

[25] Ambler, Scott W., *The Object-Relational Impedance Mismatch.*

almacenar objetos, y los profesionales de datos (modeladores de datos y administradores de bases de datos) que dicen que los modelos de objetos deben ser administrados por sus modelos de datos.

Esta división objeto – datos tiene efectos perjudiciales en el desarrollo de sistemas como son:

- Impiden al departamento de TI[3] producir software a tiempo y dentro del presupuesto.
- Si los modeladores de objetos y los modeladores de datos no pueden trabajar juntos, el esquema de objetos y el esquema de datos pueden ser incompatibles. Mientras mayor sea la incompatibilidad de esquemas es necesario escribir probar y mantener más código; este código es probablemente más lento que el código simple cuando se tienen esquemas compatibles y coordinados.
- Los modelos de objetos (diagramas de clases UML, diagramas de secuencia, etc.) orientados por el diseño de datos existente es efectivamente un trabajo de mucho esfuerzo ya que como se sabe los requerimientos son los que orientan el desarrollo de modelos en la ingeniería de software – No se inicia en el diseño.

1.1.1.4 ENCAPSULACIÓN DEL ACCESO A LA BASE DE DATOS

Como se indicó anteriormente, la encapsulación es una técnica efectiva para solventar la incompatibilidad del modelo de objetos y el modelo relacional. El paradigma de objetos está basado en construir aplicaciones que contienen datos y comportamiento encapsulado, mientras que el modelo relacional está basado en almacenamiento persistente de datos.

[3] TI. Tecnologías de Información.

La encapsulación es una estrategia útil para las necesidades de persistencia de las aplicaciones orientadas a objetos. [3]

En el desarrollo de software se utilizan las técnicas de encapsulación para la generación de capas de encapsulación de bases de datos lo cual se explica a continuación.

1.1.1.4.1 Capas de encapsulación de bases de datos

Una capa de encapsulación de base de datos esconde los detalles de implementación de la base o las bases de datos, incluyendo los esquemas físicos, al código de negocio de la aplicación. De hecho, se provee a los objetos de negocio con servicios de persistencia – la capacidad de leer datos desde, escribir datos hacia y eliminar datos desde – orígenes de datos. Las capas de encapsulación de bases de datos no son una solución mágica y no son teorías académicas; las capas de encapsulación de bases de datos son una práctica utilizada en forma común en aplicaciones pequeñas, grandes, simples y complejas. [22]

Una capa de encapsulación de bases de datos eficaz provee los siguientes beneficios:

- Reduce el acoplamiento entre en esquema de objetos y el esquema de datos, lo cual posibilita que ambos se desarrollen y evolucionen por separado.
- Implementa todo el código relacionado con los datos en un solo lugar.
- Simplifica el trabajo de los desarrolladores de software.
- Permite a los desarrolladores enfocarse en los problemas de negocio y a los administradores de bases de datos enfocarse en los esquemas de datos.
- Provee un lugar común donde implementar reglas de negocio.

---

[3] Bartels, Dirk.; Benson, Robert., *Object Persistence and Agile Software Development.*

[22] Ambler, Scott W., *Encapsulating Database Access: An Agile “Best” Practice.*

- Se saca el máximo beneficio de características específicas de bases de datos lo cual incrementa el rendimiento de la aplicación.

También se pueden enumerar algunas desventajas del uso de capas de encapsulación de bases de datos:

- Se requiere de inversión.
- Es necesario bastante mapeo o representación de objetos.
- Puede proveer poco control del acceso a la base de datos

1.1.1.4.2 Roles del personal involucrado

La figura 4 describe la interacción de los actores involucrados en el proceso de encapsulación. Se ha determinado que el Administrador de la Base de Datos (DBA *DataBase Administrator* por sus siglas en inglés) interactúe con el proceso de encapsulación para evitar alteración de datos o del modelo original, el arquitecto de software determina la estrategia de encapsulación dependiendo del ambiente y tecnología y el desarrollador implementa la capa de encapsulación basado en la estrategia para así lograr el acceso a la base de datos.

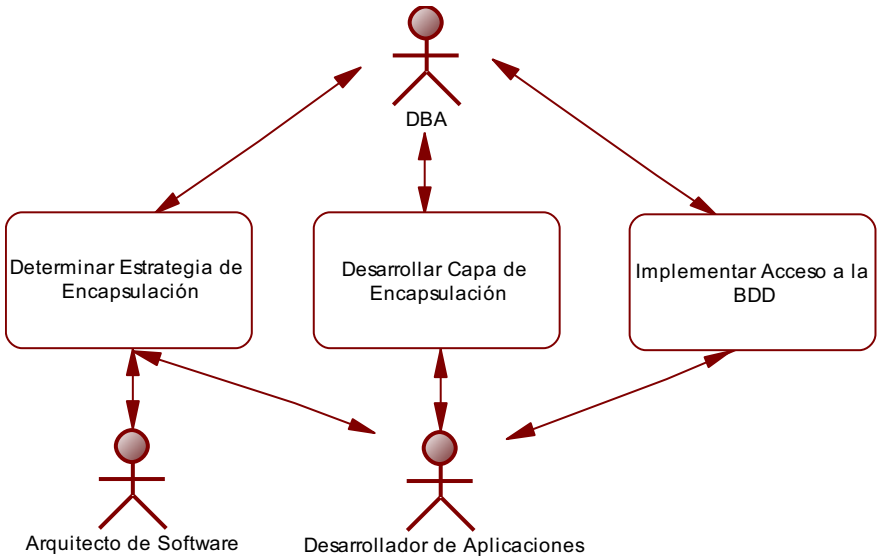


Figura 4. Descripción de interacción de actores en relación a capas de encapsulación

1.1.1.4.3 Arquitectura de la Capa de Encapsulación

La figura 5 describe la arquitectura más simple para encapsular el acceso a una base de datos relacional – la aplicación interactúa con una base de datos. En esta situación existe mayor flexibilidad para que el equipo de trabajo elija la estrategia de encapsulación que más se ajuste a su situación. Adicionalmente, es posible evolucionar el esquema de objetos y el esquema de la base de datos cuando se implementan nuevos requerimientos.

1.1.1.5 SQL ESTÁNDAR

El lenguaje de consulta estructurado SQL (Structured Query Language por sus siglas en inglés) es un lenguaje declarativo[4] de acceso a bases de datos relacionales mediante el cual es posible especificar diferentes operaciones. [27]
Aunque el SQL es un estándar ANSI[5]e ISO[6] algunos productos de bases de datos tienen sus propias extensiones al SQL estándar. Dos de sus características principales son el manejo de álgebra[7] y el cálculo relacional[8] que permiten realizar consultas o modificaciones en la base de datos de una manera relativamente sencilla, es un lenguaje de cuarta generación (4GL[9]).

[4] Lenguaje declarativo es aquel en el que se indica a la computadora qué es lo que se quiere obtener.
[5] Instituto Nacional Estadounidense de Estándares (American National Standards Institute – ANSI por sus siglas en inglés)
[6] Organización Internacional para la Estandarización (International Organization for Standarization – ISO por sus siglas en inglés)
[7] El álgebra relacional es un conjunto de operaciones que describen cómo calcular una respuesta sobre las relaciones, tal y como están definidas en el modelo relacional.
[8] El cálculo relacional es un lenguaje de consulta que describe la respuesta deseada sobre una base de datos sin especificar cómo obtenerla, a diferencia del álgebra relacional que es de tipo procedural, el cálculo relacional es de tipo declarativo; pero siempre ambos métodos logran los mismos resultados.
[9] Lenguaje de cuarta generación (4 Generation Language - 4GL por sus siglas en inglés) Es un lenguaje de alto nivel que se parece al idioma inglés.

[27] Chapple, Mike. *SQL Fundamentals.*

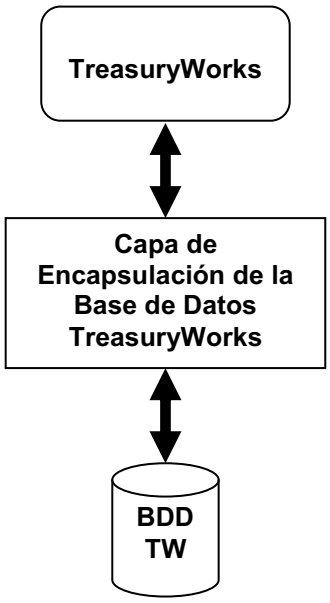


Figura 5. Una sola aplicación, una sola arquitectura base de datos

Orígenes y evolución

Los orígenes del SQL están unidos a los de las bases de datos relacionales. En 1970 Edgar Frank “Ted” Codd propone el modelo relacional y asociado a este un lenguaje de acceso a los datos basado en el cálculo de predicados. [29]

Con base en estas ideas, los laboratorios de IBM definen el lenguaje SEQUEL Lenguaje de Consulta Estructurado en Inglés (Structured English Query Language por sus siglas en inglés) que más tarde sería ampliamente implementado por el SGBD (Sistema Gestor de Bases de Datos) experimental “System R”, desarrollado en 1977 por IBM. Sin embargo, fue Oracle quien lo introdujo por primera vez en 1979 en un programa comercial.

El SEQUEL fue el predecesor de SQL, que es una versión evolucionada del primero. El SQL pasa a ser el lenguaje por excelencia de los diversos SGBD relacionales surgidos en los años siguientes y es por fin estandarizado en 1986 por el ANSI, dando lugar a la primera versión estándar de este lenguaje, el "SQL-

[29] IBM, *History of IBM, 1978 (HTML). IBM Archives*.

86" o "SQL1". Al año siguiente este estándar es también adoptado por la ISO.
Este primer estándar no cubre todas las necesidades de los desarrolladores e incluye funcionalidades de definición de almacenamiento que se consideraron suprimir. Así que en 1992 se lanza un nuevo estándar ampliado y revisado del SQL llamado "SQL-92" o "SQL2".

En la actualidad el SQL es el estándar *de facto*[10] de la inmensa mayoría de los SGBD comerciales.

Evolución del estándar:

Año	Nombre	Alias	Comentarios
1986	SQL-86	SQL-87	Primera publicación hecha por ANSI. Confirmada por ISO en 1987.
1989	SQL-89		Revisión menor.
1992	SQL-92	SQL2	Revisión mayor.
1999	SQL:1999	SQL2000	Se agregaron expresiones regulares, consultas recursivas (para relaciones jerárquicas), triggers y algunas características orientadas a objetos.
2003	SQL:2003		Introduce algunas características de XML, cambios en las funciones.
2006	SQL:2006		ISO/IEC 9075-14:2006 Define las maneras en las cuales el SQL se puede utilizar conjuntamente con XML. Define maneras importar y guardar datos XML en una base de datos SQL, manipulándolos dentro de la base de datos y publicando el XML y los datos SQL convencionales en forma XML.

[10] Un estándar de facto es aquel que no ha sido legitimado por un organismo de estandarización al efecto, se trata de una norma generalmente aceptada y ampliamente utilizada por iniciativa propia de un gran número de interesados.

2008	SQL:2008		El 17 de julio de 2008 en la oficina de la ISO en Génova se comunicó la adopción formal del estándar SQL: 2008. Entre otras cosas añade lo siguiente: Legaliza el ORDER By fuera de las definiciones de cursores. Añade disparadores tipo "INSTEAD OF". Agrega la sentencia TRUNCATE.

1.1.1.6 COMANDOS

El lenguaje SQL Estándar puede ser divido en 3 partes:

Lenguaje de Definición de Datos (LDD).
Lenguaje de Manipulación de Datos (LMD).
Lenguaje de Control de Datos (LCD).

Lenguaje de Definición de Datos (LDD)
El lenguaje de definición de datos (en inglés *Data Definition Language*, o *DDL*), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Existen cuatro operaciones básicas: CREATE, ALTER, DROP y TRUNCATE. [27]

CREATE: Este comando crea un objeto dentro de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte.

ALTER: Este comando permite modificar la estructura de un objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, modificar un trigger, etc.

---

[27] Chapple, Mike. (HTML). *SQL Fundamentals. About.com: Databases*. About.com.

DROP: Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte. Se puede combinar con la sentencia ALTER.

TRUNCATE: Este comando trunca todo el contenido de una tabla. La ventaja sobre el comando DELETE, es que si se quiere borrar todo el contenido de la tabla, es mucho más rápido, especialmente si la tabla es muy grande, la desventaja es que TRUNCATE solo sirve cuando se quiere eliminar absolutamente todos los registros, ya que no se permite la cláusula WHERE. Si bien, en un principio, esta sentencia parecería ser DML (Lenguaje de Manipulación de Datos), es en realidad una DDL, ya que internamente, el comando truncate borra la tabla y la vuelve a crear y no ejecuta ninguna transacción.

Lenguaje de Manipulación de Datos (LMD)

Un lenguaje de manipulación de datos (*Data Manipulation Language*, o *DML* en inglés) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios de la misma llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.

El lenguaje de manipulación de datos más popular hoy día es SQL (en este informe se presenta el estándar SQL-92), usado para recuperar y manipular datos en una base de datos relacional. Otros ejemplos de DML son los usados por bases de datos IMS/DL1, CODASYL u otras.

INSERT: Una sentencia *INSERT* de **SQL** agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

UPDATE: Una sentencia *UPDATE* de **SQL** es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

DELETE: Una sentencia *DELETE* de **SQL** borra cero o más registros existentes en una tabla.

Lenguaje de Control de Datos (LCD)

Un lenguaje de Control de Datos (*Data Control Language, o DCL en inglés*). Sirve para otorgar permisos a los objetos de la base de datos a los usuarios o grupos de usuarios. Permite el control de los datos y cómo se accede a ellos por parte de los usuarios.

1.1.1.7 CLÁUSULAS

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular. [28]

FROM

Utilizada para especificar la tabla de la cual se van a seleccionar los registros.

WHERE

Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.

GROUP BY

Utilizada para separar los registros seleccionados en grupos específicos.

HAVING

Utilizada para expresar la condición que debe satisfacer cada grupo.

ORDER BY

Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.

---

[28] Drake, Joshua, *Understand SQL*.

1.1.1.8 OPERADORES LÓGICOS

AND

Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad solamente si ambas condiciones son ciertas.

OR

Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos condiciones es cierta.

NOT

Negación lógica. Devuelve el valor contrario de la expresión. [30]

1.1.1.9 OPERADORES DE COMPARACIÓN

Sirven para comparar valores.

<	Menor que.
>	Mayor que.
<>	Distinto de.
<=	Menor o igual que.
>=	Mayor o igual que.
=	Igual que.
BETWEEN	Se utiliza para especificar un intervalo de valores.
LIKE	Se utiliza en la comparación de un modelo.
IN	Se utiliza para especificar registros de una base de datos. [27]

---

[30] IBM, *Structured Query Language (SQL).*

[27] Chapple, Mike. (HTML). *SQL Fundamentals*.

1.1.1.10 FUNCIONES DE AGREGADO

Las funciones de agregado son útiles dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo. [28]

AVG

Sirve para calcular el promedio de los valores de un campo determinado.

COUNT

Sirve para devolver el número de registros de la selección.

SUM

Es usada para devolver la suma de todos los valores de un campo determinado.

MAX

Es útil para devolver el valor más alto de un campo especificado.

MIN

Sirve para devolver el valor más bajo de un campo especificado.

1.1.1.11 PROCEDIMIENTOS ALMACENADOS

Un procedimiento almacenado (*stored procedure*) es un programa (o procedimiento) el cual es almacenado físicamente en una base de datos.

Generalmente son escritos en un lenguaje de bases de datos propietario como PL/SQL para Oracle database o PL/PgSQL para PostgreSQL. [20]

---

[28] Drake, Joshua, *Understand SQL*.

[20] Vieira, Robert., *Microsoft SQL Server 2008 Programming*.

La ventaja de un procedimiento almacenado es que al ser ejecutado, en respuesta a una petición de usuario, es ejecutado directamente en el motor de bases de datos, el cual usualmente corre en un servidor separado. Como tal, posee acceso directo a los datos que necesita manipular y solo necesita enviar sus resultados de regreso al usuario, deshaciéndose de la sobrecarga resultante de comunicar grandes cantidades de datos salientes y entrantes.

1.1.2 PERSISTENCIA DE OBJETOS

La persistencia de objetos es un concepto de programación que surge de la programación orientada a objetos. Utilizando diseño y lenguaje programación orientados a objetos, como C#, se hace evidente y beneficioso el considerar alguna forma de persistencia de objetos.

La persistencia de objetos es un método para hacer que objetos “transitorios” sobrevivan al ciclo de vida de los objetos almacenados en memoria por la aplicación mediante un mecanismo de persistencia que es independiente de las estructuras de datos encapsuladas de esos objetos. [3]

Los objetos “persistentes” serán almacenados, típicamente en una base de datos en disco, de esta forma, los objetos pueden ser cargados en memoria cuando se necesite aún después de reiniciar la aplicación. Los objetos persistentes mantienen su estado y comportamiento en la aplicación.

Existen varios mecanismos para implementar la persistencia de objetos:

- Escribir manualmente el código que mapea objetos de la base de datos relacional.

---

[3] Bartels, Dirk.; Benson, Robert., *Object Persistence and Agile Software Development.*

- Utilizar Esquemas de Mapeo de Objetos a Modelo Relacional, ORM (Objects to Relational Mapping por sus siglas en inglés) o Interfaces de Programación de Aplicación (APIs por sus siglas en inglés) como por ejemplo: Hibernate, Java Data Objects (JDO por sus siglas en inglés), Java Persistence API (JPA por sus siglas en inglés) y Microsoft Entity Framework, que al menos automatizan parcialmente el proceso de mapeo.

- Bases de Datos Orientadas a Objetos (ODBMS Object Oriented Database Systems por sus siglas en inglés), que pueden almacenar objetos de manera nativa, es decir, no es necesario código para mapeo de objetos.

El presente trabajo se basa en escribir el código que mapea los objetos en la base de datos relacional.

1.1.2.1 FUNDAMENTOS DEL MAPEO DE OBJETOS UTILIZANDO PATRONES DE MODELAMIENTO

La mayoría de las aplicaciones de negocios modernas utilizan las tecnologías de objetos como C# para construir las aplicaciones y bases de datos relacionales para almacenar los datos. [25]

Desafortunadamente es necesario afrontar la incompatibilidad del modelo de objetos y el modelo relacional y para esto hay que entender el proceso de mapeo de objetos a la base de datos relacional y cómo implementar estos mapeos, el término mapeo hace referencia a cómo los objetos y sus relaciones son mapeados a las tablas y sus relaciones entre ellas en la base de datos.

1.1.2.2 MAPEO DE ATRIBUTOS A COLUMNAS

El punto de inicio para mapear objetos a la base de datos relacional son los atributos de una clase. Un atributo mapea a una o más columnas de una base de

---

[25] Ambler, Scott W., *The Object-Relational Impedance Mismatch.*

datos relacional, no todos los atributos son persistentes, algunos son utilizados como cálculos temporales. Por ejemplo, el total de operaciones Spot[11] del día de una entidad bancaria, se calcula pero no se guarda este valor en la base de datos porque se calcula en la aplicación. [23]

Como algunos atributos de objetos son en sí objetos, por ejemplo, el objeto *SER_NEGOCIACION* tiene su objeto *codCliente* como un atributo, esto refleja una asociación entre dos clases que debe ser mapeada y los atributos de la clase *SER_CLIENTE* deben ser mapeados también.

El mapeo más sencillo que se puede tener es de una característica que mapea de un solo atributo a una sola columna, es aún más simple cuando cada uno tiene el mismo tipo de datos básico, por ejemplo, ambos son de tipo fecha, el atributo es un tipo de dato cadena de caracteres y la columna es de tipo carácter, o el atributo es de tipo número y la columna es de tipo flotante.

1.1.2.3 MAPEO DE CLASES A TABLAS

Las clases se mapean a tablas pero no siempre se hace directamente. Excepto por bases de datos muy simples, es poco común encontrar un mapeo de clases a tablas de tipo uno a uno. Sin embargo, desde el punto de vista académico y como punto de partida, se puede decir que una clase corresponde a una tabla en la base de datos, aunque luego se pueden hacer mejoras para el rendimiento y afinamiento de las clases mapeadas. [10]

La figura 6 describe el modelo más simple de mapeo donde se presentan dos modelos, un diagrama de clases UML (Unified Modeling Language, por sus siglas en inglés) y un modelo físico de datos.

[11] Es una operación financiera, donde la entrega o recepción de las divisas se hace inmediatamente.

[23] Ambler, Scott W., *Mapping Objects to Relational Databases: O/R Mapping In Detail.*

[10] Keller, Wolfgang, *Mapping Objects to Tables A Pattern Language*.

Ambos diagramas describen un esquema simple de un sistema de negociaciones Spot de tesorería bancaria. Se puede ver cómo los atributos de las clases pueden ser mapeados a las columnas de la base de datos. Por ejemplo, el atributo *fechaPago* de la clase *SER_NEGOCIACION* se mapea con la columna *FECHAPAGO* de la tabla *SER_NEGOCIACION*.

Aún cuando los esquemas de la figura 6 son similares, tienen algunas diferencias, lo que significa que el mapeo no es perfecto, las diferencias son:

- Existe un atributo que hace referencia al valor neto en el objeto *SER_NEGOCIACION* (*VALORNETO*). Este valor probablemente debe ser calculado usando los valores guardados en la tabla *SER_NEGOCIACION*. Cuando el objeto se carga en memoria, es necesario calcular el valor neto o se puede calcular la primera vez que se accede al objeto. Esta diferencia indica que posiblemente sea necesario modificar el modelo de datos para tener el campo de valor neto en la tabla.

- El esquema de datos indica claves primarias y foráneas. Las filas en las tablas están indicadas por claves primarias y las relaciones entre las filas se mantienen mediante claves foráneas. Las relaciones entre objetos se implementan mediante referencias a los objetos, no a través de claves foráneas. Esto implica que para que persistan los objetos y sus relaciones, es necesario saber los valores de las claves utilizados en la base de datos para identificarlos.

- Diferentes tipos de datos son utilizados en cada esquema. En la tabla *SER_NEGOCIACION*, los campos *VALORAPAGAR* y *VALORARECIBIR* son de tipo *numeric*, mientras que *valorAPagar*, *valorARecibir* y *valorNeto* en el objeto *SER_NEGOCIACION* son de tipo *double*. Cuando se implemente este mapeo, se deben convertir los valores de ida y vuelta entre las dos representaciones evitando la pérdida de información o precisión.

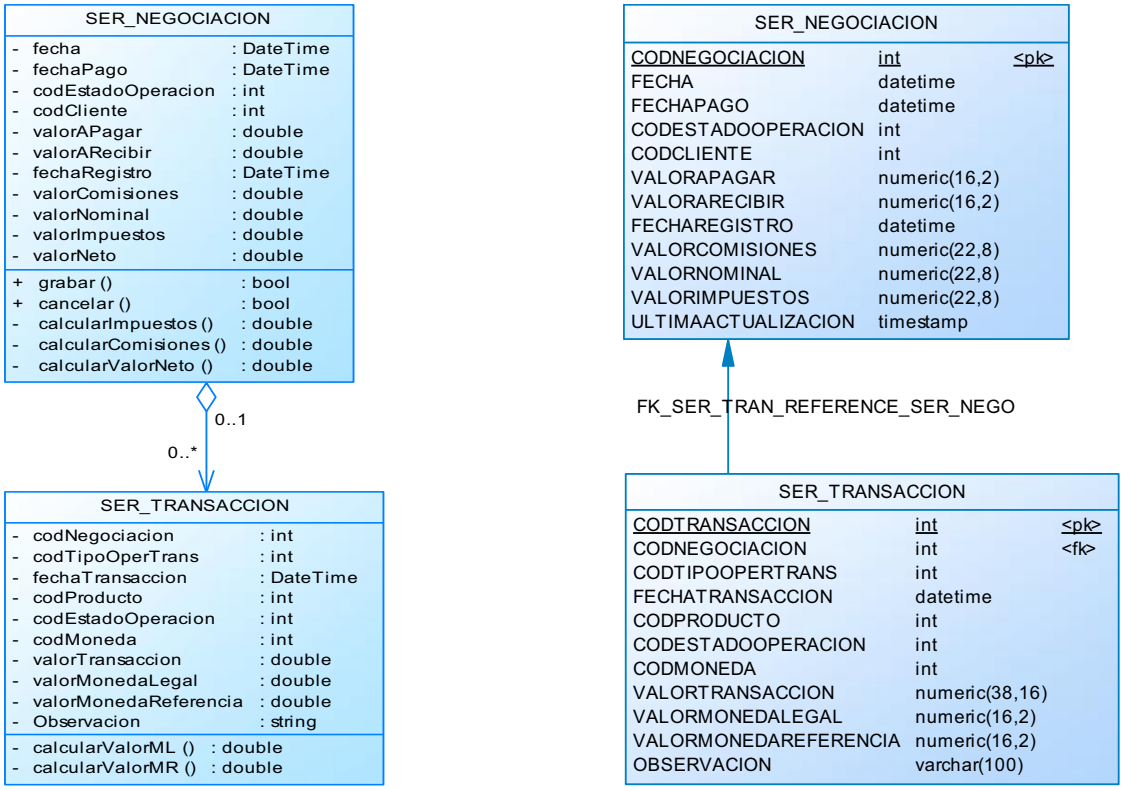


Figura 6. Ejemplo de mapeo simple.

Existe información que los objetos necesitan mantener más allá de los datos básicos para poder persistir, generalmente se necesita: información de claves primarias, especialmente si la clave primaria es una clave subrogada o clave sustituta sin significado de negocio, marcas de control de concurrencia como *"timestamp"* (secuencia de caracteres que indican fecha y hora), contadores incrementales o números de versión.

Por ejemplo, en la figura 6 la tabla *SER_NEGOCIACION* tiene como clave primaria la columna *CODNEGOCIACION* y la columna *ULTIMAACTUALIZACION* que se usa para control de concurrencia optimista que el objeto *SER_NEGOCIACION* no tiene. Para poder hacer que el objeto *SER_NEGOCIACION* persista es necesario implementar estos atributos y mantener sus valores. Esta información se la conoce como "información sombra".

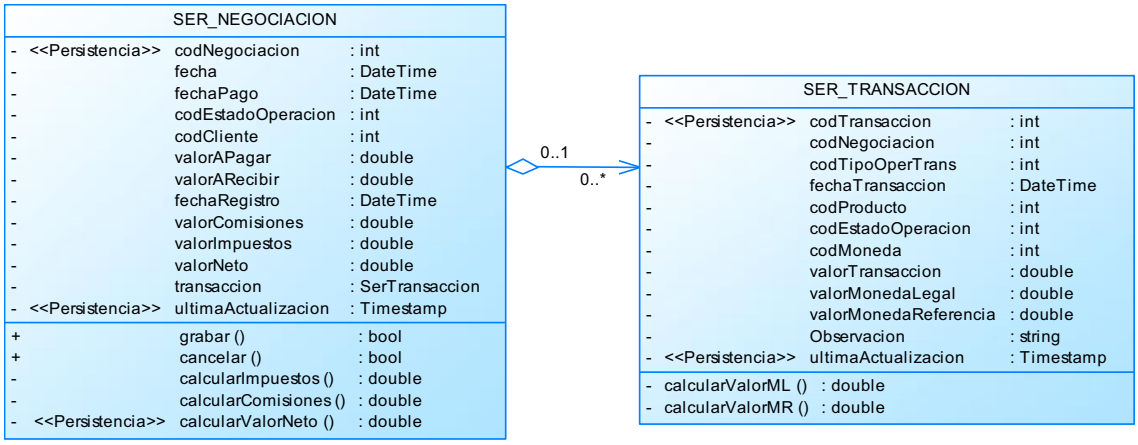


Figura 7. Inclusión de la "información sombra" al diagrama de clases.

La figura 7 muestra un diseño de modelo de clase detallado para las clases *SER_NEGOCIACION* y *SER_TRANSACCION*. A diferencia de la figura 6, la figura 7 muestra la información necesaria que las clases requieren para persistir adecuadamente. Los atributos sombra tienen asignado el estereotipo <<Persistencia>> (no es UML estándar pero sirve para indicar persistencia). Se muestra también el atributo para el andamiaje de las relaciones entre clases, como el atributo *transaccion* de tipo *SerTransaccion*, finalmente se ha agregado la operación *calcularValorNeto()* a la clase *SER_ NEGOCIACION* para calcular al valor requerido para la columna *VALORNETO* en la tabla *SER_NEGOCIACION.*

Lo que se requiere hacer es asignar las propiedades de una clase, que a veces se implementan como atributos simples y otras veces como una o más operaciones, a las columnas de la base de datos.

Existe información que también se requiere para implementar persistencia, y es una bandera booleana que indica si el objeto ya existe en la base de datos. El problema es que cuando se guarda información y el objeto ya existe, se necesita una sentencia SQL de Actualización, mientras que si el objeto no existe, se necesita una sentencia SQL de Inserción. Una práctica común es implementar una bandera de tipo booleano por ejemplo *esPersistente* que esté en verdadero si

el objeto se ha leído de la base de datos y en falso si el objeto es creado por primera vez.

Es un acuerdo común de estilo en la Comunidad UML el no mostrar la "información sombra" como claves y marcas de concurrencia en los diagramas de clases. El acuerdo también indica que no se debe modelar el código del andamiaje necesario para la persistencia. La idea es que todos saben que hay que implementar estos elementos, entonces no se debe perder el tiempo en algo obvio.

La figura 8 describe los metadatos que representan las asignaciones de propiedades requeridas para que las clases *SER_NEGOCIACION* y *SER_TRANSACCION* de la figura 7 persistan. Los metadatos son la información de los datos. La figura 8 presenta una manera de representar el mapeo de forma clara pues el concepto de mapeo de metadatos es fundamental para el funcionamiento de esquemas o capas de persistencia.

Propiedad	Columna
SER_NEGOCIACION.codNegociacion	SER_NEGOCIACION.CODNEGOCIACION
SER_NEGOCIACION.fecha	SER_NEGOCIACION.FECHA
SER_NEGOCIACION.fechaPago	SER_NEGOCIACION.FECHAPAGO
SER_NiEGOCIACION.codEstadoOperacion	SER_NEGOCIACION.CODESTADOOPERACION
SER_NEGOCIACION.codCliente	SER_NEGOCIACION.CODCLIENTE
SER_NEGOCIACION.valorAPagar	SER_NEGOCIACION.VALORAPAGAR
SER_NEGOCIACION.valorARecibir	SER_NEGOCIACION.VALORARECIBIR
SER_NEGOCIACION.fechaRegistro	SER_NEGOCIACION.FECHAREGISTRO
SER_NEGOCIACION.valorComisiones	SER_NEGOCIACION.VALORCOMISIONES
SER_NEGOCIACION.valorNominal	SER_NEGOCIACION.VALORNOMINAL
SER_NEGOCIACION.valorImpuestos	SER_NEGOCIACION.VALORIMPUESTOS
SER_NEGOCIACION.ultimaActualizacion	SER_NEGOCIACION.ULTIMAACTUALIZACION
SER_NEGOCIACION.transaccion	SER_TRANSACCION.CODTRANSACCION
SER_NEGOCIACION.calcularValorNeto()	SER_NEGOCIACION.VALORNETO
SER_TRANSACCION.codTransaccion	SER_TRANSACCION.CODTRANSACCION
SER_ TRANSACCION.codNegociacion	SER_TRANSACCION.CODNEGOCIACION
SER_TRANSACCION.codTipoOperTrans	SER_TRANSACCION.CODTIPOOPERTRANS

SER_TRANSACCION.fechaTransaccion	SER_TRANSACCION.FECHATRANSACCION
SER_TRANSACCION.codProducto	SER_TRANSACCION.CODPRODUCTO
SER_TRANSACCION.codEstadoOperacion	SER_TRANSACCION.CODESTADOOPERACION
SER_TRANSACCION.codMoneda	SER_TRANSACCION.CODMONEDA
SER_TRANSACCION.valorTransaccion	SER_TRANSACCION.VALORTRANSACCION
SER_TRANSACCION.valorMonedaLegal	SER_TRANSACCION.VALORMONEDALEGAL
SER_TRANSACCION.valorMonedaReferencia	SER_TRANSACCION.VALORMONEDAREFERENCIA
SER_TRANSACCION.observacion	SER_TRANSACCION.OBSERVACION
SER_TRANSACCION.ultimaActualizacion	SER_TRANSACCION.ULTIMAACTUALIZACION

Figura 8. Metadatos que representan el mapeo de propiedades.

