Universidad de las Ciencias
Informáticas

Facultad 2

Trabajo de Diploma para optar por el título de
Ingenieros en Ciencias Informáticas

TeleIdentificador Personal.
Desarrollo de la Plataforma Manejadora de Peticiones.
Capa de Acceso a Datos y Capa de Negocio.

Autores: Nestor Reina Marichal

Ronnier Llombart Hardy

Tutores: Ing. Madelis Pérez Gil

Ing. Yunisel Viera Vargas

Ciudad de La Habana, junio de 2009
"Año del 50 aniversario del triunfo de la Revolución"

Cada día en la vida de los hombres,
es una página imborrable de la historia.

José Martí

Declaración de autoría

Por este medio declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 2 de la Universidad de las Ciencias Informáticas para que hagan el uso que estimen pertinente con el mismo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del 2009.

_____ _____

Nestor Reina Marichal Ronnier LLomabrt Hardy

_____ _____

Ing. Yunisel Viera Vargas Ing. Madelis Pérez Gil

Agradecimientos

A nuestros compañeros de los cinco años de la carrera por brindarnos su amistad.

A los compañeros del proyecto por su colaboración en este trabajo.

A nuestros profesores, por su ejemplo de profesionalidad.

A los "colegas", especialmente a Rammel por su entrega y dedicación.

A nuestros tutores Madelis y Yunisel, por asesorarnos a lo largo de la tesis y acompañarnos en este camino que hoy culmina en el presente proyecto.

A todos aquellos que hicieron posible la confección y elaboración de este trabajo.

Dedicatoria

A mis padres, por su ejemplo, su incomparable aprecio, por permitirme ver la vida desde todas sus aristas…

A mi abuela …por su confianza…

A mis hermanos por sus preciados consejos y su apoyo incondicional…

A mi familia, por estar siempre a mi lado…

A Rosa, por quererme como un hijo más…

A mis amigos más íntimos, ellos no necesitan mención, saben que siempre estarán presentes, que forman parte de mi familia…

A las personas que forman parte de mi vida…

A Lida, por ser, por estar, por existir.

Ronnier

A mis padres por todos estos años de sacrificio, en especial a mi madre por estar siempre presente.

A mi familia por brindarme todo su apoyo.

A mi tía Normita por ser mi apoyo estos años de la carrera.

A mis abuelas.

A mí querida novia Dainivis por estar a mi lado y brindarme todo su cariño.

A mi hermano, por ser mi orgullo.

Nestor

Resumen

El despliegue de servicios de telefonía basados en VoIP[1] y las peculiaridades de éstos en comparación con el servicio tradicional, hacen necesario disponer de una tecnología que facilite las labores de identificación de cada usuario.

Uno de los primeros avances en esta área de nuevas aplicaciones reales ha sido la llegada del ENUM[2], con el objetivo de desarrollarlo en nuestro país surge el proyecto ***TeleIdentificador Personal*** que permitirá el despliegue profesional del servicio ENUM de Usuario, y como parte de este sistema se crea el trabajo ***TeleIdentificador Personal. Desarrollo de la Plataforma Manejadora de Peticiones. Capa de Acceso a Datos y Capa de Negocio***, con el objetivo concreto de implementar la Capa de Acceso a Datos y la Capa de Negocio, exponiendo a las capas superiores las funcionalidades que debe ofrecer la Plataforma Manejadora de Peticiones.

Este trabajo dará paso a lograr un soporte estandarizado para las necesidades de almacenamiento y recuperación de información del sistema antes mencionado así como una implementación reutilizable de los componentes que conforman la lógica de Negocio utilizando para ello las facilidades brindadas por Spring Framework, el Entorno de Desarrollo Integrado Eclipse 3.4, la plataforma Java 2, Enterprise Edition y el framework JUnit para realizar pruebas de unidad.

[1] **VoIP:** Tecnología que permite la transmisión de voz a través de redes IP, Internet normalmente.

[2] **ENUM**: Acrónimo de t***E***lephone ***NU***mber ***M***apping (en español Mapeo de Números Telefónicos utilizando el estándar de numeración telefónica E.164).

Índice

DECLARACIÓN DE AUTORÍA ..... **III**

AGRADECIMIENTOS ..... **IV**

DEDICATORIA ..... **V**

RESUMEN ..... **VI**

ÍNDICE ..... **VII**

INTRODUCCIÓN **1**

CAPITULO I: FUNDAMENTACIÓN TEÓRICA **5**

1.1 PRINCIPALES CONCEPTOS 6

1.1.1 Protocolo ENUM *6*

1.1.2 Plataforma Manejadora de Peticiones *8*

1.2 ESTUDIO DE MERCADO 9

1.2.1 ENUM Trial en México *9*

1.2.2 ENUM en Austria *10*

1.2.3 ENUM en Cuba *11*

1.3 PRINCIPALES TECNOLOGÍAS 11

1.3.1 Ambiente de Desarrollo Integrado (IDE) *11*

1.3.1.1 Eclipse 12

1.3.2 Framework *12*

1.3.2.1 iBatis 13

1.3.2.2 Spring 15

1.3.3 Sistemas Gestores de Bases de Datos *16*

1.3.3.1 PostgreSQL 16

1.4 LENGUAJE UTILIZADO 17

1.4.1 Java *17*

CONCLUSIONES 19

CAPITULO II: DESCRIPCIÓN Y CARACTERÍSTICAS DEL SISTEMA PROPUESTO. **20**

2.1 DESCRIPCIÓN DEL TELEIDENTIFICADOR PERSONAL 20

2.1.1 Capa de Acceso a Datos *21*
2.1.2 Capa de Negocios *22*
2.2 MODELO DE DOMINIO 23
2.2.1 Conceptos del Modelo del Dominio *23*
2.3 REQUISITOS FUNCIONALES 24
2.4 REQUISITOS NO FUNCIONALES 26
2.4.1 Usabilidad *26*
2.4.2 Seguridad *26*
2.4.3 Fiabilidad *27*
2.4.4 Portabilidad *27*
2.4.5 Soporte *27*
2.4.6 Requisitos para la documentación de usuarios en línea y ayuda del sistema *27*
2.4.7 Interfaces de Comunicación *27*
2.4.8 Hardware *28*
2.4.9 Software *28*
2.4.10 Estándares Aplicables *28*
2.5 PRINCIPALES CASOS DE USOS DE LA PLATAFORMA MANEJADORA DE PETICIONES 28
2.6 REGLAS DE NEGOCIO 31
2.6.1 Reglas de estructura *32*
2.6.2 Reglas de derivación *32*
2.6.3 Reglas de acción *32*
2.7 PATRONES DE DISEÑO UTILIZADOS 33
CONCLUSIONES 33
CAPITULO III: ELEMENTOS Y ESTRUCTURA DE LAS CAPAS DE PERSISTENCIA Y NEGOCIO 34
3.1 ELEMENTOS Y ESTRUCTURA DE LA CAPA DE PERSISTENCIA 34
3.1.1 Paquete cu.uci.tip.pmp.dataacces.dnsdao *36*
3.1.2 Paquete cu.uci.tip.pmp.dataacces.enumdao *36*
3.1.3 Paquete cu.uci.tip.pmp.dataacces.enumdao.util *37*
3.1.4 Paquete cu.uci.tip.pmp.dataacces.enumdao.config *38*
3.1.5 Paquete cu.uci.tip.pmp.dataacces.enumdao.map *42*
3.1.6 Paquete cu.uci.tip.pmp.dataacces.enumdao.dao *43*

3.2 ELEMENTOS Y ESTRUCTURA DE LA CAPA DE NEGOCIO 45
3.2.1 Paquete business.interfaces *46*
3.2.2 Paquete business.impl *54*
3.3 MODELO DE DISEÑO Y MODELO DE DATOS 54
3.4 MODELO DE IMPLEMENTACIÓN 61
3.4.1 Diagrama de Componentes *61*
CONCLUSIONES 65

CAPITULO IV: PRUEBAS AUTOMATIZADAS 66

4.1 PRUEBAS DE UNIDAD 66
4.2 PRUEBAS DE INTEGRACIÓN 73
4.2.1 Integrando JUnit con Spring *73*
CONCLUSIONES 78

CONCLUSIONES 79

RECOMENDACIONES 80

BIBLIOGRAFÍA 81

GLOSARIO DE TÉRMINOS 83

Introducción

Tradicionalmente, el teléfono ha sido uno de los sistemas de comunicación más utilizados, pero Internet ha supuesto una revolución sin precedentes en el mundo de la Informática y las Comunicaciones. Su desarrollo ha alcanzado niveles inesperados, y una de las causas fundamentales de este suceso son los múltiples servicios que brinda; esto ha hecho de Internet una red de rápida expansión, logrando una alta popularidad en todo el mundo.

Con todas las tecnologías, medios de comunicación y servicios brindados por las empresas de telecomunicaciones como correos, sitios web, móviles; los contactos se incrementan considerablemente y no solo bastará con el número telefónico para poder comunicarse, se necesitará de todos los contactos que identifiquen amigos y familiares. Esto le impone hoy en día un gran reto a las empresas de este sector.

En Cuba, la Empresa de Telecomunicaciones de Cuba S.A (ETECSA), tiene como objeto social prestar los servicios públicos de telecomunicaciones, mediante la operación, instalación, explotación, comercialización y mantenimiento de redes públicas de telecomunicaciones[3] en todo el territorio. (1)

Esta empresa tiene una alta responsabilidad en el desarrollo socio-económico del país y en especial, en la informatización de la sociedad, garantizando una efectiva conectividad. Los principales resultados obtenidos en los últimos años pudieran citarse como:

- ✓ Existencia de un plan de expansión y modernización tecnológica.
- ✓ Incremento considerable de la digitalización.
- ✓ Introducción de nuevos servicios de telecomunicaciones. (1)

Uno de los desafíos técnicos del sector de las telecomunicaciones, planteados por la inminente integración entre las redes de conmutación de circuitos y las redes de conmutación de paquetes, es la forma de dirigir las llamadas que pasan de un servicio de red a otro. (2)

[3] **Telecomunicaciones:** Tecnología que permite la transferencia de un mensaje de un punto a otro. Es toda emisión, recepción y transmisión, de signos, señales, escritos, imágenes, sonidos y cualquier tipo de datos, por cable, radio, medios ópticos u otros sistemas electromagnéticos.

Entre las diferentes soluciones que se han propuesto para resolver lo anteriormente expuesto se encuentra la "Correspondencia de Números Telefónicos" más conocido por sus siglas en idioma inglés "ENUM" –T**e**lephone **Nu**mber **M**apping– desarrollada por la Internet Engineering Task Force (IETF)[4]. (2)

Situación problemática

A lo largo de muchos años, se ha carecido de un identificador personal que permita aglutinar una serie de prestaciones que son brindadas por ETECSA.

Actualmente, en Cuba, no existe un servicio que permita dirigir las comunicaciones de una red a otra, problema que en ocasiones resulta tedioso, debido a que, tanto en la red de conmutación de circuitos como en la de paquetes, existen múltiples servicios de comunicación que si estuviesen integrados, facilitaría el intercambio entre las personas sin importar los tipos de contactos, ej. Correo electrónico, teléfono, sitio web; que se tengan.

Para el desarrollo e implantación de este novedoso servicio, denominado *ENUM de Usuario*, se necesita de un conjunto de herramientas informáticas que permitan a las operadoras controlar y regular el mismo y a los clientes acceder a las funcionalidades y beneficios rápida y eficientemente.

Al desarrollar un conjunto de aplicaciones en diversos lenguajes, se hace engorroso implementar muchas veces la misma lógica de negocio en cada una de las aplicaciones por separado. Es por esto que se hace necesario un sistema que gestione el flujo de información entre la base de datos, los DNS[5] y las aplicaciones desarrolladas para el *ENUM de Usuario*.

Por lo antes expuesto el Problema al cual se le dará solución será *¿Cómo gestionar las peticiones a la base de datos y DNS de las distintas aplicaciones clientes que complementan el servicio de ENUM de Usuario en Cuba*?

Objeto de estudio

Proceso de administración de peticiones a servidores DNS y acceso a las bases de datos de las aplicaciones empresariales.

---

[4] **IETF:** Abierta comunidad internacional de investigadores interesados en la evolución y el correcto funcionamiento de Internet.

[5] **DNS**: Acrónimo de Domain Name Server (en español Servidor de Nombres de Dominio).

Campo de acción

Capa de Acceso a Datos y Capa de Negocio de la *Plataforma Manejadora de Peticiones (PMP)*.

Objetivo General

Implementar la Capa de Acceso a Datos y Capa de Negocio para la *Plataforma Manejadora de Peticiones* del proyecto *TeleIdentificador Personal*.

Tareas de investigación

- ✓ Investigar sobre las características y ventajas de la implementación de las capas de acceso a fuentes de datos en los sistemas de software.
- ✓ Consultar la bibliografía científica técnica relacionada con el protocolo ENUM para comprender su funcionamiento.
- ✓ Estudiar los lenguajes vinculados al desarrollo de la Capa de Acceso a Datos y Capa de Negocio para evitar cometer errores en su selección.

Aportes Prácticos

Debido al interés de Cuba por brindar opciones de calidad a la población se decidió implementar el servicio *ENUM de Usuario*, que involucrará a ETECSA y a la Universidad de las Ciencias Informáticas (UCI), bajo el proyecto de nombre *TeleIdentificador Personal*. Este tendrá como objetivo principal desarrollar un identificador personal de telecomunicaciones que permita a cada llamante, conociendo sólo un número de la persona con quien desee comunicar, independientemente del contenido a intercambiar (llamada de voz, correo, videoconferencia, etcétera) establecer comunicación con el mismo.

Con el desarrollo y despliegue del *TeleIdentificador Personal*, Cuba, como nación en vías de desarrollo, contará con un servicio que facilite las telecomunicaciones, tanto a un plano empresarial como para la población. Las ventajas que aportará la implantación del ENUM en Cuba serán apreciadas por todos los residentes en el país de una forma inmediata.

Para dar cumplimiento al objetivo general y a las tareas planteadas, el actual documento está conformado por 4 capítulos:

CAPÍTULO I FUNDAMENTACIÓN TEÓRICA:

Incluye el estado del arte del tema a tratar, y el análisis de de las tendencias, tecnologías y software usados en la actualidad para resolver el problema. Se incluye además una breve descripción de las herramientas propuestas para la implementación.

CAPÍTULO II DESCRIPCIÓN Y CARACTERÍSTICAS DEL SISTEMA PROPUESTO:

Describe el objeto de estudio, problema y situación problemática, objeto de automatización, propuesta de sistema, el entorno de trabajo en que se desarrolla el sistema, requerimientos funcionales y no funcionales y los casos de uso.

CAPÍTULO III ELEMENTOS Y ESTRUCTURA DE LAS CAPAS DE PERSISTENCIA Y NEGOCIO:

Incluye una descripción detallada de toda la configuración necesaria para la funcionalidad del sistema, también incluye los diagramas de clases, además de la descripción de estas. Abarca el diseño de la base de datos el cual contiene el Diagrama de Modelos de Datos.

CAPÍTULO IV PRUEBAS AUTOMATIZADAS:

Se validan, comprueban o evalúan los componentes implementados en la capa de persistencia, además de comprobarse su correcto funcionamiento al integrarse con la lógica de la capa de negocio mediante las pruebas de unidad y de integración.

Capítulo 1

Fundamentación Teórica

A través de los años, la rama de las telecomunicaciones ha sido un tema de debate y de gran desarrollo, a partir del descubrimiento de la electricidad, grandes inventos fueron revolucionando este concepto, desde el surgimiento del telégrafo, el cual abrió las puertas al origen del teléfono y con él, la consolidación de un sistema mundial de telecomunicaciones, hasta la expansión mundial de tecnologías que se reconocería como la base de la moderna Internet.

Actualmente, es común que estas tecnologías formen parte integral de la vida social de las personas, y paralelamente, cada vez hay más usuarios de Internet, planteándose la necesidad de poder encontrar y asociar sus recursos de Internet con su número telefónico y viceversa. Por la importancia que tiene el tema a nivel internacional y nacional, muchos son los esfuerzos por lograr desarrollar sistemas que posibiliten la convergencia entre los servicios suministrados por las redes de comunicaciones y los que se proveen en la Internet, utilizando el protocolo ENUM. (2)

En este capítulo se abordan algunos de los conceptos teóricos que fueron necesarios investigar para el desarrollo de este trabajo y se ofrece un enfoque general de algunos sistemas ENUM que brindan servicios o que están en fase de prueba y desarrollo a nivel mundial. En el mismo se hace indispensable introducir algunos temas relacionados a las telecomunicaciones y las herramientas y tecnologías utilizadas para el buen desarrollo de la aplicación.