Los nombres utilizados son sencillos para entender: *SER_NEGOCIACION.fecha* se refiere al atributo *fecha* de la clase *SER_NEGOCIACION*. Asimismo, *SER_NEGOCIACION.FECHA* se refiere a la columna *FECHA* de la tabla *SER_NEGOCIACION.*

SER_NEGOCIACION.calcularValorNeto() hace referencia a la operación *calcularValorNeto()* de la clase *SER_NEGOCIACION*.

La propiedad que necesita mayor explicación es *SER_NEGOCIACION.transaccion* que hace referencia a la instancia de *SER_TRANSACCION* que se está guardando.

La figura 8 señala una parte importante de la incompatibilidad del modelo de objetos y el modelo relacional. Las clases implementan comportamiento y datos mientras que las tablas de la base de datos relacional solamente implementan datos. El resultado final es que cuando se mapean las propiedades de las clases en una base de datos relacional, se termina mapeando operaciones, como *calcularValorNeto(),* a las columnas.

Aunque no es parte de este ejemplo, generalmente es necesario mapear dos operaciones que representan una sola propiedad a una columna: Una operación para establecer el valor, por ejemplo *fijarValor()* y otra operación para recuperar el valor, por ejemplo *obtenerValor()*. Estas operaciones se denominan comúnmente *"getters"* y *"setters"* que en español significa definidores y recuperadores respectivamente.

Siempre que una columna clave se mapea a una propiedad de una clase, como el mapeo entre *SER_TRANSACCION.CODTRANSACCION* y *SER_NEGOCIACION.transaccion*, es parte del mapeo de relaciones porque las claves implementan relaciones entre en las bases de datos relacionales.

1.1.2.4 MAPEO DE ESTRUCTURAS DE HERENCIA

Las bases de datos relacionales no soportan la herencia de manera nativa, lo cual obliga a mapear las estructuras de herencia del esquema de objetos al esquema de datos. El concepto de herencia hace que el guardar objetos en una base de datos relacional tome otra perspectiva. [23]

La figura 9 muestra dos versiones de una jerarquía de clases. La primera versión describe tres clases: *Persona* (clase abstracta[12]) y dos clases concretas: *Usuario* y *Cliente*. En la segunda versión de la jerarquía de clases, se añade una nueva clase concreta: *SER_SUPERUSUARIO*. La idea es que para la segunda versión se tiene un nuevo requerimiento para tener súper usuarios pero que sean usuarios con clave de súper usuario. La nueva clase *SER_SUPERUSUARIO* soporta esta funcionalidad.

[12] Una clase abstracta es una clase que no puede instanciarse. Sirve para proveer una definición común de una clase base que múltiples clases derivadas pueden compartir. [14]

[23] Ambler, Scott W., *Mapping Objects to Relational Databases: O/R Mapping In Detail.*

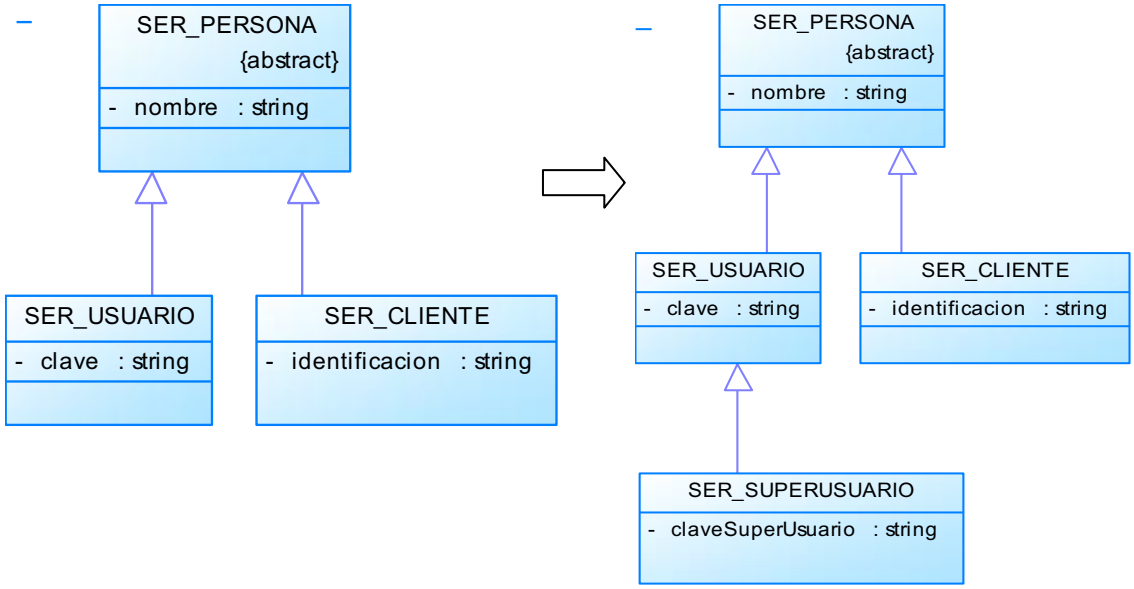


Figura 9. Dos versiones de una jerarquía de clases simple

1.1.2.4.1 Mapeo de toda la herencia de clases a una sola tabla

Este tipo de mapeo indica que se guardan todos los atributos de las clases en una tabla. La figura 10 describe el modelo de datos para las clases de la figura 9. Los atributos de cada clase son almacenados en la tabla *SER_PERSONA*, una buena estrategia para nombrar la tabla es usar el nombre de la clase base de la jerarquía de clases. [5]

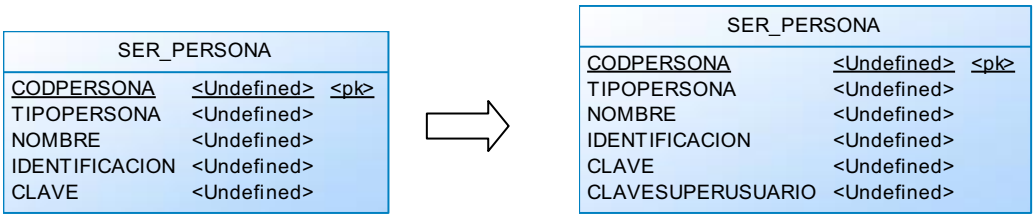


Figura 10. Mapeo a una sola tabla.

Se han agregado dos columnas a la tabla: *CODPERSONA* y *TIPOPERSONA*, la primera es la clave primaria y opcionalmente se puede poner el sufijo POID

[5] Duhl, Joshua; Barry, Douglas K., *Object Storage Fact Book: Object Relational Mapping.*

(***P**ersistent **O**bject **ID**entifier*), que significa identificador de objeto persistente y sirve para diferenciar de manera única a la persona, y la segunda es un código que indica si la persona es un usuario, cliente o ambos.

La columna *TIPOPERSONA* se requiere para identificar el tipo de objeto que puede ser instanciado de una columna específica. Por ejemplo el valor U puede indicar que es usuario, C cliente, A ambos.

En caso de requerir el tipo *Superusuario*, será necesario poner S, en estos casos las combinaciones pueden convertirse en un problema por lo que se sugiere reemplazar las combinaciones de *TIPOPERSONA* por campos de tipo booleanos.

La figura 11 muestra la mejora en el modelo donde se ha eliminado la columna *TIPOPERSONA* y se han agregado las columnas *ESUSUARIO*, *ESCLIENTE*, *ESSUPERUSUARIO* para indicar el tipo de objeto.

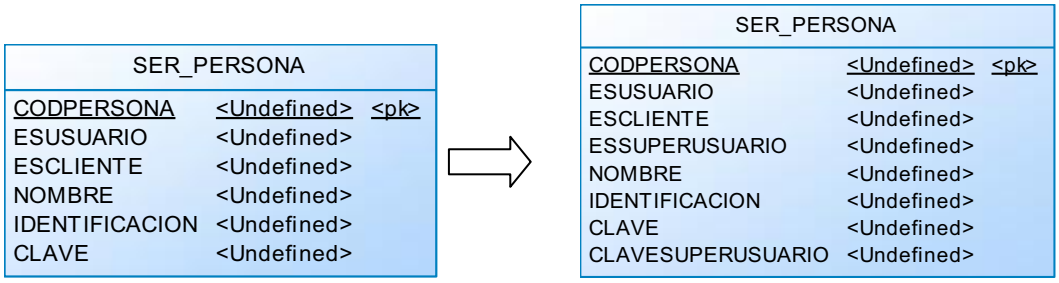


Figura 11. Mejora en el modelo

1.1.2.4.2 Mapeo de cada clase concreta a su propia tabla

Esta metodología indica que se debe crear una tabla por cada clase concreta, cada tabla debe tener los atributos implementados por la clase y los atributos heredados. [10]

---

[10] Keller, Wolfgang, *Mapping Objects to Tables A Pattern Language*.

La figura 12 muestra el modelo de datos correspondiente.

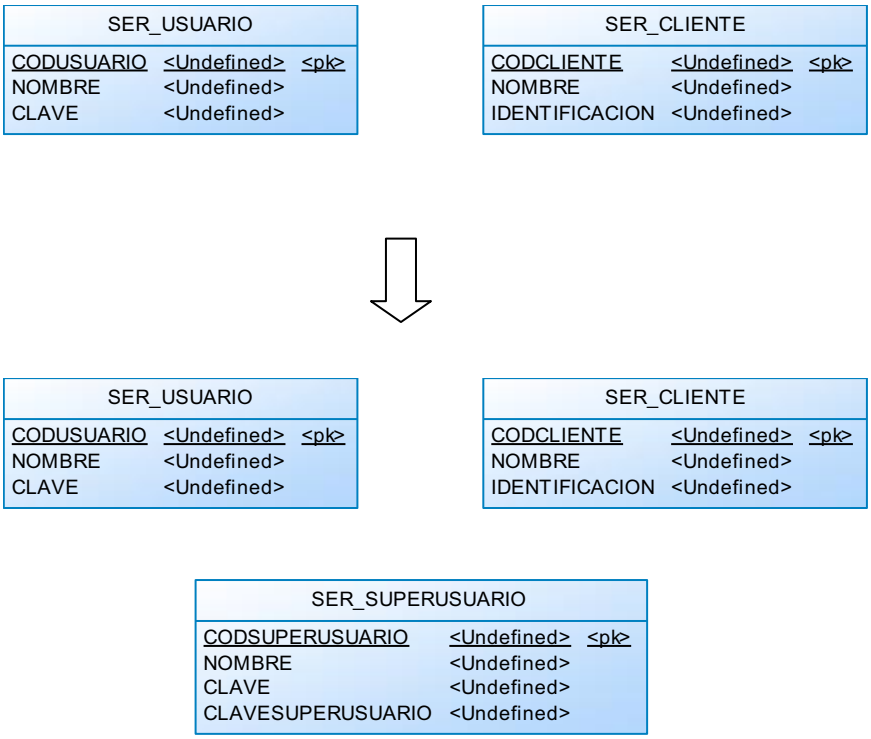


Figura 12. Mapeo de clases concretas a tablas.

De la misma forma se puede considerar el agregar columnas tipo booleano para indicar los subtipos de Persona, esto implica mayor trabajo pero es más fácil de mantener y hace algunas consultas más sencillas.

1.1.2.4.3 Mapeo de cada clase a su propia tabla

Esta estrategia plantea la creación de una tabla por clase con una columna por cada atributo de negocio y cualquier información de identificación necesaria de la clase, como para control de concurrencia y versionamiento). [5]

La figura 13 describe el modelo de datos para las clases de la figura 9.

---

[5] Duhl, Joshua; Barry, Douglas K., *Object Storage Fact Book: Object Relational Mapping.*

Los datos de la clase *SER_USUARIO* están almacenados en dos tablas, *SER_USUARIO* y *SER_PERSONA*, por lo tanto, para obtener los datos es necesario hacer una juntura de tablas o "join" o leer las dos tablas por separado.

Se puede ver cómo en este modelo se utilizan las claves primarias, *CODPERSONA* se utiliza como clave primaria de todas las tablas, para las tablas *SER_USUARIO*, *SER_CLIENTE* y *SER_SUPERUSUARIO*, *CODPERSONA* es además clave foránea que sirve para mantener la relación con la tabla *SER_PERSONA*.

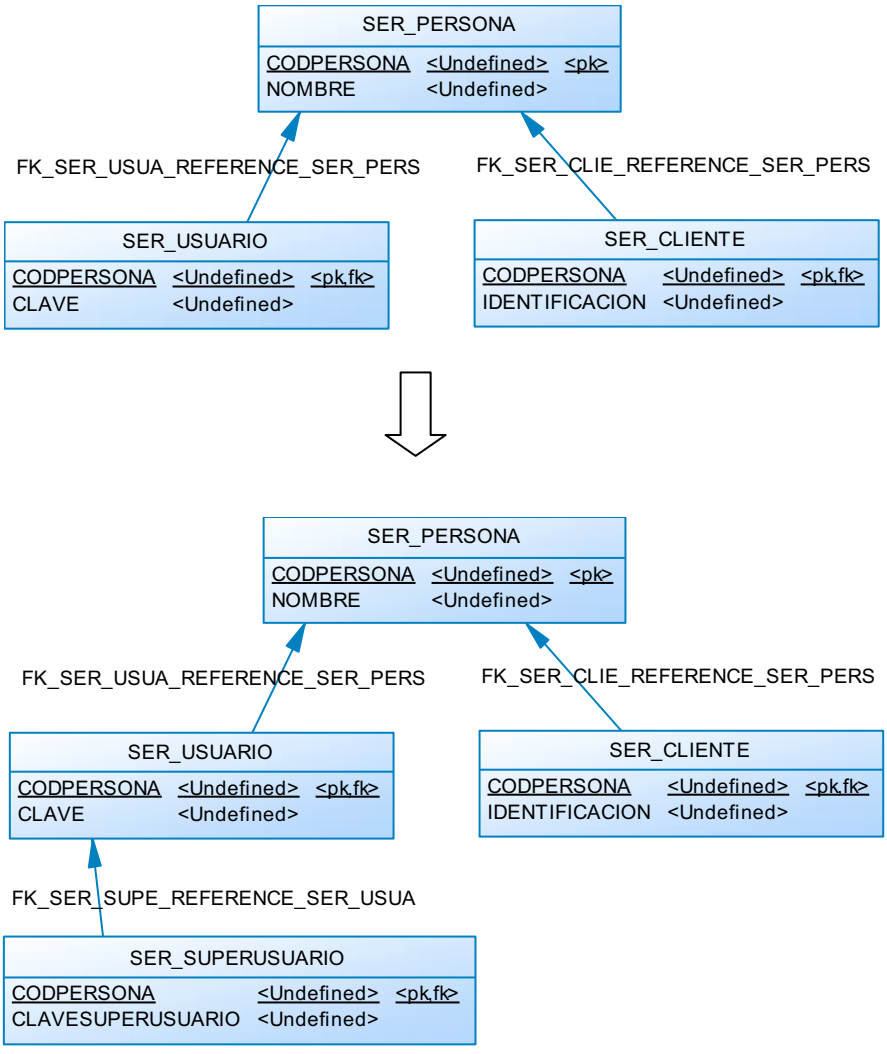


Figura 13. Mapeo de cada clase a su tabla respectiva

1.1.2.4.4 Mapeo de clases en una estructura de tablas genérica

Otra estrategia para mapear herencia es el uso de un método genérico, conocido también como método basado en meta-datos para mapear las clases. Este procedimiento no es específico a una estructura de herencia sino que soporta todas las formas de mapeo. [23]

En la figura 14 se muestra un esquema de datos para almacenar valores de atributos y recorrer estructuras de herencia. El valor de un atributo único es almacenado en la tabla Valor, por lo tanto para almacenar un objeto con diez atributos de negocio, serán necesarios diez registros, uno por cada atributo.

La columna *Valor.ObjetoPOID* almacena un identificador único para el objeto específico. La tabla *TipoAtributo* contiene filas para tipos de datos básicos como: string, int, double, decimal, etc. Esta información es requerida para convertir el valor del atributo del objeto que está almacenado como tipo varchar.

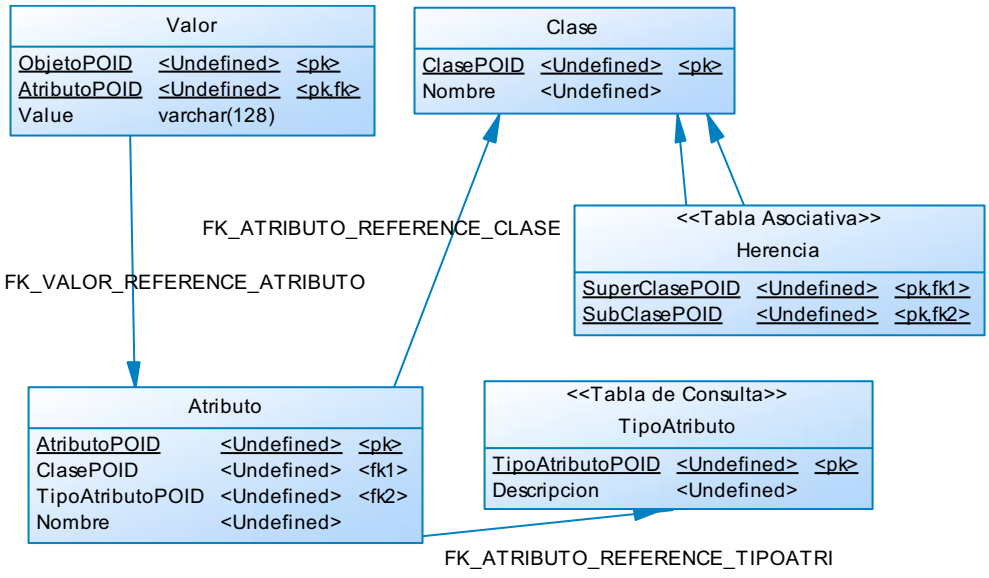


Figura 14. Esquema genérico para almacenamiento de objetos.

[23] Ambler, Scott W., *Mapping Objects to Relational Databases: O/R Mapping In Detail.*

1.1.2.5 RELACIONES DE OBJETOS, TABLAS Y MAPEO

Adicionalmente al mapeo de propiedades y herencia, es necesario el mapeo de relaciones. Existen dos categorías de relaciones a tener en cuenta para realizar el mapeo. [13]

La primera categoría está basada en la multiplicidad y tiene tres tipos:

- **Relación de uno-a-uno.** Es una relación donde el máximo de cada una de las multiplicidades es uno. Un ejemplo de esto es la relación entre *Usuario* y *Perfil* de la figura 15, un usuario mantiene uno y solo un perfil y un perfil puede ser mantenido por uno y solo un usuario (algunos perfiles pueden no ocuparse).

- **Relación de uno-a-muchos.** Se conoce también como relación de muchos-a-uno, ocurre cundo el máximo de una multiplicidad es uno y el otro es más de uno. Un ejemplo es la relación *trabaja en* entre *Usuario* y *Empresa*. Un usuario trabaja en una empresa y una empresa tiene uno o más usuarios trabajando ahí.

- **Relación muchos-a-muchos.** Es una relación donde ambas multiplicidades es mayor a uno, por ejemplo la relación asignado entre *Usuario* y *Producto*, un usuario es asignado a uno o más productos y cada producto es asignado a cero o más usuarios.

La segunda categoría está basada en la direccionalidad y contiene dos tipos: relaciones unidireccionales y bidireccionales.

- **Relaciones unidireccionales.** Es una relación unidireccional cuando un objeto sabe acerca del objeto u objetos a los cuales está relacionado pero este objeto u objetos no saben del objeto original. Un ejemplo es la relación

---

[13] Mehta, Vijay P., Pro *LINQ Object Relational Mapping with C# 2008.*

entre *Usuario* y *Perfil* en la figura 15 que se indica por la línea con una flecha abierta en el extremo. Los objetos de tipo *Usuario* saben del *perfil* que mantienen, pero los objetos de tipo *perfil* no saben qué *usuario* está ahí.

- **Relaciones bidireccionales.** Las relaciones bidireccionales se dan cuando los objetos en ambos extremos de la relación saben uno del otro, un ejemplo es la relación *trabaja en* entre *Usuario* y *Empresa*. Los objetos de tipo *Usuario* saben en qué *empresa* trabajan y los objetos de tipo *Empresa* saben qué usuarios trabajan en ellas.

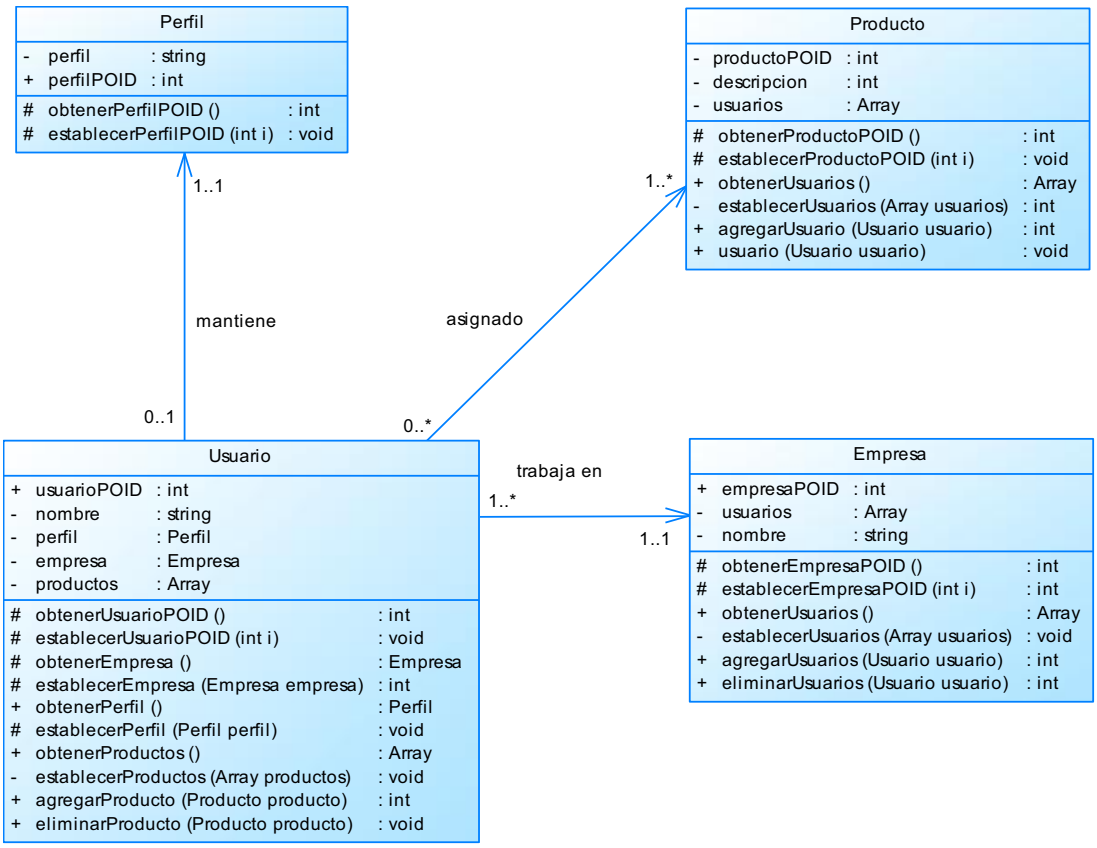


Figura 15. Relaciones entre Objetos

Es posible tener todas las combinaciones de relaciones en esquemas de objetos, sin embargo, un aspecto de la incompatibilidad del modelo de objetos y el modelo relacional es que la tecnología relacional no tiene el concepto de relaciones

unidireccionales, en las bases de datos todas las relaciones son bidireccionales porque éstas se mantienen mediante claves foráneas que pueden ser unidas o recorridas en cualquier dirección.

Las relaciones en un esquema de objetos se implementan mediante una combinación de referencias a objetos y operaciones.

Cuando la multiplicidad es uno, por ejemplo 0..1 o 1..1, la relación se implementa con una referencia a un objeto, una operación para obtener y una operación para establecer. Por ejemplo, en la figura 15 un usuario trabaja en una sola empresa, esto es implementado por la clase *Usuario* mediante la combinación del atributo *empresa*, la operación *obtenerEmpresa()* que retorna la *empresa* y *establecerEmpresa()* que establece el valor del atributo *empresa*. Los atributos y operaciones necesarios para implementar una relación son conocidos como andamiaje.

Cuando la multiplicidad es muchos, por ejemplo 0..*, 1..* la relación se implementa con una colección de atributos, como un arreglo (*Array* en inglés) y operaciones para manejar ese arreglo. Por ejemplo, la clase *Empresa* implementa un atributo arreglo llamado *usuarios*, *obtenerUsuarios()* para obtener los valores y *establecerUsuarios(…)* para fijar los valores, *agregarUsuario(…)* para agregar un usuario al arreglo y *eliminarUsuario(…)* para quitar un usuario del arreglo.

Cuando la relación es unidireccional, el código se implementa solamente en el objeto que conoce de los otros objetos. Por ejemplo, entre *Usuario* y *Perfil*, solo la clase *Usuario* implementa la asociación. Las relaciones bidireccionales en cambio se implementan en ambas clases, como se puede ver en la relación muchos a muchos entre *Usuario* y *Producto*.

Las relaciones en bases de datos relacionales se mantienen mediante claves foráneas. En una relación uno a uno la clave foránea debe ser implementada por una de las tablas. En la figura 16 se puede ver que la tabla *Perfil* incluye

UsuarioPOID como clave foránea de la tabla *Usuario* para implementar la asociación.

Para implementar una relación uno a muchos, se implementa una clave foránea de la "tabla uno" a la "tabla muchos". Por ejemplo, *Usuario* incluye la columna *EmpresaPOID* para implementar la relación *trabaja en* a la tabla *Empresa*.

Para implementar las relaciones muchos a muchos es importante la creación de una tabla asociativa como la tabla *UsuarioProducto* de la figura 16 que incluye las claves primarias de las tablas que asocia. Con esto se convierte la relación muchos a muchos en dos relaciones uno a muchos mediante la tabla asociativa.

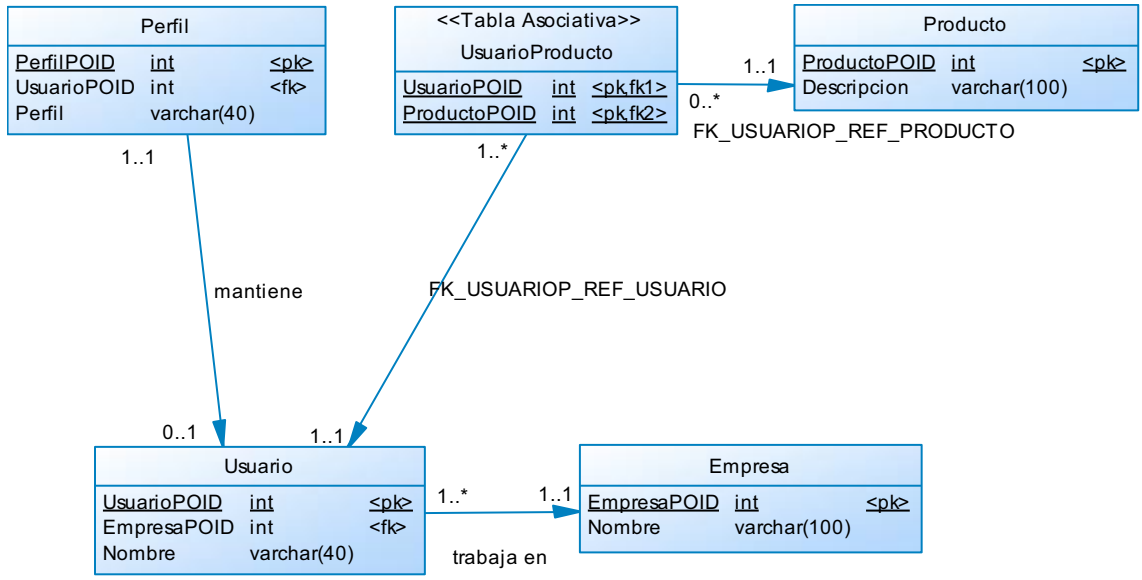


Figura 16. Relaciones en una base de datos relacional.

Como las claves foráneas se utilizan para juntar tablas, todas las relaciones en una base de datos relacional son efectivamente bidireccionales. Por esto no importa en qué tabla se implementa la relación uno a uno, el código para juntar las tablas es prácticamente el mismo. Por ejemplo en el esquema de la figura 16 el código SQL para juntar las tablas *Perfil* y *Usuario* puede ser:

*SELECT * FROM Perfil, Usuario*
WHERE Perfil.UsuarioPOID = Usuario.UsuarioPOID

Si se tiene la clave foránea implementada en la tabla Usuario el código es:

*SELECT * FROM Perfil, Usuario*
WHERE Perfil.PerfilPOID = Usuario.PerfilPOID

1.1.2.6 MAPEO DE RELACIONES

El mapeo de relaciones se describe desde el punto de vista del mapeo de relaciones de objetos en la base de datos relacional.

Se pueden implementar relaciones con multiplicidades distintas entre objetos y bases de datos, por ejemplo, se puede implementar una relación de objetos uno a uno con una relación de datos uno a muchos o muchos a muchos. Esto es porque una relación de datos uno a uno es un subconjunto de una relación de datos de uno a muchos y una relación de uno a muchos es un subconjunto de una relación muchos a muchos. [10]

La figura 17 describe el mapeo de propiedades entre el esquema de objetos de la figura 15 y el esquema de datos de la figura 16. Se ha mapeado las propiedades de negocio y la información sombra de los objetos, pero no se ha mapeado la información de andamiaje (atributos y operaciones necesarios para implementar relaciones) como *Usuario.perfil* y *Usuario.productos*. Estos atributos de andamiaje se representan mediante la información sombra mapeada en la base de datos.

---

[10] Keller, Wolfgang, *Mapping Objects to Tables A Pattern Language*.

Propiedad	Columna
Perfil.perfil	Perfil. Perfil
Perfil.perfilPOID	Perfil. PerfilPOID
Usuario.nombre	Usuario.Nombre
Usuario.usuarioPOID	Usuario.UsuarioPOID
Usuario.usuarioPOID	UsuarioProducto.UsuarioPOID
Empresa.nombre	Empresa.Nombre
Empresa.empresaPOID	Empresa.EmpresaPOID
Producto.descripcion	Producto.Descripcion
Producto.productoPOID	Producto.ProductoPOID
Producto.productoPOID	UsuarioProducto.ProductoPOID

Figura 17. Mapeo de Propiedades

1.1.2.6.1 Mapeo Uno a Uno

Se considera para esto la relación de objetos uno a uno entre *Usuario* y *Perfil*. Asumiendo que cada vez que un objeto *Perfil* o *Usuario* se lee en memoria la aplicación automáticamente recorrerá la relación y automáticamente lee el objeto correspondiente. Otra opción es recorrer la relación manualmente en el código con un enfoque de lectura diferida donde el objeto se lee al momento de ser requerido. En la figura 18 se puede ver cómo se mapean las relaciones de objetos. [13]

[13] Mehta, Vijay P., Pro *LINQ Object Relational Mapping with C# 2008.*

Relación de Objeto	Desde	Hasta	Card.	Lectura Autom.	Columna(s)	Propiedad de Andamiaje
mantiene	Usuario	Perfil	Uno	Si	Perfil.UsuarioPOID	Usuario.perfil
mantenido por	Perfil	Usuario	Uno	Si	Perfil.UsuarioPOID	Usuario.perfil
trabaja en	Usuario	Empresa	Uno	Si	Usuario.EmpresaPOID	Usuario.empresa
ha trabajado en	Empresa	Usuario	Muchos	No	Usuario.EmpresaPOID	Perfil.usuarios
asignado	Usuario	Producto	Muchos	No	Usuario.UsuarioPOID UsuarioProducto.UsuarioPOID	Usuario.productos
asignado a	Producto	Usuario	Muchos	No	Producto.ProductoPOID UsuarioProducto.ProductoPOID	Producto.usuarios

Figura 18. Mapeo de las relaciones de objetos.

La lógica para obtener un objeto *Perfil* paso a paso es:

1. El objeto *Perfil* se lee en memoria.
2. La relación se recorre automáticamente.
3. El valor en la columna *Perfil.UsuarioPOID* se usa para identificar el usuario único que necesita ser leído en memoria.
4. Se busca en la tabla *Usuario* el registro con el valor de *UsuarioPOID*.
5. El objeto *Usuario*, si hay alguno, es leído e instanciado.
6. El valor del atributo *Usuario.perfil* se establece con la referencia al objeto *Perfil*.

La lógica para obtener un objeto *Usuario* paso a paso es:

1. El objeto *Usuario* se lee en memoria.
2. La relación se recorre automáticamente.
3. El valor en la columna *Usuario.UsuarioPOID* se utiliza para identificar el perfil único que necesita ser leído en memoria.
4. En la tabla *Perfil* se busca una fila con el valor *UsuarioPOID*.
5. El objeto *Perfil* es leído e instanciado.
6. El valor del atributo *Usuario.perfil* se establece con una referencia al objeto *Perfil*.

Para almacenar los objetos en la base de datos, como las relaciones se recorren automáticamente y para mantener la integridad referencial, se crea una transacción.

El siguiente paso es agregar sentencias de actualización por cada objeto en la transacción.

Cada sentencia de actualización incluye los atributos de negocio y los valores clave asignados en la figura 18, como las relaciones se implementan mediante claves foráneas y como esos valores son actualizados, la relación efectivamente persiste. La transacción es enviada a la base de datos y se ejecuta.

1.1.2.6.2 Mapeo Uno a Muchos

Una relación de uno a muchos es la relación *trabaja en* entre *Usuario* y *Empresa* de la figura 15. Un usuario trabaja en una sola empresa y una empresa tiene varios usuarios trabajando ahí.

Algo interesante de esta relación es que se puede recorrer la misma automáticamente desde *Usuario* hasta *Empresa*, conocido como lectura en cascada, pero no se puede recorrer en dirección contraria, es decir, de *Empresa* a *Usuario*. [21]

Cuando un usuario se lee en memoria la relación se recorre automáticamente para leer la empresa en la que trabaja. Como no son necesarias varias copias de la misma empresa, por ejemplo, si se tienen diez usuarios que trabajan para la empresa de *TI* es preferible referirse al mismo objeto empresa *TI* en memoria. Esto implica el desarrollo de una estrategia y una opción es implementar un caché que asegura que una sola copia de un objeto existe en memoria o simplemente hacer que la clase *Empresa* implemente su propia colección de instancias en

[21] Yoder, Joseph; Johnson, Ralph; Wilson, Quince, *Connecting Business Objects to Relational Databases Whitepaper.*

memoria (un mini caché).
Si la aplicación lo requiere, se lee el objeto *Empresa* en memoria, luego se establece el valor del atributo *Usuario.empresa* la referencia apropiada del objeto *Empresa*. De igual forma la operación *Empresa.agregarUsuario(…)* se invoca para añadir el objeto *usuario* a su colección.

El almacenamiento es igual que en la relación uno a uno, cuando los objetos son guardados también se guarda los valores de sus claves primarias y foráneas, por lo tanto la relación se graba automáticamente.

1.1.2.6.3 Mapeo Muchos a Muchos

Para implementar las relaciones de muchos a muchos es necesario el concepto de tabla asociativa, una entidad cuyo único propósito es mantener la relación entre dos o más tablas en una base de datos relacional. [11]

En la figura 15 existe una relación muchos a muchos entre *Usuario* y *Producto*, en el esquema de datos de la figura 16 se hizo necesario introducir la tabla asociativa *UsuarioProducto* para implementar la relación muchos a muchos entre las tablas *Usuario* y *Producto*.

En las bases de datos relacionales los atributos de una tabla asociativa son generalmente la combinación de las claves primarias de las tablas involucradas, en este caso *UsuarioPOID* y *ProductoPOID*, asimismo el nombre de la tabla asociativa es típicamente la combinación de los nombres de las tablas que asocia o el nombre de la asociación que implementa, en este caso se ha elegido combinar los nombres de las tablas: *UsuarioProducto*.

Teniendo en cuenta las multiplicidades de la figura 15, la regla es que las multiplicidades se “cruzan” cuando se introduce la tabla asociativa como se indica

[11] Keller, Wolfgang, *Object/Relational Access Layers A Roadmap, Missing Links and More Patterns.*

en la figura 12. Una multiplicidad de uno siempre se introduce en los bordes externos de la relación dentro del esquema de datos para conservar la multiplicidad general de de la relación original.

La relación original indica que un usuario es asignado a uno o más productos y un producto tienen cero o más usuarios asignados. En el esquema de datos se puede ver que esto es cierto aún con la tabla asociativa que mantiene la relación. La idea es no perder la multiplicidad original.

Si un objeto usuario está en memoria y se necesitan leer todos los productos asignados a éste, la aplicación necesita hacer lo siguiente:

1. Crear una sentencia de selección (*SELECT* en inglés) que junte las tablas *UsuarioProducto* y *Producto*, se eligen todos los registros de *UsuarioProducto* con un valor *UsuarioPOID* igual al usuario, así se tiene la lista de productos.
2. Ejecutar la sentencia en la base de datos.
3. Los registros que representan estos productos se organizan en objetos tipo *Producto*, parte de esto es verificar si el objeto *Producto* está ya en la memoria. Si ya está, se refresca el objeto con los nuevos valores.
4. La operación *Usuario.agregarProducto(…)* se invoca para cada objeto tipo *Producto* para construir la colección.

El proceso para leer los usuarios asignados a un producto es similar. Para almacenar la relación, desde el punto de vista del objeto *Usuario*, los pasos son:

1. Iniciar una transacción.
2. Agregar sentencias de actualización para cada objeto *producto* modificado.
3. Agregar sentencias de inserción para la tabla *Producto* por cada producto nuevo creado.
4. Agregar sentencias de inserción para la tabla *UsuarioProducto* por los nuevos productos asignados.

5. Agregar sentencias de eliminación para la tabla *Producto* por las productos eliminados. Esto puede no ser necesario si las eliminaciones en el objeto individual ya se han realizado.
6. Agregar sentencias de eliminación para la tabla *UsuarioProducto* por cualquier producto eliminado. Esto puede no ser necesario si las eliminaciones en el objeto individual ya se han realizado.
7. Agregar sentencias de eliminación para la tabla *UsuarioProducto* por cualquier producto que ya no sea asignada a ningún usuario
8. Ejecutar la transacción.

Estas relaciones son interesantes por la inclusión de la tabla asociativa. Dos clases de negocio son mapeadas a tres tablas de datos para poder mantener la relación.

1.1.2.6.4 Mapeo de Relaciones Recursivas

Una relación recursiva, también llamada reflexiva, es una relación en la cual la misma entidad (clase, tabla, etc.) está en ambos lados de la relación. Por ejemplo la relación *dirige* de la figura 19 es recursiva, representa el concepto que un usuario puede dirigir otros usuarios.

La relación agregada que la clase *Equipo* mantiene con sí misma es recursiva, un equipo puede ser parte de uno u otros equipos, los equipos se refieren a grupos de trabajo en distintas áreas. [23]

La figura 19 describe un modelo de clases que incluye dos relaciones recursivas y el modelo de datos resultante al cual puede ser mapeado.

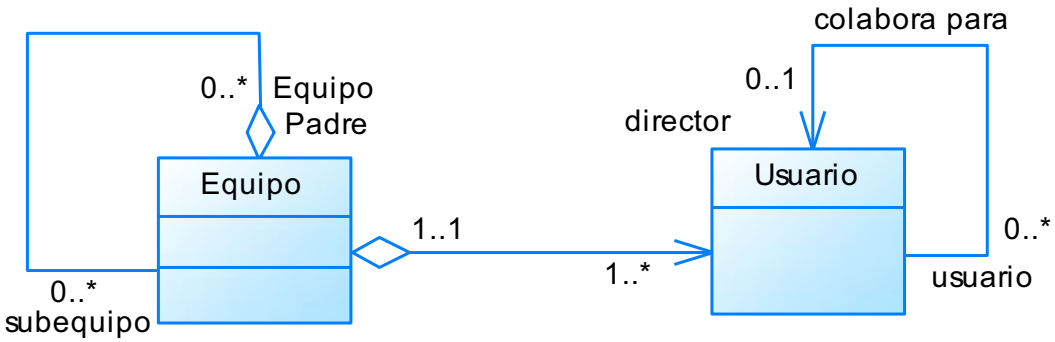
La agregación recursiva muchos a muchos es mapeada a la tabla asociativa *Subequipos* de la misma forma en que se mapea una relación normal de muchos

[23] Ambler, Scott W., *Mapping Objects to Relational Databases: O/R Mapping In Detail.*

a muchos, la única diferencia es que ambas columnas son claves foráneas en la misma tabla.

De forma similar, la asociación *dirige* uno a muchos es mapeada de la misma forma que una relación normal de uno a muchos, la columna *UsuarioDirectorPOID* se refiere a otra columna en la tabla *Usuario* donde se guardan los datos del director.

<< Modelo de Clases >>



<< Modelo Físico de Datos >>

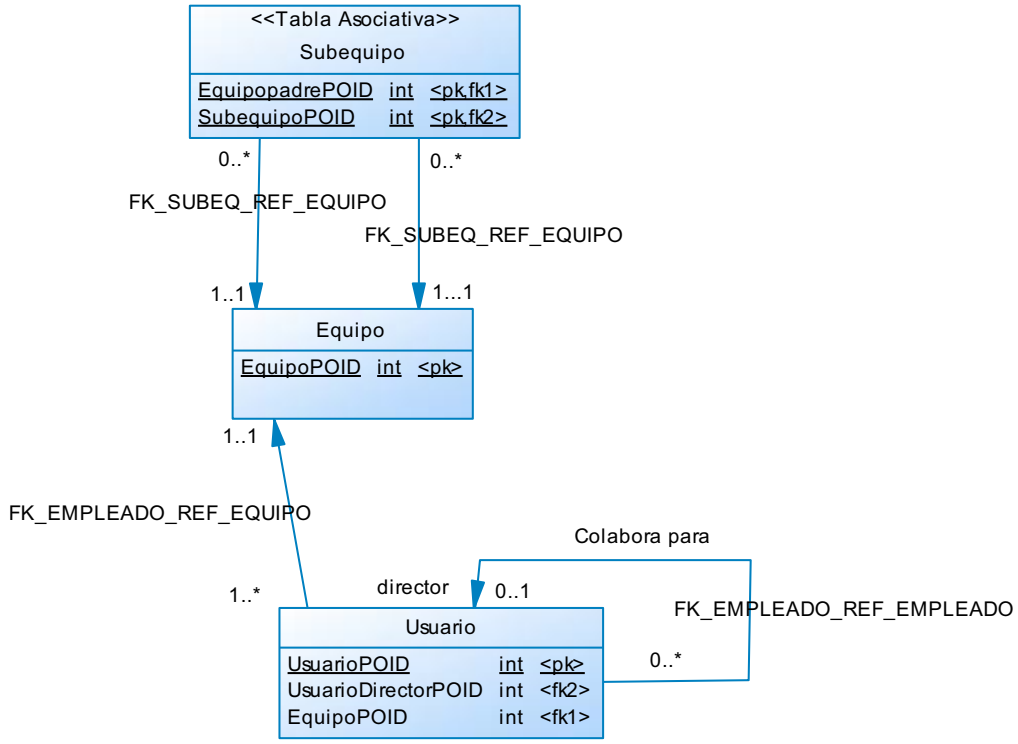


Figura 19. Mapeo de Relaciones Recursivas.

1.1.2.7 OBJETOS CONCURRENCIA Y BLOQUEO DE FILAS

La concurrencia es el acceso simultáneo de varios usuarios y/o procesos al mismo registro en la base de datos relacional. Como esto sí es posible, si la aplicación lo permite, y los usuarios efectivamente acceden al mismo registro en la base de datos, es necesaria una estrategia para controlar la concurrencia. [20]

El mecanismo utilizado en bases de datos relacionales para controlar esto es el bloqueo. Existen dos técnicas principales para realizar el bloqueo de filas: Bloqueo Optimista y Bloqueo Pesimista.

1.1.2.7.1 Bloqueo Optimista

Es una técnica donde un ítem es bloqueado en el mecanismo de persistencia solamente el tiempo en el que se accede a él desde el mecanismo de persistencia.

Por ejemplo, si un objeto *cliente* es editado, se pone un bloqueo en el mecanismo de persistencia por el tiempo que toma leer el objeto en memoria, luego el bloqueo se quita inmediatamente. El objeto es editado y luego cuando es necesario grabar, se bloquea nuevamente, se graba y se desbloquea inmediatamente.

Esta técnica permite varias personas trabajar con el objeto de manera simultánea, pero se puede sobrescribir el trabajo de otros. Este tipo de bloqueo es el mejor para procesos en línea.

---

[20] Duhl Vieira, Robert., *Microsoft SQL Server 2008 Programming*.

1.1.2.7.2 Bloqueo Pesimista

Esta técnica de bloqueo hace que el ítem sea bloqueado en el mecanismo de persistencia todo el tiempo que está en memoria. Por ejemplo, cuando se edita un objeto *cliente*, se pone un bloqueo en el mecanismo de persistencia, el objeto es traído a memoria y editado, luego el objeto puede ser guardado nuevamente en el mecanismo de persistencia y ahí el objeto se desbloquea. [26]

Esta estrategia garantiza que un ítem no será modificado en el mecanismo de persistencia mientras esté en memoria, pero se impide que otros usuarios trabajen con él mientras otro esté "ocupando" el objeto. El bloqueo pesimista es útil para procesos por lote (*batch* en inglés) que necesitan asegurar consistencia en los datos que escriben.

1.1.3 TIPOS DE CAPAS DE PERSISTENCIA

Entre los tipos de persistencia más comunes utilizados por los desarrolladores en sus sistemas, se han encontrado los siguientes:

***Código SQL incrustado directamente en las clases de negocio*.** La ventaja que puede tener esta técnica es la rapidez en la escritura de código y puede ser útil únicamente en proyectos pequeños o prototipos.

La desventaja es que se acoplan las clases de negocio con el esquema de la base de datos, lo cual implica que el menor cambio en la base de datos como el cambio de nombre de una columna requiera el escribir nuevamente el código fuente. También resulta en un código difícil de mantener y de extender.

La figura 20 muestra gráficamente la técnica mencionada.

---

[26] Tore Bostrup, "Introduction to Relational Databases - Part 1: Theoretical Foundation".

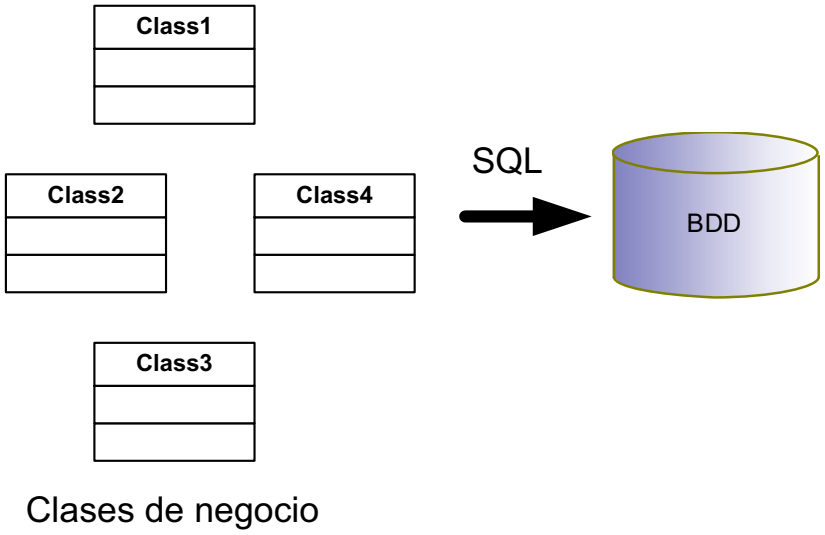


Figura 20. Código SQL incrustado en clases de negocio.

Otro tipo de persistencia es el ***código SQL incrustado en clases de datos separadas*.** Esta metodología encapsula las sentencias y el código SQL en clases de datos, que aunque es mejor que la anterior, es adecuada únicamente para proyectos pequeños o prototipos de menos de 50 clases pues es necesaria una re-compilación del código con cada cambio en la base de datos.

El desarrollo de procedimientos almacenados en la base de datos es un ejemplo de este tipo de persistencia.

Lo mejor que se puede decir de este tipo de persistencia es que al menos se tiene encapsulado el código SQL incrustado en un solo lugar: las clases de datos o los procedimientos almacenados. [24]

La figura 21 muestra clases de datos correspondientes a clases de negocio.

[24] Ambler, Scott, *The Design of a Robust Persistence Layer For Relational Databases.*

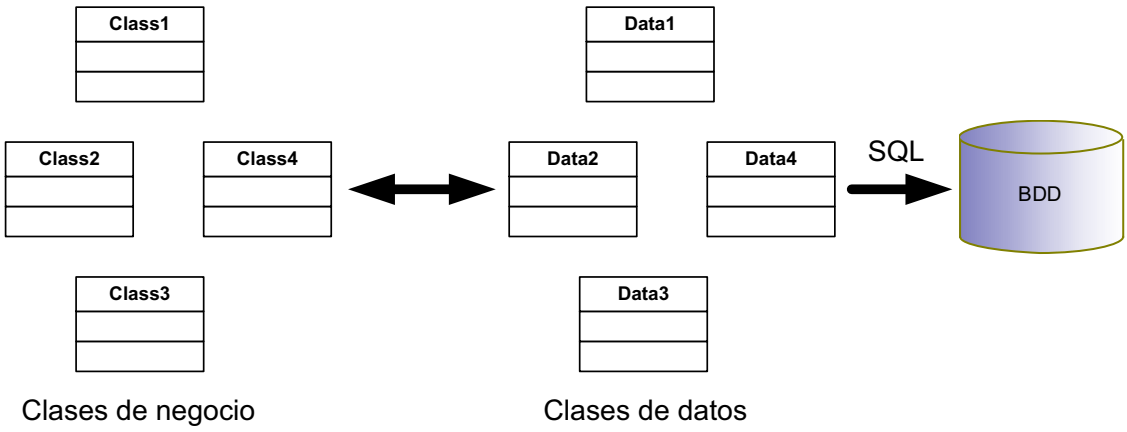


Figura 21. Creación de clases de datos que corresponden a las clases de negocio.

La creación de una ***capa de persistencia robusta*** es el tipo de persistencia que propone esta tesis. Este tipo de persistencia mapea objetos a un mecanismo de persistencia, en este caso una base de datos relacional, de tal manera que los cambios simples del esquema relacional, no afectan el código orientado a objetos.

La ventaja de este tipo de persistencia es que los programadores de aplicaciones no necesitan saber del esquema de la base de datos relacional, de hecho, ni siquiera necesitan saber que sus objetos son guardados en una base de datos relacional. Este mecanismo de persistencia permite la creación de aplicaciones de gran escala y misión crítica. La desventaja es que hay un impacto en el rendimiento de las aplicaciones, un menor impacto si la capa está bien construida pero impacto al fin.

En la figura 22 se puede ver gráficamente una capa de persistencia entre las clases de negocio y la base de datos.

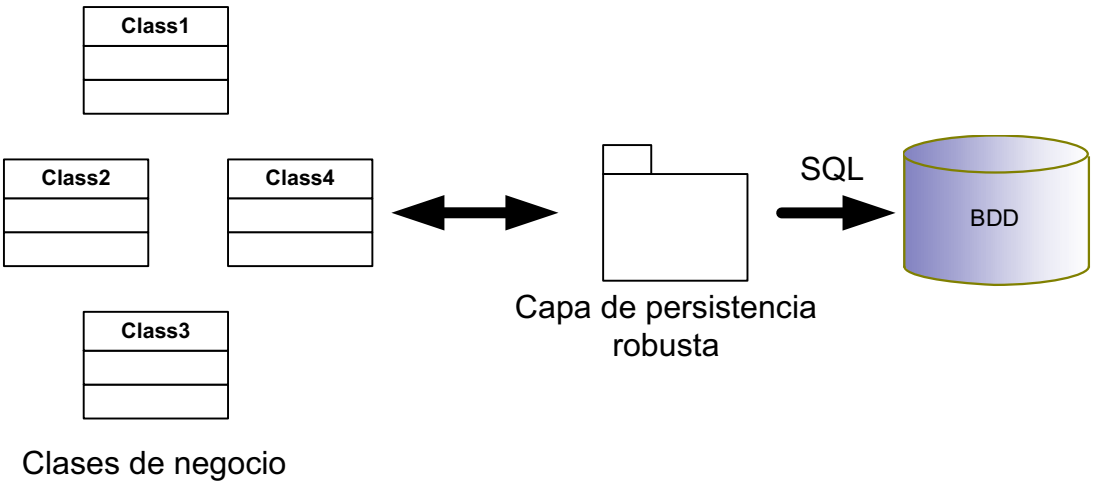


Figura 22. Una capa de persistencia robusta.

1.1.4 LA ARQUITECTURA DE CLASES

La arquitectura de clases sugerida a los programadores cuando se desarrollan aplicaciones está basada en al Patrón de Capas (Buschmann, Meunier, Rohnert, Sommerlad, Stal, 1996), la idea es que una clase en una capa dada puede interactuar con otras clases en esa capa o con clases en capas adyacentes. El mantener el código en capas, lo hace más sencillo de mantener y mejorar porque se reduce el acoplamiento y se incrementa la robustez. [7]

La figura indica que los usuarios interactúan directamente con la capa de interfaz de usuario, que generalmente está formada por clases que implementan pantallas y reportes. Estas clases pueden enviar mensajes a clases de negocio o dominio, clases controladoras o de proceso y a la clase de sistema.

---

[7] Esposito, Dino; Saltarello Andrea, *Architecting Microsoft® .NET Solutions for the Enterprise*.

La capa de dominio o negocio implementa las clases de dominio o negocio de la aplicación, por ejemplo, en una entidad bancaria se tienen las clases Cliente y Cuenta. La capa controladora o de proceso implementa la lógica de negocio que implica la interacción de clases de dominio o incluso interacción con otras clases de proceso, por ejemplo, el cálculo de impuestos y comisiones por una negociación de mercado de capitales de un cliente donde interactúan instancias de las clases Cliente, Cuenta, Impuestos y Comisiones.

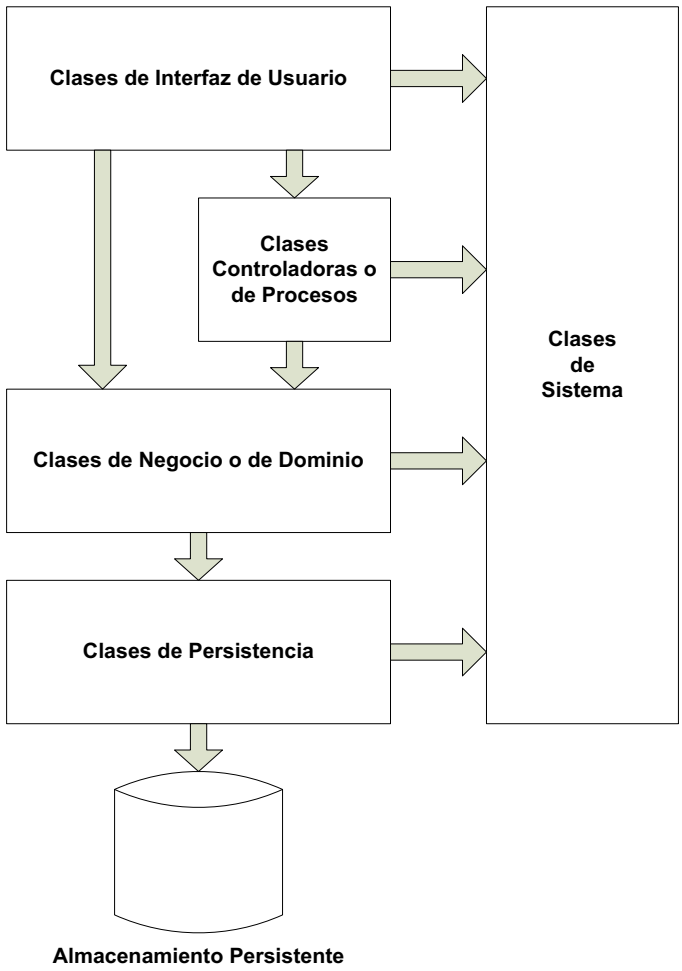


Figura 23. Arquitectura de clases sugerida.

La capa de sistema implementa clases que permiten el acceso a funcionalidad del sistema operativo como envío de correo electrónico o impresión. Las clases de dominio o negocio pueden enviar mensajes a las clases de sistema y de persistencia.

La capa de persistencia encapsula el comportamiento necesario para almacenar objetos en mecanismos de persistencia como bases de datos relacionales, bases de datos orientadas a objetos, archivos, etc.

Esta arquitectura de clases aumenta la robustez del código y se reduce el acoplamiento de la aplicación. La figura 23 muestra que para que la capa de interfaz de usuario obtenga información, debe interactuar con la capa de negocio o de dominio, la misma que interactúa con la capa de persistencia para obtener objetos almacenados en el mecanismo de persistencia. Esta es una característica importante de la arquitectura de capas que no permite que la interfaz de usuario acceda directamente a la información almacenada en el mecanismo de persistencia y se desacopla la interfaz de usuario del esquema de persistencia, esto implica que se puede cambiar la forma en que se guardan los objetos, reorganizar las tablas de una base de datos relacional o cambiar el mecanismo de persistencia a otro proveedor, sin escribir nuevamente las pantallas y reportes.

1.2 REDES LAN MULTICAPA

Las Tecnologías de la Información y la Telecomunicación (TIC), tienen gran relevancia en los sistemas informáticos y en la vida misma. El propósito principal de las TICs es obtención de información, procesamiento y distribución de la misma y con el rápido avance tecnológico crece la capacidad para manipular información y crece también la demanda de mejores y más sofisticados servicios de procesamiento de información.

La combinación entre computadoras y comunicaciones cambian la manera en que los sistemas de computadoras están organizados, las necesidades de computación, almacenamiento y distribución de información se realizan en varias computadoras independientes pero interconectadas.

La información sobre redes que se encuentra en este capítulo se basa únicamente en las redes LAN como parte del tema de investigación.

1.2.1 CONCEPTOS Y DEFINICIONES

Red de Computadoras. Una red de computadoras actualmente llamada también red informática, es un conjunto de equipos, computadoras o dispositivos, conectados mediante cables, ondas u otro medio de transporte de datos, que comparten información y servicios. [18]

Redes LAN. Las redes LAN (*Local Area Networks*), o Redes de Area Local en español, se conocen comúnmente como LANs y son redes privadas dentro de un edificio o campus de pocos kilómetros en tamaño, se utilizan para interconectar computadoras personales y equipos en oficinas, fábricas e intercambiar información y recursos, las redes LAN se distinguen de las otras redes por su tamaño, tecnología de transmisión y topología.

Como el tamaño de las redes LAN es limitado, implica que el peor caso del tiempo de transmisión es conocido, lo cual hace posible el uso de distintos diseños de redes y simplifica la administración de la misma.

Las redes LAN pueden usar una tecnología de transmisión formada por un cable al cual todas las computadoras o equipos están conectados, como en una oficina. Las LANs tradicionales trabajan a velocidades de 10 Mbps a 100 Mbps, tienen un bajo retardo del orden de los microsegundos o nanosegundos, y generan pocos errores. Las redes LAN más modernas operan hasta 10 Gbps.

Las redes LAN utilizan tecnología de transmisión *broadcast* (un solo canal de transmisión compartido por todos los equipos que reciben todos los mensajes pero los procesan solo si es para ellos). Las topologías más utilizadas en esta tecnología de transmisión se muestran en la figura 24.

---

[18] Tanenbaum Andrew S., *Computer Networks, Fourth Edition*.

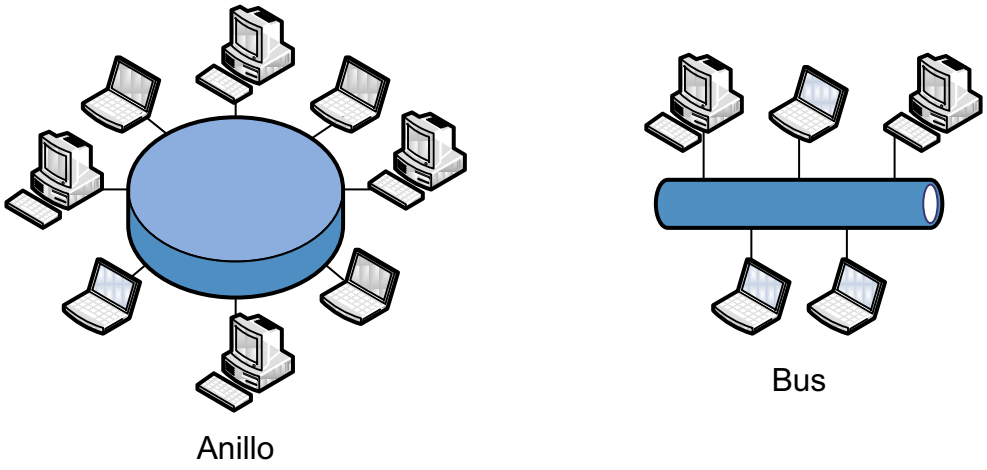


Figura 24. Topologías de las redes LAN.

Bus. En una red con topología de bus, en un instante una computadora es maestra y puede transmitir, las otras se abstienen de transmitir. Para resolver conflictos cuando dos o más computadoras quieren transmitir al mismo tiempo, se requiere un mecanismo de arbitraje. Este mecanismo puede ser centralizado o distribuido. Por ejemplo el mecanismo IEEE[13] 802.3, conocido como *Ethernet* tiene control descentralizado y opera a 10 Mbps hasta 10 Gbps. Las computadoras en una Ethernet pueden transmitir cuando quieran y si algún paquete colisiona, cada computadora espera un lapso aleatorio e intenta de nuevo.

Anillo. En un anillo, cada bit se propaga alrededor por sí mismo sin esperar al resto del paquete al cual pertenece. También es necesario un sistema para controlar accesos simultáneos al anillo por ejemplo que los equipos tengan turnos para transmitir. Ejemplos de este tipo de redes son: IEEE 802.5 conocido como *IBM token ring* y las redes *FDDI (Fiber Distributed Data Interface)* que en español significa Interfaz de Distribución de Datos por Fibra y es un estándar para la transmisión de datos sobra cables de fibra óptica.

[13] IEEE (Institute of Electrical and Electronic Engineers). Instituto de Ingenieros Eléctricos y Electrónicos.

Las redes se encuentran organizadas como una pila de capas o niveles para resucir la complejidad de diseño. El número de capas, el nombre, el contenido, y la función de cada capa difiere en cada red. El propósito de cada capa es proveer servicios a las capas superiores ocultando los detalles de implementación de los servicios.

La capa *n* de una computadora mantiene una comunicación con la capa *n* de otra computadora. El conjunto de reglas y convenciones utilizado en la comunicación se conoce como protocolo de la capa n.

El **Protocolo** es un acuerdo de comunicación entre dos entidades que indica cómo proceder en la comunicación. Los datos no se transmiten directamente de la capa *n* de una máquina a la capa *n* de otra máquina sino que cada capa transmite información de datos y control a la capa inmediatamente inferior hasta que se alcanza la última capa. La figura 25 muestra un ejemplo de capas y debajo de la capa 1 se encuentra el medio físico donde realmente ocurre la comunicación.