1.1 Principales Conceptos

1.1.1 Protocolo ENUM

Inicialmente, un grupo de trabajo de la Internet Task Force (IETF), analizó la posibilidad de utilizar nombres de dominios numéricos con la estructura jerárquica que marca la norma E.164[6], debido a su buena aceptación por el sistema DNS, que permitiría interrogar a un servidor de este tipo para encontrar información asociada a ese número, el DNS contestaría a estas peticiones con una serie de registros NAPTR[7] que especifican las diferentes formas de contactar con el propietario de dicho número, ya sea mediante una página web , un e-mail o un móvil, y si ese número además, es un número telefónico, empezarían a tener un punto de encuentro dos redes que hasta ahora funcionaban de forma independiente, surgiendo un esquema que facilitaría el direccionamiento de las comunicaciones en un entorno convergente, denominado ENUM (Figura 1). (3)

El protocolo ENUM (Electronic NUMber / tElephone NUmber Mapping) correlaciona los números de los planes de numeración nacionales conformes con la Recomendación E.164 (Plan Internacional de Numeración de las Telecomunicaciones Públicas) de la UIT-T[8] con los identificadores de recursos uniformes (URI) almacenados en las bases de datos jerárquicas y físicamente distribuidas del Sistema de Nombres de Dominio o DNS definido en la RFC 3761[9] de la IETF. (2)

Es evidente que la necesidad de soluciones como el ENUM se debe a que los números telefónicos no tienen significado en una red IP[10]. Además existen fuertes indicadores que los números E.164

---

[6] **E.164:** Recomendación de la Unión Internacional de Telecomunicaciones que asigna a cada país un código numérico (código de país) usado para las llamadas internacionales.

[7] **NAPTR**: Acrónimo de Naming Authority Pointer, nuevo tipo de registro DNS que admite expresiones regulares o forma normalizada de escribir las cadenas usadas especialmente para buscar patrones de textos determinados. De este modo se puede determinar los métodos o servicios disponibles para contactar un nodo específico identificado mediante un número de la Recomendación E.164.

[8] **UIT-T:** Sector de Normalización de las Telecomunicaciones de la Unión Internacional de Telecomunicaciones (UIT).

[9] **RFC 3761:** Recomendación de la IETF del año 2004 que analiza el uso de sistema de nombres de dominio (DNS) para almacenamiento de números E.164.

[10] **IP**: Acrónimo de Internet Protocol, (en español Protocolo de Internet). Protocolo de comunicación estándar de entrega de paquetes entre dos computadores dentro de Internet.

permanecerán como el identificador para los servicios de voz a largo plazo, entre los más significativos se encuentran que: (2)

- ✓ Son comprensibles universalmente.
- ✓ Son únicos.
- ✓ Son neutrales con respecto a la tecnología.
- ✓ Se asientan en un fuerte sistema de Normas Internacionales.

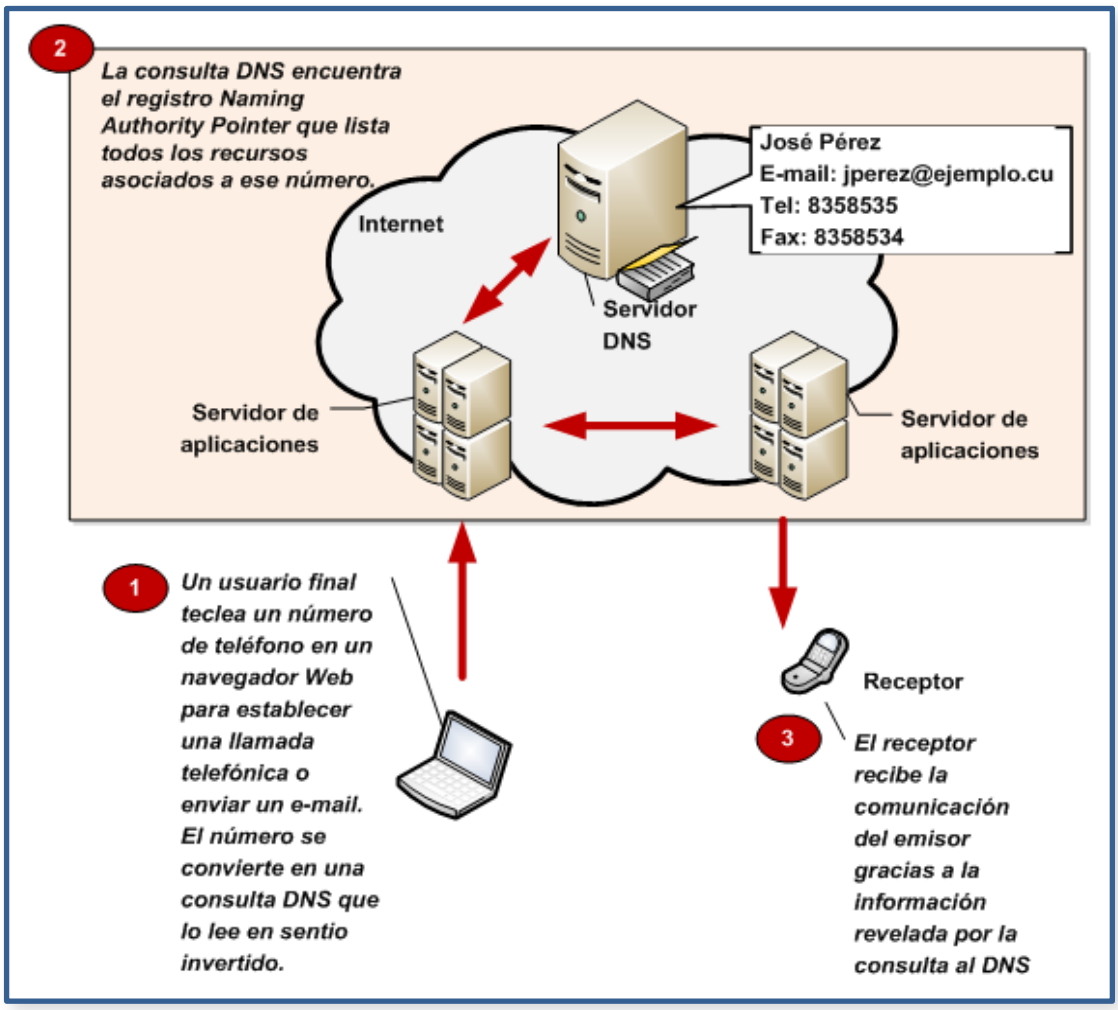

Figura 1: Funcionamiento del ENUM.

En síntesis el ENUM es una base de datos en la jerarquía DNS. Como con cualquier base de datos una cuestión crucial es quien tiene los derechos para acceder o cambiarla. La base de datos DNS de la Internet es de disponibilidad pública. De esta forma cualquier usuario puede tomar información del DNS. No obstante sólo los dueños de los dominios pueden cambiar la información lo que constituye un claro concepto. Con ENUM este asunto no es tan claro como en los DNS actuales. De ahí que surgen dos importantes preguntas: (2)

- ✓ ¿Quién puede usar la información de los registros ENUM?
- ✓ ¿Quién tiene autorización para cambiar o poblar los registros ENUM?

Estas preguntas han dado lugar a clasificar el ENUM en dos tipos: *ENUM de Usuario* y *ENUM de Infraestructura*.

El implementado para Cuba es el *ENUM de Usuario* que no es más que la correspondencia de números telefónicos E.164 con los Identificadores de Recursos Uniforme (URI) utilizando el Sistema de Nombre de Dominio (DNS) en el dominio *e164.arpa*, con la restricción de que tanto el mantenimiento como el uso de los registros está bajo la autoridad del usuario. (2)

1.1.2 Plataforma Manejadora de Peticiones

El desarrollo del servicio *ENUM de Usuarios* implica la integración y producción de un conjunto de sistema que permitirán que los usuarios interactúen con el mismo y que ETECSA lo pueda brindar y administrar. Esto generará el uso de varias tecnologías y herramientas de trabajo. Lograr una integración exitosa de estas garantizará la calidad del mismo. Para ellos se definió una arquitectura que es flexible a esta diferencia donde todas las peticiones de los usuarios son tramitadas por la *Plataforma Manejadora de Peticiones*, garantizando así centralizar toda la información y el control del acceso a los DNS, estableciendo un modo estandarizado del acceso a los datos.

Principales funcionalidades:

- ✓ *Gestionar las peticiones de los clientes:* Permitirá darle respuesta a las peticiones de los clientes del servicio con la calidad y la rapidez requerida.

- ✓ *Estandarizar el acceso a la información:* Todo el acceso a los DNS y la BD[11] ENUM se realizará a través de la plataforma, separando así esta responsabilidad de las aplicaciones clientes.
- ✓ *Integrar aplicaciones:* La plataforma contará con una capa de servicios que permitirá integrar y comunicar todas las aplicaciones que se desarrollen por separado y que requiera interactuar con la información de los subscriptores almacenada en el BIND[12] DNS y la base de datos ENUM.

1.2 Estudio de Mercado

Internacionalmente, ENUM cada vez gana más aceptación a pesar de sus restricciones, alargándose la lista de países que se muestran interesados por dar el novedoso salto en sus sistemas de telecomunicaciones.

País	Estado
Alemania, Austria, Inglaterra, Polonia, Finlandia, Israel	Empresa en explotación
España, MERCOSUR, Singapur	En estudio
Cuba	En desarrollo
Suecia, Irlanda, Japón, EUA	Pruebas concluidas
Canadá, México	En desarrollo
Francia, China, Australia, Brasil, Grecia, Holanda, Noruega, Eslovaquia, Corea del Sur, Suiza, Hungría	Plataforma experimental

Tabla 1: ENUM a nivel mundial, Octubre del 2008.

1.2.1 ENUM Trial en México

ENUM Trial México es un proyecto que se realizó en conjunto NIC México (Network Information Center – México, organización encargada de la administración del nombre de dominio territorial MX) y Tecnológico

[11] **BD**: Base de Datos

[12] **BIND**: Acrónimo de Berkeley Internet Name Domain. Es el servidor de DNS más comúnmente usado en Internet, especialmente en sistemas Unix, en los cuales es un estándar por defecto.

de Monterrey Campus Monterrey; a través de su proyecto se convierten en pioneros en México en tecnología ENUM, al utilizar, experimentar y ofrecer una plataforma que hace uso de dicha tecnología. El objetivo fundamental de ENUM Trial México consistió en que sus usuarios de Internet y las operadoras telefónicas del país pudieran experimentar y familiarizarse con la tecnología ENUM así como analizar los aspectos operacionales y técnicos del aprovisionamiento de ENUM para el código de país +52. (4)

En general, el funcionamiento de *ENUM Trial México*, consiste que al obtener un número telefónico privado, el usuario puede fácilmente crear registros NAPTR (del inglés: Naming Authority Pointer) mediante una interfaz Web bajo el dominio .enum.org. Por ejemplo, si el sistema asigna el teléfono 80001, entonces automáticamente se genera un dominio 1.0.0.0.8.enum.org.mx en los servidores de DNS de ENUM Trial México. El dominio generado tiene la información de los servidores SIP que pueden ser utilizados para contactarse a un dispositivo de VoIP o página Web o dirección de correo electrónico. Para usar el número telefónico, el usuario debe tener un dispositivo de VoIP enlazado con un servidor SIP que soporte realizar búsquedas ENUM. El servidor SIP deberá ser configurado para realizar búsquedas bajo el dominio enum.org.mx. (4)

1.2.2 ENUM en Austria

Austria en uno de los primeros países que se interesó por el protocolo ENUM poniendo en marcha el proyecto *ENUM.at* el cuál es una filial al 100% de la fundación Internet Privatstiftung Austria (IPA), convirtiéndose así en una empresa afiliada de nic.at, que se encarga de la administración y registro de todos los dominios. (5)

Sobre la base del contrato entre RTR (del inglés: Rundfunk und Telekom Regulierungs-GmbH) y *ENUM.at*, el primer paso de un uso comercial de ENUM basada en servicios de Internet se ha hecho. *ENUM.at* actualmente ha sido asignado para administrar la zona ENUM 3.4.e164.arpa. (5)

Por primera vez ENUM proporciona un estándar global para vincular los números de teléfonos con los recursos de Internet, permitiendo así el simple tratamiento de estos servicios de Internet a través de un número de teléfono. Austria es el primer país del mundo donde ENUM está disponible para los servicios comerciales.

1.2.3 ENUM en Cuba

El desarrollo del *ENUM de Usuarios* en Cuba hace especial énfasis en permitir el uso de los teléfonos de la red nacional de telefonía para acceder a los servicios de Internet, permitiendo evaluar de forma concreta algunos de los primeros servicios de convergencia entre estas dos redes(redes de conmutación de circuitos y redes de conmutación de paquetes). A pesar de los esfuerzos que realiza la dirección del país por informatizar la sociedad, no se cuenta todavía con una infraestructura tecnológica que permita adecuadamente gestionar y acceder mediante computadoras a las funcionalidades y beneficios del servicio *ENUM de Usuarios*, esto es un elemento a considerar a la hora de desarrollar los sistemas que interactuarán con el servicio, sin embargo sí se cuenta con una gran cantidad de teléfonos fijos o de la red pública de servicios telefónicos PSTN (del inglés: Public Service Telephone Network), por lo que uno de los principales retos será la utilización de estos teléfonos para acceder a todos los servicios suministrados por las redes IP.

El desarrollo del ENUM de Usuarios encierra grandes retos, y aunque parezca sencillo constituye un proceso largo y complicado por lo que su puesta en funcionamiento puede demorar más de 2 años de trabajo continuo.

La producción de todas las aplicaciones que permitirán interactuar a las operadoras de servicios y a sus clientes con el ENUM de Usuarios deberán de estar diseñadas e implementadas según las características de las personas a las que van dirigidas, sus recursos y habilidades. El desarrollo del ENUM cubano está caracterizado por el uso tecnologías libres y de código abierto por lo que constituye otro paso más del país en lograr su soberanía tecnológica.

1.3 Principales Tecnologías

1.3.1 Ambiente de Desarrollo Integrado (IDE)

Un ambiente de desarrollo integrado o en inglés Integrated Development Environment (IDE) es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien, poder utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica

(GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Estos proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Java, C#, Visual Basic, Object Pascal, etcétera.

1.3.1.1 Eclipse

Aunque Eclipse es escrito en Java y su principal uso es como IDE para Java, este es neutral. El soporte para desarrollo en Java es proveído por un componente enchufado o plugin, pero además están disponibles plugins para otros lenguajes, como C/C++, Cobol, C#. (6)

Alguna de las ventajas que aporta el Eclipse como IDE de desarrollo:

- ✓ Es administrado y dirigido por un consorcio de compañías de desarrollo de software con un interés comercial en promover Eclipse como plataforma compartida para herramientas de desarrollo de software.
- ✓ En principio permite ejecutar un programa sobre cualquier plataforma.
- ✓ Es una extensible plataforma de código abierto (open source) para desarrollar herramientas.
- ✓ Eclipse compila en tiempo real, es decir, no espera a que tengas el código listo para compilarlo tras la pulsación de sus botones, sino que, conforme vas escribiendo el código, te va avisando de los errores que va cometiendo.
- ✓ Posee un completamiento de código muy cómodo para los desarrolladores.

Eclipse además, brinda un conjunto de facilidades como: ambiente de trabajo ordenado, integración total con SVN y soporte de refactorización que lo convierten en un IDE muy utilizado en el mundo profesional dado su alto grado de aceptación.

1.3.2 Framework

Recientemente, el interés en reutilizar software ha sido cambiado de la reutilización de componentes simples a diseño de sistemas enteros o estructuras de aplicaciones. Un sistema software que pudiera ser reutilizado en este nivel para la creación de aplicaciones completas es llamado framework. Los frameworks son basados en la idea que deberían permitir la producción fácil de un conjunto de sistemas específicos pero similares, dentro de un cierto dominio comenzando desde una estructura genérica.

Brevemente, los frameworks son arquitecturas genéricas integradas por un conjunto de componentes extensibles. Además, los frameworks pueden contener subframeworks los cuales representen subconjuntos de componentes de un sistema más grande. (12)

Un framework contiene clases o estructuras que implementan los componentes de una aplicación genérica también como componentes concretos que satisfacen tareas especializadas. Para desarrollar programas completos, los desarrolladores buscan e instancian los componentes apropiados.

No hay una definición común para los frameworks, pero la mayoría tiene un tema común: la reutilización. Una definición ampliamente aceptada es dada por R. E. Johnson, and B. Foote en 1988 en su publicación "Designing Reusable Classes":

"Un framework es un conjunto de clases que personifican un diseño abstracto para soluciones de una familia de problemas relacionados..."

1.3.2.1 iBatis

Uno de los problemas con el que se encuentran los desarrolladores en Java, que en sus aplicaciones interactúan con una base de datos, es que lo hacen desde un lenguaje orientado a objetos hacia un sistema que tiene un modelo relacional, esto debido en ocasiones a la no existencia de un Object-Relational Mapping (ORM) totalmente preparado para las operaciones que con los modelos relacionales se presentan a la hora de desarrollar la capa de persistencia correspondiente a una determinada solución de software.

IBatis es un framework o motor de persistencia que brinda una solución a este problema.

Alguna de los principales beneficios que aporta iBatis como framework de persistencia y alguna de las razones por las cuales se utiliza para la implementación de la Capa de Acceso a Datos son las siguientes: (13)

- ✓ *Simplicidad:* iBatis es ampliamente considerado como uno de los más simples framework de persistencia disponible en la actualidad. La simplicidad es la esencia de los objetivos del equipo de desarrollo de iBatis, y tiene prioridad sobre casi todo lo demás. Esta sencillez se consigue mediante el mantenimiento de una muy sólida base sobre la cual se construye iBatis: JDBC y SQL. Es fácil de usar para los desarrolladores de Java, ya que funciona como JDBC, sólo que con

mucho menos código. Sus archivos de configuración pueden ser fácilmente entendidos por casi cualquier persona con alguna experiencia en programación SQL.