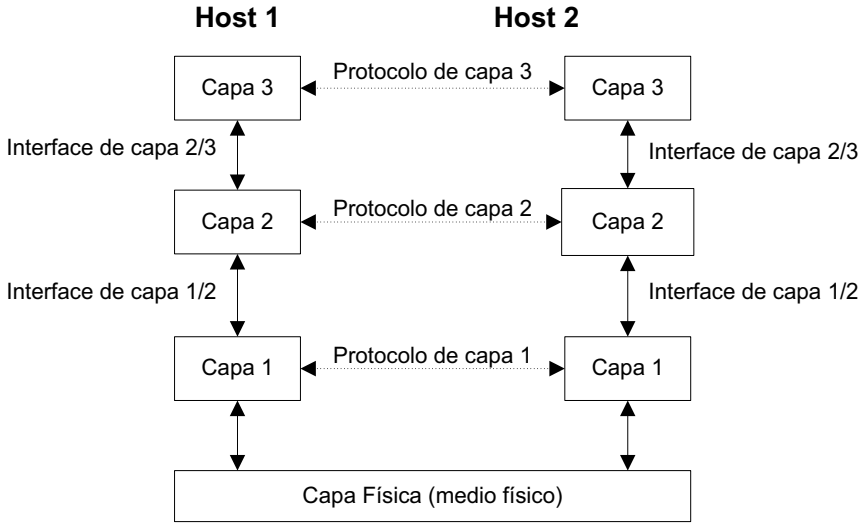


Figura 25. Ejemplo de una Red dividida en capas.

Existe una **Interface** entre capas, la cual que define las operaciones y servicios disponibles de la capa inferior a la superior.

El conjunto de capas y protocolos se conoce como la **Arquitectura de Red**. La especificación de una arquitectura de red contiene suficiente información como para implementar programas o construir hardware para cada capa cumpliendo con el protocolo establecido. Los detalles de implementación o la especificación de las interfaces son parte de la arquitectura, pues están dentro de la máquina y no son visibles desde el exterior, no es necesario tampoco que las interfaces de todas las computadoras de la red sean las mismas, siempre que cada máquina pueda usar correctamente los protocolos. La lista de protocolos en un sistema, un protocolo por capa, se conoce como **pila de protocolos.**

Las capas pueden prestar dos tipos de servicios a las capas superiores: Servicio Orientado a Conexión y Servicio no Orientado a Conexión.

En el Servicio **Orientado a Conexión**, el servicio de usuario establece una conexión, utiliza la conexión y suelta la conexión, es como un conducto entre dos máquinas o equipos donde generalmente se mantiene el orden de envío de los bits.

El Servicio **No Orientado a Conexión** parte de que cada mensaje es direccionado a través del sistema independientemente de los otros, es decir, puede que el segundo mensaje enviado llegue antes que el primer mensaje.

Es importante diferenciar el servicio del protocolo, el servicio es un conjunto de primitivas u operaciones que un capa provee a la capa inmediatamente superior, un servicio es una interface entre dos capas.

El protocolo en cambio, es un conjunto de reglas que establecen el formato y el significado de los paquetes o mensajes que se intercambian entre dos máquinas o pares (*peers* en el argot de las redes) en una misma capa. Las entidades o pares pueden cambiar de protocolos siempre que no modifiquen el servicio visible a los usuarios, así, el servicio y el protocolo están desacoplados.

1.2.2 USOS DE LAS REDES DE COMPUTADORAS

En el caso de esta investigación se indican los usos de las redes desde una perspectiva corporativa, las redes no se construyen y mejoran porque si, sino que cada red o cada mejora en una red en particular son impulsadas por necesidades de negocio. Las justificaciones para la creación de redes o mejoras de las mismas deben mostrar claramente por qué y cómo son necesarias para el correcto funcionamiento del negocio o si éstas juegan un rol importante para que la compañía cumpla sus objetivos, consistentemente con el costo y el esfuerzo del personal. [8]

Los usos de las redes de computadoras desde el punto de vista corporativo son los siguientes:

Compartir Archivos. Originalmente, la compartición de archivos fue la principal razón para instalar una red. El uso de la red para compartir archivos, requiere de un directorio compartido o una unidad de disco a la cual tengan acceso los usuarios, también se hace necesaria una lógica para asegurarse que los archivos no sean modificados por más de una persona al mismo tiempo. Adicionalmente los sistemas operativos de redes administran la seguridad al compartir archivos permitiendo que los usuarios tengan accesos de lectura, edición, etc. Sobre distintos recursos.

Compartir Impresoras. El compartir impresoras en la red reduce el número de impresoras que la empresa necesita, lo cual puede significar el uso de impresoras de mejor calidad o más sofisticadas. La forma más común de llevar a cabo la compartición de impresoras es mediante el uso de colas de impresión, una cola de impresión mantiene los trabajos hasta que finalice los que está ejecutando actualmente, luego automáticamente, envía los trabajos de impresión que tenga en la cola a la impresora.

---

[8] Hallberg, Bruce, *Networking a Beginners Guide Second Edition*.

Servicios de Aplicaciones. Así como se comparten archivos en una red, las redes se utilizan también para compartir aplicaciones. Por ejemplo, se puede instalar el *Sistema de Operaciones Spot Treasury Works*, del caso de estudio, en el servidor y que lo utilice cualquier estación en la red. Otro servicio de aplicación que se utiliza en una red es la instalación compartida que consiste en mantener copias de instaladores de aplicaciones en el servidor, luego correr el programa de instalación desde el servidor para cada estación en lugar de tener CD-ROM en cada estación.

Correo Electrónico. El correo electrónico es un recurso muy importante en la red actualmente, ayuda en las comunicaciones dentro de la compañía y fuera de ella. Los sistemas de correo electrónico son generalmente de dos tipos: correo basado en archivos y cliente/servidor. El correo basado en archivos consiste básicamente en un conjunto de archivos en un lugar compartido en el servidor, éste no provee acceso directo a los archivos sino que un computador (llamado *gateway server*) se encarga de la conexión mediante software especializado.

El sistema de correo electrónico basado en cliente/servidor, contiene los mensajes y maneja las interconexiones, estos son sistemas especializados como Microsoft Exchange o Lotus Notes, son más seguros y más poderosos que los sistemas basados en archivos y ofrecen características adicionales para el negocio.

Acceso Remoto. Se utilizan también las redes para que los usuarios accedan remotamente a la red y sus recursos (archivos, correo, etc.) cuando están trabajando fuera de la compañía o desde el hogar. La forma como se conectan los usuarios depende de las necesidades que tengan los usuarios remotos, la cantidad de usuarios y el costo. Por ejemplo: Servicio de Acceso Remoto (*RAS Remote Access Service*), Redes Privadas Virtuales.

1.2.3 MODELOS DE REFERENCIA

Los modelos o arquitecturas más importantes son el modelo de referencia OSI y el modelo de referencia TCP/IP. Los protocolos asociados al modelo OSI ya casi no se utilizan, el modelo es general y válido para entender y visualizar las características de las capas de las redes y su interacción. El modelo TCP/IP en cambio no es muy utilizado pero sus protocolos son aceptados y utilizados ampliamente.

1.2.3.1 EL MODELO DE REFERENCIA OSI.

Este modelo es conceptual y está basado en la propuesta desarrollada por la Organización Internacional de Estándares (*ISO International Standards Organization*) como primer paso para la estandarización internacional de protocolos utilizados en varias capas.

El modelo se denomina **Modelo de Referencia ISO OSI** (*Open Systems Interconnection*, en español: *Interconexión de Sistemas Abiertos*) porque se ocupa de la conexión de sistemas abiertos para la comunicación con otros sistemas.
La figura 26 muestra el Modelo de Referencia OSI, donde PDU significa *Unidad de Datos de Protocolo* o *Protocol Data Unit* por sus siglas en inglés. [18]
Como se puede ver, este modelo tiene siete capas, los principios aplicados para llegar a las siete capas son:

1. Una capa debe crearse donde sea necesaria un abstracción diferente.
2. Cada capa debe realizar una función bien definida.
3. La función de cada capa debe tener en cuenta los protocolos estandarizados internacionalmente.
4. La información que cruza las interfaces debe ser mínima.

---

[18] Tanenbaum Andrew S., *Computer Networks, Fourth Edition*.

5. El número de capas debe ser lo suficiente para distinguir las funciones y no poner una capa con doble funcionalidad, y debe ser lo suficientemente pequeño para no hacer la arquitectura compleja.

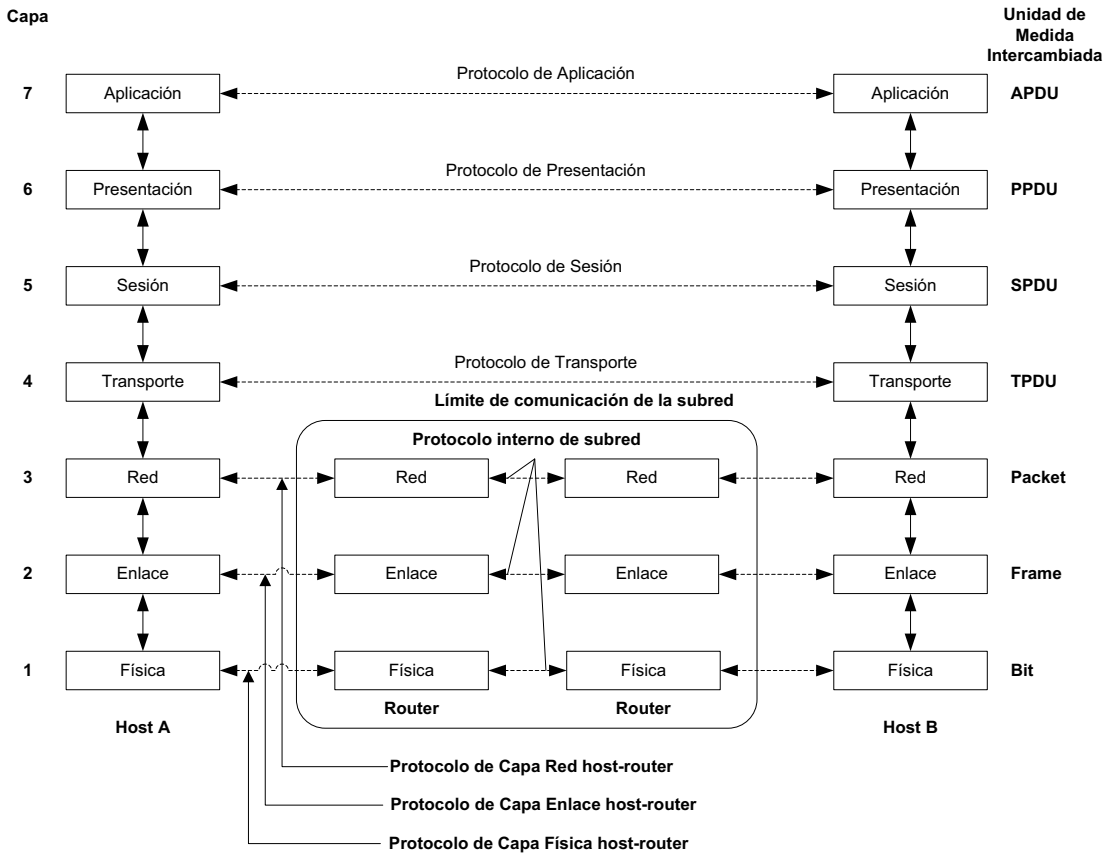


Figura 26. El Modelo de Referencia OSI.

El Modelo OSI no es una arquitectura de red porque no define servicios y protocolos en cada capa, más bien indica lo que cada capa debe hacer. La ISO ha publicado estándares para cada capa pero no son parte del Modelo OSI.

La Capa Física. Es la encargada de transmitir bits sobre un medio de comunicación. El diseño de esta capa tiene que asegurar que cuando se envía un bit 1 se reciba un bit 1 en el destino, también define cuántos voltios se utilizan para representar 1 o 0, cuántos nanosegundos dura un bit, si la transmisión es simultánea en ambas direcciones, cómo se establece y cierra la conexión,

cuántos pines tiene el conector de red y para qué se usa. Esta capa tiene relación con aspectos mecánicos, eléctricos, de tiempo y el medio físico de transmisión.

La Capa de Enlace. Esta capa se encarga de transformar la transmisión de bits en una vía que parece libre de errores de transmisión a la capa de red, esto lo hace separando los datos de entrada en *frames,* que son la unidad de datos de esta capa y que tienen un tamaño de cientos o miles de bytes, y transmitiendo los *frames* secuencialmente. Si el servicio es confiable, el receptor envía un *frame* de acuse de recibo confirmando que recibió correctamente el *frame*.

La regulación de tráfico, para que un transmisor más rápido no sobrecargue un receptor más lento, es la tarea de esta capa que se logra mediante un *buffer* (memoria intermedia) en el receptor, esta regulación de flujo y el manejo de errores están integrados en el mecanismo de regulación de tráfico de esta capa.

Las redes *broadcast* necesitan de controles por parte de esta capa: El control de acceso al canal compartido. Esta capa tiene una subcapa que se encarga de esto denominada: Subcapa de Control de Acceso al Medio.

La Capa de Red. La capa de red controla la operación de la subred, su principal tarea es determinar cuántos paquetes, la unidad de datos de esta capa, son dirigidos desde el origen al destino. Los ruteadores, dispositivos encargados del direccionamiento de los paquetes, pueden estar basados en tablas estáticas que están en la red y casi no cambian, estas tablas también se pueden determinar al inicio de una conversación o pueden ser dinámicas basadas en la carga de la red.

La capa de red se encarga de la interconexión de redes heterogéneas.

La Capa de Transporte. Su función principal es aceptar datos de las capas superiores y si es necesario dividirla en partes más pequeñas, luego pasar estos datos a la capa de red y asegurarse que todas las partes lleguen correctamente al otro lado, esto debe realizarse de una manera que aísle a las capas superiores de los cambios de hardware.

Esta capa también define el tipo de servicio que presta a la capa de sesión y a los usuarios de la red. El tipo de conexión de la capa de transporte más popular es el canal punto a punto libre de errores que envía mensajes o bytes en el orden que fueron enviados, también hay el transporte de mensajes sin garantía del orden de entrega, el envío de mensajes a múltiples destinos. El tipo de servicio se determina cuando se establece la conexión.

La capa de transporte es una capa de extremo a extremo, es decir, que un programa en la máquina origen conversa con un programa similar en la máquina destino, utilizando cabeceras de mensajes y mensajes de control. En las capas inferiores los protocolos están entre cada máquina y sus vecinos inmediatos y no entre los extremos finales, que pueden estar separados por varios ruteadores.
La Capa de Sesión. Permite a los usuarios, en diferentes máquinas o dispositivos, establecer sesiones entre ellos. Las sesiones ofrecen varios servicios como: **Control de diálogo** (indicando a quién le toca transmitir), **administración de *token*** (previniendo que dos partes realicen una operación crítica al mismo tiempo mediante un grupo de bits a manera de seña llamada *token*) y **sincronización** (permitir continuar transmisiones largas desde donde se quedaron, por ejemplo por una caída del sistema).

La Capa de Presentación. Se encarga de la sintaxis y la semántica de la información que se transmite. Para que dos computadoras con diferente representación de datos puedan comunicarse, las estructuras de datos a intercambiar pueden estar definidas de manera abstracta, junto con una codificación estándar, este mecanismo es manejado por la capa de presentación.

La Capa de Aplicación. Esta capa contiene varios protocolos utilizados por los usuarios. El protocolo usado ampliamente es el protocolo **http** que significa protocolo para la transferencia de hipertexto, *hypertext transfer protocol* por sus siglas en inglés, este protocolo es la base para la WWW. Existen otros protocolos en esta capa para la transferencia de archivos, para correo electrónico, para noticias, etc.

1.2.3.2 EL MODELO DE REFERENCIA TCP/IP.

El modelo de referencia TCP/IP es una especificación establecida por DARPA[14] para establecer las reglas de ARPANET[15], este modelo es ahora mantenido por la IETF[16]. Se desarrolló mucho antes que el modelo OSI y en lugar de las siete capas del modelo OSI, el modelo TCP/IP tiene 5 capas como se puede ver en la figura 27. [4]

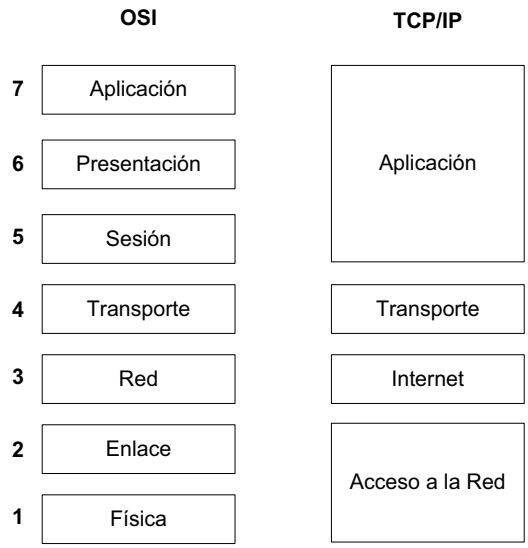


Figura 27. Modelo de Referencia TCP/IP comparado con Modelo OSI.

La Capa Aplicación. Esta capa en el modelo TCP/IP asume la funcionalidad de las capas de presentación y sesión del modelo OSI, todos los protocolos de la capa superior se manejan aquí.

La Capa de Transporte. Realiza las mismas funciones que su par del modelo OSI. Los protocolos principales en esta capa son TCP (*Transmission Control*

[14] DARPA (Defense Advanced Research Projects Agency). Agencia de Investigación de Proyectos Avanzados de Defensa.
[15] ARPANET (Advanced Research Projects Agency Network). Red de la Agencia de Investigación de Proyectos Avanzados.
[16] IETF (Internet Engineering Task Force). Grupo de Trabajo en Ingeniería de Internet. Es una organización internacional abierta de normalización que regula las propuestas y estándares de Internet.
[4] Comer, Douglas, *Internetworking with TCP/IP Vol. I: Principles, Protocols, and Architecture (4th Edition)*.

Protocol) en español: Protocolo de Control de Transmisión y UDP (*User Datagram Protocol*) en español: Protocolo de Datagrama de Usuario. TCP es un protocolo orientado a conexión, por lo tanto, provee envío confiable. UDP por otra parte, es no orientado a conexión y provee envío no confiable.

La Capa Internet. Esta capa realiza las mismas funciones que la capa 3 del modelo OSI. Se encarga de direccionar paquetes del origen al destino, se puede hacer esto dentro de una red LAN, varias redes LAN, redes MAN[17] o WAN[18]. Aquí se utiliza el protocolo de internet o IP (*Internet Protocol*) y los paquetes se denominan: paquetes IP.

La Capa de Acceso a la Red. Esta capa es la combinación de las capas Física y Enlace del modelo de referencia OSI, en el modelo TCP/IP no se presta mayor importancia a esta capa ni se definen protocolos, simplemente indica que debe proveer conexión para poder transmitir datos de un lugar a otro utilizando cualquier mecanismo.

1.2.4 PRINCIPALES ESTÁNDARES DE LAS REDES LAN

La IEEE ha reconocido los estándares a ser desarrollados para que los dispositivos LAN de diferentes fabricantes puedan comunicarse entre sí. La visión general de la IEEE 802 y una arquitectura estándar describen cómo estos dispositivos deben estar interconectados en redes LAN. Los estándares principales son los siguientes: [6]

802.2 *Logical Link Control* (Control de Enlace Lógico). Las dos capas inferiores del modelo OSI, la capa física y enlace, se tratan en el estándar 802.2. Además, se divide la capa de enlace en dos subcapas: *Logical Link Control* (LLC), en español: Control de Enlace Lógico, y *Media Access Control* (MAC), en español:

[17] MAN (Metropolitan Area Netwok), en español: Red de Área Metropolitana.
[18] WAN (Wide Area Network), en español: Red de Área Extendida.

[6] Edwards, James; Bramante, Richard, *Networking Self – Teaching Guide.*

Control de Acceso al Medio. Esto facilita el mapeo entre diversas capas físicas de redes LAN en la familia de estándares 802 de las redes LAN.
La estrategia es que la capa LLC sirve como interface entre las capas superiores y la capa física sin importar el medio físico utilizado para la construcción de la LAN.

802.3 CSMA/CD Método de Acceso y Capa Física. Las siglas CSMA/CD significan: *Carrier Sense Multiple Access with Collision Detection*, en español: Acceso Múltiple con Detección de Portadora y Detección de Colisiones. El estándar IEEE 802.3 contiene un grupo de estándares que administran las características únicas de la Capa Física que se usa en la red. Estas normas fueron evolutivas y se emitieron conforme aparecían nuevas tecnologías de comunicación con distintas características. El estándar define la estructura MAC para CSMA/CD.

802.5 "*Token Ring*" Método de Acceso y Capa Física. El estándar IEEE 802.5 define el protocolo "Token Ring" que, aunque no tiene una traducción oficial en español, se refiere a una topología de "Anillo con paso de Testigo". Es diferente a CSMA/CD donde múltiples equipos pueden transmitir al mismo tiempo, ocasionando colisiones, cuando ocurre una colisión, se retransmite.

Con *Token Ring* un solo equipo puede transmitir: el que tenga en su poder el *token* o testigo. La transmisión es secuencial en un patrón fijo, después que un equipo termina su transmisión, este pasa el *token* al siguiente equipo. Cuando el número de equipos o estaciones es pequeño este método es mejor que CSMA/CD, pero se incrementa el número de equipos o estaciones, ya no es óptimo.

El *"Token Ring"* fue creado por IBM aunque prácticamente son iguales, se pueden indicar dos diferencias entre el 802.5 y el estándar de IBM: En la especificación de IBM, el número máximo de nodos en un anillo es 260, mientras que el estándar IEEE 802.5 limita a 250 nodos. IBM permite 8 campos para designar la ruta

cuando se utiliza enrutamiento de origen, mientras que el estándar IEEE 802.5 permite hasta 14 campos.

IEEE 802.11. Es un conjunto de estándares que definen la operación de una red de comunicaciones utilizando radio frecuencias, estos estándares se confunden y se denominan generalmente con el término Wi-Fi, pero tienen algunas diferencias con los estándares de la Alianza Wi-Fi[19]. Como existen muchos productos para redes LAN inalámbricos en el mercado, la Wi-Fi Alliance está en el proceso de certificación de estos productos antes que se completen las enmiendas al estándar 802.11. Entre los productos que se venden bajo las normas se encuentran:

IEEE 802.11. Es el estándar base para las redes inalámbricas.

IEEE 802.11a. La ventaja de este estándar es el uso de la banda de 5GHz, la cual no es muy concurrida. La principal desventaja es que las señales son fácilmente absorbidas y degradan la calidad de la señal al pasar por objetos sólidos.

IEEE 802.11b. Este estándar utiliza la banda de 2.4 GHz y proporciona una velocidad de datos de 4.5 Mbps con una velocidad máxima de 11 Mbps. Su principal desventaja es que puede recibir interferencia de equipos que utilizan la misma banda como microondas, teléfonos inalámbricos y dispositivos Bluetooth. El aumento del rendimiento total y la reducción del costo han hecho que este estándar sea rápidamente aceptado como estándar definitivo para las redes LAN inalámbricas.

IEEE 802.11g. Debido a la demanda de productos de mayor velocidad, se han introducido dispositivos que soportan el estándar IEEE 802.11a y b y este estándar, es decir, estos productos soportan los tres estándares en un solo dispositivo. Sin embargo, si en una red LAN con estándar 802.11g se incluye un dispositivo que soporta solo el estándar IEEE 802.11b, se reducirá la velocidad de

[19] Wi-Fi es una marca de la *Wi-Fi Alliance*, organización comercial que adopta, prueba y certifica que los equipos cumplen los estándares 802.11 relacionados a las redes LAN inalámbricas.

la red a la del dispositivo con menor velocidad. AL igual que el estándar IEEE 802.11b este estándar comparte la misma banda y puede recibir interferencia.

IEEE 802.11.2007. Este es un estándar que contiene todas las enmiendas al estándar 802.11 desde su introducción. A la fecha este es el estándar más completo que define la operación de una red LAN inalámbrica.

IEEE 802.11n. Este estándar fue emitido directamente y contiene enmiendas que agregan funcionalidad adicional al estándar 802.11 que incluye la tecnología MIMO *Multiple Input, Multiple Output*, en español: Entrada Múltiple, Salida Múltiple, esta tecnología utiliza múltiples antenas para transmisión y recepción, lo que significa aumento significativo de velocidad y rendimiento sin aumentar el ancho de banda. Este estándar hace uso de las bandas de 2.4 GHz y 5.4 GHz y tiene una velocidad de 600 Mbps en la capa física.

1.3 METODOLOGÍA DE DESARROLLO

La metodología que se utiliza es RUP *Rational Unified Process*, en español: Proceso Unificado de Rational[20].

1.3.1 RUP

El proceso de software propuesto por RUP está dirigido por los Casos de Uso, está centrado en la arquitectura, es iterativo e incremental.

Proceso dirigido por Casos de Uso. Una de las características del RUP es que el proceso está dirigido por Casos de Uso.

“Los Casos de Uso son una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no solo en términos de funciones

[20] Rational Software es una familia de software de IBM para el despliegue, diseño, construcción, pruebas y administración de proyectos en el proceso desarrollo de software.

que sería bueno contemplar. Se define un Caso de Uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los Casos de Uso representan los requisitos funcionales del sistema."[21]

En el Proceso Unificado, los Casos de Uso guían el diseño, implementación y prueba, no solamente especifican los requerimientos del mismo, son un elemento integrador y una guía de trabajo.

La figura 28 muestra cómo los Casos de Uso integran las actividades, basándose en los casos de uso se crean los modelos de análisis y diseño, luego la implementación y finalmente la verificación de que cada Caso de Uso esté correctamente implementado, es decir, que cumpla los requerimientos. [12]

Los Casos de Uso permiten la trazabilidad entre los modelos generados durante las distintas actividades del proceso de desarrollo, todos los modelos deben estar sincronizados con el modelo de Casos de Uso. Un ejemplo de trazabilidad se puede observar en el diagrama de actividad de la Figura 29.

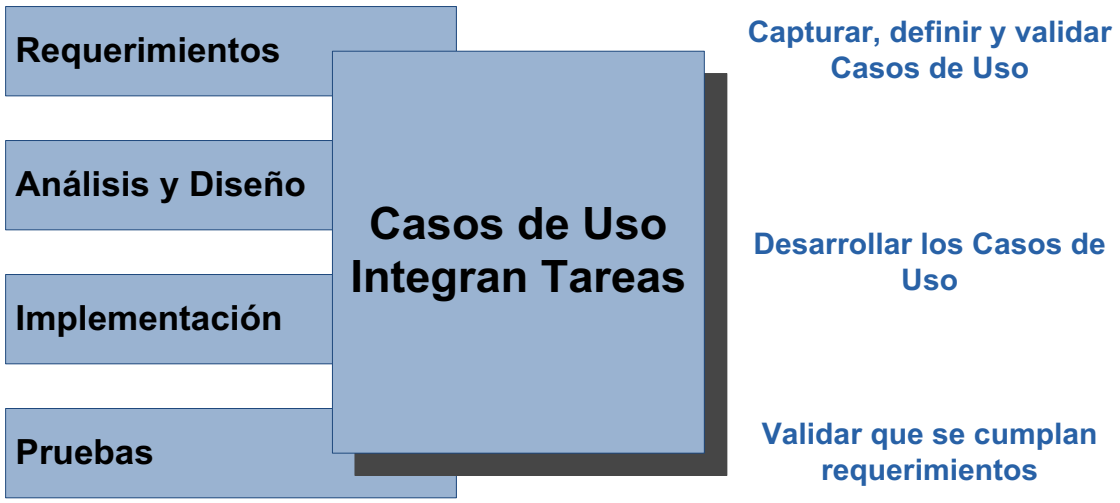


Figura 28. Casos de Uso integran tareas.

[21] Kruchten, P., The Rational Unified Process: An Introduction, 2000 Addison Wesley.

[12] Letelier, Patricio. *Introducción a RUP*.

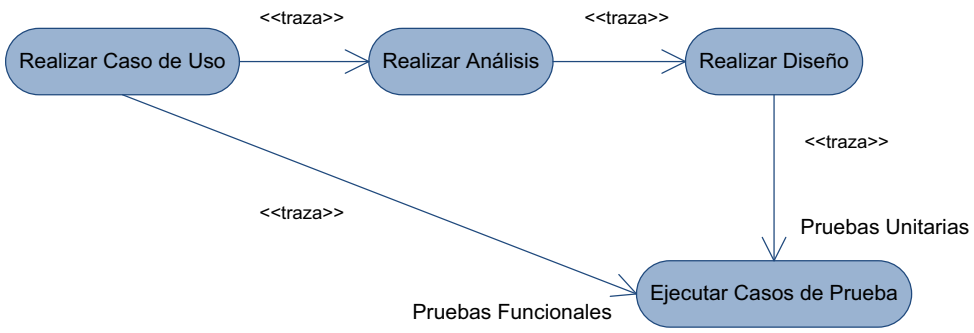


Figura 29. Trazabilidad de las actividades.

Proceso centrado en la arquitectura. La arquitectura comprende los aspectos organizativos más significativos del sistema, tiene relación con la manera en que tiene que ser construido el sistema y en qué orden.

"La arquitectura de un sistema es la organización o estructura de sus partes más relevantes, lo que permite tener una visión común entre todos los involucrados (desarrolladores y usuarios) y una perspectiva clara del sistema completo, necesaria para controlar el desarrollo"[22]

La arquitectura debe tomar en cuenta la calidad del sistema, rendimiento, reutilización y evolución del mismo, por lo que debe ser flexible durante el proceso de desarrollo. Elementos como la plataforma de software, el sistema operativo, el sistema de gestión de base de datos, protocolos de red, sistemas heredados, etc. Influencian directamente en la arquitectura del sistema y estas restricciones son requerimientos no funcionales del sistema.

En RUP se presta atención especial a la estructuración de una buena arquitectura para que no sean necesarios cambios fuertes durante la construcción y el mantenimiento del sistema.

[22] Kruchten, P., The Rational Unified Process: An Introduction, 2000 Addison Wesley.

La arquitectura y los Casos de Uso deben evolucionar de manera paralela durante el proceso de desarrollo del sistema, pues cada producto tiene una función y una forma, la funcionalidad está definida por los casos de uso y la forma la proporciona la arquitectura. Los Casos de Uso deben ajustarse a la arquitectura cuando se ejecutan y la arquitectura debe permitir el desarrollo de todos los Casos de Uso actuales y futuros.

En la Figura 30 se puede observar la evolución de la arquitectura durante las fases de RUP, donde se tiene una arquitectura más robusta en las fases finales.

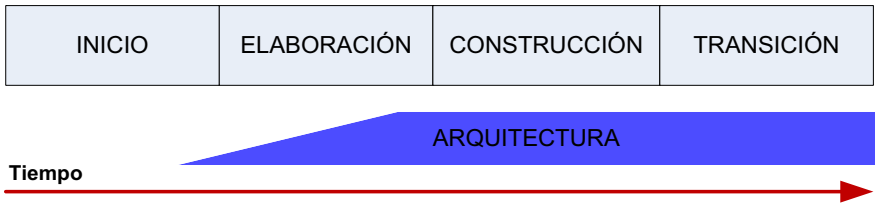


Figura 30. Desarrollo de la arquitectura durante fases del RUP.

Proceso Iterativo e Incremental. En RUP significa que el trabajo se divide en partes pequeñas o mini proyectos, esto permite que el equilibrio entre Casos de Uso y arquitectura se vaya logrando en cada mini proyecto y durante todo el proceso de desarrollo. Cada mini proyecto se puede ver como una iteración o recorrido por los flujos de trabajo fundamentales de la cual se obtiene un incremento que produce un avance en el producto.

La Figura 31 muestra la iteración como una cascada, donde la iteración pasa por: Requisitos, Análisis, Diseño, Implementación y Pruebas e Integración pero también es necesaria la planificación de la iteración, análisis de la iteración y actividades de la iteración.

Al decir que el RUP es un proceso iterativo e incremental se indica que consta de una serie de iteraciones, cada iteración se encarga de una parte de la

funcionalidad total del sistema pasando por los flujos de trabajo (Requisitos, Análisis, Diseño, Implementación, Pruebas e Integración) y perfeccionando la arquitectura. La iteración se analiza cuando termina, así, se pueden identificar requisitos nuevos o modificaciones a los existentes que afectan la iteración siguiente.

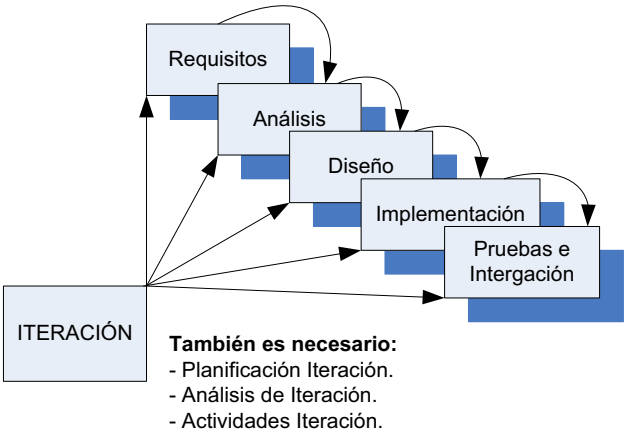


Figura 31. Iteración

1.3.1.1 FASES DEL RUP

El RUP divide al proceso en cuatro fases, en cada fase se realizan iteraciones que varían en número según el proyecto en las que se realiza un esfuerzo distinto en cada actividad. La figura 32 representa el ciclo de vida del RUP y muestra cómo cambia el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto.

Las elevaciones horizontales de cada disciplina dan un estimado del esfuerzo relativo a lo largo de las fases. Se puede ver, por ejemplo, se puede ver que el modelado de negocio se realiza en su mayoría en la fase Inicial aunque continúa hasta iniciada la fase de Transición.

El trabajo en la Distribución no empieza sino hasta iniciada la fase de Elaboración y el mayor esfuerzo comienza a mitad de la fase de Construcción.

Así, el diagrama del Ciclo de Vida de RUP es una visión aproximada de cuánto y cuándo de cada disciplina se ejecuta en cada fase. [2]

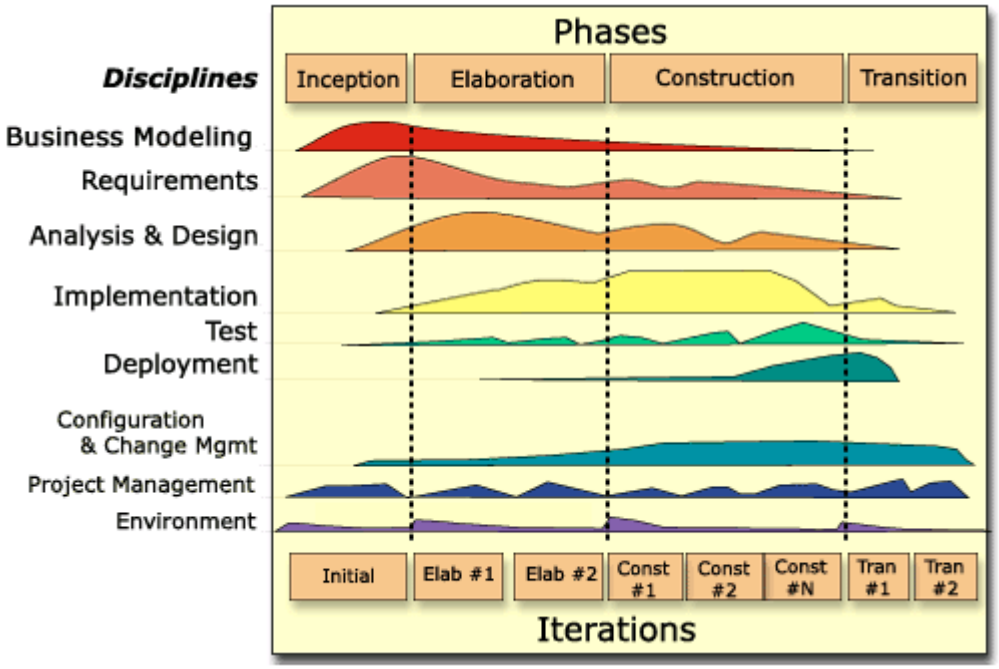


Figura 32. Ciclo de vida de RUP[23]

Como se puede apreciar en la Figura 32 el RUP está dividido en cuatro fases: Inicio, Elaboración, Construcción y Transición, en inglés: *Inception, Elaboration, Construction, Transition* respectivamente.

Al final de cada fase se tienen hitos bien definidos, es aquí donde se puede evaluar el avance del proyecto incluyendo lo que se ha hecho y los planes para continuar. Cada fase tiene un conjunto de objetivos que se concretan en las iteraciones de cada fase para que se pueda cumplir el hito de la fase.

En la Figura 33 se pueden ver las actividades principales e hitos de cada fase.

---

[23] Diagrama de dominio público, autoría de IBM.

[2] Ambler, Scott W, *A Manager's Introduction to The Relational Unified Process (RUP)*.

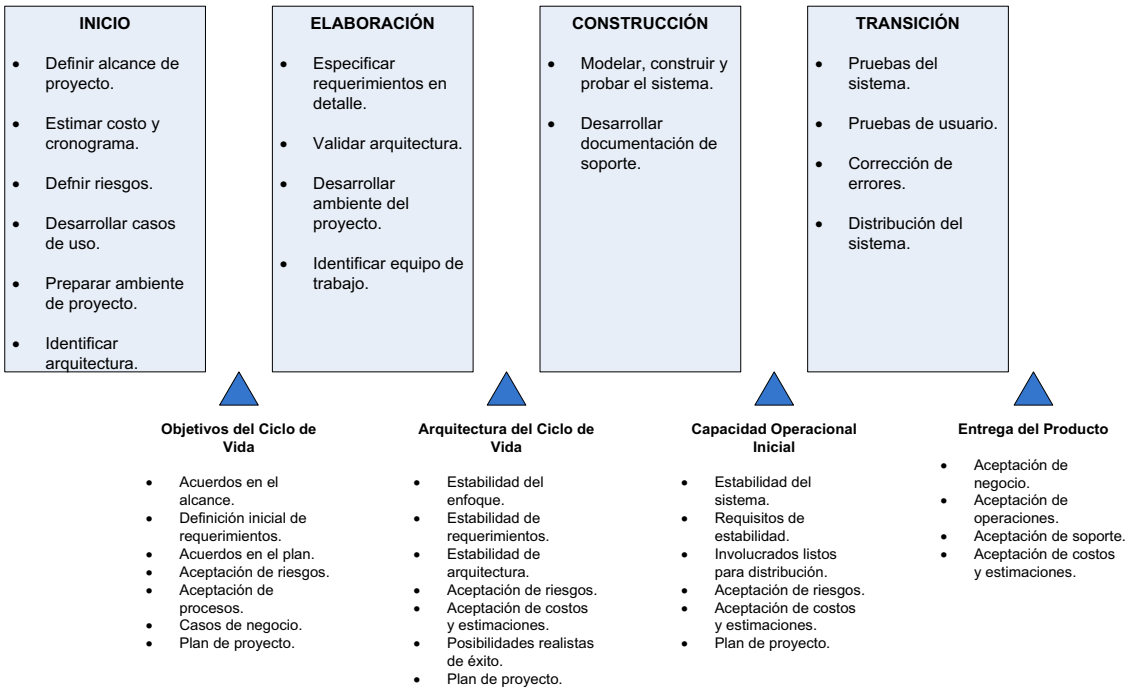


Figura 33. Fases del RUP e hitos.

La Fase Inicial. Los principales objetivos de esta fase es lograr el consenso de las partes involucradas en relación a los objetivos del proyecto y obtener financiamiento. Esto se logra con el desarrollo de un modelo de requerimientos de alto nivel que delimitan el alcance del proyecto y un prototipo de la interfaz de usuario. En esta fase también se instala el ambiente de trabajo y se adapta el proceso para el equipo, se desarrolla un plan de alto nivel que indica cómo se desarrollará el proyecto. El hito, al final de la fase, son los Objetivos del Ciclo de Vida, *LCO* por sus siglas en inglés (*Lifecycle Objectives*), donde los involucrados evalúan el estado del proyecto y deben estar de acuerdo en los siguientes puntos:

- El alcance del proyecto.
- Que los requerimientos iniciales han sido identificados aún sin mayor detalle.
- Que el plan de desarrollo de software es realista.
- Que los riesgos han sido identificados y serán administrados apropiadamente.
- Que los casos de negocio para el proyecto tienen sentido.

- Que el proceso de desarrollo está adaptado de manera adecuada al equipo.

La Fase de Elaboración. Durante esta fase se especifican los requerimientos en mayor detalle y se verifica la arquitectura del sistema. El detalle de los requisitos es el suficiente solamente para entender los riesgos de arquitectura y asegurar que existe una comprensión del alcance de cada requisito para su posterior planificación. Para verificar la arquitectura se implementa y prueba una estructura básica extremo a extremo de código que soporta los casos de uso de alto riesgo del sistema. Al final de esta fase se tiene el hito: Arquitectura del Ciclo de Vida, *LCA* por sus siglas en inglés (*Lifecycle Architecture*), donde los involucrados evalúan el estado del proyecto y deben estar de acuerdo en los siguientes puntos:

- El enfoque del proyecto está estable y es realista.
- Los requerimientos del proyecto, aunque pueden variar aún.
- La arquitectura es estable y suficiente para satisfacer los requerimientos.
- Los riesgos continúan siendo administrados.
- Los gastos actuales son aceptables y se han realizado estimaciones razonables para costos y cronogramas futuros.
- El equipo del proyecto tiene oportunidades reales de tener éxito.
- Que se tengan planes de iteraciones detalladas para las siguientes iteraciones de *Construcción*, así como un plan de proyecto de alto nivel.

La Fase de Construcción. En esta fase el objetivo principal es desarrollar el sistema hasta que esté listo para la implementación. La prioridad ahora son los requerimientos y se analiza y diseña para completar su especificación y se programa y prueba el software. Si es necesario se distribuyen versiones alfa y beta del sistema, interna o externamente, para obtener retroalimentación por parte de los usuarios. Al final de esta fase se debe revisar el hito: Capacidad Operativa Inicial, *IOC* por sus siglas en inglés (*Initial Operational Capability*) donde los involucrados en el sistema deben acordar en lo siguiente:

- El software y la documentación de soporte son aceptables para la implementación.
- Los involucrados en el sistema están listos para la implementación.
- Los riesgos continúan siendo administrados de manera efectiva.
- Los gastos actuales son aceptables y se han realizado estimaciones razonables para costos y cronogramas futuros.
- Que se tengan planes de iteraciones detalladas para las siguientes iteraciones de *Transición*, así como un plan de proyecto de alto nivel.

La Fase de Transición. Esta fase se enfoca en poner el sistema en producción. Se realizan pruebas del sistema por el equipo de control de calidad y los usuarios, se realizan las correcciones pertinentes y el afinamiento del sistema. En esta fase también se realiza la capacitación a los usuarios finales, el personal de soporte y operaciones. Al finalizar esta fase se revisa el hito: Entrega del Producto, *PR* por sus siglas en inglés (*Product Release*) donde las partes involucradas acuerdan lo siguiente:

- El sistema, incluyendo la documentación y el soporte está listo para la implementación.
- Los gastos actuales son aceptables y se han realizado estimaciones razonables para costos futuros.
- El sistema puede estar en operación una vez en producción.
- El sistema tendrá el soporte adecuado cuando esté en producción

1.3.1.2 DISCIPLINAS DEL RUP

Las fases del RUP están divididas en una o más iteraciones. Las iteraciones se encargan de una parte del sistema, cada iteración tiene un plan detallado con un objetivo específico, las iteraciones se basan en el trabajo realizado por iteraciones anteriores y se ensamblan al final del sistema haciendo el proceso incremental. Las iteraciones se pueden ver en la parte inferior de la Figura 32 donde cada fase está dividida en una o más iteraciones. Cuando termina una iteración

especialmente en la fase de Construcción, una pequeña porción del sistema se ha completado e incluso puede ser distribuida a los usuarios como una versión alfa o beta.

La naturaleza iterativa del RUP se refleja en el enfoque de sus disciplinas, que son una agrupación lógica de actividades que se realizan durante un proyecto. El corazón del RUP son sus disciplinas, no las fases. Durante cada iteración se va hacia adelante y atrás por las actividades de las disciplinas realizando las tareas necesarias para cumplir los objetivos de la iteración, es decir, durante una iteración se selecciona un parte de los requerimientos, se analiza, diseña, codifica, prueba e integra con los productos de las iteraciones anteriores. Las disciplinas del RUP son: [12]

1. Modelado de Negocio
2. Requerimientos
3. Análisis y Diseño
4. Implementación
5. Pruebas
6. Distribución
7. Administración de Configuración y Cambios
8. Administración de Proyecto
9. Ambiente

Modelado de Negocio. Aquí se procura llegar a un mejor entendimiento de la organización donde se implantará el producto. Los objetivos son:

- Entender la estructura y dinámica de la organización a la que va orientada el sistema.
- Identificar la problemática actual de la organización y eventuales mejoras.
- Asegurar que clientes, usuarios finales y desarrolladores entiendan el negocio de la organización.

---

[12] Letelier, Patricio. *Introducción a RUP*.

- Desarrollar un modelo de dominio que refleje el negocio.

Para lograr los objetivos, es necesario definir procesos, roles y responsabilidades de la organización mediante un modelo de Casos de Uso de negocio y un Modelo de Objetos de Negocio, complementariamente puede desarrollarse un glosario.

Requerimientos. Es una de las disciplinas más importantes porque se define qué es lo que tiene que hacer el sistema. Los requisitos son el contrato que se debe cumplir y los usuarios finales deben comprender y aceptar los mismos. Los objetivos en esta disciplina son los siguientes:

- Establecer y mantener un acuerdo entre los clientes e inversores del sistema sobre lo que el sistema hace y sus necesidades.
- Definir el alcance del sistema.
- Planificar los contenidos técnicos de las iteraciones.
- Estimar costos y tiempo de desarrollo del sistema.
- Definir una interfaz de usuario enfocada en necesidades y funciones del usuario.

Los requerimientos pueden ser funcionales y no funcionales, donde los primeros representan funcionalidad misma del sistema y se representan con Casos de Uso, los otros son atributos que debe tener el sistema, por ejemplo: fiabilidad, eficiencia, portabilidad, etc.

Análisis y Diseño. El objetivo aquí es analizar los requerimientos y diseñar una solución a ser implementada, es decir, se especifica cómo realizar el sistema. Los objetivos son los siguientes:

- Transformar los requerimientos al diseño del sistema.
- Desarrollar una arquitectura para el sistema (arquitectura candidata).
- Desarrollar elementos de prueba conceptuales para validar la arquitectura candidata.

- Adaptar el diseño para que sea consistente con el entorno de implementación y la arquitectura.
- Diseño de la base de datos, red, etc.

Los resultados finales más importantes de esta disciplina son: El modelo de diseño y la documentación de la arquitectura de software.

Implementación. En esta disciplina se implementa el código fuente del sistema y se realizan pruebas unitarias, cada desarrollador es responsable de probar su código, en cada iteración se realiza lo siguiente:

- Planificar los subsistemas a ser implementados y en qué orden serán integrados.
- Decidir en qué orden implementar los elementos del subsistema, cada desarrollador decide el orden.
- Indicar posibles errores de diseño.
- Realizar pruebas unitarias.
- Integrar el sistema según el plan de integración.

La integración es incremental, es decir, se añade un elemento a la vez para localizar posibles fallos y probar los componentes de manera exhaustiva.

Pruebas. El objetivo principal es el evaluar la calidad del sistema, no sirve para aceptar o rechazar el producto al final del proceso. Las pruebas son integradas durante todo el ciclo de vida del sistema. Las actividades principales son:

- Encontrar y documentar defectos en el sistema.
- Asesorar sobre la calidad del software.
- Desarrollar casos de prueba.
- Verificar funcionalidad del producto según el diseño.
- Comprobar que los requerimientos estén correctamente implementados.

Distribución. El objetivo principal es producir una distribución del producto y entregarlo a los usuarios.

- Probar el producto en ambiente de producción.
- Empaquetar el software para su distribución.
- Distribuir el software.
- Instalar el software.
- Proporcionar soporte a los usuarios.
- Capacitar a los usuarios.
- Migrar sistemas o bases de datos heredados.

Administración de Configuración y Cambios. El objetivo aquí es el control de versiones. Las principales actividades son:

- Administrar control de cambios.
- Planificar configuración de control.
- Configuración de ambiente de generación de versiones.
- Administrar tiempos y versiones entregables.

Administración del proyecto. En esta disciplina se administra objetivos, riesgos y restricciones para desarrollar el producto, también se deben coordinar tareas con el personal. Las actividades críticas son:

- Facilitar una estructura de trabajo para la gestión de proyectos de software.
- Proveer mejores prácticas para planificación, contratación de personal, ejecución y monitoreo del proyecto.
- Disponer una estructura de trabajo para manejar los riesgos.

Ambiente. La finalidad principal en este punto es el entregar soporte al proyecto usando herramientas y procedimientos adecuados. Se indica una especificación de las herramientas o utilidades que serán utilizadas, las actividades críticas incluyen:

- Seleccionar y adquirir las herramientas o utilidades.
- Configurar las herramientas para ajustarlas al proyecto actual.
- Configurar el proceso.

1.3.2 LENGUAJE DE MODELADO UNIVERSAL UML Y ARTEFACTOS DEL PROYECTO

Se conoce como artefacto o producto a una porción de información producido, modificado o utilizado durante el proceso de desarrollo de software. Estos artefactos o productos son resultados tangibles del proyecto, por ejemplo: Un documento como el documento de requerimientos, un modelo como el modelo lógico del sistema o un elemento que pertenece al modelo como una entidad del modelo. [17]

El Lenguaje de Modelado Universal UML por sus siglas en inglés (*Unified Modeling Language*) es un lenguaje gráfico para visualización, especificación, construcción y documentación de los artefactos que se producen durante el proceso de desarrollo de software. UML provee un estándar para realizar proyectos de sistemas, información conceptual como procesos de negocios y funcionalidad del sistema, información concreta como clases en un lenguaje de programación específico, esquemas de bases de datos y componentes de software reutilizables.

Para el presente proyecto se han considerado los siguientes artefactos y se utilizará UML en los que sea necesario:

- **Diagramas de Actividad**, que indicarán los flujos de trabajo del sistema.
- **Características del Producto**, desde el punto de vista de necesidades del usuario, es decir, los requerimientos.

---

[17] RUMBAUGH, James., JACOBSON Ivar., BOOCH., Grady. *Unified Modeling Language User Guide 2nd Edition*.

- **Glosario**, que permitirá establecer una terminología clara.
- **Modelo de Casos de Uso**, donde se presenta la funcionalidad del sistema y los actores.
- **Especificación de Casos de Uso**, solamente para los casos que requieran detalle.
- **Modelo de Análisis y Diseño**, donde se incluyen Diagramas de Clases.
- **Modelo Físico**, donde se muestran las tablas del sistema.

1.4 DESCRIPCIÓN DE LAS HERRAMIENTAS

En esta sección se presenta una descripción de las herramientas de lenguaje utilizadas en el desarrollo de este proyecto: Lenguaje C#, Visual Studio .NET 2010.

1.4.1 .NET Y VISUAL STUDIO .NET

A continuación se presentan las características principales del Framework de .NET:

Programación orientada a objetos, el .NET Framework y el C# están completamente basados en los principios de orientación a objetos desde el inicio.

Óptimo diseño, la biblioteca de clases base está diseñada desde abajo hacia arriba de una manera intuitiva.

Independencia del lenguaje, en .NET todos los lenguajes como VB, C#, C++ administrado, se compilan a un lenguaje intermedio común, esto significa que los lenguajes son interoperables. [15]

---

[15] Nagel, Christian; Evjen, Bill; Glynn, Jay; Watson, Karli; Skinner, Morgan. *Professional C# 4 and .NET 4*.

Mejor soporte para páginas Web dinámicas, las páginas Web dinámicas ASP .NET, el código es compilado y puede ser escrito en lenguaje de alto nivel como C# o VB 2010, se cuenta incluso con soporte para Ajax[24].

Acceso a datos eficiente, el .NET Framework contiene un conjunto de componentes conocido como ADO.NET, que proporciona un acceso eficiente a bases de datos relacionales, archivos, directorios, etc. El soporte para XML está incorporado, lo que permite manipular los datos e importar o exportar a plataformas no Windows.

Código compartido, .NET ha renovado completamente la manera en que las aplicaciones comparten el código, introduciendo el concepto de *assembly*[25], en lugar de las DLL. Los *assemblies* tienen control de versión y *assemblies* de diferentes versiones pueden residir uno junto a otro.

Seguridad mejorada, cada assembly puede contener información que indica precisamente qué usuario o qué categoría de usuario o proceso tiene permitido ejecutar qué métodos y en qué clases.

Instalación sin impacto, existen dos tipos de assemblies, los compartidos y privados. Los compartidos con librerías disponibles para todo el software, mientras que los privados son completamente independientes por lo que el proceso de instalación es simple, no hay entradas en el registro, solo se ponen los archivos apropiados en el directorio adecuado.

Soporte para servicios Web, .NET tiene un soporte completo para el desarrollo de servicios Web como si estuviera desarrollando cualquier otra aplicación.

Visual Studio 2010, .NET viene con un ambiente integrado de desarrollo, Visual Studio 2010, con el cual se puede trabajar con cualquier lenguaje: C++, C# o VB

[24] Ajax, acrónimo de *Asynchronous JavaScript And XML,* es una tecnología asíncrona en que ciertos datos de una página Web se piden al servidor y se cargan en segundo plano sin cambiar el diseño o comportamiento de la página, no hace falta recargar la página.
[25] Assembly es el nombre que dio Microsoft a una unidad lógica de código parcialmente compilado.

2010, también con ASP.NET o XML. Visual Studio integra las mejores características de ambiente de los respectivos lenguajes.

C#, este lenguaje fue desarrollado desde cero, completamente orientado a objetos creado específicamente para el uso con .NET.

Un componente fundamental del .NET Framework, es su ambiente de interpretación en tiempo de ejecución, conocido como Lenguaje Común en Tiempo de Ejecución o CLR por sus siglas en inglés: *Common Language Runtime*. El código que se ejecuta bajo control del CLR se le denomina código administrado.

Antes de que el CLR ejecute el código, este debe ser compilado, en .NET la compilación se realiza en dos pasos:

- Compilación del código fuente a Lenguaje Intermedio, IL por sus siglas en inglés: *Intermediate Languaje*.

- Compilación del IL a código específico de la plataforma por el CLR.

Este proceso de compilación de dos etapas es muy importante, la existencia del lenguaje IL es la clave para proveer muchos de los beneficios de .NET.

El lenguaje IL, al igual que el *Bytecode* de Java, comparten la idea de ser lenguajes de bajo nivel con una sintaxis simple (basada en códigos numéricos más que en texto), los cuales pueden ser traducidos rápidamente a código nativo de la máquina donde se ejecuta. El tener esta sintaxis de código universal bien definida tiene ventajas significantes: independencia de plataforma, mejora del rendimiento e interoperabilidad de lenguaje.

Independencia de Plataforma. Significa que el mismo archivo con las instrucciones en IL puede ser puesto en cualquier plataforma; en ejecución, la etapa final de compilación puede completarse satisfactoriamente y así el

código se ejecutará en esa plataforma particular. Esta independencia de plataforma es solamente teórica actualmente, pues al realizar una implementación completa de .NET está disponible solamente para Windows.

Mejora del Rendimiento. El código IL es siempre compilado “justo a tiempo”, y se la conoce como compilación JIT por sus siglas en inglés *Just in Time*. En lugar de compilar la aplicación completa de una sola vez (lo que puede llevar a una demora en el arranque de la aplicación), el compilador JIT compila simplemente una porción de código mientras sea requerido (justo a tiempo).

Cuando el código ha sido compilado una vez, el código nativo resultante se almacena hasta que se sale de la aplicación, por lo que no es necesario volver a compilar la próxima vez que esa porción de código se ejecuta. Microsoft argumenta que este proceso es más eficiente que compilar todo el código, porque existe la probabilidad de que grandes porciones de código no se ejecuten en un momento dado y usando la compilación JIT ese código nunca se compilará. El compilador JIT sabe exactamente en qué procesador se ejecutará el código, lo que significa que, en la segunda etapa de compilación, se puede optimizar el código ejecutable final para aprovechar cualquier característica particular de instrucciones de máquina ofrecidas por ese procesador particular.

Interoperabilidad de Lenguaje. El uso del lenguaje IL, facilita la interoperabilidad de lenguaje. Se puede obtener IL de un código fuente escrito en un lenguaje por ejemplo Visual Basic 2010, y este código puede interactuar con código compilado a IL de fuentes escritas en otro lenguaje, por ejemplo C#.

Otro elemento fundamental de la plataforma .NET es el Sistema de Tipos común o ***CTS*** por sus siglas en inglés: *Common Type System*. La especificación del CTS describe completamente todos los tipos de datos posibles y términos de programación soportados por el tiempo de ejecución, especifica cómo estas entidades pueden interactuar entre sí y detalla cómo están representadas en formato de metadatos de .NET.

Un lenguaje dado de .NET puede no soportar alguna característica definida por el CTS, existe entonces la Especificación Común de Lenguaje o CLS por sus siglas en inglés: *Common Language Specification*. El CLS es una especificación que define un conjunto de reglas de lenguaje básicas requeridas por la mayoría de las aplicaciones. Las reglas de CLS definen un subconjunto del CTS, es decir, todas las reglas que se aplican al CTS se aplican también a CLS, salvo que se definan reglas más estrictas en CLS. CLS ayuda a mejorar y garantizar la interoperabilidad entre lenguajes mediante la definición de un conjunto de características en las que se pueden basar los programadores y que están disponibles en una gran variedad de lenguajes.

La plataforma .NET provee una biblioteca de clases base disponible para todos los lenguajes de programación de .NET. Esta biblioteca de clases, no solamente encapsula primitivas como manejo de hilos, entrada y salida (I/O), sistemas de representación gráfica, interacción con varios dispositivos de hardware externos, sino también provee soporte para servicios requeridos para la mayoría de aplicaciones del mundo real.

Por ejemplo, la biblioteca de clases base define tipos que facilitan el acceso a datos, manipulación de documentos XML, programación de seguridad, construcción de interfaces de usuario basadas en Web o escritorio.

La Figura 34 muestra cómo se relacionan el CLR, CTS, CLS y la biblioteca de clases base.

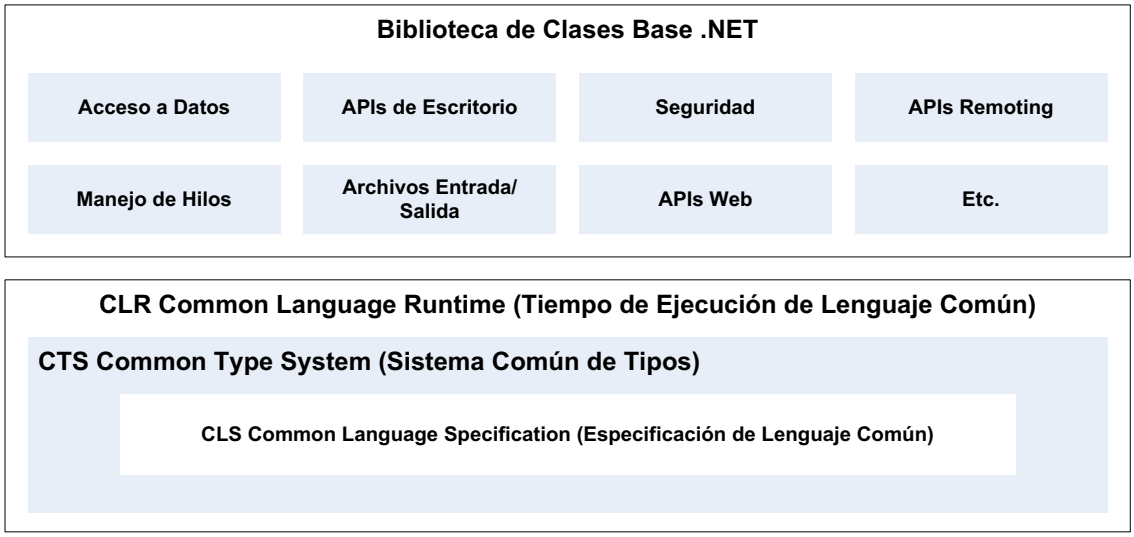


Figura 34. Relación entre CLR, CTS, CLS y la biblioteca de clases base.

1.4.2 CUESTIONES DEL LENGUAJE C#

El lenguaje C# es un lenguaje nuevo, desde el punto de vista que es creado desde cero, elaborado exclusivamente para esta plataforma. C# es un lenguaje de programación cuya sintaxis es similar a la de Java, esto porque Java y C# forman parte de la familia de lenguajes C. [19]

Varias de las construcciones de sintaxis de C# están modeladas de varios aspectos de Visual Basic 6.0 y C++. Por ejemplo, como VB6, C# permite la declaración de métodos con parámetros variables (mediante arreglos). Como C++, C# permite sobrecargar operadores, crear estructuras, enumeraciones y punteros a funciones o “callback” (mediante delegados).

C# soporta también características tradicionalmente encontradas en lenguajes funcionales (Ej. LISP o Haskell) como expresiones lambda y tipos anónimos. Además, con la llegada de LINQ (Language Integrated Query), en español: Lenguaje Integrado de Consulta, C# soporta una serie de características que lo hacen único entre los lenguajes de programación, sin embargo, la mayor parte de

[19] Troelsen, Andrew. *Pro C# 2010 and the .NET 4 Platform (5th Edition)*.

C# está influenciado por lenguajes basados en C.

El hecho que C# sea un híbrido de muchos lenguajes, el resultado es un producto sintácticamente limpio y simple como VB6 y provee casi tanta potencia y flexibilidad como C++. Las siguientes son características principales de C#:

C# no requiere de punteros, el lenguaje no requiere la manipulación directa de punteros.

Administración automática de memoria a través del recolector de basura (Garbage Collector).

Sintáctica formal para clases, interfaces, estructuras, enumeraciones y delegados.

Habilidad de sobrecargar operadores de tipos personalizados sin complejidad.

Soporte para la programación basada en atributos. Esto permite comentar tipos y sus miembros para calificar su comportamiento.

Con el lanzamiento de .NET 2.0, C# se actualizó y las características más notables son:

Posibilidad de crear tipos y miembros *genéricos*, con lo cual se puede hacer código eficiente y de tipos seguros los cuales se definen al momento de interactuar con el elemento genérico.

Soporte para métodos anónimos que permiten la creación de funciones en línea en cualquier lugar que un tipo delegado es requerido.

Capacidad para definir un tipo único en múltiples archivos de código (o si es necesario una representación en memoria) utilizando la palabra clave *partial*. Las definiciones de tipo parcial permiten dividir la definición de una clase, estructura o interfaz en varios archivos.

.NET 2.5 añadió mejoras al lenguaje como:

Soporte para consultas bien definidas como LINQ que usado para interactuar con varias formas de datos.

Soporte para tipos anónimos que permiten modelar la forma de un tipo más que su comportamiento. Los tipos anónimos consisten en definiciones en línea de tipos, sin necesidad de especificar un nombre de tipo.

Capacidad para extender funcionalidad de un tipo existente (sin sub-clases) haciendo uso de los métodos de extensión.

Inclusión del operador lambda “=>” que simplifica el uso de delegados, donde en el lado izquierdo van los parámetros de entrada, si existen, y en el derecho las instrucciones.

Nueva sintaxis de implementación de objetos que permite establecer valores de propiedades al momento de la creación del objeto.

La actual plataforma .NET 4.0, contiene nuevas características para el C#:

Soporte para parámetros opcionales de métodos, así como argumentos con nombre para referirse al nombre en lugar de la posición del argumento.

Búsqueda dinámica de miembros en tiempo de ejecución mediante la palabra clave “dynamic”. El tipo “dynamic” permite que las operaciones en las que se produce omitan la comprobación de tipo en tiempo de compilación. En su lugar, se resuelven estas operaciones en tiempo de ejecución.