- ✓ *No se impone al diseño actual de la aplicación o de la base de datos*: Si se tiene un pequeño sistema que ya está implementado parcialmente o incluso con alguna versión liberada, todavía es fácil introducir iBatis en la capa de persistencia. Lo mismo podría no ser cierto en el caso de herramientas de mapeo de objetos relacionales o generadores de código que hacen suposiciones sobre el diseño de la aplicación o bien de la base de datos.
- ✓ *Productividad:* La finalidad primordial de cualquier buen framework es hacer al desarrollador más productivo. Generalmente existe un framework para hacerse cargo de las tareas comunes, reducir código y resolver problemas arquitectónicos complejos. El código consistente y la fácil configuración de iBatis ha logrado reducir la cantidad de código en la capa de persistencia a un importante 62%. Este ahorro se debe principalmente al hecho de que ningún código JDBC debe ser escrito.
- ✓ *División del trabajo:* Debido a que el código SQL es separado en gran parte del código fuente de la aplicación, los programadores de SQL pueden escribir la sentencia SQL correspondiente, sin tener que preocuparse por el código Java involucrado. Esto no es una cosa fácil de hacer con JDBC. iBatis posibilita separar el trabajo del programador de SQL y el programador Java manteniendo la consistencia que se requiere.
- ✓ *Portabilidad:* Debido a su diseño relativamente simple, puede ser aplicado para cualquier tipo de lenguaje o de plataforma. Hasta el momento, iBatis soporta las tres plataformas de desarrollo más populares: Java, Ruby y C#.Net. iBatis trabaja con más idiomas y más tipos de aplicaciones que cualquier otro framework, independientemente del diseño de la aplicación.
- ✓ *Open source:* iBatis es libre, se encuentra bajo la licencia Apache 2.0.

Todas estas ventajas y características hacen muy atractivo al framework iBatis para utilizarlo en cualquier aplicación profesional, más si se utiliza también el framework Spring que le brinda más funcionalidades y prestaciones. Básicamente, todo lo que necesita es una instancia de *SqlMapClient*, configurado con los archivos de mapeo, y accesible por los diferentes DAOs de la capa de persistencia. La gestión de transacciones es exactamente la misma que en un escenario de JDBC, utilizando las clases de Spring *DataSourceTransactionManager* o *JtaTransactionManager*.

1.3.2.2 Spring

Spring Framework es un framework de aplicación de código abierto que ayuda a hacer el desarrollo en JEE[13] mucho más fácil. Ayuda a estructurar aplicaciones completas en una manera consistente y productiva para crear arquitecturas coherentes. (14)

Es el más popular y el más ambicioso de todos los framework de peso ligero. Es el único framework que interviene en todas las capas arquitectónicas de una aplicación JEE. Además está diseñado para facilitar una flexibilidad arquitectónica. (14)

Los principales valores de Spring, según Rod Johnson, en su libro: "Professional Java Development with the Spring Framework", se pueden resumir en:

- ✓ *Ayuda a promover la reusabilidad de código:* Ayuda a evitar la necesidad de tomar decisiones importantes y duras, como es si una aplicación alguna vez usará JTA o JNDI; las abstracciones de Spring permitirán desarrollar el código en ambientes diferentes si esta lo requiere.
- ✓ *Facilita el diseño Orientado a Objetos (OO) en aplicaciones JEE:* ¿Cómo una aplicación JEE, escrita en Java – un lenguaje OO– no es OO? En realidad muchas aplicaciones JEE no merecen el nombre OO. Con Spring es más fácil eliminar los impedimentos del diseño OO en aplicaciones JEE tradicionales haciendo el código más coherente, con más bajo acoplamiento y más reusable.
- ✓ *Facilita buenas prácticas de programación, tales como la programación a interfaces:* El uso de un contenedor de Inversión de Control reduce grandemente la complejidad del código a interfaces, más que a clases. El uso de los objetos a través de estas interfaces protege los requerimientos, los cuales pudieran cambiar en el desarrollo de la aplicación.
- ✓ *Permite la extracción de valores de configuración desde el código java a archivos XML o archivos de propiedades:* Mientras que algunos valores de configuración pudieran ser programados en Java, todas las aplicaciones de mediana y alta complejidad necesitan algunas configuraciones externalizadas del código fuente, para permitir la administración sin recompilar las clases nuevamente.
- ✓ *Está diseñado a fin que las aplicaciones lo usen para que las pruebas sean lo más fácil posible:* Hasta donde sea posible, los objetos de las aplicaciones serán POJOs los cuales son fáciles de

[13] **JEE**: Java Enterprise Edition.

probar. La dependencia sobre las APIs de Spring normalmente serán en forma de interfaces que son fáciles para simular.

- ✓ *Promueve la selección arquitectónica:* Spring apunta para facilitar el reemplazo de cada capa. Por ejemplo, con una capa media de Spring, se pudiera ser capaz de cambiar de un framework de mapeo objeto relacional (ORM) a otro con un impacto mínimo sobre el código de la lógica de negocio, o cambiar de Struts a Spring MVC o WebWork sin algún impacto en la capa media.

1.3.3 Sistemas Gestores de Bases de Datos

Un Sistema Gestor de Base de Datos (SGBD) es un tipo de software muy específico o conjuntos de ellos que permite manejar de manera clara, sencilla y ordenada altos volúmenes de datos. Actualmente existe una amplia gama de SGBD con características propias, no obstante todos deben tener en cuenta los siguientes aspectos de manera general: abstracción de la información, la independencia de los datos, redundancia mínima, consistencia en los datos, seguridad, integridad, respaldo y recuperación, tiempo de respuesta y control de concurrencia.

Existen SGBD muy potentes tanto en la clasificación de software propietario como software libre. Como propietarios podemos encontrar Oracle y Microsoft SQL Server que lideran el mercado por sus altas prestaciones dado muchos años de experiencia mientras que como opción libre se encuentra, entre otros, MySQL, gestor muy utilizado en la web por su simplicidad de uso, y PostgreSQL que representa una opción muy confiable y comprometedora y es catalogado como la mejor y más avanzada base de datos Open Source del mundo.

1.3.3.1 PostgreSQL

PostgreSQL es un sistema gestor de base de datos objeto-relacional desarrollado sobre licencia open source. Ofrece una alternativa ante otros sistemas gestores de bases de datos comerciales. Similar a proyectos de software libres como Apache y GNU/Linux, PostgreSQL no es controlado por ninguna compañía en específico, pero responde al esfuerzo de una comunidad global de desarrolladores. (15)

Está demostrado que PostgreSQL en sus versiones más recientes puede competir con los grandes productos comerciales como Oracle 10g y Microsoft SQL Server 2008 –la última versión estable es 8.3.7. No por gusto, empresas como IBM, Google, Skype, Deutch Bank y Fujitsu le dan soporte y lo usan en las

implementaciones de sus grandes bases de datos; además, también el dominio de Internet .ORG está soportado por dicho software.

Algunas de sus características más distintivas:

- ✓ *Es multiplataforma:* Está disponible para 34 plataformas en su última versión estable.
- ✓ *Cumple con las características ACID (Atomicity, Consistency, Isolation and Durability:* En español Atomicidad, Consistencia, Aislamiento y Durabilidad) para realizar transacciones seguras.
- ✓ *Aporta potencia y flexibilidad adicional:* Como son las restricciones (constraints), disparadores (triggers), reglas (rules) e integridad transaccional.
- ✓ *Diseñado para ambientes de alto volumen:* Usa una estrategia de almacenamiento de filas para conseguir una mejor respuesta en ambientes de grandes volúmenes. Ejemplo de esto lo demuestran los 32 TB máximo de tamaño de tabla, 1.6 TB máximo de tamaño de registro y 1 GB máximo de tamaño de campo.
- ✓ *Gran escalabilidad:* Es ajustable al número de procesadores y a la cantidad de memoria que posee el sistema de forma eficiente, por este motivo es capaz de soportar una mayor cantidad de peticiones simultáneas.

Puede ser utilizado, modificado y distribuido gratuitamente, para cualquier propósito, ya sea con fines privados, comerciales o académicos.

1.4 Lenguaje Utilizado

1.4.1 Java

Java es un lenguaje de programación independiente de la plataforma, creado por Sun MicroSystems. Alcanzó su madurez con la popularización de Internet y es en cierta manera el heredero legítimo de C++, eliminando la mayoría de sus complejidades como por ejemplo: la herencia múltiple y la creación de punteros.

Java presenta características que lo convierten en un lenguaje seguro, estándar y de alto nivel, algunas de las principales características se muestran a continuación:

- ✓ *Orientado a Objetos:* Basado en C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Soporta las tres características propias de la orientación a objetos: encapsulación, herencia y polimorfismo.
- ✓ *Distribuido:* Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como HTTP[14] y FTP[15]. La característica de ser distribuido es debido a que proporciona las librerías y herramientas para que los programas puedan distribuirse, es decir, que se corran en varias máquinas.
- ✓ *Interpretado:* Se traduce el código fuente a un código intermedio (bytecode), que es interpretado por La Máquina Virtual de Java, lo cual permite que se pueda ejecutar en cualquier sistema operativo.
- ✓ *Robusto:* Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo.
- ✓ *Seguro:* No se permite el acceso ilegal a memoria ya que no se trabaja con punteros.
- ✓ *Portabilidad:* Una de las principales características de Java es su portabilidad. En lugar de compilarse a código nativo de la máquina, los programas de Java se traducen al formato bytecode que es el mismo en cualquier sistema operativo. Estos bytecodes son procesados directamente por La Máquina Virtual de Java que es dependiente de la máquina en uso. La portabilidad radica en que se pueden programar las aplicaciones solo una vez y ejecutar en cualquier plataforma. Por ejemplo, cuando se compila un programa Java en una plataforma Windows/Intel, se obtiene la misma salida compilada (o los mismos bytecodes) que en un sistema Macinstosh o Unix
- ✓ *Altas prestaciones:* No se pierde tiempo optimizando código que no se ejecutará.
- ✓ *Multihilo:* Permite la ejecución de varias tareas a la vez.
- ✓ *Dinámico:* No conecta todos los módulos que comprende una aplicación hasta el tiempo de ejecución.

[14] **HTTP**: Acrónimo de HyperText Transfer Protocol (en español Protocolo de Transferencia de Hipertexto).

[15] **FTP**: Acrónimo de File Transfer Protocol (en español Protocolo de Transferencia de Archivos).

La programación en Java, permite el desarrollo de aplicaciones tanto bajo el esquema Cliente/Servidor, como de aplicaciones distribuidas, lo que lo hace capaz de conectar dos o más computadoras, ejecutando tareas simultáneamente, y de esta forma logra distribuir el trabajo a realizar.

Java, además, reúne un conjunto de elementos a su favor como por ejemplo: es un lenguaje maduro, con una amplia documentación y cuenta con muchos años de explotación, que lo ajustan mejor a las características de la solución a brindar.

La Máquina Virtual Java (JVM).

Recibe este nombre porque es una máquina imaginaria que se implementa emulando por software una máquina real. Es un programa ejecutable para una plataforma específica, capaz de interpretar y ejecutar el bytecode, el cual es generado por el compilador del lenguaje Java. Su misión principal es la de garantizar la portabilidad de las aplicaciones Java.

Las tareas principales de la JVM son las siguientes:

- ✓ Reservar espacio en memoria para los objetos creados.
- ✓ Liberar la memoria no usada (garbage collector).
- ✓ Asignar variables a registros y pilas.
- ✓ Llamar al sistema huésped para ciertas funciones, como los accesos a los dispositivos.
- ✓ Vigilar el cumplimiento de las normas de seguridad de las aplicaciones Java.

En la JVM se encuentra el motor que en realidad ejecuta el programa Java y es la clave de muchas de las características principales de Java, como la portabilidad, la eficiencia y la seguridad.

Conclusiones

En este capítulo se profundizó en algunos de los conceptos necesarios para la desarrollo de este trabajo. Se realizó un estudio de mercado el cual permitió apreciar el estado del arte en que se encuentran algunos países que explotan o implementan el ENUM, así como un breve recorrido por la evolución de las diferentes capas arquitectónicas implicadas en el desarrollo de la Plataforma Manejadora de Peticiones, específicamente la capa de Acceso a Datos y la capa de Negocio. Además se realizó un análisis completo de las tecnologías, herramientas y lenguajes que serán utilizadas a lo largo del desarrollo de la Plataforma.

Capítulo 2

Descripción y Características del sistema propuesto.

Para realizar cualquier sistema es necesario conocer ciertos conceptos, pautas y sobre todo comprender lo que se va a realizar, cómo y por qué.

En el transcurso de este capítulo se describe brevemente la solución propuesta del sistema a implementar, presentando las funcionalidades que brinda descritas en forma de requerimientos funcionales. Se especifica también las propiedades que el sistema posee mediante los requisitos no funcionales.

Se exponen conceptos que serán manejados tanto en la capa de de Acceso a Datos como en la de Negocio, así como los elementos organizacionales con el fin de lograr un mejor entendimiento de la estructura de la Plataforma, así como las reglas que deben satisfacerse para cumplir los objetivos y funcionalidades del *TeleIdentificador Personal.*

2.1 Descripción del TeleIdentificador Personal

El proyecto *TeleIdentificador Personal* (TIP) es un desarrollo en conjunto entre la Universidad de las Ciencias Informáticas (UCI) y la Empresa de Telecomunicaciones de Cuba (ETECSA), su futura implantación en el país no solo marcará un hito en las telecomunicaciones, sino que al prestar este servicio de avanzada se le proveerán soluciones efectivas a las cuestiones de cómo aunar sus contactos que hoy se manifiestan en la población.

TIP tiene para su desarrollo varios módulos entre ellos la *"Plataforma Manejadora de Peticiones"* (PMP) (Figura 2), la cual permite la integración de diversas aplicaciones donde todas las peticiones realizadas

por los usuarios son tramitadas por la PMP, garantizando así centralizar toda la información y el control del acceso a los DNS, estableciendo un modo estandarizado del acceso a los datos.

Dentro de las funcionalidades que brinda se destacan:

- ✓ *Gestionar las peticiones de los clientes:* Permite darle respuesta a las consultas hechas por los usuarios del servicio con la calidad y la rapidez requerida. El cliente tendrá la posibilidad de mostrar sus contactos, modificar la preferencia de los mismos y realizar búsquedas personalizadas.
- ✓ *Estandarizar el acceso a la información:* Todo el accedo a los DNS y la BD ENUM se realizará a través de la plataforma, específicamente a través de la capa de Acceso a Datos, separando así, esta responsabilidad de las aplicaciones clientes.
- ✓ *Integrar aplicaciones:* La plataforma cuenta con una capa de servicios que permite integrar y comunicar todas las aplicaciones que se desarrollen por separado y que requiera interactuar con la información de los subscriptores almacenada en el BIND DNS y la base de datos ENUM.

Para desarrollar la PMP se utilizó una arquitectura en capas, específicamente la arquitectura de tres capas.

La arquitectura en capas soporta un diseño basado en niveles de abstracción crecientes lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. Permite realizar optimizaciones y refinamientos enfocando los cambios en un solo lugar y proporciona amplia reutilización dada la división bien definida de responsabilidades. (8)

Para brindarle solución al problema científico planteado, se decidió implementar la Capa de Acceso a Datos y la capa de Negocio de la PMP.

2.1.1 Capa de Acceso a Datos

La capa de Acceso a Datos es la parte lógica, dentro de la arquitectura de sistemas multicapa que se encarga de implementar aquellas clases encargadas de realizar todas las operaciones con las bases de datos dentro de la aplicación. Contiene las clases que interactúan con la base de datos; y estas clases se implementan como una necesidad de mantener la cohesión y el bajo acoplamiento que ayudan a reducir

la dependencia entre las otras clases y demás capas de la arquitectura, además de maximizar la flexibilidad de la aplicación. (9)

2.1.2 Capa de Negocios

La capa de Negocio es la parte de un sistema donde reside la lógica de la aplicación y normalmente las integraciones con otros servicios. La capa de negocio recibe peticiones de la capa de servicios, procesa la lógica de negocio basada en las peticiones, y media en los accesos a los recursos de la capa de persistencia.

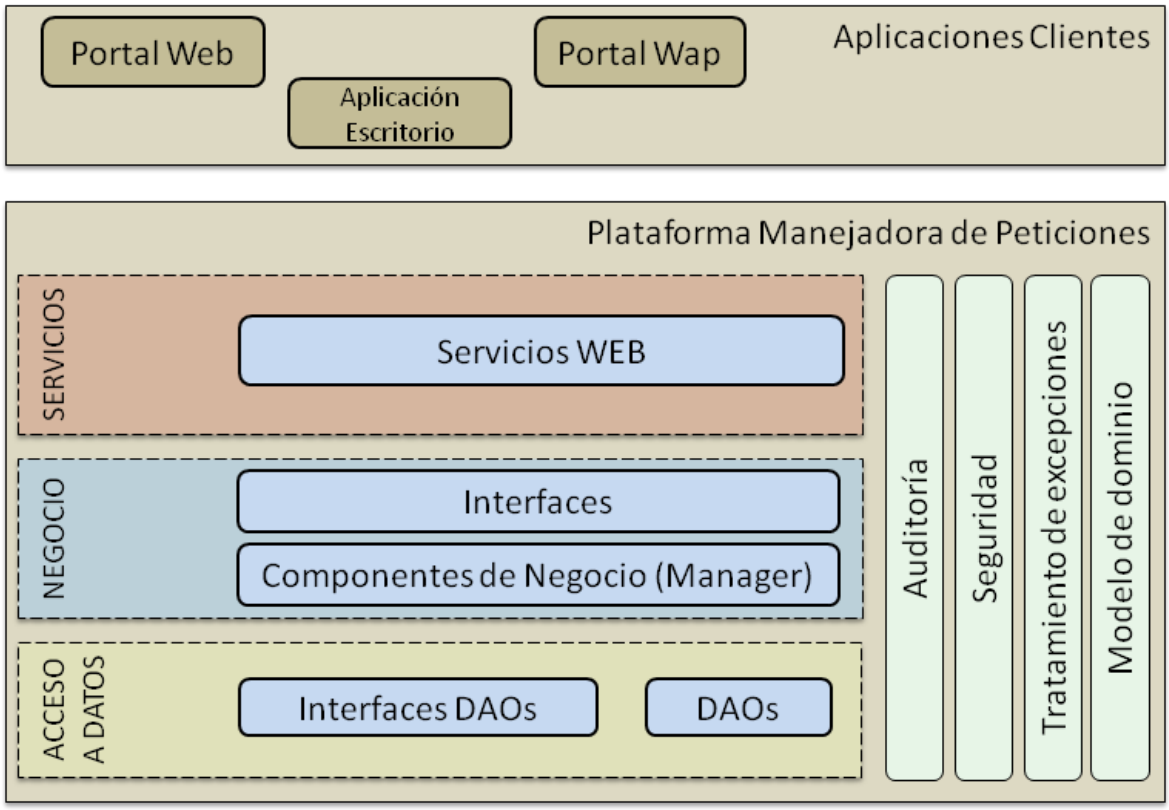

Figura 2: Estructura de la Plataforma Manejadora de Peticiones.

2.2 Modelo de Dominio

Teniendo en cuenta la necesidad de separar y representar los conceptos más importantes relacionados con la realidad física del sistema se propone realizar un Modelo de Dominio (Figura 3), esto se realiza si se determina que los procesos del negocio no están lo suficientemente definidos, como es el caso del desarrollo de la Plataforma Manejadora de Peticiones, esto permite mostrar de manera visual los principales conceptos que se manejan en el dominio de la plataforma y de esta forma utilizar un vocabulario común que ayude a usuarios, clientes, desarrolladores e interesados a entender el contexto en que se ubica el sistema, logrando una captura correcta de requisitos. (16)

2.2.1 Conceptos del Modelo del Dominio

Registrador: Define a la persona con privilegios para gestionar toda la información de los subscriptores, registrar solicitudes de subscripción, modificar la información de los subscriptores así como eliminar un subscriptor que quiera prescindir del servicio ENUM.

Subscripción: Define el proceso de que un usuario esté subscripto o no al servicio ENUM de Usuario.

Subscriptor: Define a aquella persona que tiene que estar subscrito al servicio ENUM.

Usuario: Define a aquella persona que solicita estar subscrito al servicio ENUM.

Contacto: Define la información relacionada con los servicios que posee un usuario con servicio ENUM, como pueden ser correo electrónico, beeper, teléfono, página web, SMS, etc. O sea las formas de establecer contacto o comunicación con dicho usuario.

Tipo de contacto: Constituye un codificador para validar los contactos.

Servicio ENUM: Define el servicio de telecomunicaciones que le asigna a cada persona un id personal para las telecomunicaciones.

Identificador: Define un número que se le asigna a cada persona y que engloba todos sus contactos.

Los conceptos que anteriormente se precisaron son los que se relacionan entre sí conformando la naturaleza del problema a analizar. A continuación se muestra el Modelo de Dominio que se definió:

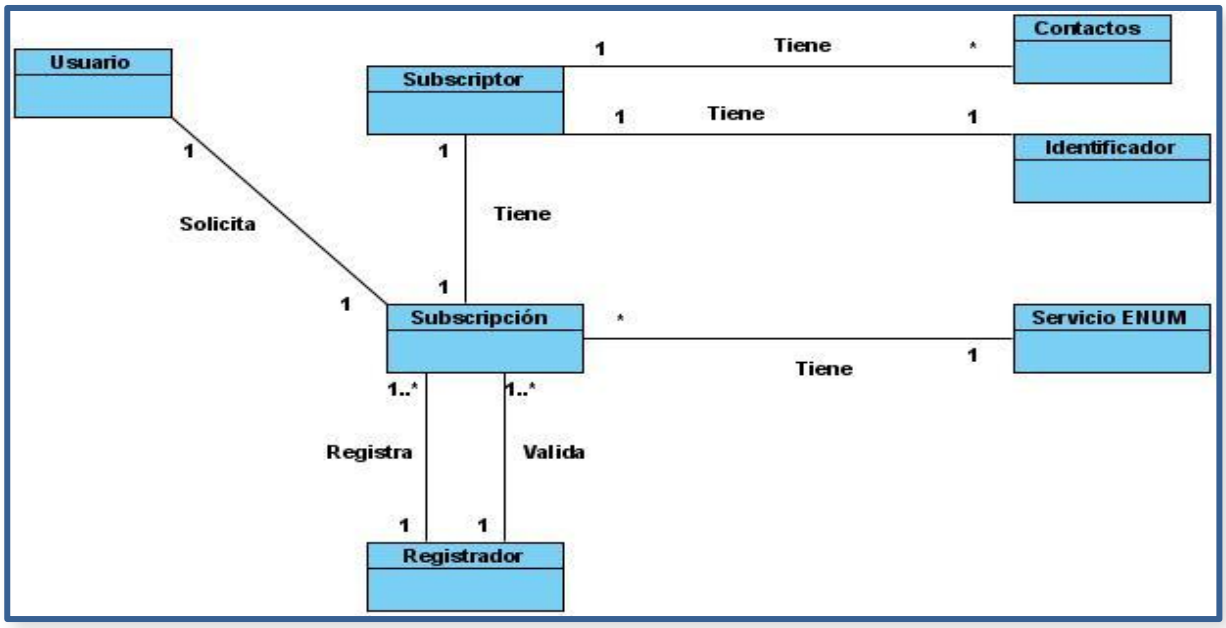

Figura 3: Modelo de Dominio.

2.3 Requisitos Funcionales

El propósito fundamental de la descripción de los requisitos funcionales es guiar el desarrollo hacia el sistema correcto. Los requerimientos funcionales definen las funciones que el sistema debe ser capaz de realizar, son el punto de partida para identificar qué debe hacer el sistema. Deben comprenderlo tanto los desarrolladores como los usuarios. (16)

Los requerimientos funcionales que debe cumplir el sistema se listan a continuación:

RF1 Autenticar usuario ENUM: Permite a los usuarios ENUM autenticarse introduciendo sus credenciales, verificando automáticamente la veracidad de los datos y asignándoles los permisos correspondientes según corresponda a sus privilegios.

RF2 Gestionar contraseña: Permite que los usuarios cambien su contraseña, o la recuperen en caso de olvido de la misma, mediante una palabra clave.

RF2.1: Cambiar contraseña.

RF2.2: Solicitar una nueva contraseña.

RF3 Actualizar el orden de los contactos: Permite actualizar el orden en que aparecerán los contactos.

RF4 Actualizar preferencia de los contactos: Permite al subscriptor cambiar la preferencia de aparición de sus contactos.

RF5 Buscar contactos del subscriptor: Permite a los usuarios realizar una búsqueda de los contactos de un subscriptor dado su número TIP.

RF6 Buscar datos personales del subscriptor: Permite realizar la búsqueda de los datos personales de un subscriptor, esto datos personales son: Número TIP, Nombre, Primer Apellido, Segundo Apellido. Esta búsqueda se puede realizar por los siguientes criterios; número TIP, Nombre, Apellidos, Provincia y Municipio.

RF7 Planificar contactos: Permite al subscriptor planificar sus contactos, estableciendo el intervalo de tiempo en el que desee que cierto contacto no esté visible en el servidor DNS.

RF8 Gestionar subscripción: Permite a los registradores adicionar una subscripción, eliminarla, cambiar el estado de la misma y actualizar los datos de un subscriptor asociado a una subscripción.

RF8.1: Registrar subscripción.

RF8.2: Actualizar subscripción.

RF8.3: Eliminar subscripción.

RF9 Gestionar contactos personales: Permite al registrador adicionar, actualizar y eliminar los contactos a los subscriptores, en caso de que los mismos lo soliciten.

RF9.1: Insertar nuevo contacto.

RF9.2: Modificar contacto.

RF9.3: Eliminar contacto.

RF10 Generar Reportes: Permite generar reportes de la cantidad de consultas realizadas a los contactos de un subscriptor.

RF11 Gestionar tipos de contactos: Permite al administrador de la Plataforma Manejadora de peticiones adicionar, modificar o eliminar tipos de contactos.

RF11.1: Insertar nuevo tipo de contacto.

RF11.2: Modificar tipo de contacto.

RF11.3: Eliminar tipo de contacto.

RF12 Gestionar oficina comercial: Permite al administrador de la Plataforma Manejadora de peticiones adicionar, modificar o eliminar oficinas comerciales por provincias y municipios.

RF12.1: Insertar nueva oficina comercial.

RF12.2: Modificar oficina comercial.

RF12.3: Eliminar oficina comercial.

RF13 Gestionar auditor de administradores de la PMP: Permite al administrador de la Plataforma Manejadora de Peticiones adicionar, actualizar y eliminar a otros administradores de la plataforma, así como a usuarios con el rol de auditor que son los encargados de generar los reportes de la plataforma.

2.4 Requisitos no Funcionales

Los requisitos no funcionales especifican propiedades y restricciones del sistema, como restricciones de entorno o de la implementación, rendimiento, dependencia de la plataforma, facilidad de mantenimiento, extensibilidad y fiabilidad, entre otros que se mencionarán posteriormente. (16)

2.4.1 Usabilidad

- ✓ Para interactuar con la plataforma se necesitan conocimientos intermedios de programación y manejo de servicios web.
- ✓ Se necesita de un período de 5 días de capacitación para comenzar a interactuar con la plataforma.

2.4.2 Seguridad

- ✓ Los datos manejados mediante los servicios web expuestos por la plataforma constituyen datos generalmente personales y de vital importancia para el desarrollo del servicio ENUM de Usuarios en Cuba, por lo que se debe garantizar la seguridad y restringir el acceso a estos servicios web.

2.4.3 Fiabilidad

- ✓ El sistema tiene que estar disponible todo el tiempo, por gestionar servicios de telecomunicaciones y ser encuestado en todo momento.
- ✓ Tiempo medio de reparación: 1 hora.
- ✓ Toda la información que gestiona el sistema es de carácter personal y está debidamente verificada por lo que debe de ser exacta y real.
- ✓ Se identifican 2 tipos de errores significativos: errores menores (tiempo excedido, error en algún parámetro) errores críticos (pérdida de conexión con la BD, DNS).
- ✓ Tiempo de respuesta por transición promedio: 5 seg, max 50 seg.

2.4.4 Portabilidad

- ✓ El sistema deberá correr sobre plataforma Windows/Linux.

2.4.5 Soporte

- ✓ Se debe de generar un documento detallado que explique el funcionamiento del sistema.

2.4.6 Requisitos para la documentación de usuarios en línea y ayuda del sistema

- ✓ Se debe documentar cada una de las funcionalidades del sistema.
- ✓ Documentación del código, siguiendo el formato especificado en el estándar de codificación.
- ✓ Se debe brindar una descripción con cada uno de los servicios web, especificando en cada caso la funcionalidad del servicio y los parámetros de entrada y salida de los mismos.

2.4.7 Interfaces de Comunicación

- ✓ La comunicación con el sistema se hará a través del protocolo SOAP[16].
- ✓ La comunicación con la BD y los DNS a través del conjunto de protocolos que conforman TCP/IP[17].

[16] **SOAP**: Acrónimo de Simple Object Access Protocol (Protocolo Simple de Acceso a Objetos) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

2.4.8 Hardware

- ✓ Los servidores donde se alojará la aplicación, servidor de base de datos y servidor DNS deben cumplir los siguientes requerimientos:
  - Tarjeta de red Gigabit.
  - 1 GB de RAM o superior.
  - Procesador 3.0 GHz o superior.

2.4.9 Software

- ✓ Para el servidor donde se desplegará la Plataforma Manejadora de Peticiones:
  - Máquina Virtual de Java versión 1.6.
  - Servidor web Jakarta Apache Tomcat versión 6.0.
- ✓ Para los servidores de Base de Datos y DNS:
  - Sistema Gestor de Base de Datos PostgreSQL versión 8.3.

2.4.10 Estándares Aplicables

- ✓ El estándar de codificación se realizó basado en el estándar de SUM Microsystem.

2.5 Principales Casos de Usos de la Plataforma Manejadora de Peticiones

En esta sección se representan los casos de usos arquitectónicamente significativos (Figura 4) así como un breve resumen de cada uno de ellos. Es necesario aclarar que en la *Plataforma Manejadora de Peticiones* los casos de usos no contarán con un interfaz visual pues serán expuestos como servicios web.

---

[17] **TCP/IP**: Acrónimo de Transmission Control Protocol/Internet Protocol (Protocolo de Control de Transmisión / Protocolo Internet). Es un conjunto de protocolos de red en la que se basa Internet y que permiten la transmisión de datos entre redes de computadoras.

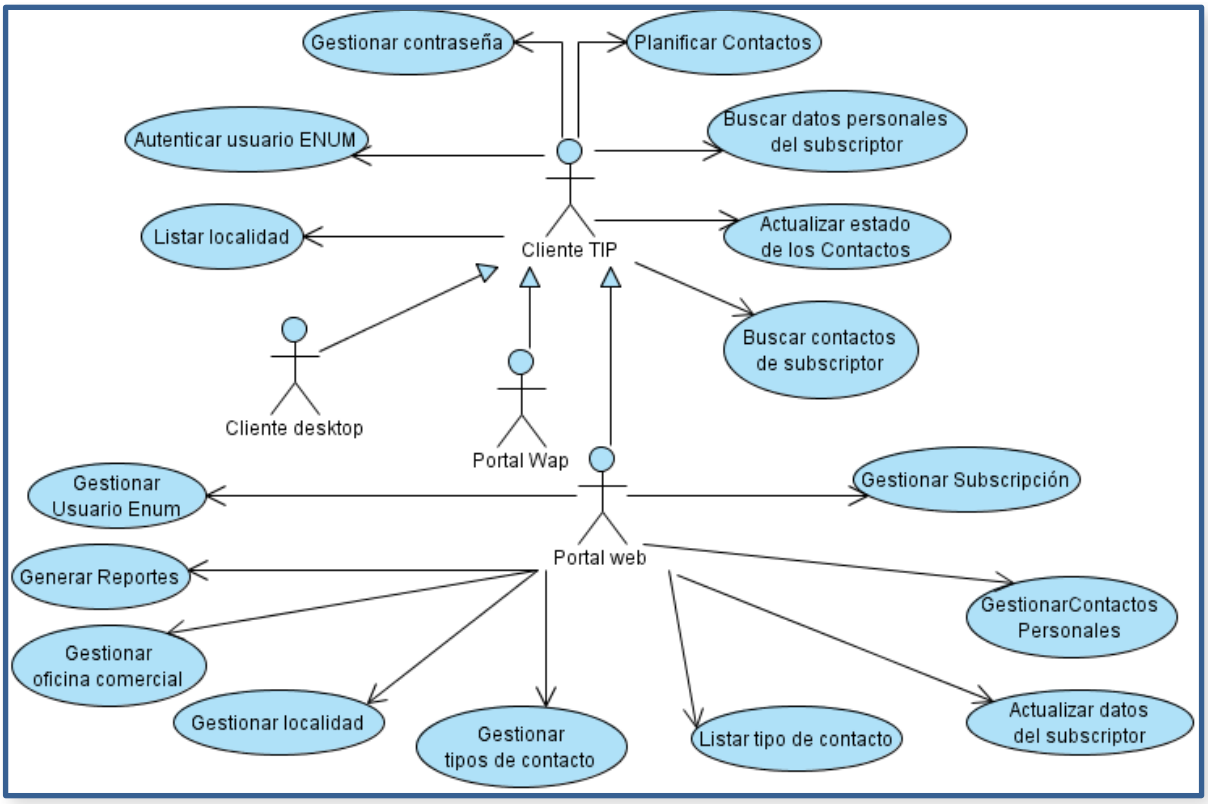

Figura 4: Vista de Casos de Uso de la Plataforma Manejadora de Peticiones.