Es importante indicar que el lenguaje C# produce código capaz de ejecutarse únicamente en plataforma .NET, es decir, dentro del tiempo de ejecución de .NET.

Oficialmente hablando, el término que describe el código dirigido a .NET se llama código administrado o en inglés: *managed code*. La unidad binaria que contiene el código administrado se denomina assembly. Por el contrario, el código que no puede ser ejecutado en el tiempo de ejecución de .NET se llama código no administrado o en inglés: *unmanaged code*.

1.4.3 SQL SERVER 2008

Microsoft SQL Server 2008, es un producto de base de datos relacional con T-SQL[26] (Transact SQL) como su lenguaje de programación principal. SQL Server 2008 tiene integración con el CLR de .NET, lo cual introduce una nueva familia de tipos de datos basados en tipos de datos del CLR definidos por el usuario, permitiendo capacidades jerárquicas y geoespaciales en la base de datos. [16]

Algunas de las ventajas de SQL Server 2008 son:

- La conexión con SQL Server resulta fácil con numerosas opciones de conectividad de datos y tecnologías para importar y exportar datos, como: Flujo de Datos Tabular o TDS por sus siglas en inglés, *Tabular Data Stream*; XML, Servicios de Integración, Copia de Archivos a Bases de Datos (Bulk Copy), Conectividad Nativa SQL, OLE DB, ODBC, consultas distribuidas con el Coordinador de Transacciones Distribuidas o DTC por sus siglas en inglés: *Distributed Transaction Coordinator*.

- El motor funciona bien con datos: XML, espacial, palabras dentro de texto, y datos blob (*binary large object*), que son elementos utilizados en las bases de datos para almacenar datos de gran tamaño que cambian de forma dinámica.

[26] T-SQL es una extensión de SQL, propiedad de Microsoft y Sybase.

[16] Nielsen, Paul. *Microsoft SQL Server 2008 Bible.*

- SQL Server tiene un conjunto de herramientas y componentes para trabajar con datos multidimensionales, análisis de datos, creación de cubos y "*data mining*" que consiste en extracción no trivial de información que reside implícitamente en los datos.

- SQL tiene una solución completa de reportes muy vistosos que permite a los usuarios crear reportes y realizar un seguimiento de quién vio un reporte y cuándo.

- SQL Server expone un impresionante nivel de detalle en diagnóstico con herramientas como: Performance Studio, SQL Trace/Profiler y Vistas y Funciones de Administración de Base de Datos.

- SQL Server incluye varias opciones de alta disponibilidad con diversos grados de latencia, el rendimiento, la distancia física y la sincronización.

- SQL Server puede administrarse de forma declarativa mediante la administración basada en políticas.

- La herramienta SQL Server Management Studio es una interfaz de usuario madura para el desarrollador de base de datos y el administrador.

El Motor de Base de Datos de SQL Server, llamado también Motor Relacional, es el núcleo del SQL Server. Es el componente que controla todo el trabajo de base de datos relacional. Como el SQL es un lenguaje descriptivo, significa que describe la consulta solo al motor y el motor asume el trabajo desde aquí. Dentro del Motor Relacional hay varios componentes y procesos clave, incluidos los siguientes:

- **Algebrador:** Verifica la sintaxis y transforma una consulta a una representación interna que es utilizada por los siguientes componentes.

- **Optimizador de Consultas:** Determina cómo mejorar el proceso de consulta basada en los costos de diferentes tipos de operaciones de ejecución de consultas. Los planes estimados y actuales de ejecución de la consulta pueden verse gráficamente, o en XML, usando Management Studio o el analizador de SQL.

- **Motor de Consultas o Procesador de Consultas:** Ejecuta las consultas de acuerdo con los planes generados por el Optimizador de Consultas.

- **Motor de Almacenamiento:** Trabaja para el Motor de Consultas y administra la lectura y escritura al disco.

- **Administrador de Buffer:** Analiza las páginas de datos utilizadas y precargadas de los archivos en memoria, reduciendo así la dependencia en el rendimiento de E/S del disco.

- **Punto de Control:** Proceso que escribe los datos de páginas modificadas de la memoria al archivo de datos.

- **Monitor de Recursos:** Optimiza el caché del plan de consulta en respuesta a la memoria disponible, lo hace manera inteligente removiendo planes de consulta antiguos.

- **Administrador de Bloqueos:** Administra el alcance de los bloqueos dinámicamente para balancear el número de bloqueos con el tamaño del bloqueo.

- **SQLOS:** SQL Server consume muchos recursos y por esta razón necesita control directo de los recursos disponibles (memoria, hilos, solicitud de E/S, etc.). Simplemente dejando la administración de recursos a Windows no es suficiente para SQL Server. SQL Server incluye su propia capa de OS, *SQLOS*, que gestiona la totalidad de sus recursos internos.

SQL Server 2008 soporta la instalación de hasta 16 (Edición *Workgroup*) o 50 (Edición *Estándar* o *Enterprise*) instancias del Motor Relacional en un servidor físico. Aunque las instancias comparten algunos componentes, cada una funciona como una instalación independiente y completa de SQL Server.

1.4.4 INTERACCIÓN DEL LENGUAJE C# CON LOS DATOS

Para la interacción con la base de datos en el caso de estudio se utiliza LINQ como extensión del lenguaje C# para acceso a los datos.

LINQ es un conjunto de extensiones del .NET Framework que incluyen el lenguaje integrado de consulta (*Language-INtegrated Query – LINQ por sus siglas en inglés*) y operaciones de actualización y transformación de datos. LINQ extiende al lenguaje C# con sintaxis de lenguaje nativo para consultas y proporciona librerías necesarias para tomar ventaja de estas capacidades.

LINQ permite escribir consultas utilizando C# directamente contra las entidades, donde las entidades son las representaciones de las tablas físicas de la base de datos. Estas consultas están representadas mediante un árbol de comandos de consulta, las cuales se ejecutan contra un objeto contexto.

La creación y ejecución de consultas LINQ contra entidades tiene la siguiente secuencia:

1. Construcción de una instancia del objeto consulta del objeto contexto.
2. Creación de una consulta LINQ en C# utilizando la instancia del objeto consulta.
3. Conversión de operadores de consulta LINQ estándar y expresiones a árboles de comandos.
4. Ejecución de la consulta, en representación de árbol de comandos, contra la fuente de datos. Las excepciones se pasan directo al cliente.
5. Visualización de resultados de consulta al cliente.

Objeto Contexto. La clase *ObjectContext* es la clase principal para interactuar con datos mapeados a objetos que son instancias de los tipos de entidad que se definen en un modelo conceptual. Una instancia de la clase *ObjectContext* encapsula lo siguiente:

- Una conexión con la base de datos, en forma de un objeto *EntityConnection*.
- Los metadatos que describen el modelo, en forma de un objeto *MetadataWorkspace*.
- Un objeto *ObjectStateManager* que realiza el seguimiento de los objetos durante las operaciones de creación, actualización y eliminación.

Ejemplo:

```
// Creación del objeto context.
ObjectContext context =
    new ObjectContext("name= TREASURYWORKSEntities");

// Se puede especificar un repositorio de datos por defecto,
// en caso de haber más de un repositorio.
context.DefaultContainerName = "TREASURYWORKSEntities";

ObjectSet<SER_NEGOCIACION> query =
context.CreateObjectSet<SER_NEGOCIACION>();

// Iterar por el arreglo de negociaciones. Cambiando el estado.
foreach (SER_NEGOCIACION nego in query)
    nego.CODESTADOOPERACION = 2; // Cambia a estado CONFIRMADA.
```

Objeto Consulta. Una instancia de la clase *ObjectQuery* representa una consulta que retorna una colección de cero o más entidades tipificadas. Un objeto consulta se puede construir de un objeto contexto existente o de forma manual, pero siempre pertenece a ese contexto.

El contexto como se indicó anteriormente provee información de conexión y meta datos requeridos para construir y ejecutar la consulta. La clase *ObjectQuery* implementa la interface *IQueryable*, cuyos métodos permiten que las consultas se construyan de forma incremental. Se puede también dejar que el compilador infiera el tipo de las entidades mediante el uso de la palabra reservada de C#: var.

Ejemplo:

```
// Retorna negociaciones pertenecientes al cliente dado
ObjectQuery<SER_NEGOCIACION> query =
    context.SER_NEGOCIACION
    .Where("it.CODCLIENTE = @cliente",
    new ObjectParameter("cliente", codCliente));
```

Creación de la consulta. Las instancias de la clase *ObjectQuery* que implementan las interface *IQueryable,* sirven como origen de datos para consultas LINQ a entidades. En una consulta se especifica qué es lo que se quiere obtener del origen de datos, se puede también especificar orden, agrupamiento y forma en la que deben ser devueltos los datos. En LINQ, una consulta es almacenada en una variable, esta variable no ejecuta acción alguna y no devuelve datos, simplemente almacena información de la consulta. Luego de crear la consulta, se debe ejecutar la misma para obtener los datos requeridos.

Conversión de la consulta. Para ejecutar una consulta LINQ contra las entidades, la consulta LINQ debe ser convertida a una representación de árbol de comandos para que pueda ser ejecutada contra las entidades.
Las consultas LINQ a entidades se componen de operadores estándar de LINQ (Ejemplo: Select, Where, GroupBy) y expresiones (Ejemplo: X > 2, SER_NEGOCIACION.CODESTADOOPERACION, etc.). Los operadores de LINQ no están definidos por una clase, sino más bien son métodos en una clase.

En LINQ, las expresiones pueden contener cualquier cosa permitida por los tipos en el espacio de nombres *System.Linq.Expressions* y, por extensión, cualquier

cosa que se pueda representar en una función *lambda*[27]. Este es un superconjunto de las expresiones que están permitidas por las entidades, las cuales, por definición están restringidas a operaciones permitidas por la base de datos y soportadas por *ObjectQuery*.

Los operadores y expresiones están representados por una jerarquía de tipo único, que se colocan en un árbol de comandos. El árbol de comandos es utilizado por el Framework de las entidades para ejecutar la consulta. Si la consulta LINQ no se puede expresar como un árbol de comandos, se produce una excepción cuando la consulta se está convirtiendo.
La conversión de LINQ a entidades incluye dos sub-conversiones: la conversión de los operadores de consulta estándar, y la conversión de las expresiones.

Ejecución de la consulta. Después que se ha creado una consulta LINQ, se convierte a una representación en forma de árbol de comandos, la misma que se ejecuta contra los datos. En tiempo de ejecución de la consulta, las expresiones o componentes de la consulta se evalúan en el cliente o en el servidor.

Materialización. Es el proceso de devolver los datos solicitados al cliente, estos datos son de tipos pertenecientes al CLR[28]. En LINQ no se devuelven registros de datos, siempre se tienen tipos de datos pertenecientes al CLR definidos por el usuario, por la entidad o generados por el compilador (en caso de ser tipos anónimos). La materialización de objetos se realiza por el framework de las entidades, las excepciones que se generan cuando no se puede mapear entre las entidades y el CLR, se envían al cliente durante la materialización.
Los resultados de las consultas pueden devolver generalmente los siguientes datos:

- Una colección de cero o más objetos de tipo de la entidad o una proyección de tipos complejos definidos en el modelo conceptual.

[27] Una expresión lambda es una función o una subrutina sin nombre que se puede utilizar donde haya un delegado válido. Las expresiones lambda pueden ser funciones o subrutinas y tener una sola línea o varias líneas. Puede pasar valores del ámbito actual a una expresión lambda.[6]
[28] El *Common Language Runtime* o CLR (Lenguaje común en tiempo de ejecución) es el componente de máquina virtual de la plataforma .Net de Microsoft. [6]

- Tipos de datos soportados por el CLR.
- Colecciones en línea (colecciones almacenadas de manera serializada).
- Tipos anónimos.

Ejemplo:

```
//Proporciona funcionalidad para evaluar consultas con respecto a un origen de
//datos concreto en el que se especifica el tipo de los datos.
IQueryable<SER_NEGOCIACION> _negociaciones;

//Contexto de las entidades
using (var context = new TREASURYWORKSEntities())
            {
              //Creación de las consultas necesarias para obtener los datos
                var fechaEmpresa = context.SER_EMPRESAGENERAL.Where(m =>
m.CODEMPRESA == 1).Select(m => new { m.FECHAACTUAL }).Single().FECHAACTUAL;

                _negociaciones = from neg in context.SER_NEGOCIACIONES where
neg.FECHA == fechaEmpresa select neg;

                var operaciones = context.SER_NEGOCIACIONES
                                  .Where(neg => neg.FECHA == fechaEmpresa)
                                  .Select(neg => new
                                  {
                                      neg.CODNEGOCIACION,
                                      CLIENTE = neg.SER_CLIENTE.DESCRIPCION,
                                      MONEDA1 = neg.SER_MONEDA.SIGLA,
                                      MONEDA2 = neg.SER_MONEDA1.SIGLA,
                                      neg.FECHAPAGO,
                                      neg.VALORAPAGAR,
                                      neg.VALORARECIBIR,
                                      neg.VALORIMPUESTOS,
                                      neg.VALORCOMISIONES,
                                      neg.VALORNETO,
                                      OBSERVACION =
neg.SER_OPERACION.FirstOrDefault().OBSERVACIONES,
                                      OPERACION =
neg.SER_OPERACION.FirstOrDefault().SER_TIPOOPERACIONTRANSACCION.NOMBRE
                                  });
                //En este punto se materializa la consulta
                uwgNegociaciones.DataSource = operaciones.ToList();
                uwgNegociaciones.DataBind();
            }
```

CAPÍTULO 2. ANÁLISIS Y DISEÑO DE LA APLICACIÓN

2.1 REQUERIMIENTOS PARA EL USO Y CONSTRUCCIÓN DE LA APLICACIÓN Y CASO DE ESTUDIO.

El caso de estudio es una aplicación financiera que, en términos generales, sirve para realizar operaciones de compra y venta de divisas, estas operaciones se denominan spot[29] y se utiliza para esto un tipo de cambio denominado tipo de cambio spot. Las operaciones de compra y venta de divisas permitidas en la aplicación se realizan mediante una negociación spot donde se utilizan productos parametrizados, en el caso de esta aplicación se tiene el producto EUR/USD, este producto consta de dos monedas Euro y Dólar y sirve para realizar operaciones de compra y venta de Euros ya que la moneda local es el dólar y la moneda extranjera es el Euro.

Para realizar la negociación, es necesario seleccionar un producto que consta básicamente de un par de monedas que son la moneda del producto y la moneda de pago. Se debe también seleccionar el cliente, una estructura u oficina de pago en caso que se pague en otro lugar, el tipo de cambio spot[30] con la cual se realiza la operación y el tipo de cambio del día que viene de manera automática como tipo de cambio de referencia. Se deben calcular los impuestos, comisiones y el valor neto en caso de tener parametrizado.

La negociación tiene también una forma de pago, es decir, la manera en que se va a cancelar la operación, por ejemplo: efectivo, cheque, etc.

Cuando se guarda una negociación, se graban también una operación y una operación spot con el detalle del producto, también se tiene un registro del estado de la operación con la negociación original asociada y el usuario que realizó el cambio de estado de la negociación, los estados de las operaciones son: por

[29] Operación para intercambio de divisas en el momento, estas operaciones se completan máximo en dos días hábiles.
[30] Tipo de cambio o precio corriente de un producto spot.

confirmar, confirmada, y rechazada. El estado de las operaciones se modifica, luego de ingresadas, en el control de operaciones donde se pueden ver las operaciones realizadas en el día.

Por motivos contables se requieren almacenar una transacción y *voucher*[31] de cada negociación que se realice, es importante indicar que la parte contable no forma parte de la aplicación de este trabajo.

Una vez indicado de manera general el funcionamiento del caso de estudio, se requiere de una capa de persistencia para hacer que los objetos persistan, es decir, leer, grabar y eliminar objetos desde y hacia un tipo de almacenamiento persistente. La capa de persistencia debe tener las siguientes características:

- **Tipo de mecanismo de persistencia**, el mecanismo a usar en este caso es una base de datos relacional.

- **Encapsulación del mecanismo de persistencia**, lo que quiere decir que se deben enviar mensajes a los objetos para obtener, eliminar y grabar datos, la capa de persistencia se debe encargar de cómo hacerlo, solo en casos excepcionales se deben escribir sentencias directamente al mecanismo de persistencia, en este caso, a la base de datos.

- **Acciones de varios objetos**, la capa de persistencia debe soportar el obtener varios objetos de forma simultánea o eliminar varios objetos según un criterio específico.

- **Transaccionalidad,** una colección de acciones en varios objetos es una transacción, por ejemplo puede ser una combinación acciones como consultar, grabar o eliminar objetos. Las transacciones no pueden finalizar en un estado intermedio, si por alguna razón una de las acciones que pertenecen a la transacción falla, el sistema deshace (*rollback*) todas las acciones realizadas hasta ese punto como si la orden de la transacción

[31] Comprobante con datos de la negociación.

nunca se hubiera realizado, solo cuando se han ejecutado con éxito todas las acciones de la transacción, los cambios se hacen permanentes (*commit*).

- **Flexibilidad,** la capa de persistencia debe permitir agregar clases nuevas y actualizar de manera sencilla, así los programadores y administradores de base de datos pueden hacer su trabajo, por ejemplo: aumentar objetos y aumentar tablas respectivamente.

- **Identificadores de objetos,** son necesarios identificadores del objetos únicos que identifiquen de manera única a un objeto. Estos identificadores son equivalentes a las claves de las bases de datos que identifican de manera única a una fila de una tabla.

- **Consultas en lenguaje SQL**, El escribir código SQL en un código orientado a objetos contraviene los principios de encapsulación pues se está acoplando la aplicación directamente al esquema de base de datos. Las consultas SQL se realizan excepcionalmente por motivos de rendimiento y codificar las consultas en el código debe hacerse en casos justificados.

La capa de persistencia debe permitir a los desarrolladores concentrarse en su trabajo, es decir, desarrollar aplicaciones, sin preocuparse en cómo sus objetos serán almacenados.

Además, la capa de persistencia debe permitir a los administradores de bases de datos (*DBA's database administrators*) administrar las bases de datos sin preocuparse de introducir *bugs* o errores en la aplicación existente, la capa de persistencia permite a los administradores de bases de datos mover tablas, renombrar tablas, renombrar columnas y reorganizar tablas sin afectar las aplicaciones que acceden a esas tablas.

2.2 ANÁLISIS Y DISEÑO

Se analizan en detalle los requerimientos del sistema del caso de estudio y se describe el funcionamiento y cómo realizar el sistema, para esto se desarrollan los siguientes artefactos:

- **Especificación de Requerimientos.** Detalles de los requerimientos del sistema.
- **Modelo de Casos de Uso**, donde se presenta la funcionalidad del sistema y los actores.
- **Especificación de Casos de Uso**, solamente para los casos que requieran detalle.
- **Diagramas de Actividad**, indica los flujos de trabajo del sistema.
- **Glosario**, establece una terminología clara.
- **Diagramas de Clases,** describen los objetos que forman parte de la aplicación.
- **Modelo Físico**, donde se muestran las tablas del sistema.
- **Diccionario de Datos.**

2.2.1 ESPECIFICACIÓN DE REQUERIMIENTOS

Requerimiento TW1. Los usuarios deben ser autenticados antes de ingresar al sistema para parametrizar el sistema, realizar negociaciones o procesar operaciones.

Requerimiento TW2. Usuario autenticado como Trader[32] puede realizar una negociación de compra o venta de divisas. Para una negociación compra (operación por defecto), se debe seleccionar un cliente, una estructura de pago, un portafolio, el monto, seleccionar las formas de pago y calcular el total. Calcular los impuestos, comisiones y el valor neto en caso de tener parametrizado. Con

[32] En términos financieros, un Trader es alguien que compra y vende instrumentos financieros.

cada negociación de compra se generarán una operación de compra y una operación spot de compra, el estado de le negociación es “Por Confirmar”. También se graba un registro del estado de la operación con la negociación original asociada y el usuario que realizó el cambio de estado de la negociación. Por motivos contables se guardará una transacción y un voucher de la negociación de la compra. En caso de error se grabará un log.

Requerimiento TW3. Usuario autenticado como Trader puede realizar una negociación de compra o venta de divisas. Para una negociación venta es necesario seleccionar la operación venta, se debe seleccionar un cliente, una estructura de pago, un portafolio, el monto, seleccionar las formas de pago y calcular el total. Calcular los impuestos, comisiones y el valor neto en caso de tener parametrizado. Con cada negociación de venta se generarán una operación de venta y una operación spot de venta, el estado de le negociación es “Por Confirmar”. También se graba un registro del estado de la operación con la negociación original asociada y el usuario que realizó el cambio de estado de la negociación. Por motivos contables se guardará una transacción y un voucher de la negociación de la venta. En caso de error se grabará un log.

Requerimiento TW4. Usuario autenticado como Back Office[33] revisa todas las operaciones ingresadas y confirma o rechaza las operaciones, al confirmar la operación, ésta cambia de estado a “Confirmada” o “Rechazada”. También se graba un registro del estado de la operación con la negociación original asociada y el usuario que realizó el cambio de estado de la negociación. En caso de error se grabará un log.

Requerimiento TW5. Usuario autenticado como Administrador parametriza o modifica los datos de catálogos y datos para negociar en la aplicación. Los datos necesarios son los siguientes: País, Moneda, Usuario, Empresa, Cliente, Estructura, Portafolio, Tipo de Operación Transacción, Tipo de Forma de Pago, Forma de Pago por Moneda, Forma de Pago, Nombre de Tipo de Cambio por

[33] El back office en finanzas es quien procesa las operaciones ingresadas por el trader y puede cambiar el estado de la operación.

Tipo de Producto, Tipo de Cambio, Tipo de Producto, Comisión, Comisión por Operación, Impuesto, Impuesto por Operación, Producto.

2.2.2 MODELO DE CASOS DE USO

Los casos de uso indican qué es lo que el sistema del caso de estudio debe hacer. Cada caso de uso representa una pieza de funcionalidad del sistema, se presenta el caso de uso y su descripción correspondiente.

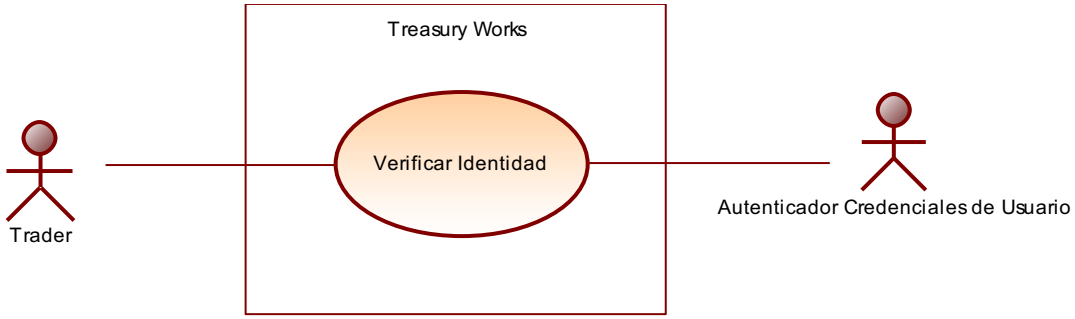


Figura 35. Caso de Uso: Verificar Identidad.

Nombre Caso de Uso	**Verificar Identidad**
Requerimientos relacionados	TW1
Objetivo	Autenticar al usuario en la aplicación.
Precondiciones	El usuario debe estar ingresado en la tabla de usuarios de la base de datos de la aplicación.
Fin de Condición Exitosa	Usuario es autenticado y puede ingresar a la aplicación.
Fin de Condición Fallida	Usuario es rechazado.
Actores Principales	Administrador, Trader, Back Office.
Actores Secundarios	Autenticador de credenciales de usuario.
Disparador	Usuario solicita ingreso a la aplicación mediante nombre de usuario y contraseña.
Casos de Uso Base	
Flujo Principal	1. Usuario ingresa nombre y contraseña en la pantalla de inicio del sistema. 2. Administrador de credenciales verifica datos del usuario en la base de datos en la tabla de usuario. 3. Usuario es autenticado en la aplicación.

Extensiones	2.1 Administrador de credenciales no verifica datos del usuario en la tabla de usuario. 2.2 Usuario es rechazado.

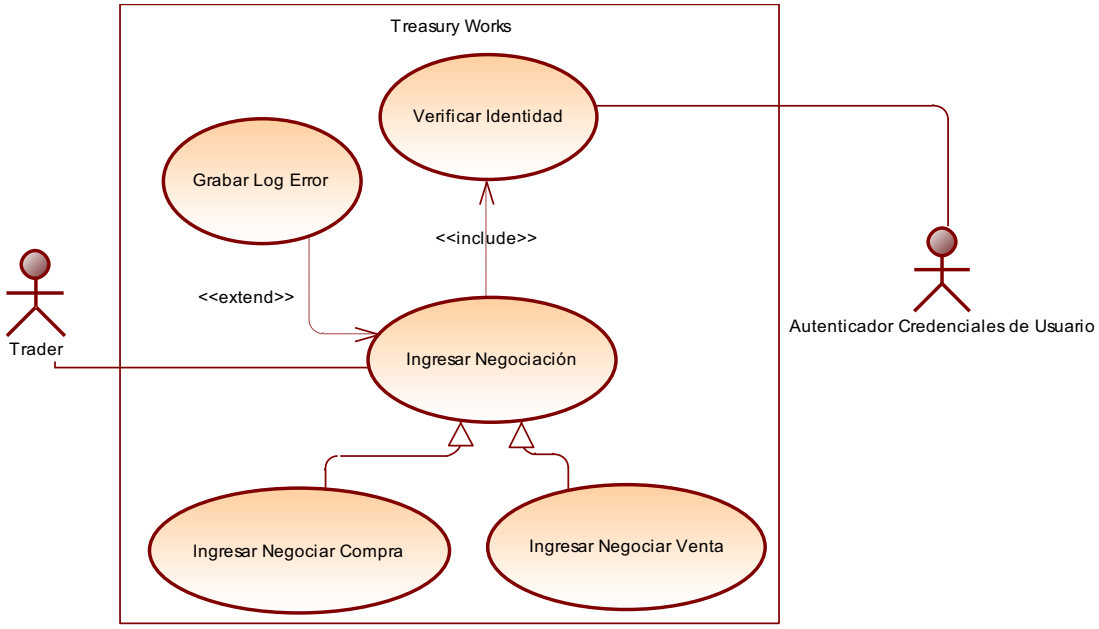


Figura 36. Caso de Uso: Ingresar Negociación

Nombre Caso de Uso	**Ingresar Negociación**
Requerimientos relacionados	TW2, TW3
Objetivo	Ingresar una negociación de un producto en el sistema.
Precondiciones	Usuario debe estar autenticado en el sistema como Trader.
Fin de Condición Exitosa	Negociación es creada correctamente.
Fin de Condición Fallida	Se rechaza la creación de una nueva negociación.
Actores Principales	Trader.
Actores Secundarios	Autenticador de credenciales de usuario.
Disparador	Trader solicita al sistema la creación de una nueva negociación presionando grabar.
Casos de Uso Base	
Flujo Principal	1. include::Verificar Identidad. Usuario se autentica

	en la aplicación. 2. El Trader ingresa a página de Negociación. 3. El Trader selecciona el tipo de operación. 4. El Trader proporciona los datos necesarios. 5. La negociación nueva es creada.
Extensiones	

Nombre Caso de Uso	**Ingresar Negociación Compra**
Requerimientos relacionados	TW2
Objetivo	Ingresar una nueva negociación compra de un producto en el sistema.
Precondiciones	Usuario debe estar autenticado en el sistema como Trader.
Fin de Condición Exitosa	Negociación compra es creada correctamente con estado “Por Confirmar”.
Fin de Condición Fallida	Se rechaza la creación de una nueva negociación compra.
Actores Principales	Trader.
Actores Secundarios	Autenticador de credenciales de usuario.
Disparador	Trader solicita al sistema la creación de una nueva negociación compra presionando grabar.
Casos de Uso Base	Ingresar Negociación
Flujo Principal	1. include::Verificar Identidad. Usuario se autentica en la aplicación. 2. El Trader ingresa a página de Negociación. 3. El Trader selecciona el tipo de operación compra (operación por defecto). 4. El Trader proporciona los datos necesarios: • Ingresar Cliente. • Seleccionar Estructura de Pago. • Seleccionar Portafolio. • Ingresar Monto de la compra. • Seleccionar Formas de Pago. 5. Aplicación calcula el total. 6. Aplicación calcula Impuestos y Comisiones (en caso de tener). 7. Aplicación calcula valor neto.

	8. Trader graba la negociación. 9. La negociación nueva es creada. 10. Se crean una operación y una operación spot. 11. Se almacena registro de operación nueva en la tabla de estados con el estado “Por Confirmar”. 12. Se genera una transacción y un voucher de la negociación compra ingresada.
Extensiones	8.1 Sistema rechaza la creación de una nueva negociación compra. 8.2 No se genera una nueva negociación Compra. 8.3 Se almacena un Log con el error o los errores.

Nombre Caso de Uso	**Ingresar Negociación Venta**
Requerimientos relacionados	TW3
Objetivo	Ingresar una nueva negociación venta de un producto en el sistema.
Precondiciones	Usuario debe estar autenticado en el sistema como Trader.
Fin de Condición Exitosa	Negociación venta es creada correctamente con estado “Por Confirmar”.
Fin de Condición Fallida	Se rechaza la creación de una nueva negociación venta.
Actores Principales	Trader.
Actores Secundarios	Autenticador de credenciales de usuario.
Disparador	Trader solicita al sistema la creación de una nueva negociación venta presionando grabar.
Casos de Uso Base	Ingresar Negociación
Flujo Principal	1. include::Verificar Identidad. Usuario se autentica en la aplicación. 2. El Trader ingresa a página de Negociación. 3. El Trader selecciona el tipo de operación venta. 4. El Trader proporciona los datos necesarios: • Ingresar Cliente. • Seleccionar Estructura de Pago. • Seleccionar Portafolio. • Ingresar Monto de la venta. • Seleccionar Formas de Pago.

	5. Aplicación calcula el total. 6. Aplicación calcula Impuestos y Comisiones (en caso de tener). 7. Aplicación calcula valor neto. 8. Trader graba la negociación. 9. La negociación nueva es creada. 10. Se crean una operación y una operación spot. 11. Se almacena registro de operación nueva en la tabla de estados con el estado “Por Confirmar”. 12. Se genera una transacción y un voucher de la negociación venta ingresada.
Extensiones	8.1 Sistema rechaza la creación de una nueva negociación venta. 8.2 No se genera una nueva negociación Venta. 8.3 Se almacena un Log con el error o los errores.