- ✓ ***Gestionar usuario ENUM:*** El caso de uso permite a las aplicaciones clientes adicionar, actualizar y eliminar a los usuarios ENUM. Para adicionar y actualizar un usuario se debe enviar a la plataforma los datos del usuario en cuestión; y para eliminarlo, se requiere el identificador del mismo.
- ✓ ***Gestionar contraseña:*** El caso de uso permite a las aplicaciones clientes que los usuarios cambien su contraseña o la recuperen en caso de olvido de la misma. Para modificar la contraseña se debe enviar el usuario, la contraseña actual y la nueva contraseña a la plataforma. Para recuperar la contraseña la aplicación cliente debe enviar el usuario que desea realizar dicha acción, su pregunta de seguridad y la respuesta de la misma; si los datos son válidos la plataforma retorna una nueva contraseña, en caso contrario, un mensaje de error.
- ✓ ***Gestionar oficina comercial:*** El caso de uso le permite a las aplicaciones clientes adicionar, modificar o eliminar oficinas comerciales. Para adicionar una nueva oficina se debe enviar a la

plataforma el identificador del municipio al cual pertenece y la oficina; para actualizarla es necesaria la oficina, y para eliminarla el identificador de la misma.

- ✓ ***Gestionar localidad:*** El caso de uso permite a las aplicaciones clientes adicionar, modificar o eliminar los municipios, provincias y países. Para adicionar o modificar un país se debe enviar a la plataforma el país en cuestión. Para adicionar una provincia se enviará el identificador del país al cual pertenece y la provincia; para modificarla se enviará la provincia. Para adicionar un municipio se requiere el identificador de la provincia a la cual pertenece y el municipio; para modificarlo se enviará el municipio. En el caso de la eliminación solamente se requiere el identificador del país, municipio o provincia a eliminar.
- ✓ ***Gestionar subscripción:*** El caso de uso permite a las aplicaciones clientes adicionar una subscripción, eliminarla y cambiar el estado de la misma. Para adicionar una subscripción se debe enviar a la plataforma el usuario del registrador y los datos del subscriptor requeridos. Para modificar el estado de una subscripción se enviará el identificador de la subscripción, el usuario del registrador y el nuevo estado; y en caso que se desee eliminarla, el identificador de la subscripción y el usuario del registrador. En cualquiera de los tres casos, se enviará a la plataforma la descripción de la acción realizada (opcional).
- ✓ ***Gestionar contactos personales:*** El caso de uso permite a las aplicaciones clientes adicionar, actualizar y eliminar los contactos a los subscriptores. En cualquiera de los casos se debe enviar a la plataforma el número TIP del subscriptor y el contacto en cuestión.
- ✓ ***Gestionar tipos de contacto:*** El caso de uso le permite a las aplicaciones clientes adicionar, modificar o eliminar tipos de contactos. Para adicionar un nuevo contacto se debe enviar a la plataforma el número TIP del subscriptor y el contacto, para actualizarlo es necesario el contacto, y para eliminarlo el identificador del mismo.
- ✓ ***Autenticar usuario ENUM:*** El caso de uso permite a las aplicaciones clientes autenticar a los Usuarios ENUM; para ello se debe enviar el usuario y la contraseña a la plataforma, donde se realiza el proceso de autenticación, devolviéndose a las aplicaciones clientes los permisos asociados al usuario en caso de que la autenticación haya sido exitosa y un mensaje de error en caso contrario.

- ✓ ***Planificar contactos:*** El caso de uso permite a las aplicaciones clientes realizar la planificación de los contactos de un subscriptor. Para ello se debe enviar a la plataforma el número TIP del subscritor, el identificador del contacto y los intervalos de tiempo en que desee que dicho contacto no esté visible en el servidor DNS.
- ✓ ***Actualizar estado de los contactos:*** El caso de uso permite a las aplicaciones clientes modificar la visibilidad o la preferencia de los contactos de un subscriptor. Para esto se debe enviar a la plataforma el número TIP del subscritor, el identificador del contacto y la visibilidad o preferencia del mismo, en dependencia de la operación que se desee realizar.
- ✓ ***Actualizar datos del subscriptor:*** El caso de uso permite a las aplicaciones clientes actualizar los datos personales de un subscriptor. Para ello, se debe enviar a la plataforma un subscriptor con los datos actualizados.
- ✓ ***Buscar datos personales del subscriptor:*** El caso de uso permite realizar la búsqueda de los datos personales de un subscriptor, estos datos pueden ser: Número TIP, Nombre, Primer Apellido, Segundo Apellido, Municipio, Provincia. Esta búsqueda se puede realizar por los siguientes criterios; número TIP, Nombre, Apellidos, Provincia y Municipio.
- ✓ ***Buscar contactos del subscriptor:*** El caso de uso permite a las aplicaciones clientes realizar una búsqueda de los contactos de un subscriptor dado su número TIP. Estos contactos pueden ser correo electrónico, Beeper, Teléfono, página web, SMS, etc., o sea las formas de establecer contacto o comunicación con dicho subscriptor.
- ✓ ***Listar localidad:*** El caso de uso permite a las aplicaciones clientes listar los municipios, provincias y países. Para listar los municipios se debe enviar a la plataforma el identificador de la provincia cuyo listado de municipios se requiere. En cualquiera de los otros dos casos se hace necesario el envío de ninguna información.
- ✓ ***Listar tipos de contactos:*** El caso de uso permite a las aplicaciones clientes listar los tipos de contactos que existen.

2.6 Reglas de Negocio

Las reglas de negocio describen políticas que deben cumplirse o condiciones que deben satisfacerse para alcanzar los objetivos misionales, por lo que regulan algún aspecto del negocio.

En el sistema se han definido varias reglas de negocio:

2.6.1 Reglas de estructura

- ✓ Modelo de datos:
  - Un subscriptor solo podrá contar con un solo identificador personal en su servicio ENUM.
  - Para llenar la solicitud de subscripción la persona debe aportar al menos un contacto.
- ✓ Relación:
  - La persona solicita el servicio ENUM. Para esto aporta su(s) contactos.
  - Cuando se aprueba su servicio ENUM le es otorgado un número ENUM.

2.6.2 Reglas de derivación

- ✓ Cuando una persona solicita el servicio ENUM debe llenar una solicitud de subscripción y se convierte en un usuario.
- ✓ Cuando a un usuario se le aprueba la solicitud de subscripción se convierte en un subscriptor.

2.6.3 Reglas de acción

- ✓ Flujo:
  - Para realizar cualquier cambio en su solicitud de subscripción el usuario deberá llenar una solicitud de actualización.
  - La persona que aprueba, modifica y/o elimina las solicitudes de subscripción y de actualización es el registrador.
  - El registrador tendrá acceso de forma exclusiva a la base de datos Directorio ENUM.
  - Una vez hecha efectiva la subscripción al servicio el subscriptor tiene la posibilidad de modificar su información, en este caso solo se podrá modificar el identificador personal asignado y la información personal registrada, así como eliminar su subscripción en el servicio.
  - Para adicionar un nuevo contacto relacionado por algún subscriptor se debe primeramente registrar una solicitud, en caso de que el proceso de validación de la información

suministrada por el subscriptor sea válido se procede a realizar dicha funcionalidad, este mismo proceso se realiza igualmente para modificar algún contacto, en el caso de que la petición realizada sea la de eliminar un contacto se procede en el momento.

2.7 Patrones de diseño utilizados

Los patrones para el desarrollo de software son uno de los últimos avances de la Tecnología Orientada a Objetos, representando una forma de resolver problemas de ingeniería del software aportando soluciones de manera elegante y efectiva. (17)

Por otra parte, las metodologías Orientadas a Objetos tienen como uno de sus principios “no reinventar la rueda” para la resolución de diferentes problemas. Por lo tanto los patrones se convierten en una parte muy importante para poder conseguir la reutilización. (17)

En la solución propuesta se utilizaron patrones como:

Singleton (Instancia única): Garantiza que solamente se crea una instancia de la clase y provee un punto de acceso global a él. Todos los objetos que utilizan una instancia de esa clase usan la misma instancia.

DAO (Data Acces Object): encapsula la lógica de acceso a datos y proporcionar una interfaz consistente a los componentes de la Capa de Negocio, separando estas dos lógicas para una mayor reutilización. (18) Este patrón es referenciado en el Capítulo 3.

Conclusiones

A partir del análisis de los requisitos funcionales y la descripción de lo que se desea implementar, se está en condiciones de desarrollar un sistema con calidad que satisfaga al cliente. Para su puesta en práctica se utilizará las pautas de codificación establecidas para una mejor organización y entendimiento del producto, llegando a la conclusión de las grandes ventajas que la Plataforma Manejadora de Peticiones brinda al proyecto *TeleIdentificador Personal* como intermediaria de todas las aplicaciones que se implementan paralelamente a este producto.

Capítulo 3

Elementos y Estructura de las capas de Persistencia y Negocio

En capítulos anteriores se expuso un grupo de herramientas, tecnologías y conceptos necesarios para realizar el análisis y descripción de la solución propuesta al problema planteado en el diseño teórico de la investigación.

Para ello será objetivo de este capítulo describir la estructura de las capas de Acceso a Datos y Negocio y las principales configuraciones necesarias para implementar la Plataforma Manejadora de Peticiones.

Dicha implementación se presenta a partir de los diagramas de clases que implementan las funcionalidades de los Casos de Usos mencionados en el capítulo pasado, además del modelo de datos correspondiente a las entidades persistentes.

3.1 Elementos y Estructura de la Capa de Persistencia

La Capa de Persistencia o de Acceso a Datos encapsula el conjunto de clases y componentes responsables de almacenar y recuperar datos, incluyendo la integración directa de la aplicación con la base de datos; una capa de persistencia es la base de la arquitectura en capas. Su implementación estará dividida en varios paquetes (Figura 5) para su mejor utilización y entendimiento.

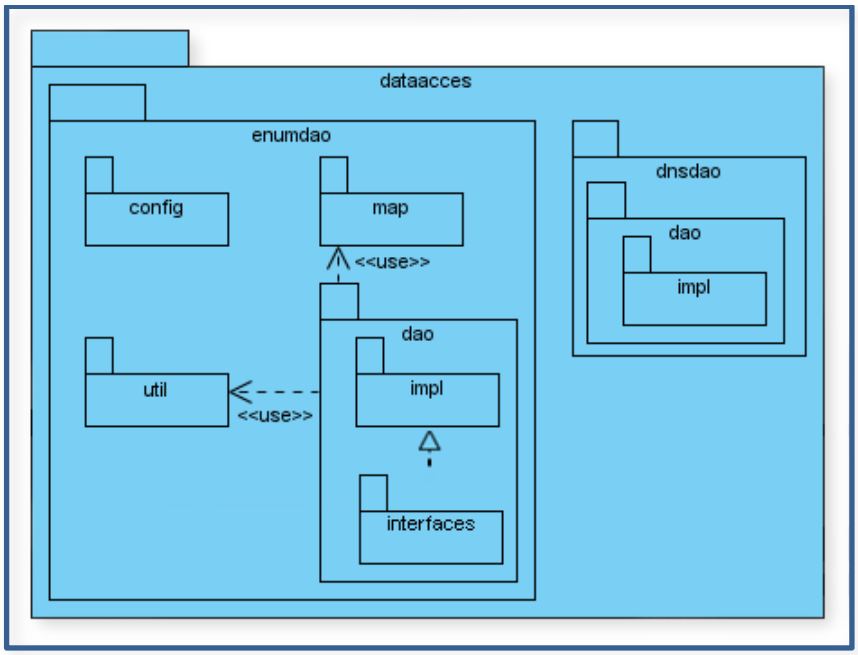

Figura 5: Diagrama de paquetes de la capa de persistencia.

Desde el entorno de desarrollo se observaría de la siguiente forma (Figura 6):

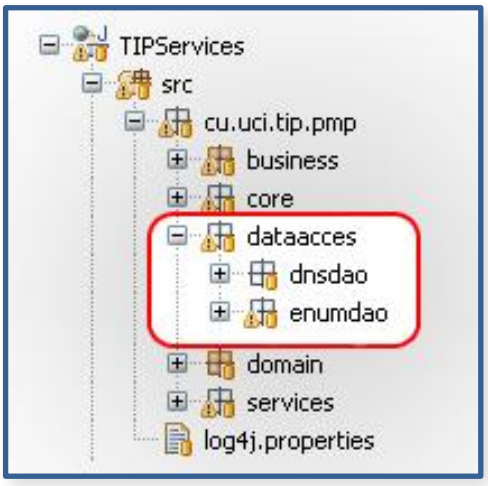

Figura 6: Estructura de la capa de persistencia desde el entorno de desarrollo.

Dentro de cada paquete se definen un grupo de clases y ficheros XML los cuales se detallan en los próximos epígrafes.

3.1.1 Paquete *cu.uci.tip.pmp.dataacces.dnsdao*

Este paquete es el encargado de encapsular un grupo de subpaquetes y clases para el trabajo con el DNS, específicamente para el trabajo con los contactos de los subscriptores. Aunque en el paquete *enumdao* se trabaja también con esos contactos, es necesaria la interacción con el DNS, puesto que los contactos cuando pasan al estado de activo u oculto se transfieren automáticamente a la base de datos del DNS.

Interfaces	Implementaciones
IBIND.java	BIND.java

Tabla 2: Clases de acceso a datos para el trabajo con la base de datos del DNS

3.1.2 Paquete *cu.uci.tip.pmp.dataacces.enumdao*

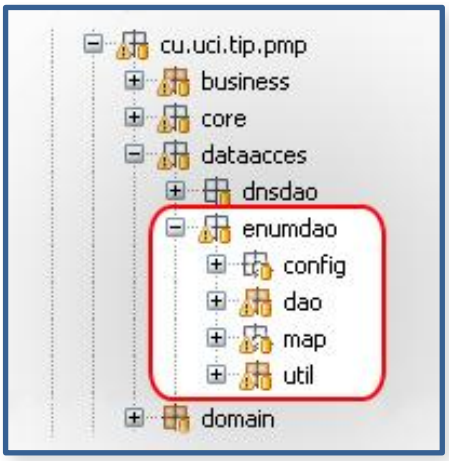

Figura 7: Estructura del paquete *enumdao* desde el entorno de desarrollo.

Este paquete es el encargado de encapsular un grupo de subpaquetes en los cuales se encuentran las interfaces de las clases de acceso a datos o DAO (Data Access Object) y las implementaciones concretas de las mismas. Dentro de él se encuentran también las configuraciones necesarias para la conexión al Sistema Gestor de Base de Datos, los archivos de mapeo necesarios para interactuar con los objetos del dominio y un paquete donde se agrupan algunas clases de utilidad para la capa de persistencia.

3.1.3 Paquete *cu.uci.tip.pmp.dataacces.enumdao.util*

Dentro de este paquete se definen la clase IbatisUtil (Tabla 3) la cual utiliza el soporte que proporciona Spring para el trabajo con iBatis en el paquete **`org.springframework.orm.ibatis`** mediante el cual se expone las funcionalidades necesarias para el manejo de los objetos persistentes y transitorios.

Usando el contenedor de Inversion de Control que brinda Spring se logra crear una instancia única de la clase IbatisUtil, la cual será utilizada por los objetos de acceso a datos de la aplicación.

Nombre:	`public class` **`IbatisUtil`** `implements` **`InitializingBean`**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**`private sqlMapClientTemplate`**: **`SqlMapClientTemplate = null`**
Descripción:	*Contiene la instancia del* ***`SqlMapClientTemplate`*** *de Spring.*
Métodos:	
Nombre:	**`private IbatisUtil`** ()
Descripción:	*Constructor de la clase.*
Nombre:	**`public setSqlMapClientTemplate`** (**`sqlMapClientTemplate`**: **`SqlMapClientTemplate`**) : **`void`**
Descripción:	*Setearle el* ***`sqlMapClientTemplate`*** *al IbatisUtil.*
Nombre:	`public int` **`delete(String statementName, Object parameterObject)`**
Descripción:	*Elimina la instancia persistente dada.*
Nombre:	`public` **`Object insert(String statementName, Object parameterObject)`**
Descripción:	*Persiste la instancia transitoria dada.*
Nombre:	**`public`** `List` **`queryForList`**`(String statementName, Object parameterObject)`
Descripción:	*Obtiene más de un resultado de la ejecución de una consulta a la base de datos.*
Nombre:	`public` **`Object queryForObject(String statementName, Object parameterObject)`**
Descripción:	*Obtiene un solo resultado de la ejecución de una consulta a la base de datos.*
Nombre:	`public int` **`update(String statementName, Object parameterObject)`**
Descripción:	*Modifica la instancia persistente dada.*

Tabla 3: Descripción de la clase IbatisUtil.

3.1.4 Paquete *cu.uci.tip.pmp.dataacces.enumdao.config*

Como iBatis está orientado a operar con múltiples entornos, es necesario setearle un grupo de configuraciones para que se integre perfectamente la aplicación al gestor de base de datos PostgreSQL. Este paquete es el encargado de contener toda la configuración necesaria para la conexión de la capa de persistencia con el Sistema Gestor de Base de Datos.

El archivo **`ibatis.properties`** servirá para configurar dicha conexión.

```
ibatis.connection.url=jdbc:postgresql://10.7.20.3:5901/#####
ibatis.connection.driverClassName=org.postgresql.Driver
ibatis.connection.username=####
ibatis.connection.password=####

ibatis.dbcp.initialSize=10
ibatis.dbcp.maxActive=1000
ibatis.dbcp.maxIdle=100
ibatis.dbcp.minIdle=10
ibatis.dbcp.maxWait=60
```

Como se puede apreciar, se especifica la cadena de conexión donde se indica la identificación del equipo en el cual se encuentra el servidor de base de datos y el nombre a la cual se hará la conexión, el driver para la conexión (JDBC) encargado de enlazar la aplicación con el gestor, incluyendo la autenticación de acceso al servidor. También contiene las propiedades para el pool de conexiones, inicialmente crea 10 conexiones y teniendo la posibilidad de asignar 1000 conexiones activas concurrentemente, incluyendo las conexiones perezosas que pueden ir de 10 hasta 100 esperando 60 segundos para lanzar una excepción si no existen conexiones para asignar. Todas estas propiedades pueden ser gestionadas una vez desplegada la aplicación.

La ventaja de tener estas propiedades en el **`.properties`** facilita el trabajo con la capa de Acceso a Datos al tenerlas todas en un mismo archivo de configuración y no modificar el archivo **`dataacces-config.xml`** para evitar futuros errores en la aplicación y así lograr que la aplicación sea más portable y escalable.

Lo fundamental para configurar iBatis es el archivo de configuración de mapeos SQL (**`sqlMapConfig.xml`**).

```
<sqlMapConfig >
      <settings
            cacheModelsEnabled="true"
            useStatementNamespaces="true"
      />
      <!-- configuración de los elementos <sqlMap> -->

</sqlMapConfig>
```

Cada elemento **<settings>** establece un grupo de configuraciones mediante la inserción de los atributos al elemento.

En este caso se han especificado los atributos:

- ✓ **cacheModelsEnabled**: Caching es una técnica para mejorar el rendimiento de la aplicación permitiendo que se mantenga en memoria un grupo de datos utilizado recientemente suponiendo que será necesario consultarlo en un futuro próximo. El **cacheModelsEnabled** se emplea para indicar sí iBatis desea utilizar el almacenamiento en caché o no.
- ✓ **useStatementNamespaces**: se emplea para indicarle a iBatis que se utiliza los nombres de mapeos para llamar a las declaraciones de las consultas SQL, facilitando que se puedan tener consultas con igual nombre en archivos de mapeos diferentes sin que se solapen.

El archivo **dataaccess-config.xml** es el encargado de configurar completamente el acceso a datos de la PMP utilizando las propiedades definidas en el **ibatis.properties** anteriormente expuesto. Este archivo está compuesto por un grupo de propiedades y beans detallados a continuación:

propertyConfigurer: Este bean permite leer el archivo **.properties** desde el classpath para luego darle la configuración necesaria a iBatis.

```
<bean id="propertyConfigurer"
       class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
              <property name="location"
                 value="cu/uci/tip/pmp/dataacces/enumdao/config/ibatis.properties" />

</bean>
```

datasource: Independientemente del framework de persistencia que se utilice, es necesario configurar una referencia a un datasource (fuente de datos). Spring provee varias opciones para configurar los beans de fuente de datos, uno de los cuales es el del pool de conexiones[18].

[18] **Pool de conexiones**: Se le denomina al manejo de una colección de conexiones abiertas a una base de datos de manera que puedan ser reutilizadas al realizar múltiples consultas o actualizaciones.

Jakarta Commons Database Connection Pools (DBCP) incluye múltiples orígenes de datos los cuales provén el pool de conexiones, él utilizado en este caso es el *BasicDataSource* muy simple de configurar en Spring.

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
                                                        destroy-method="close">
            <!--se setean un grupo de propiedades para realizar la conexión al Gestor
            de Base de Datos obtenidas del ibatis.properties-->
</bean>
```

sqlMapClientTemplate: El framework Spring implementa una plantilla para los objetos de acceso a datos que se crean en iBatis, lo que permite crear una clase, en este caso IbatisUtil, que utilice todas las funcionalidades que brinda la plantilla de Spring para luego ampliar y facilitar la implementación de los DAOs.

Esta plantilla necesita que se le indiquen varias configuraciones, esto se realiza mediante la propiedad *sqlMapClient*. De esta manera se le setean los archivos de mapeos necesarios para cargar el contexto de la capa de persistencia y el traductor de excepciones, encargado de traducir las excepciones JDBC a las de Spring ORM.

```
<bean id="sqlMapClientTemplate"
            class="org.springframework.orm.ibatis.SqlMapClientTemplate">
      <property name="sqlMapClient">
            ...
      </property>
      <property name="exceptionTranslator">
            ...
      </property>
</bean>
```

ibatisUtil: Este bean es el encargado de instanciar la clase IbatisUtil si no ha sido creada.