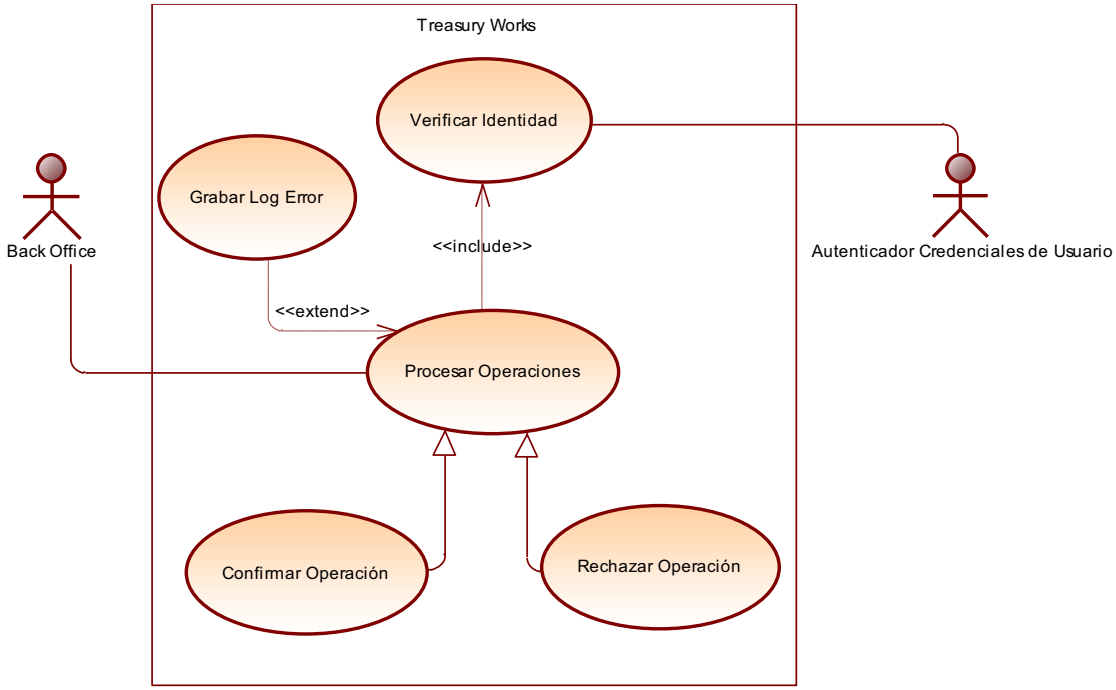


Figura 37. Casos de Uso: Confirmar Operaciones / Rechazar Operaciones

Nombre Caso de Uso	Confirmar Operaciones
Requerimientos relacionados	TW4
Objetivo	Confirmar operaciones.
Precondiciones	Usuario debe estar autenticado en el sistema como Back Office.
Fin de Condición Exitosa	Operación es confirmada correctamente.
Fin de Condición Fallida	No se confirma la operación.
Actores Principales	Back Office.
Actores Secundarios	Autenticador de credenciales de usuario.
Disparador	Back Office solicita al sistema la confirmación de operaciones presionando botón de confirmar.
Casos de Uso Base	
Flujo Principal	1. include::Verificar Identidad. Usuario se autentica en la aplicación. 2. El Back Office ingresa a página de Operaciones. 3. El Back Office confirma las operaciones. 4. Se cambia el estado de la Negociación y Operación asociada a “Confirmada”. 5. Se almacena registro con operación en la tabla de estados con el estado “Confirmada”.
Extensiones	3.1 Sistema rechaza la confirmación de operaciones. 3.2 Se rechaza el cambio de estado de las operaciones. 3.3 Almacena un Log con el error o los errores.

Nombre Caso de Uso	Rechazar Operaciones
Requerimientos relacionados	TW4
Objetivo	Rechazar operaciones.
Precondiciones	Usuario debe estar autenticado en el sistema como Back Office.
Fin de Condición Exitosa	Operación es rechazada correctamente.
Fin de Condición Fallida	No se rechaza la operación.
Actores Principales	Back Office.
Actores Secundarios	Autenticador de credenciales de usuario.
Disparador	Back Office solicita al sistema rechazar operaciones presionando botón de confirmar.

Casos de Uso Base	
Flujo Principal	1. include::Verificar Identidad. Usuario se autentica en la aplicación. 2. El Back Office ingresa a página de Operaciones. 3. El Back Office rechaza las operaciones. 4. Se cambia el estado de la Negociación y Operación asociada a “Rechazada”. 5. Se almacena registro con operación en la tabla de estados con el estado “Rechazada”.
Extensiones	3.1 Sistema no rechaza las operaciones. 3.2 Se deniega el cambio de estado de las operaciones. 3.3 Almacena un Log con el error o los errores.

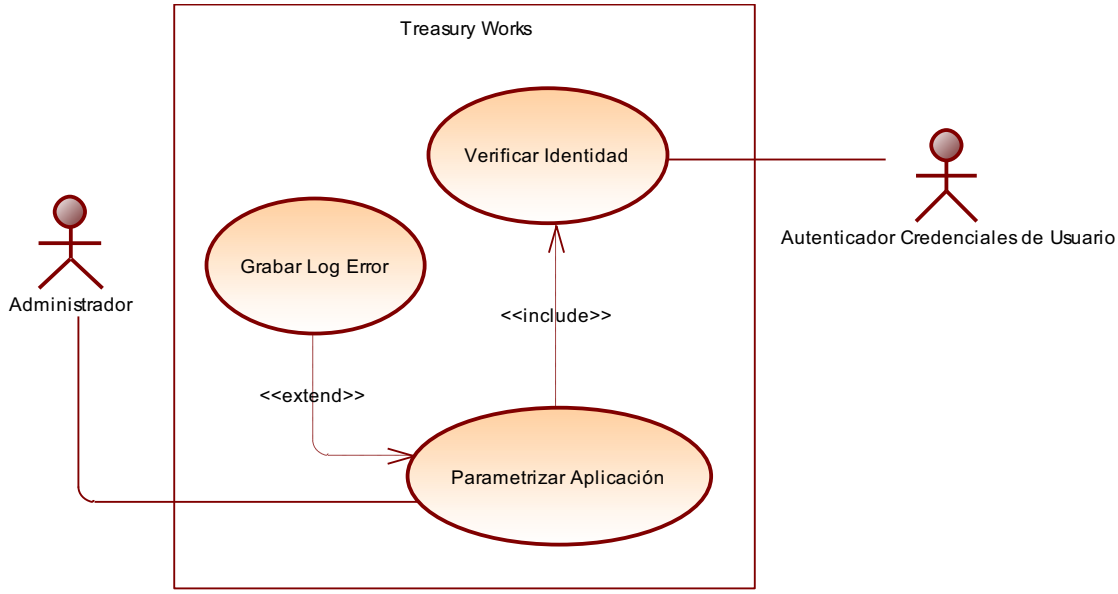


Figura 38. Caso de Uso: Parametrizar Aplicación

Nombre Caso de Uso	Parametrizar Aplicación
Requerimientos relacionados	TW5
Objetivo	Parametrizar el Sistema.
Precondiciones	Usuario debe estar autenticado en el sistema como Administrador.

Fin de Condición Exitosa	Catálogos y datos básicos son ingresados al sistema.
Fin de Condición Fallida	No se pueden grabar los datos.
Actores Principales	Administrador.
Actores Secundarios	Autenticador de credenciales de usuario.
Disparador	Administrador solicita al sistema grabar los datos ingresados en las pantallas mediante botón grabar.
Casos de Uso Base	
Flujo Principal	1. include::Verificar Identidad. Usuario se autentica en la aplicación. 2. Administrador ingresa a catálogos o datos básicos. 3. Administrador ingresa información básica: • Países. • Monedas. • Usuarios. • Empresas. • Clientes. • Estructuras. • Portafolios. • Tipos de Operación Transacción. • Tipos Forma Pago. • Formas Pago por Monedas. • Formas de Pago. • Nombres Tipo Cambio por Tipo de Producto. • Tipos de Cambio. • Tipos de Productos. • Comisiones. • Comisiones por Operaciones. • Impuestos. • Impuestos por Operaciones. • Productos. 4. Administrador graba los datos en las pantallas respectivas.
Extensiones	4.1 Sistema no graba la información. 4.2 Se deniega la creación/modificación de datos base. 4.3 Almacena un Log con el error o los errores.

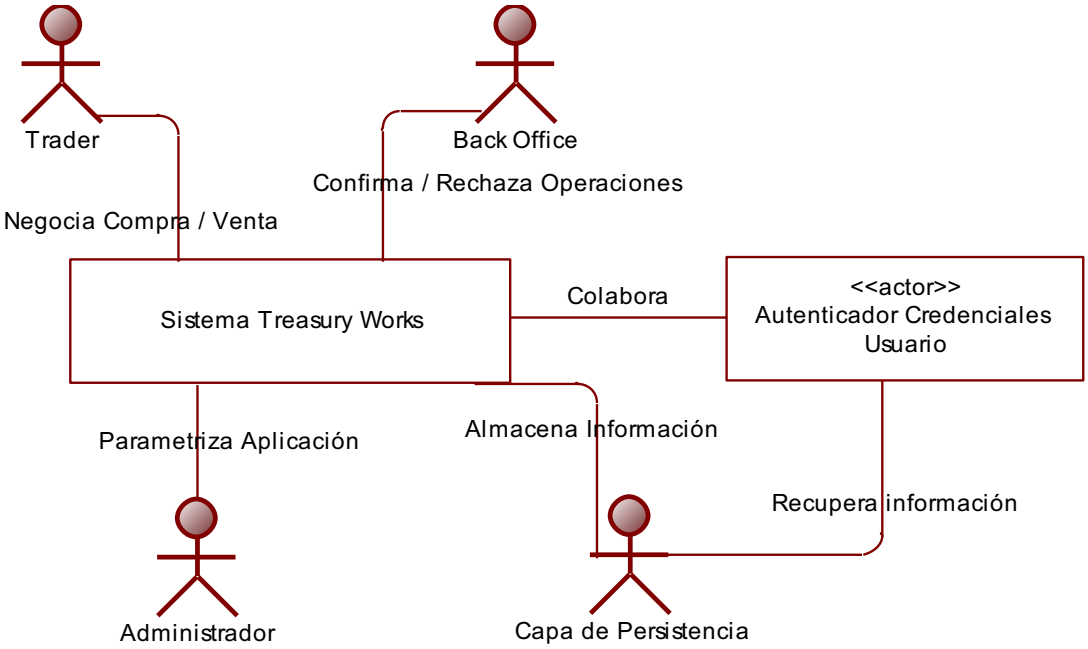


Figura 39. Caso de Uso: Visión General de la Aplicación

2.2.3 DIAGRAMAS DE ACTIVIDAD

Los diagramas de actividad representan los flujos de trabajo paso a paso tanto de negocio como operacionales del sistema del caso de estudio y permiten especificar cómo el sistema cumple sus objetivos.

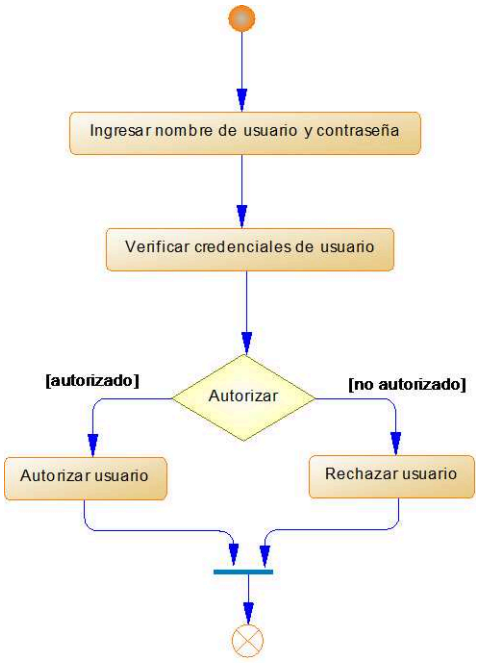


Figura 40. Diagrama de Actividad: Verificar Identidad.

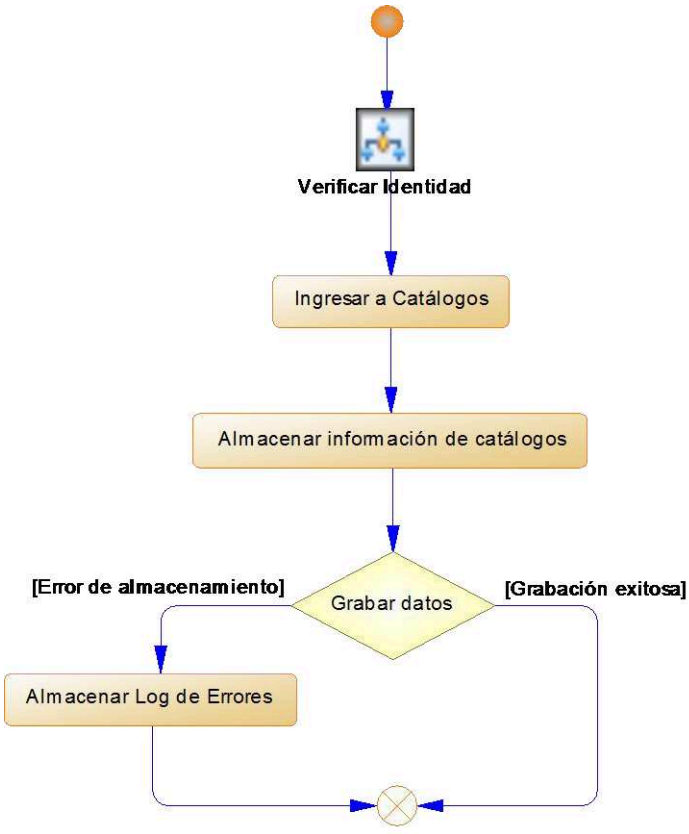


Figura 41. Diagrama de Actividad: Parametrizar Aplicación

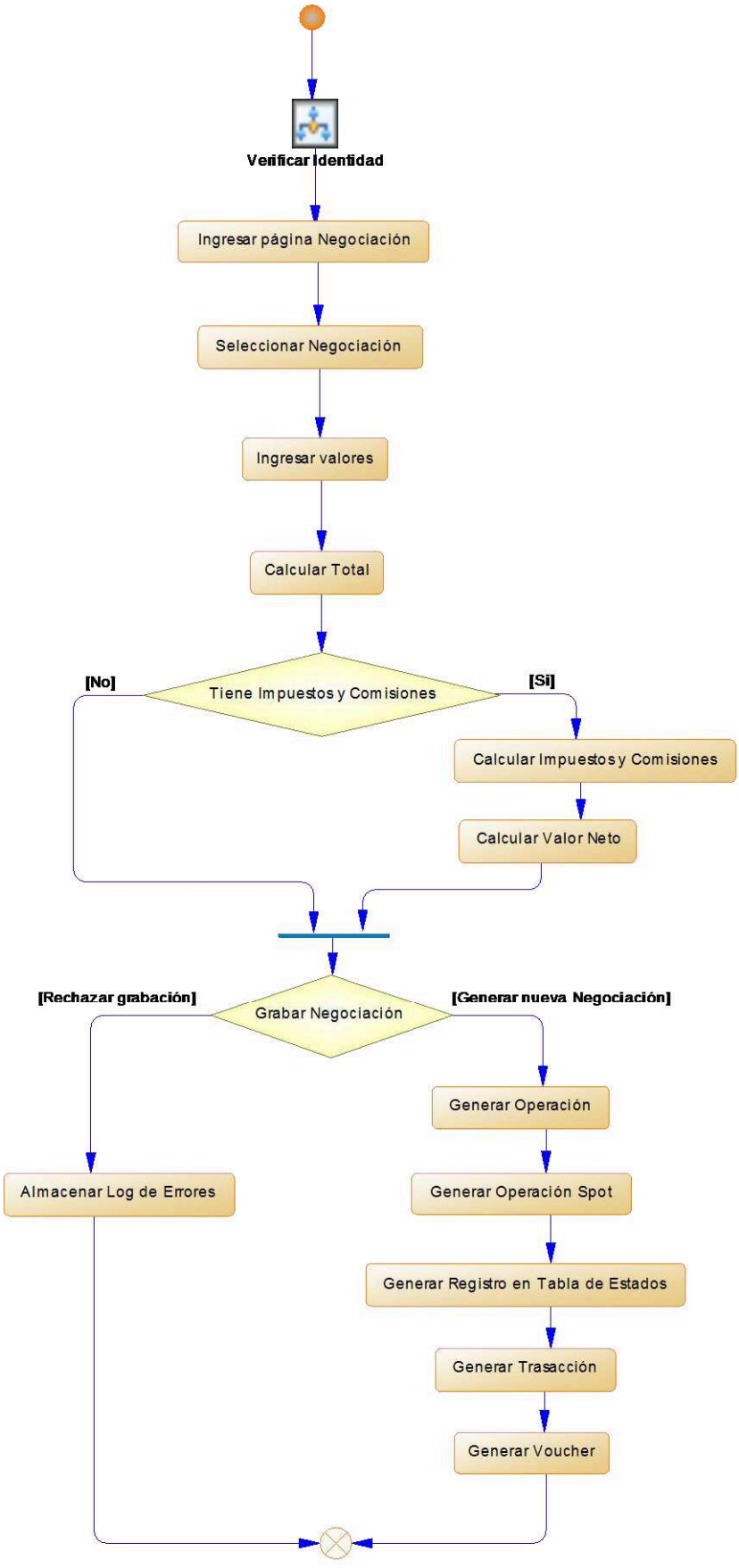


Figura 42. Diagrama de Actividad: Ingresar Negociación.

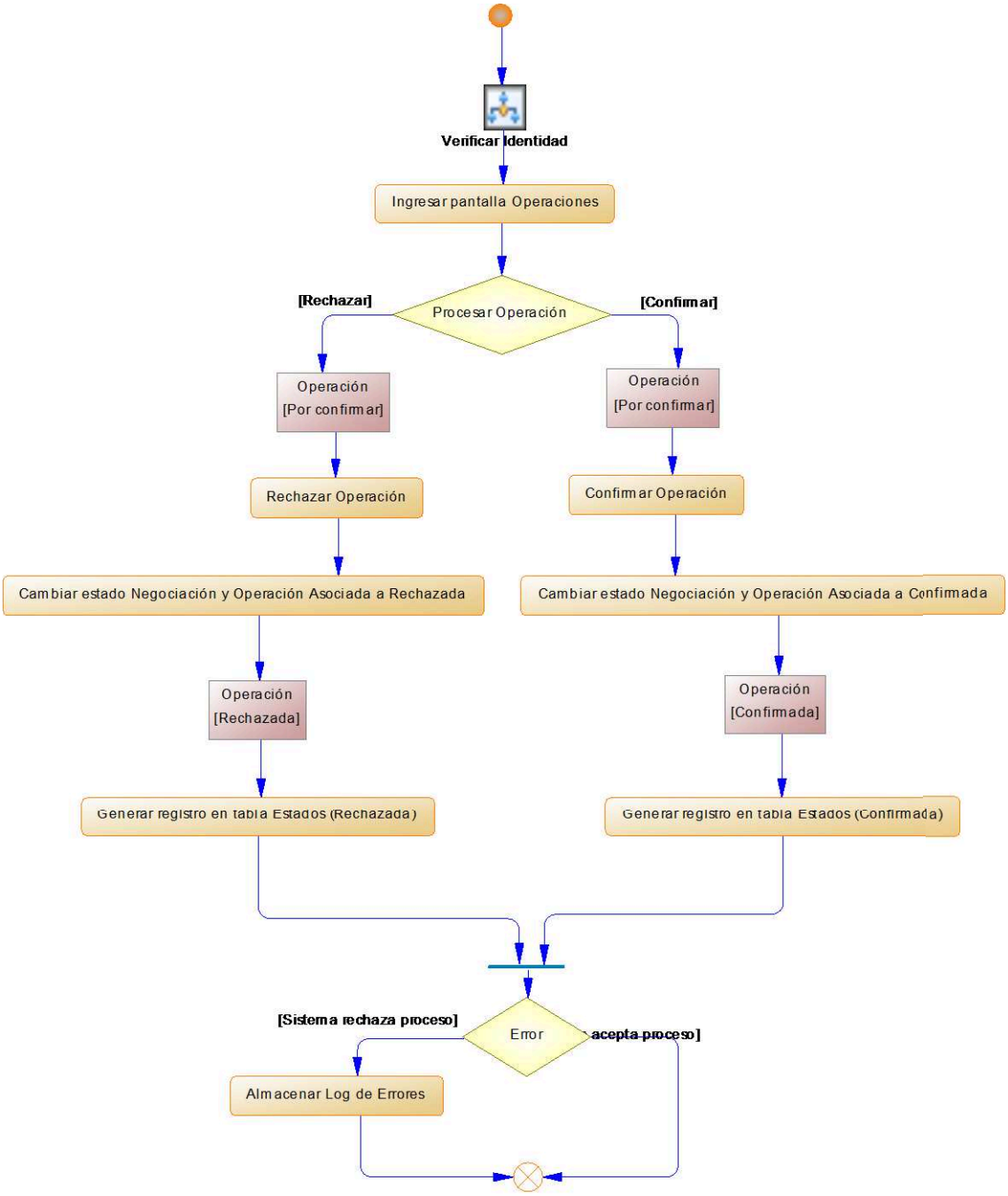


Diagrama 43. Diagrama de Actividad: Procesar Operación.

2.2.4 GLOSARIO DE TÉRMINOS UTILIZADOS EN EL SISTEMA.

Negociación Spot. Operación para intercambio de divisas en el momento, estas operaciones se completan máximo en dos días hábiles.

Tipo de Cambio Spot. Tipo de cambio o precio corriente de un producto spot.

Forma de Pago. Modalidad en la que se va a cancelar una operación.

Estado de la Operación. Indica la situación de la operación, en este caso se tienen los estados: Por Confirmar, que indica que la operación aún no se verifica; Confirmada, que indica que la operación fue revisada y aprobada; Rechazada, indica que la operación tiene algún problema y no puede ser Confirmada.

Voucher. Comprobante con datos de la negociación, este registro es útil para contabilidad.

Trader. En términos financieros, un Trader es alguien que compra y vende instrumentos financieros.

Estructura de Pago. Oficina o agencia donde se realiza el pago.

Portafolio. Es un conjunto de inversiones de una persona, empresa o fondo.

Valor Neto. Valor luego del cálculo de impuestos y comisiones.

Back Office. El back office en finanzas es quien procesa las operaciones ingresadas por el Trader y puede cambiar el estado de la operación.

Confirmar Operación. Indica que la operación ha sido revisada y aprobada por el Back Office.

Rechazar Operación. Indica que la operación ha sido revisada pero rechazada por el Back Office, debido a algún inconveniente.

2.2.5 DIAGRAMAS DE CLASES

Estos diagramas representan los objetos involucrados en el caso de estudio. Las figuras: Figura 44, Figura 45, Figura 46 y Figura 47 representan el diagrama de clases.

2.2.6 MODELO FÍSICO DE DATOS

El diagrama físico muestra las tablas involucradas en el caso de estudio. Las figuras: Figura 48, Figura 49, Figura 50 y Figura 51 muestran el diagrama físico de datos.

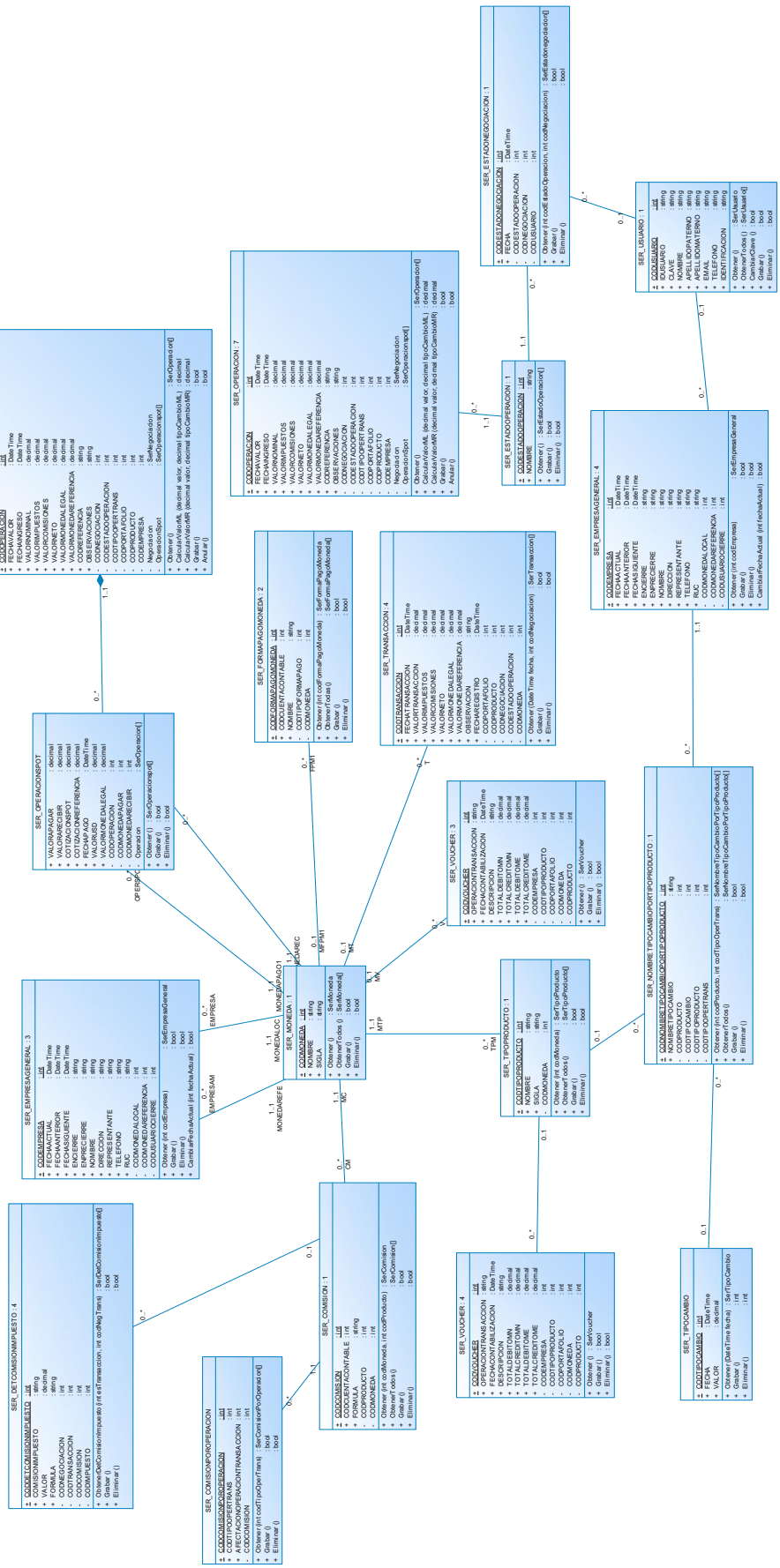


Figura 44. Diagrama de clases del caso de estudio.

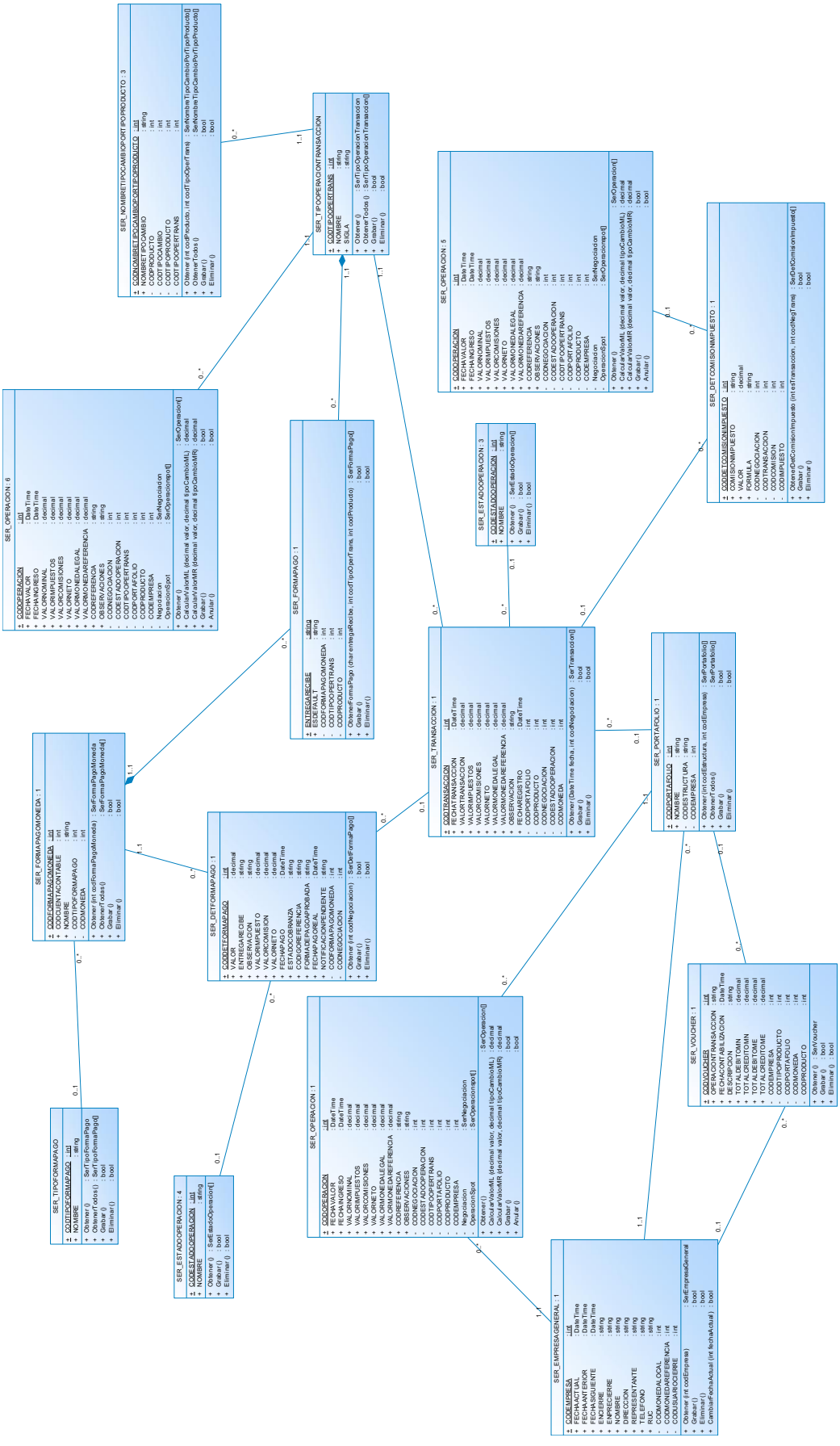


Figura 45. Diagrama de clases del caso de estudio (continuación).

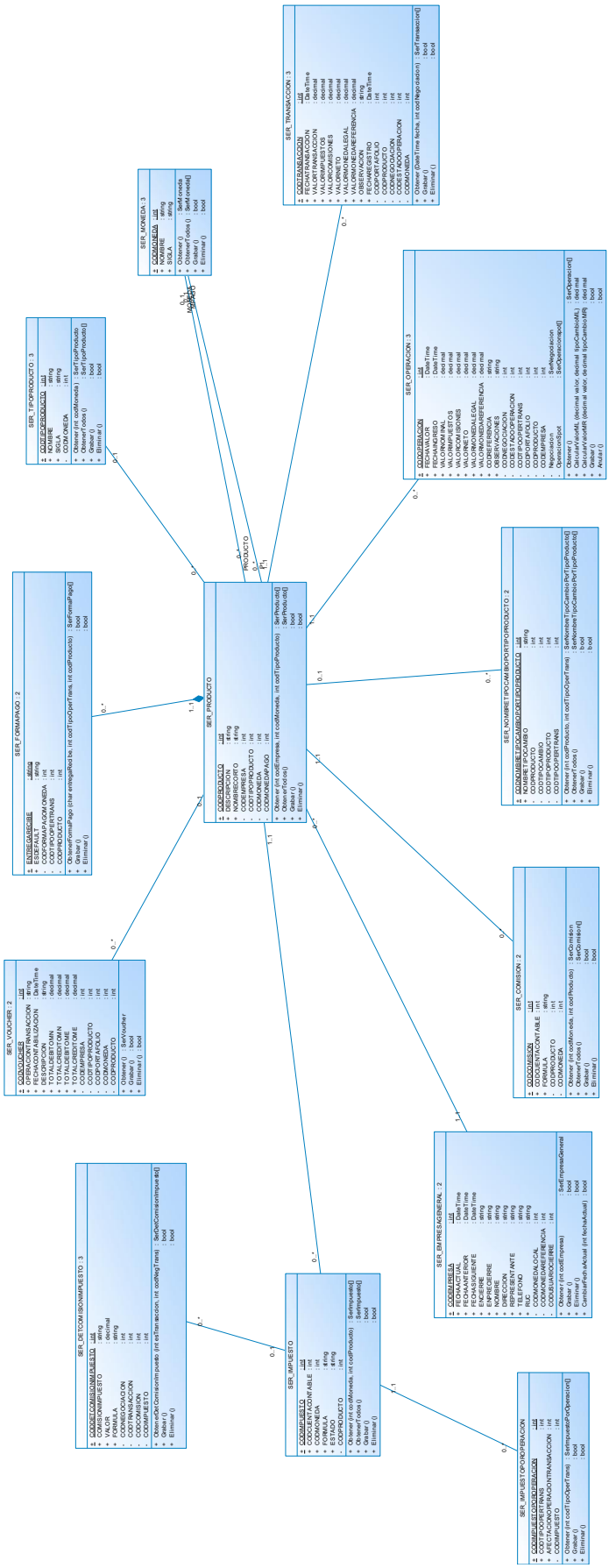


Figura 46. Diagrama de clases del caso de estudio (continuación).