```
<bean id="ibatisUtil"
            class="cu.uci.tip.pmp.dataacces.enumdao.util.IbatisUtil"
            factory-method="getInstance"
            depends-on="sqlMapClientTemplate">
      <property name="sqlMapClientTemplate" ref="sqlMapClientTemplate" />
</bean>
```

Con esta configuración declarada, la Plataforma Manejadora de Peticiones está lista para comenzar a persistir y recuperar objetos desde la base de datos.

3.1.5 Paquete *cu.uci.tip.pmp.dataacces.enumdao.map*

El almacenamiento y recuperación de los datos es solucionada mediante el mapeo de las clases persistentes que proporciona el framework iBatis. Este paquete es el encargado de contener todos los archivos de mapeo correspondientes a cada una de las clases persistentes u objetos de negocio (Figura 8).

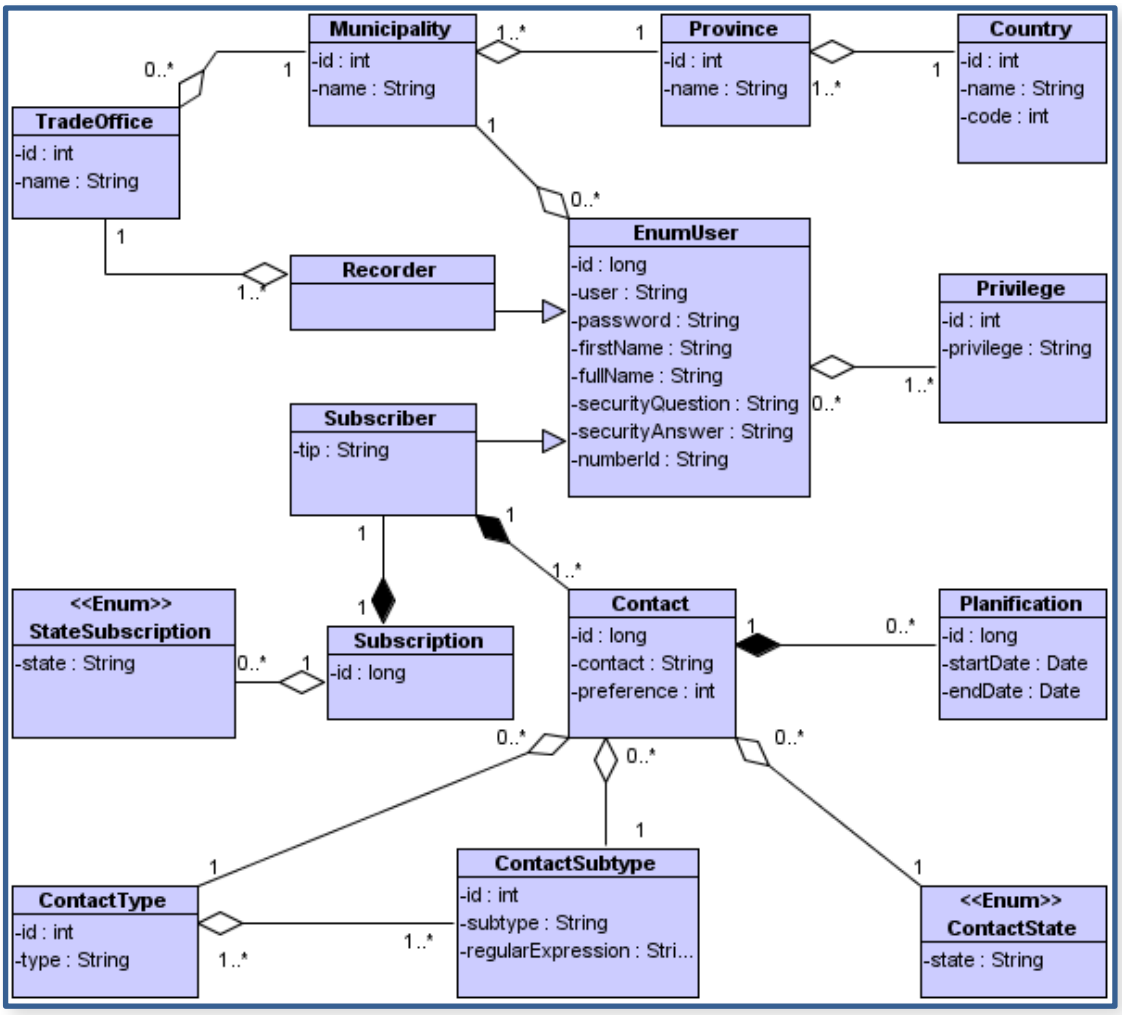

Figura 8: Diagrama de clases persistentes.

En los archivos **`.xml`** correspondientes se declaran los **`<resultMap>`** que no son más que las propiedades que vamos a obtener cuando se ejecute una consulta SQL. Cada archivo de mapeo puede contener varios **`<resultMap>`** en dependencia de los resultados de las consultas *select* que se tengan. En este fichero se contendrá también las consultas SQL que se ejecutarán como parte del acceso a datos.

3.1.6 Paquete *cu.uci.tip.pmp.dataacces.enumdao.dao*

Muchas de las aplicaciones J2EE[19] necesitan consultar datos persistentes en algún momento. En diferentes aplicaciones, el almacenamiento se lleva a cabo con diferentes mecanismos, y hay marcadas diferencias en las APIs utilizadas para acceder a los diferentes Sistemas Gestores de Bases de Datos (SGBD). Algunas pueden utilizar el API de JDBC para acceder a datos que residen en un SGBD relacional (SGBDR). Este API permite el acceso y manipulación de datos mediante el uso de consultas SQL, que es el estándar para acceder a las tablas de las bases de datos relacionales. (20)

Pero si se desea utilizar objetos que encapsulen todo el acceso a las fuentes de datos y maneje las conexiones con el SGBD, lo más conveniente en el entorno de desarrollo es la utilización del patrón DAO (Data Acces Object).

El objeto de acceso a datos (también conocido simplemente como DAO), implementa todo el código requerido para interactuar con la fuente de datos. Independientemente de qué tipo de SGBD se utilice, el DAO siempre proporciona una API uniforme al desarrollador. La capa de negocio utiliza las interfaces implementadas por los DAOs y este oculta completamente los detalles de la implementación a las capas superioras. Debido a que las interfaces no cambian cuando se modifica la implementación de los DAOs, esto permite migrar una aplicación con un SGBD específico a otra sin alterar la implementación de las capas superiores. Esencialmente, el DAO actúa como un adaptador entre las diferentes capas y la fuente de datos. (20)

Alguna de las ventajas que brinda este patrón son: (20)

- ✓ *Transparencia*: La capa de negocio puede utilizar la base de datos sin conocer los detalles específicos de su implementación. El acceso es transparente porque los detalles de implementación se ocultan dentro del DAO.

[19] **J2EE:** Acrónimo de Java 2 Enterprise Edition. Plataforma de programación —parte de la Plataforma Java— para desarrollar y ejecutar software en lenguaje Java.

- ✓ *Migración*: Una capa de acceso a datos implementada por el patrón DAO, facilita migrar de una base de datos a otra. La capa de negocio no tiene ningún conocimiento de la implementación del acceso a datos. Así, la migración implicaría cambios sólo en la capa de acceso a datos. Además, si se emplea una estrategia de fábrica de DAOs, es posible implementar una fábrica concreta para cada SGBD.
- ✓ *Complejidad del código en la capa de negocio*: Debido a que los DAOs gestionan todas las complejidades del acceso a datos, simplifica el código en el negocio y otros capas que necesiten del acceso a la base de datos. Todas las consultas SQL figuran en los DAO, en el caso de iBatis, en los **`.xml`** de configuración de cada entidad persistente y no en el negocio. Esto mejora la legibilidad de código y la productividad.
- ✓ *Centraliza todo el acceso a datos una capa separada*: Debido a que todas las operaciones de acceso a datos están delegadas a los DAOs, la capa de acceso a datos puede considerarse como la capa que aísla el resto de la aplicación del código de acceso a datos. Esta centralización hace que la aplicación sea más fácil de mantener y gestionar.

Dado estas ventajas se decidió utilizar este patrón, y el paquete *dao* es el encargado de encapsular toda la declaración e implementación de las clases del acceso a datos (Tabla 4) que se pueda requerir para manejar la persistencia de la aplicación.

Interfaces	Implementaciones
IBaseDao.java	BaseDao.java
ICountryDAO.java	CountryDAOImpl.java
IContactStateDAO.java	ContactStateDAOImpl.java
IRecorderDAO.java	RecorderDAOImpl.java
IEnumUserDAO.java	EnumUserDAOImpl.java
IContactTypeDAO.java	ContactTypeDAOImpl.java
IContactSubtypeDAO.java	ContactSubtypeDAOImpl.java
IMunicipalityDAO.java	MunicipalityDAOImpl.java
IContactDAO.java	ContactDAOImpl.java
ISubscriptionDAO.java	SubscriptionDAOImpl.java
ITradeOfficeDAO.java	TradeOfficeDAOImpl.java

IProvinceDAO.java	ProvinceDAOImpl.java
IUserPrivilegeDAO.java	UserPrivilegeDAOImpl.java
IRequestLogDAO.java	RequestLogDAOImpl.java
IPrivilegeDAO.java	PrivilegeDAOImpl.java
ISubscriberDAO.java	SubscriberDAOImpl.java
IPlanificationDAO.java	PlanificationDAOImpl.java

Tabla 4: Clases de Acceso a Datos y sus interfaces.

3.2 Elementos y Estructura de la Capa de Negocio

La Capa de Negocio es la responsable de implementar las reglas que rigen el comportamiento de la aplicación. Los métodos expuestos por los componentes de la Capa de Negocio son la única puerta de entrada posible a la aplicación, por lo que deben ofrecer toda la funcionalidad requerida por el sistema. Esta capa se comunica con la Capa de Presentación, para recibir las solicitudes y presentar los resultados, y con la Capa de Persistencia, para solicitar al gestor de base de datos almacenar o recuperar datos de él.

Para implementar las reglas de negocio, se han encapsulado las clases e interfaces dentro de diferentes paquetes.

El uso de paquetes aporta varias ventajas frente a la programación sin paquetes. En primer lugar, permite encapsular funcionalidad en unidades con un cierto grado de independencia, ocultando los detalles de implementación. De esta forma se pueden conseguir diseños (e implementaciones) más limpios y elegantes.

Por otra parte, se potencia la reutilización de las clases desarrolladas. Es posible definir interfaces de uso de cada paquete, para que otros paquetes o aplicaciones puedan utilizar la funcionalidad implementada.

Además, el uso de paquetes permite la reutilización de los nombres de las clases, ya que el espacio de nombres de un paquete es independiente del de otros paquetes. El lenguaje Java impone la restricción de que una clase debe tener un nombre único, dentro del paquete al cual pertenece. Sin embargo, es posible que dos clases tengan el mismo nombre, siempre y cuando pertenezcan a paquetes distintos.

En el sistema propuesto se han organizado los paquetes de la siguiente manera para su mejor utilización y entendimiento (Figura 9):

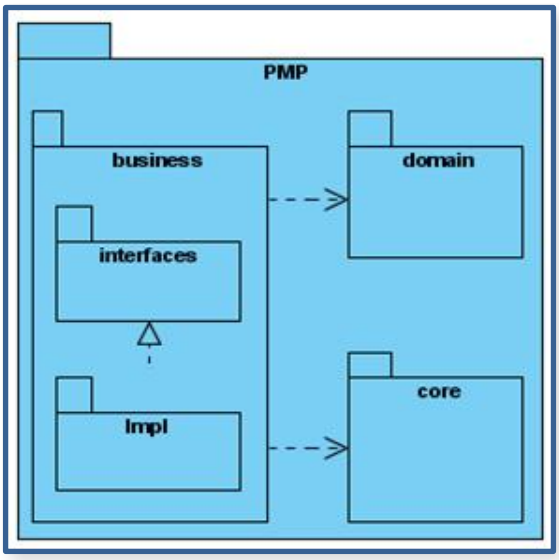

Figura 9: Diagrama de paquetes de la capa de negocio.

Desde el entorno de desarrollo se observaría de la siguiente manera (Figura 10):

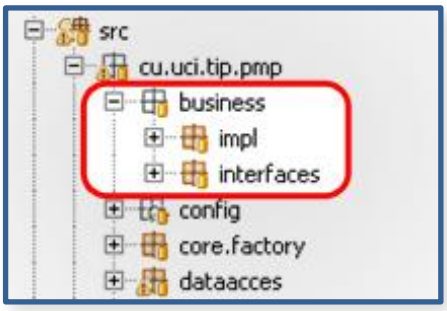

Figura 10: Estructura de la capa de negocio desde el entorno de desarrollo.

Dentro de cada paquete se definen un conjunto de clases e interfaces como beneficio de la infraestructura que aporta Spring Framework, los cuales se detallan en próximos epígrafes.

3.2.1 Paquete *business.interfaces*

Dentro de este paquete se definen un grupo de interfaces que permitirán ocultar los detalles de implementación, de tal forma que en caso de producirse una modificación en alguna de ellas, tenga la

mínima repercusión posible en el resto de las clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases, reduciendo así el acoplamiento, posibilitando que el código sea legible y más fácil de realizarle pruebas.

Interfaz IPasswordManager

Nombre:	`public interface IPasswordManager`
Descripción:	Contiene una colección de métodos relacionados con la gestión de la contraseña y la autenticación de usuarios.
Métodos:	
	`public setPassword (password: String, newPassword: String, user: String): Boolean`
	`public recoverPassword (user: String, securityAswerd: String): String`
	`public authenticate(user: String, password: String): Boolean`
	`public getSecurityQuestion(user: String): String`

Interfaz IEnumUserManager

Nombre:	`public interface IEnumUserManager`
Descripción:	Define una lista de métodos relacionados con la gestión de información referida a los usuarios ENUM.
Métodos:	
	`public addEnumUser (enumUser: EnumUser): int`
	`public deleteById (idEnumUser: int): Boolean`
	`public updateEnumUser (enumUser: EnumUser): Boolean`
	`public selectById (idEnumUser: int): EnumUser`
	`public selectByUserNameAndPassword (userName: String, password: String): EnumUser`
	`public selectAll (): List<EnumUser>`
	`public selectByUser (userName: String): EnumUser`

Interfaz IRecorderManager

Nombre:	`public interface IRecorderManager`
Descripción:	Define una lista de métodos relacionados con la gestión de información referida a los Registradores.
Métodos:	
	`public addRecorder (recorder: Recorder): int`
	`public deleteById (idRecorder: int): Boolean`
	`public updateRecorder (recorder: Recorder): Boolean`
	`public selectById (idRecorder: int): Recorder`
	`public selectAll (): List<Recorder>`

Interfaz IContactManager

Nombre:	`public interface IContactManager`
Descripción:	Define una lista de métodos relacionados con la gestión de información referida a los contactos de un Subscriptor.
Métodos:	
	`public addContact (contact: Contact): int`