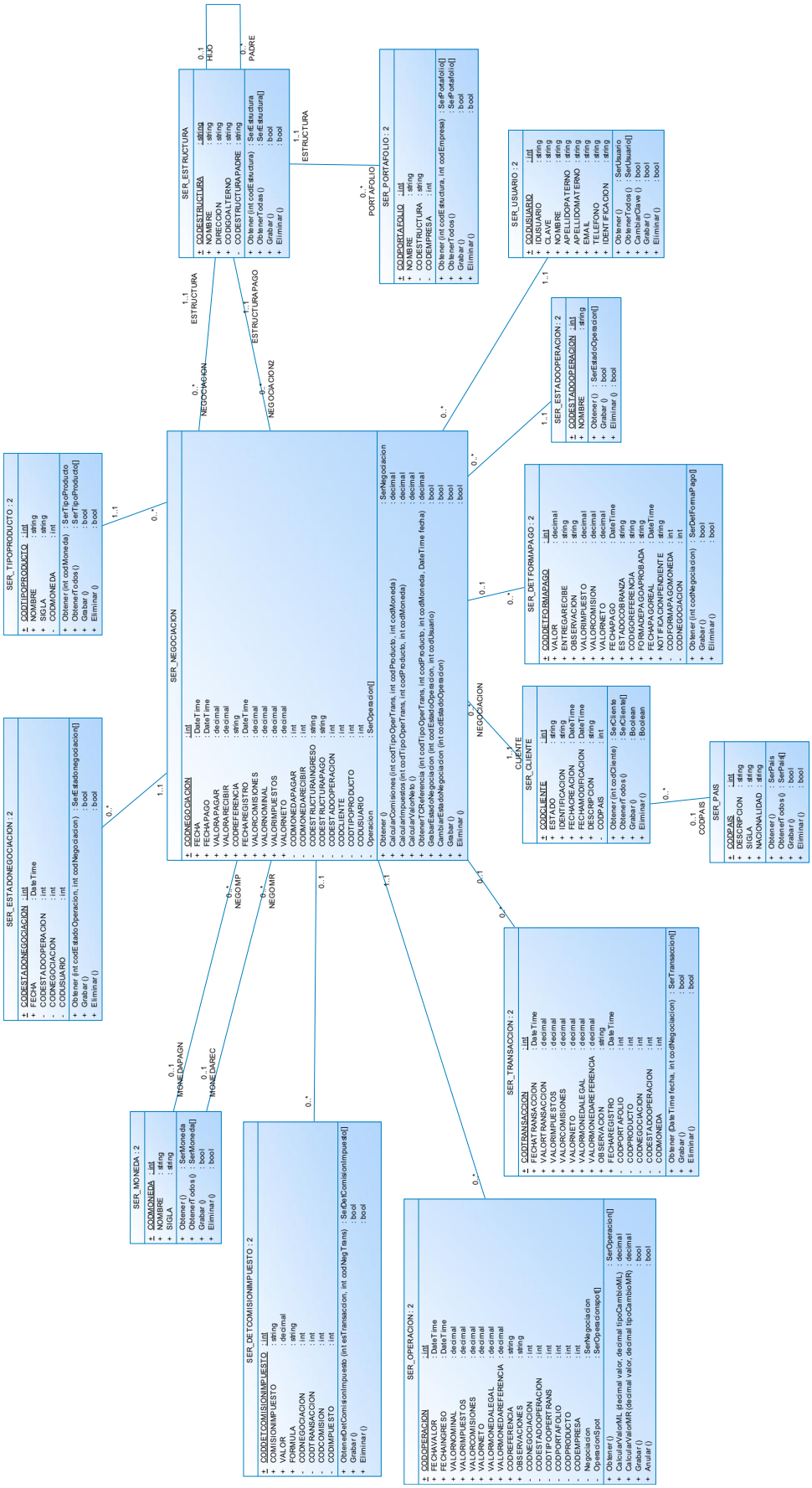


Figura 47. Diagrama de clases del caso de estudio (continuación).

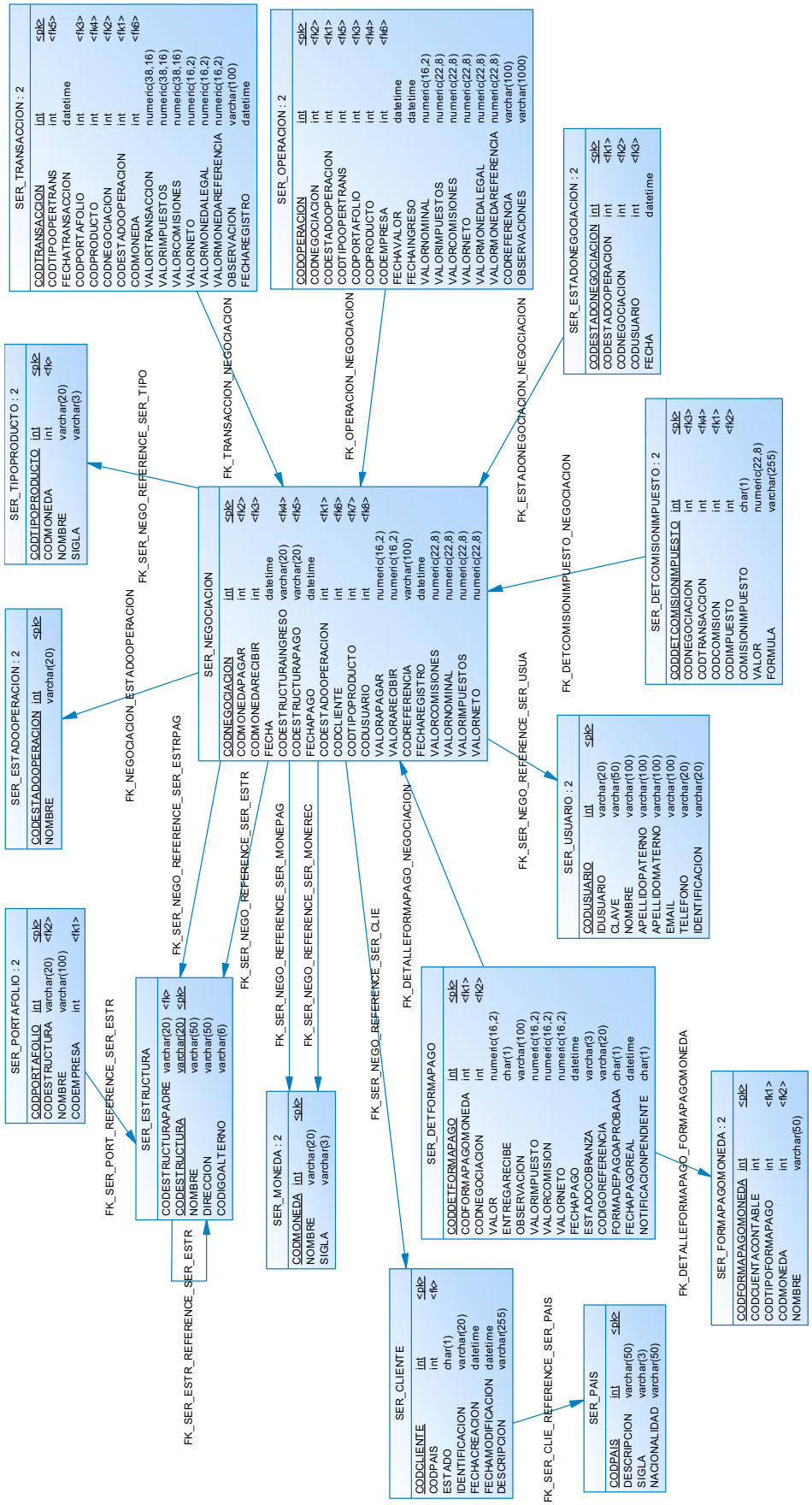


Figura 48. Diagrama físico de datos.

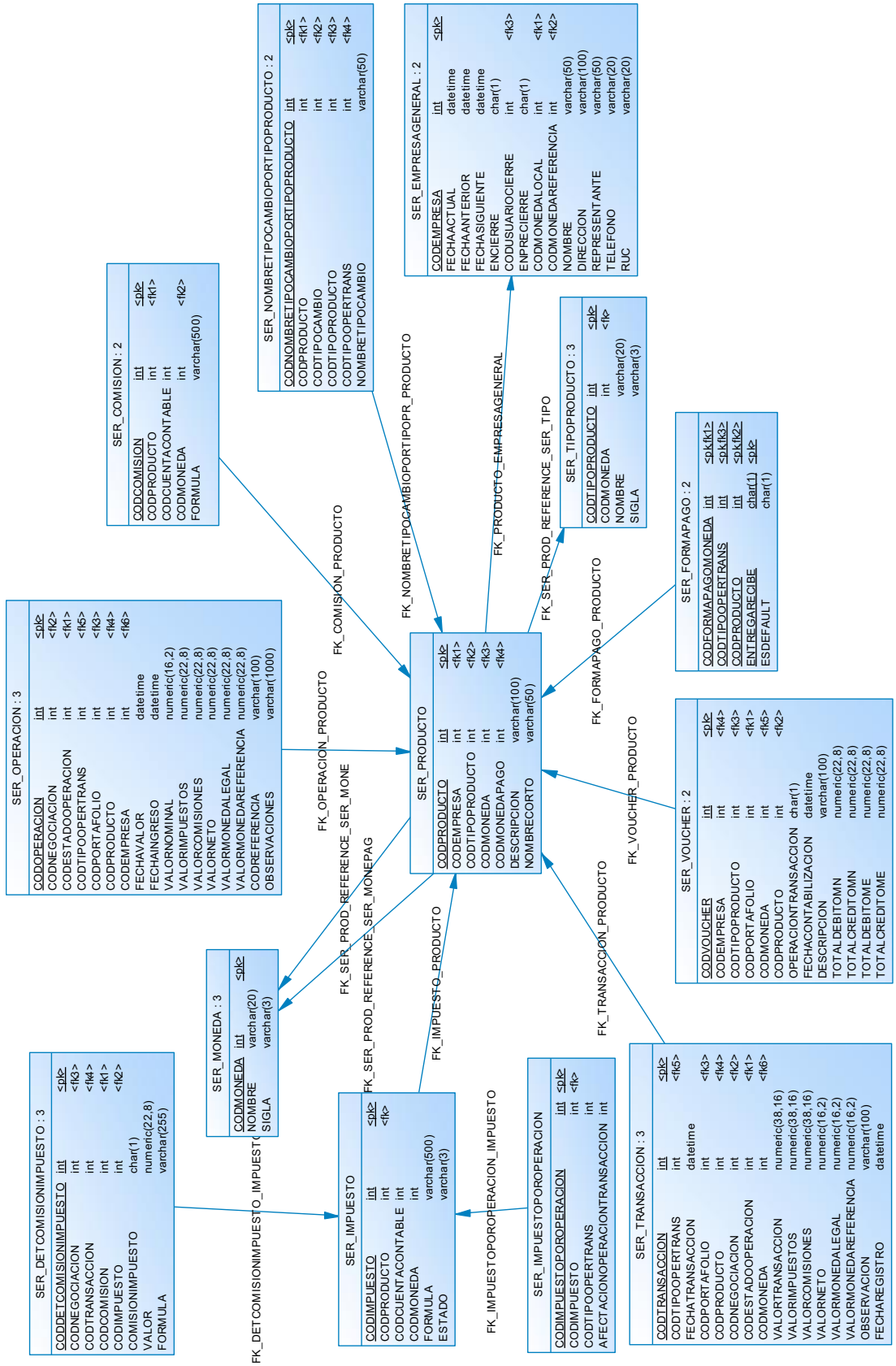


Figura 49. Diagrama físico de datos (continuación).

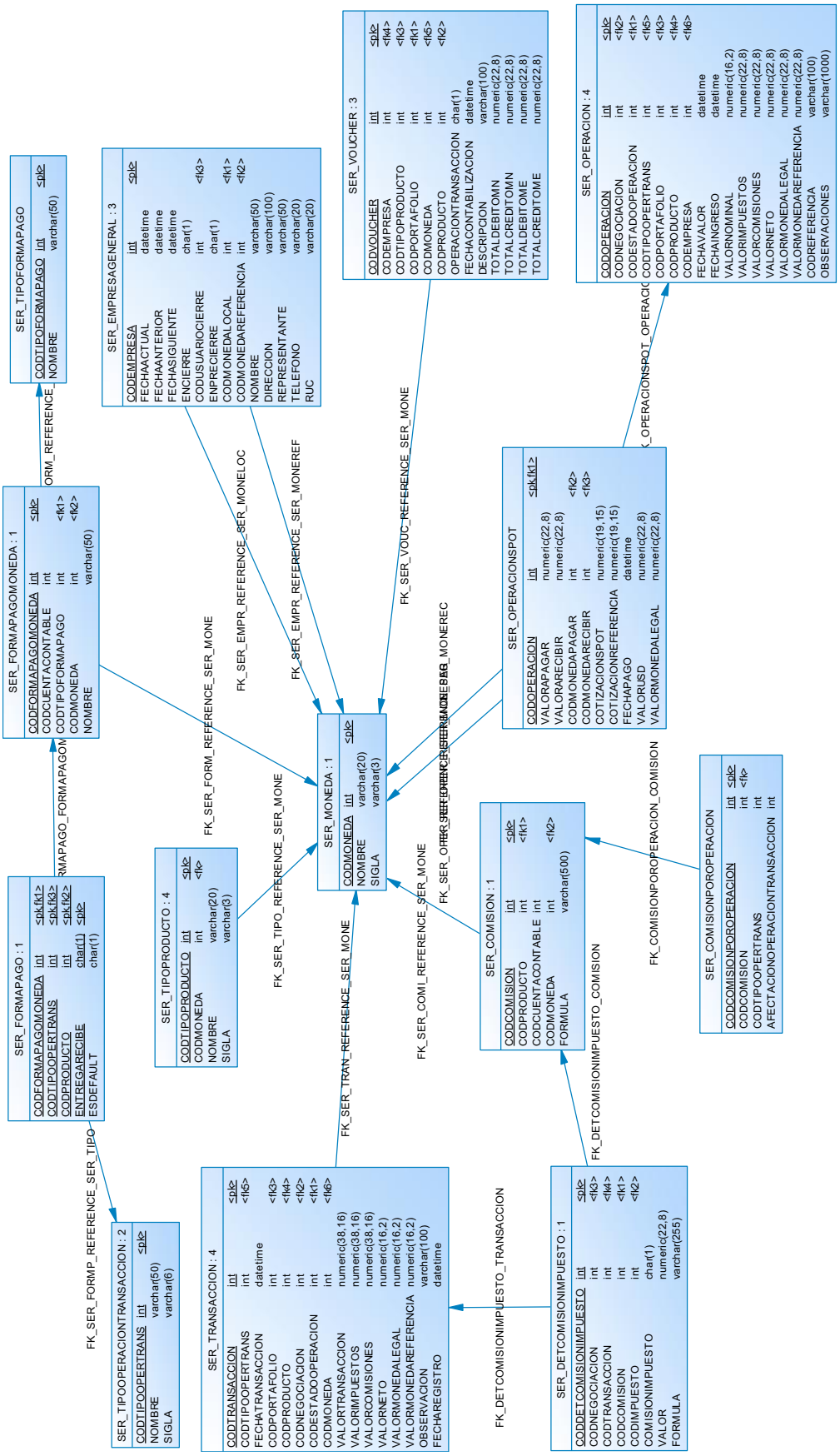


Figura 50. Diagrama físico de datos (continuación).

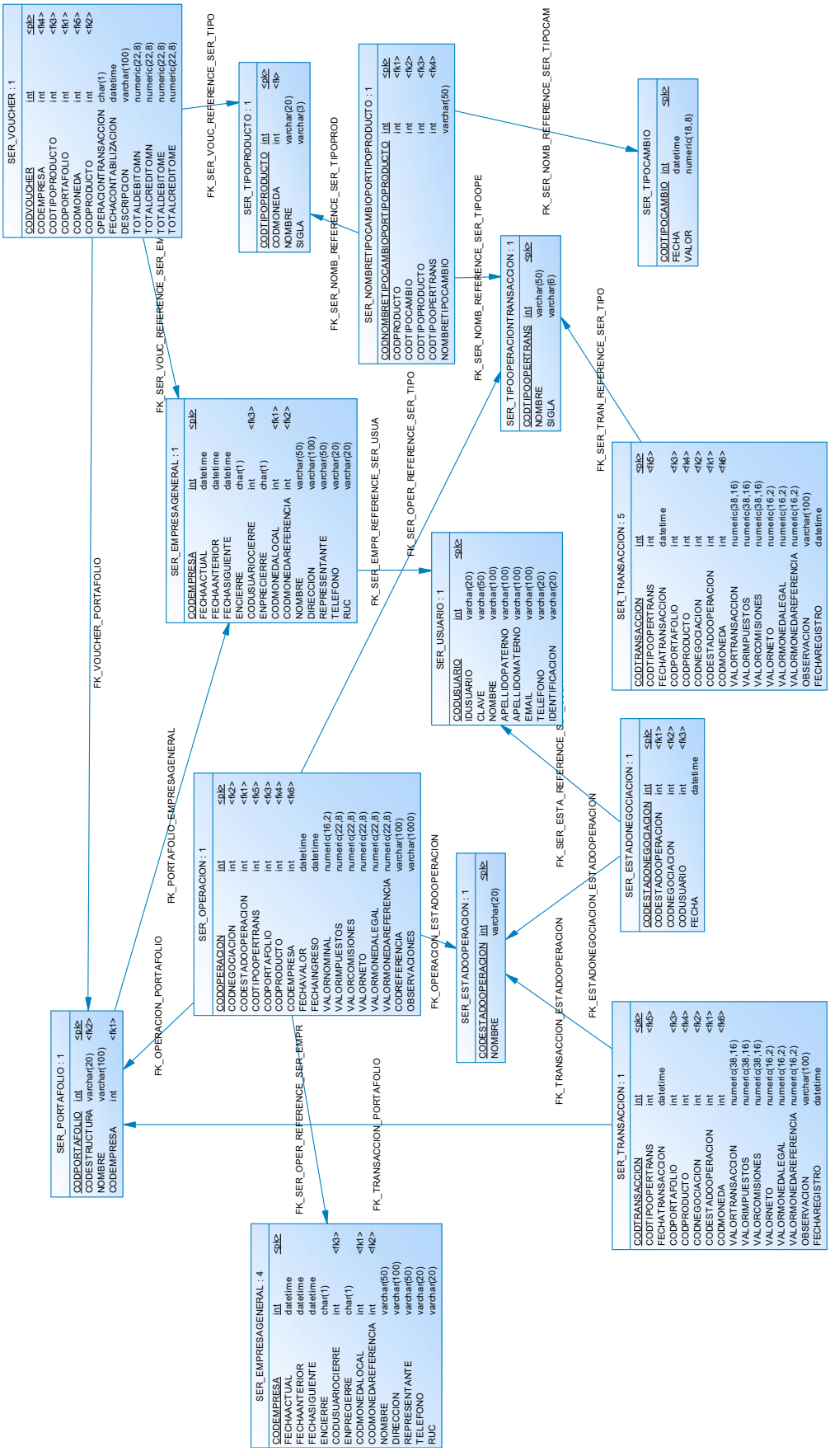


Figura 51. Diagrama físico de datos (continuación).

2.2.7 DICCIONARIO DE DATOS

El diccionario de datos muestra los metadatos que contienen las características lógicas y puntuales de los datos que se van a utilizar en el caso de estudio. A continuación se presenta una muestra del diccionario de datos para la tabla SER_CLIENTE con atributos y columnas. El contenido completo del diccionario de datos se presenta en el Anexo 1.

Tabla SER_CLIENTE

Name	SER_CLIENTE
Comment	Datos de los clientes con los cuales se negocia.

Referencias

Name	Code	Child Table	Foreign Key Columns	Parent Role	Child Role
Reference_63	REFERENCE_63	SER_NEGOCIACION	CODCLIENTE		

Columnas

Name	Comment	Domain
CODCLIENTE	Código único de la tabla.	IDREGLARGO
CODPAIS	Código del país de origen.	IDREGCORTO
ESTADO	Estado del cliente. A->Activo, I->Inactivo	ESTADO
IDENTIFICACION	Identificación del cliente.	NOMBRECORTO
FECHACREACION	Fecha de creación del registro.	FECHAHORA
FECHAMODIFICACION	Fecha de modificación del registro.	FECHAHORA
DESCRIPCION	Descripción del cliente.	DESCRIPCIONCORTA

CAPÍTULO 3. IMPLEMENTACIÓN Y PRUEBA DE LA CAPA DE PERSISTENCIA

3.1 IMPLEMENTACIÓN Y PRUEBAS

Se presenta como ejemplo la clase de la entidad que corresponde a la tabla SER_CLIENTE. El código completo se encuentra en el Anexo 2. El código presentado corresponde al lenguaje C# 4.0.

Es importante indicar que los comentarios marcados con "///" son comentarios de documentación XML. Las etiquetas XML contenidas en estos comentarios sirven para crear documentación para el código.

```
/// <summary>
/// Create a new SER_CLIENTE object.
/// </summary>
/// <param name="cODCLIENTE">Initial value of the CODCLIENTE property.</param>
/// <param name="iDENTIFICACION">Initial value of the IDENTIFICACION
property.</param>
/// <param name="fECHACREACION">Initial value of the FECHACREACION property.</param>
/// <param name="dESCRIPCION">Initial value of the DESCRIPCION property.</param>
public static SER_CLIENTE CreateSER_CLIENTE(global::System.Int32 cODCLIENTE,
global::System.String iDENTIFICACION, global::System.DateTime fECHACREACION,
global::System.String dESCRIPCION)
        {
            SER_CLIENTE sER_CLIENTE = new SER_CLIENTE();
            sER_CLIENTE.CODCLIENTE = cODCLIENTE;
            sER_CLIENTE.IDENTIFICACION = iDENTIFICACION;
            sER_CLIENTE.FECHACREACION = fECHACREACION;
            sER_CLIENTE.DESCRIPCION = dESCRIPCION;
            return sER_CLIENTE;
        }
```

Ejemplo de código de relación con la entidad que representa a la tabla SER_NEGOCIACION:

```
/// <summary>
        /// No Metadata Documentation available.
        /// </summary>
        [XmlIgnoreAttribute()]
        [SoapIgnoreAttribute()]
        [DataMemberAttribute()]
        [EdmRelationshipNavigationPropertyAttribute("TREASURYWORKSModel",
"FK_SER_NEGO_REFERENCE_SER_CLIE", "SER_CLIENTE")]
        public SER_CLIENTE SER_CLIENTE
        {
            get
            {
                return
((IEntityWithRelationships)this).RelationshipManager.GetRelatedReference<SER_CLIENTE
>("TREASURYWORKSModel.FK_SER_NEGO_REFERENCE_SER_CLIE", "SER_CLIENTE").Value;
            }
            set
            {

((IEntityWithRelationships)this).RelationshipManager.GetRelatedReference<SER_CLIENTE
>("TREASURYWORKSModel.FK_SER_NEGO_REFERENCE_SER_CLIE", "SER_CLIENTE").Value = value;
            }
        }
        /// <summary>
        /// No Metadata Documentation available.
        /// </summary>
        [BrowsableAttribute(false)]
        [DataMemberAttribute()]
        public EntityReference<SER_CLIENTE> SER_CLIENTEReference
        {
            get
            {
                return
((IEntityWithRelationships)this).RelationshipManager.GetRelatedReference<SER_CLIENTE
>("TREASURYWORKSModel.FK_SER_NEGO_REFERENCE_SER_CLIE", "SER_CLIENTE");
            }
            set
            {
                if ((value != null))
                {
```

```
((IEntityWithRelationships)this).RelationshipManager.InitializeRelatedReference<SER_
CLIENTE>("TREASURYWORKSModel.FK_SER_NEGO_REFERENCE_SER_CLIE", "SER_CLIENTE", value);
                }
            }
        }
```

3.2 PRUEBAS SOBRE UNA BASE DE DATOS RELACIONAL

Las pruebas realizadas sobre la base de datos relacional corresponden a las operaciones CRUD, que significa que se realizaron las operaciones fundamentales sobre los datos como son: Crear, Obtener, Actualizar y Eliminar. Los casos de prueba completos se presentan en el Anexo 3.

Ejemplo de caso de prueba de grabación de una negociación:

	TOTAL CASOS						CONDICIONES DE PRUEBA					
N°	4	76	11	PROCESOS	1	Estado	2	Estado	3	Estado	4	Estado
	TOTAL CASOS DE PRUEBA/ PROCESO	OK	NOK	CAMBIOS								
1	4	4	0	NEG - NEGOCIACION SPOT	Negociación Spot : Compra en moneda extranjera. Verificar que la operación se grabe correctamente, se desplegará un mensaje indicando que la operación se ha almacenado correctamente.	ok	Negociación Spot: Venta en moneda extranjera. Verificar que la operación se grabe correctamente, se desplegará un mensaje indicando que la operación se ha almacenado correctamente.	ok	Negociación Spot: Verificar cálculo de impuestos y comisiones.	ok	Negociación Spot: Comprobar actualización de tablas asociadas a la negociación.	ok

CAPÍTULO 4. CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

Se concluye que una capa de persistencia permite la abstracción del sistema de almacenamiento de datos simplificando el código y transparentando el acceso al mecanismo de almacenamiento de datos.

Se puede concluir que una capa de persistencia mantiene el código de acceso al mecanismo de almacenamiento de datos evitando la práctica común de embeber código SQL dentro de las clases de negocio.

Una capa de persistencia permite tener roles definidos, es decir, los programadores resuelven problemas de negocio programando en lenguajes de alto nivel como C# sin importar la tecnología de almacenamiento de objetos utilizada, asimismo la capa de persistencia se encarga de realizar operaciones CRUD en los datos, sin importar reglas de negocio, interfaces, comunicaciones y otras cosas que no tengan que ver con almacenamiento persistente de datos.

Se concluye también que el uso de una capa de persistencia hace posible que aplicaciones orientadas a objetos utilicen bases de datos relacionales como mecanismo de persistencia sin tener que acoplar el esquema de objetos al esquema de datos.

Se puede concluir que las características de la tecnología de Microsoft .NET y C# tienen potencialidades que permiten la implementación de capas de persistencia de objetos de manera ágil y transparente, pudiendo acceder al mecanismo de persistencia mediante una aplicación Web en una red LAN.

RECOMENDACIONES

Se recomienda el desarrollo de una capa de persistencia para proyectos grandes o medianos para poder medir los beneficios en el desarrollo de aplicaciones a mediano plazo.

En proyectos pequeños se recomienda el uso de procedimientos almacenados, capas de acceso a datos y acceso a datos nativos de cada lenguaje como ADO de .NET.

Se recomienda el estudio de capas de abstracción de acceso a bases de datos disponibles bajo licenciamiento libre.

Se recomienda el diseño y construcción de una capa de persistencia para proyectos nuevos en etapa de diseño y no para proyectos en producción, pues el cambiar a un esquema de capa de persistencia implica prácticamente el rehacer toda la aplicación.

Se recomienda diferenciar las arquitecturas "*Multi Layer*" y "*Multi Tier o N-Tier*", que se puede traducir como "Multi Capa" y "Multi Nivel" respectivamente. La primera se refiere a una separación lógica de funcionalidad y la segunda se refiere a una separación física cuando se distribuye e implementa la aplicación. Esto es porque en una arquitectura "Multi Tier" se tiene que considerar Servicios para consumir y exponer datos, servicios u objetos entre los niveles.

REFERENCIAS BIBLIOGRÁFICAS

1. Ambler, Scott, *Agile Database Techniques: Effective Strategies for the Agile Software Developer.* Wiley Publishing Inc., USA, 2003.

2. Ambler, Scott W., *A Manager's Introduction to The Relational Unified Process (RUP)*. Ambysoft, USA, 2005.

3. Bartels, Dirk.; Benson, Robert., *Object Persistence and Agile Software Development.* Versant Corporation. White Paper.

4. Comer, Douglas, *Internetworking with TCP/IP Vol. I: Principles, Protocols, and Architecture (4th Edition).* Prentice Hall, USA, 2000.

5. Duhl, Joshua; Barry, Douglas K., *Object Storage Fact Book: Object Relational Mapping.* Barry & Associates, 2001.

6. Edwards, James; Bramante, Richard, *Networking Self – Teaching Guide.* Wiley Publishing, Inc., USA, 2009.

7. Esposito, Dino; Saltarello Andrea, *Architecting Microsoft® .NET Solutions for the Enterprise*. Microsoft Press, USA, 2008.

8. Hallberg, Bruce, *Networking a Beginners Guide Second Edition*. Osborne / McGraw Hill, USA, 2001.

9. Kauffman, John.; Ferracchiati, Claudio Fabio., Otros, *Beginning ASP.NET Databases Using VB.NET (Peperback).* Wrox, USA. 2003.

10. Keller, Wolfgang, *Mapping Objects to Tables A Pattern Language Whitepaper.* USA, 2004.

11. Keller, Wolfgang, *Object/Relational Access Layers A Roadmap, Missing Links and More Patterns Whitepaper.* USA, 2004.

12. Letelier, Patricio. *Introducción a RUP.* Universidad Politécnica de Valencia, Departamento de Sistemas Informáticos y Computación, España, 2004.

13. Mehta, Vijay P., Pro *LINQ Object Relational Mapping with C# 2008.* Apress, USA, 2008.

14. Microsof, *MSDN Library for Visual Studio 2008.* Microsoft Corporation, USA, 2008.

15. Nagel, Christian; Evjen, Bill; Glynn, Jay; Watson, Karli; Skinner, Morgan. *Professional C# 4 and .NET 4.* Wiley Publishing, Inc., USA, 2010.

16. Nielsen, Paul. *Microsoft SQL Server 2008 Bible.* Wiley Publishing Inc., USA, 2009.

17. RUMBAUGH, James., JACOBSON Ivar., BOOCH., Grady. *Unified Modeling Language User Guide 2nd Edition.* Adisson-Wesley, USA, 2005.

18. Tanenbaum Andrew S., *Computer Networks, Fourth Edition.* Prentice Hall. USA, 2003.

19. Troelsen, Andrew. *Pro C# 2010 and the .NET 4 Platform (5th Edition).* Appress, USA, 2010.

20. Vieira, Robert., *Microsoft SQL Server 2008 Programming*. Wiley Publishing, USA, 2009.

21. Yoder, Joseph; Johnson, Ralph; Wilson, Quince, *Connecting Business Objects to Relational Databases Whitepaper.*USA, 2000.

22. Ambler, Scott W., *Encapsulating Database Access: An Agile "Best" Practice.* http://www.agiledata.org/essays/implementationStrategies.html 2005.

23. Ambler, Scott W., *Mapping Objects to Relational Databases: O/R Mapping In Detail* http://www.agiledata.org/essays/mappingObjects.html 2005.

24. Ambler, Scott, *The Design of a Robust Persistence Layer For Relational Databases* http://www.ambysoft.com/essays/persistenceLayer.html 2005.

25. Ambler, Scott, *The Object-Relational Impedance Mismatch.* http://www.agiledata.org/essays/impedanceMismatch.html 2006.

26. Bostrup, Tore, *Introduction to Relational Databases - Part 1: Theoretical Foundation.* http://www.15seconds.com/issue/020522.htm 2002.

27. Chapple, Mike. (HTML). *SQL Fundamentals. About.com: Databases*. About.com. http://databases.about.com/od/sql/a/sqlfundamentals.htm?terms=SQL. 2007.

28. Drake, Joshua, *Understand SQL*. http://www.faqs.org/docs/ppbook/c1164.htm. 2002.

29. IBM, *History of IBM, 1978 (HTML). IBM Archives*. http://www-03.ibm.com/ibm/history/history/year_1978.html. 2007.

30. IBM, *Structured Query Language (SQL)*. http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/c0004100.htm. 2006.