	`public deleteContact (idContact: int): Boolean`
	`public updateContact (contact: Contact): Boolean`
	`public selectById (idContact: int): Contact`
	`public searchContactsByTip (tip: String): List<Contact>`
	`public selectAll (): List<Contact>`
	`public saveContactWithPlanification (contact: Contact): void`
	`public updateState (contact: Contact):Boolean`

Interfaz ISearchSubscriberInformation

Nombre:	public interface ISearchSubscriberInformation
Descripción:	Contiene una colección de métodos relacionados con la gestión de información de los Subscriptores.
Métodos:	
	public paginatedAdvanceSearch (*text*: String, *idMunicipality:* int, *idProvince*: int, *idCountry*: int, *limit*: int, *offset*: int): List<Subscriber>
	public lookupSubscriberById (*idSubscriber*: int): Subscriber
	public allSubscriber (): Subscriber
	public addSubscriber (*subscriber*: Subscriber): int
	public addSubscriberWithContacts (*subscriber*: Subscriber): int
	public deleteSubscriber (*idSubscriber*: int): Boolean
	public updateSubscriber (*subscriber*: Subscriber): Boolean

Interfaz ISubscriptionManager

Nombre:	public interface ISubscriptionManager
Descripción:	Define una lista de métodos relacionados con la gestión de información referida a las subscripciones.
Métodos:	
	public addSupscription (*subscription*: Subscription): int
	public deleteSupscription (*idSubscription*: int): Boolean
	public updateSupscription (*subscription*: Subscription): Boolean
	public setSupscriptionState (*idSubscription*: int, *state*: String):Boolean
	public selectSupscriptionById (*idSubscription*: int): Subscription

	public getSupscriptionBySubscriber (*tip*: String): Subscription

Interfaz IMunicipalityManager

Nombre:	public interface IMunicipalityManager
Descripción:	Define una lista de métodos relacionados con la gestión de información referida a los municipios.
Métodos:	
	public addMunicipality (*municipality*: Municipality): Boolean
	public deleteMunicipality (*idMunicipality*: int): Boolean
	public updateMunicipality(*municipality*: Municipality): Boolean
	public getMunicipalitiesByProvinceId(*idProvince*: int): List<Municipality>
	public selectAll(): List<Municipality>
	public selectById(*idMunicipality*: int): Municipality

Interfaz IProvinceManager

Nombre:	public interface IProvinceManager
Descripción:	Contiene una colección de métodos relacionados con la gestión de información de las provincias.
Métodos:	
	public addProvince (*province*: Province): Boolean
	public deleteProvince (*idProvince*: int): Boolean
	public updateProvince (*province*: Province): Boolean
	public getProvincesByCountryId (*idCountry*: int): List<Province>
	public selectAll (): List<Province>

	public selectByID (*idProvince*: int): Province

Interfaz ICountryManager

Nombre:	public interface ICountryManager
Descripción:	Contiene una colección de métodos relacionados con la gestión de información de los países.
Métodos:	
	public addCountry (*country*: Country): int
	public deleteCountry (*idCountry*: int): Boolean
	public updateCountry (*country*: Country): Boolean
	public countryById (*idCountry*: int): Country
	public listAllCountry(): List<Country>

Interfaz IPrivilegeManager

Nombre:	public interface IPrivilegeManager
Descripción:	Define una lista de métodos relacionados con la gestión de información referida a los privilegios de un usuario.
Métodos:	
	public addPrivilege (*privilege*: Privilege): int
	public deleteById (*idPrivilege*: int): Boolean
	public updatePrivilege (*privilege*: Privilege): Boolean
	public selectAll (): List<Privilege>
	public selectByID (*idPrivilege*: int): Privilege

Interfaz ITradeOfficeManager

Nombre:	public interface ITradeOfficeManage
Descripción:	Contiene una colección de métodos relacionados con la gestión de información de las Oficinas Comerciales.
Métodos:	
	public addTradeOffice (*tradeOffice*: TradeOffice): int
	public deleteTradeOffice (*idTradeOffice*: int): Boolean
	public updateTradeOffice (*tradeOffice*: TradeOffice): Boolean
	public listTradeOfficeByMunicipalityId (*idMunicipality*: int): List<TradeOffice>
	public selectAll (): List <TradeOffice >
	public selectById (*idTradeOffice*: int): TradeOffice

Interfaz IContactTypeManager

Nombre:	public interface IContacTypeManager
Descripción:	Contiene una colección de métodos relacionados con la gestión de información de los tipos de contactos que poseen los usuarios.
Métodos:	
	public addContacType (*contacType*: ContacType): Boolean
	public deleteContacType (*idContacType*: int): Boolean
	public updateContacType (*contacType*: ContacType): Boolean
	public getAllContactType (): List<ContacType>

Interfaz IContactSubtypeManager

Nombre:	public interface IContactSubtypeManager
Descripción:	Define una lista de métodos relacionados con la gestión de información referida a los subtipos de contacto de un usuario.
Métodos:	
	public addContactSubtype (*idContactType*: int, *contactSubtype:* ContactSubtype): int
	public deleteContactSubtype (*idContactSubtype*: int): Boolean
	public updateContactSubtype (*contactSubtype*: ContactSubtype): Boolean
	public selectAll (): List<ContactSubtype>
	public selectByTypeId (*id*: int): List<ContactSubtype>
	public selectById (*id*: int): ContactSubtype

Interfaz IDomainManager

Nombre:	public interface IDomainManager
Descripción:	Define una lista de métodos relacionados con la gestión de información referida al dominio.
Métodos:	
	public update (*host*: Host): Boolean
	public selectAll (): List<Host>

Interfaz IPlanificationContactManager

Nombre:	public interface IPlanificationContactManager
Descripción:	Define una lista de métodos relacionados con la gestión de información referida a la planificación de los contactos de un usuario.
Métodos:	
	public addPlanification (*planification*: Planification): int
	public deletePlanification (*idContac*: int): Boolean

	`public updatePlanification (planification: Planification): Boolean`
	`public selectByContactId (idContac: int): List<Planification>`

3.2.2 Paquete *business.impl*

En este paquete radicará toda la implementación de las operaciones definidas por las clases del diseño, las cuales están coleccionadas en el paquete de interfaces antes descrito.

3.3 Modelo de Diseño y Modelo de Datos

El modelo de diseño es un modelo de objetos que describe la realización de los casos de uso, y sirve como una abstracción de la implementación y el código fuente que se genera. Es un artefacto extensivo y compuesto, que comprende todas las clases de diseño, subsistemas, paquetes, colaboraciones y relaciones entre ellos. Por lo que el modelo de diseño, es una entrada esencial para las actividades de la implementación y las pruebas. Por otra parte, el modelo de datos describe las representaciones lógicas y físicas de los datos persistentes utilizados por la aplicación.

Los diseños de ambos modelos propuestos, concretamente, los artefactos Diseño de Clases y Modelo de Datos, permiten obtener una visión objetiva para la implementación de los componentes de acceso a datos y negocio; el diseño de clases por su parte, asegura que las clases proporcionen el comportamiento que requieren las realizaciones de los casos de uso, así como también, la consistencia con la arquitectura de software, y la información suficiente para evitar ambigüedades en las clases implementadas. (20)

La representación del Diseño de Clases propuesto por los Arquitectos, se muestra a continuación, atendiendo a las clases que involucran directamente a la persistencia de datos y a la capa de negocio para la realización de cada uno de los casos de uso arquitectónicamente significativos.

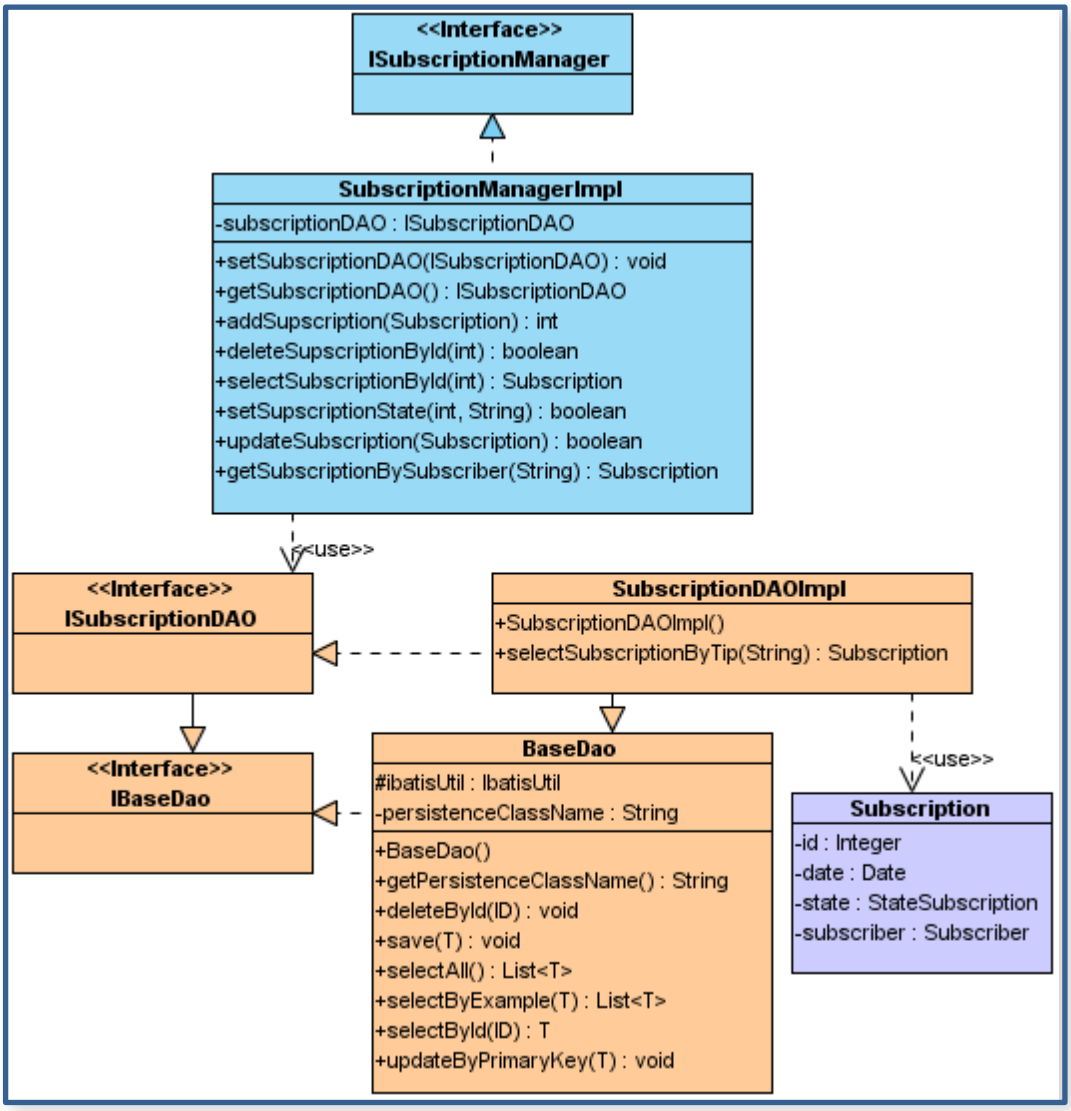

Figura 11: Diagrama de clases correspondiente al Caso de Uso: Gestionar Subscripción.

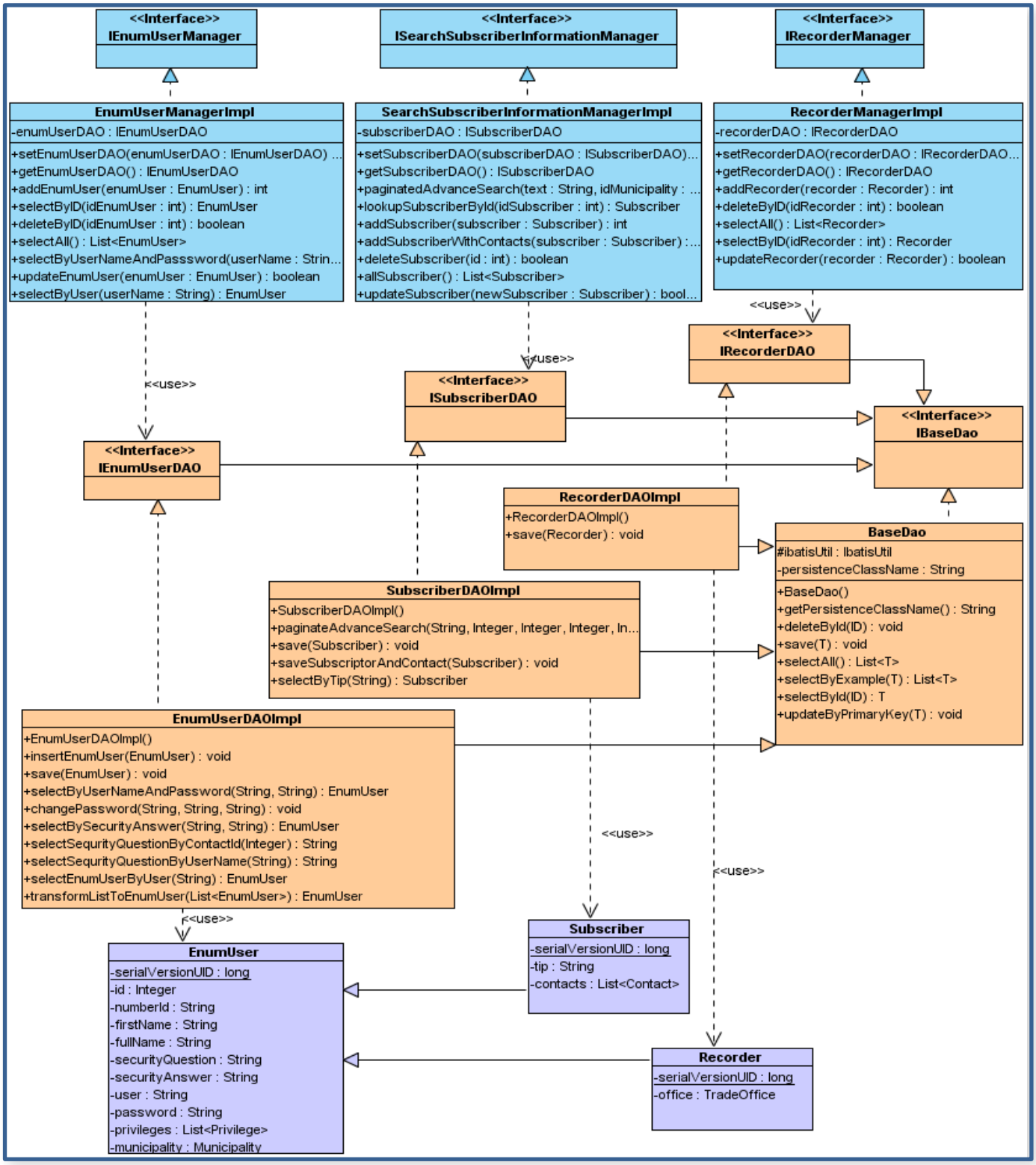

Figura 12: Diagrama de Clases correspondiente a los Casos de Usos: Gestionar usuario ENUM, Gestionar contraseña, Autenticar usuario ENUM, Actualizar datos del Subscriptor y Buscar datos personales del Subscriptor.

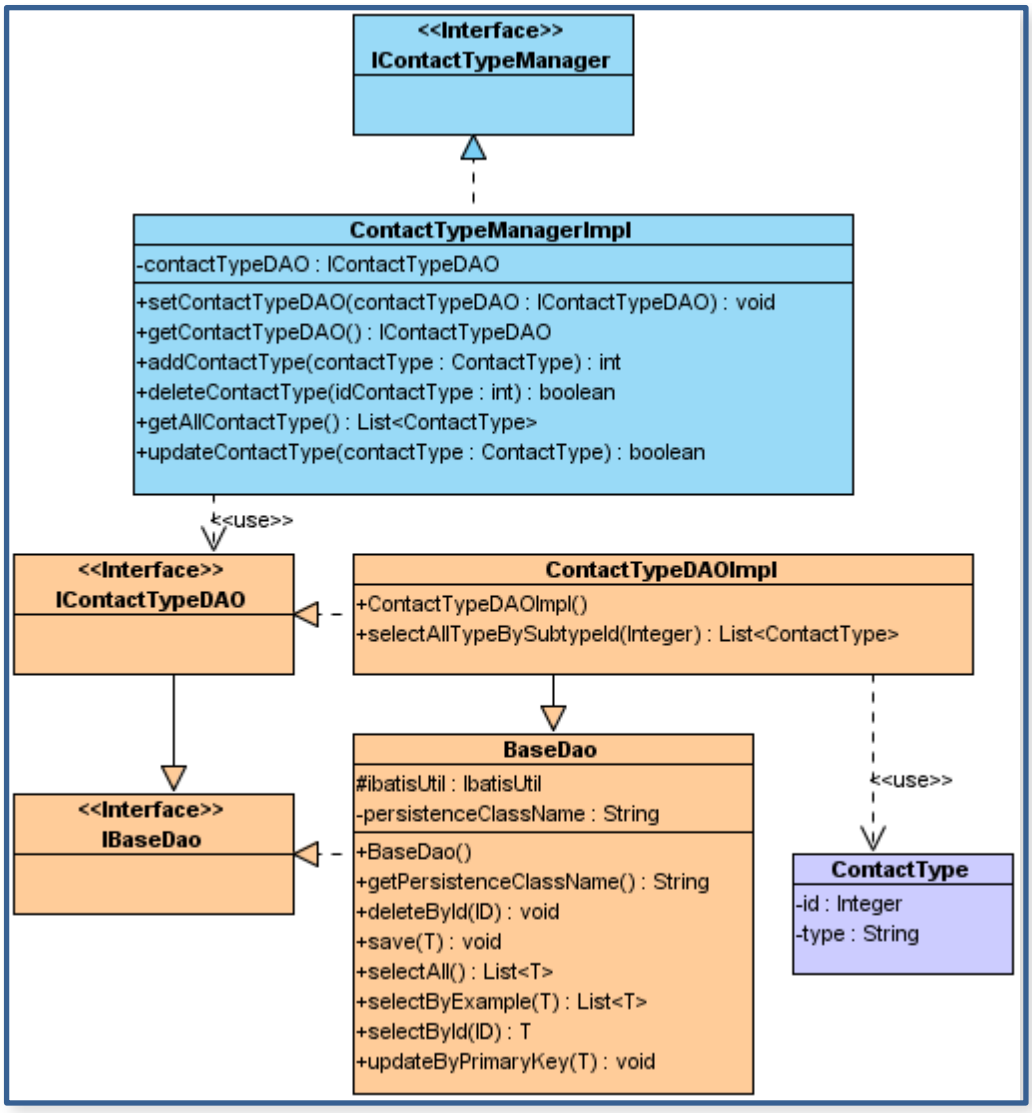

Figura 13: Diagrama de clases correspondiente a los Casos de Usos: Gestionar tipos de contactos y Listar tipos de contactos.

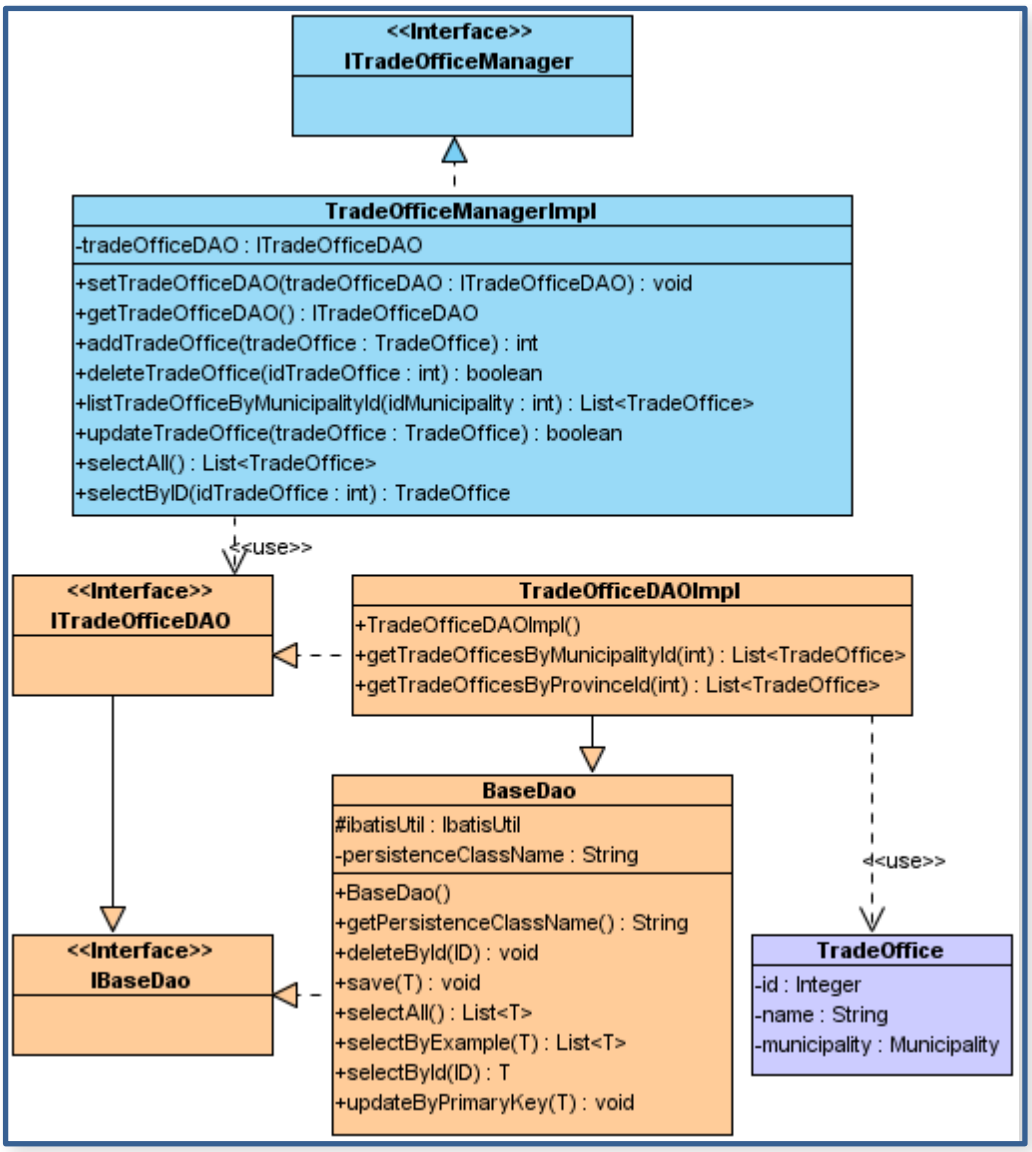

Figura 14: Diagrama de clases correspondiente al Caso de Uso: Gestionar Oficina Comercial.

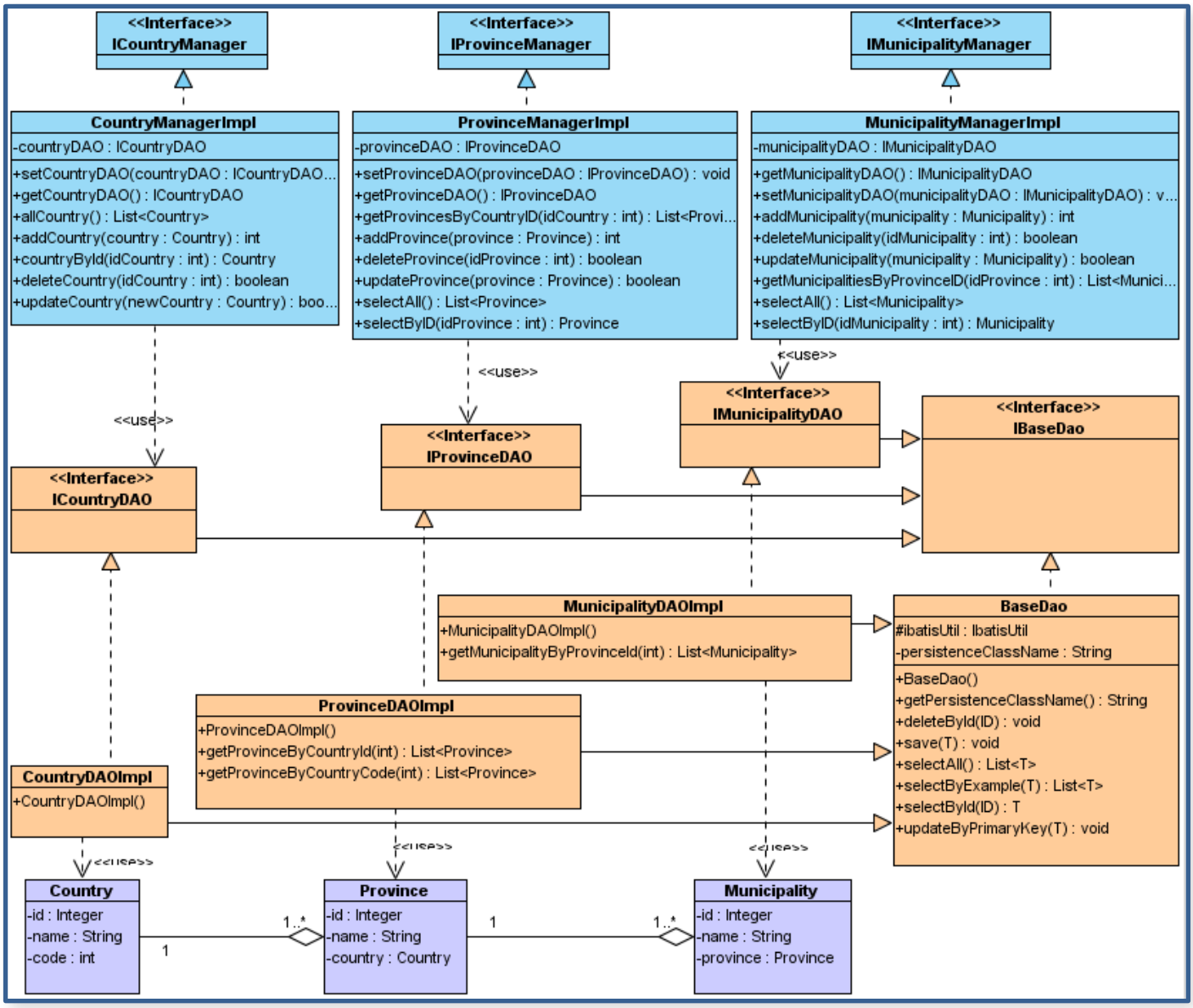

Figura 15: Diagrama correspondiente a los Casos de Usos: Gestionar localidad y Listar localidad.

La representación del Modelo de Datos propuesto por el Diseñador de Base de Datos, en este caso, se muestra conteniendo cada una de las entidades persistentes implicadas en el desarrollo (Figura 16).

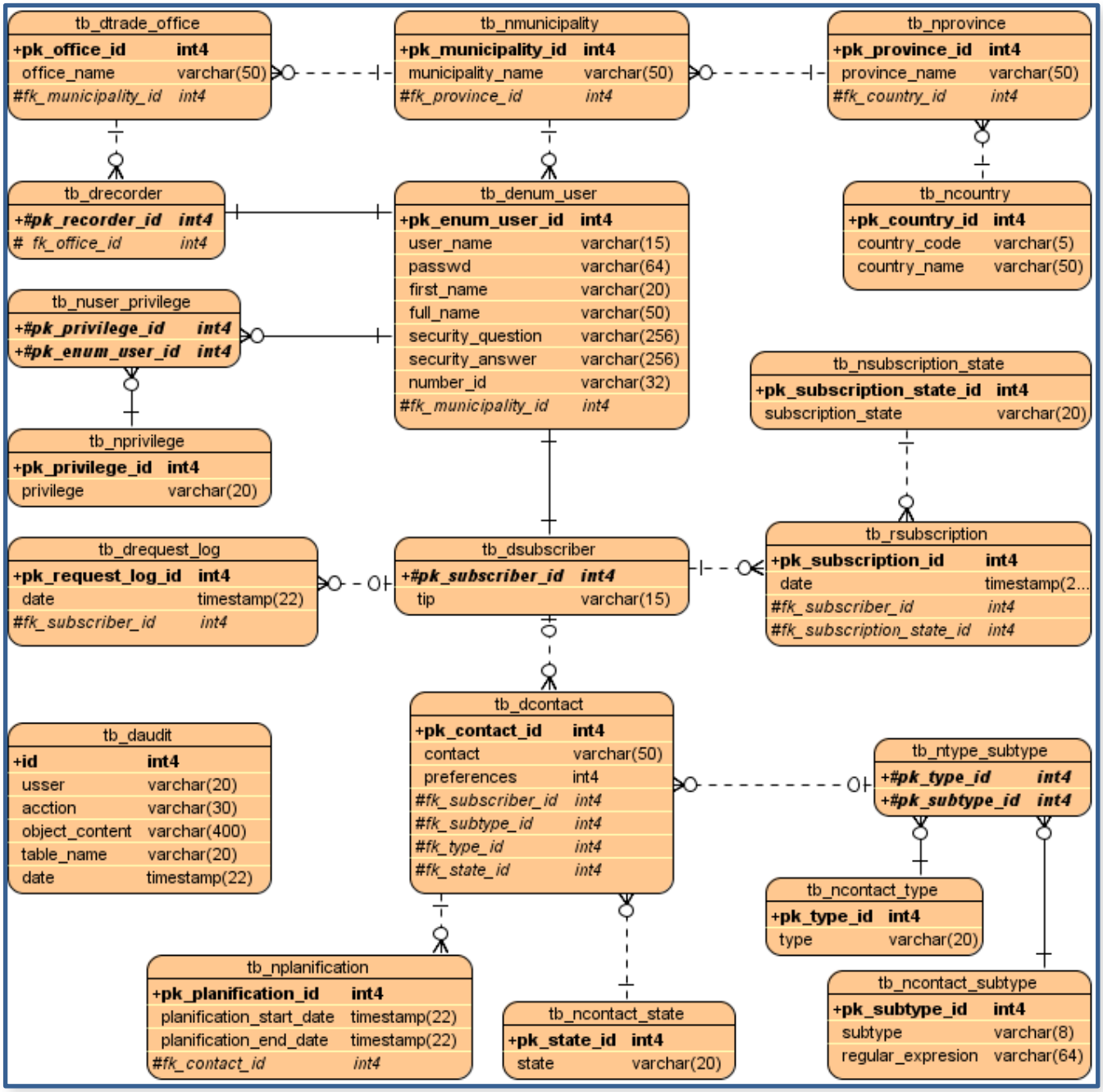

Figura 16: Diagrama Entidad - Relación.

3.4 Modelo de Implementación

Los componentes son la parte física y reemplazable de un sistema que está compuesto por un conjunto de interfaces y proporciona la realización de dicho conjunto. Se usan para modelar los elementos físicos que pueden hallarse en un nodo por lo que empaquetan elementos como clases, colaboraciones e interfaces. Son independientes entre ellos y tienen su propia estructura e implementación. Tienen relaciones de traza con los elementos del modelo que implementan. Uno de los objetivos de los diagramas que componen este modelo es de describir los componentes a construir y su organización. (20)

3.4.1 Diagrama de Componentes

Los diagramas de componentes se utilizan para modelar la vista estática de un sistema. Muestran la organización y las dependencias lógicas entre un conjunto de componentes software, sean éstos componentes de código fuente, librerías, binarios o ejecutables. El uso más importante de estos diagramas es mostrar la estructura de alto nivel del modelo de implementación. (20)

Para el caso de la capa de Persistencia y de Negocio, se implementó un grupo de funcionalidades agrupadas es diferentes componentes, mostrados de forma general en la Figura 17.

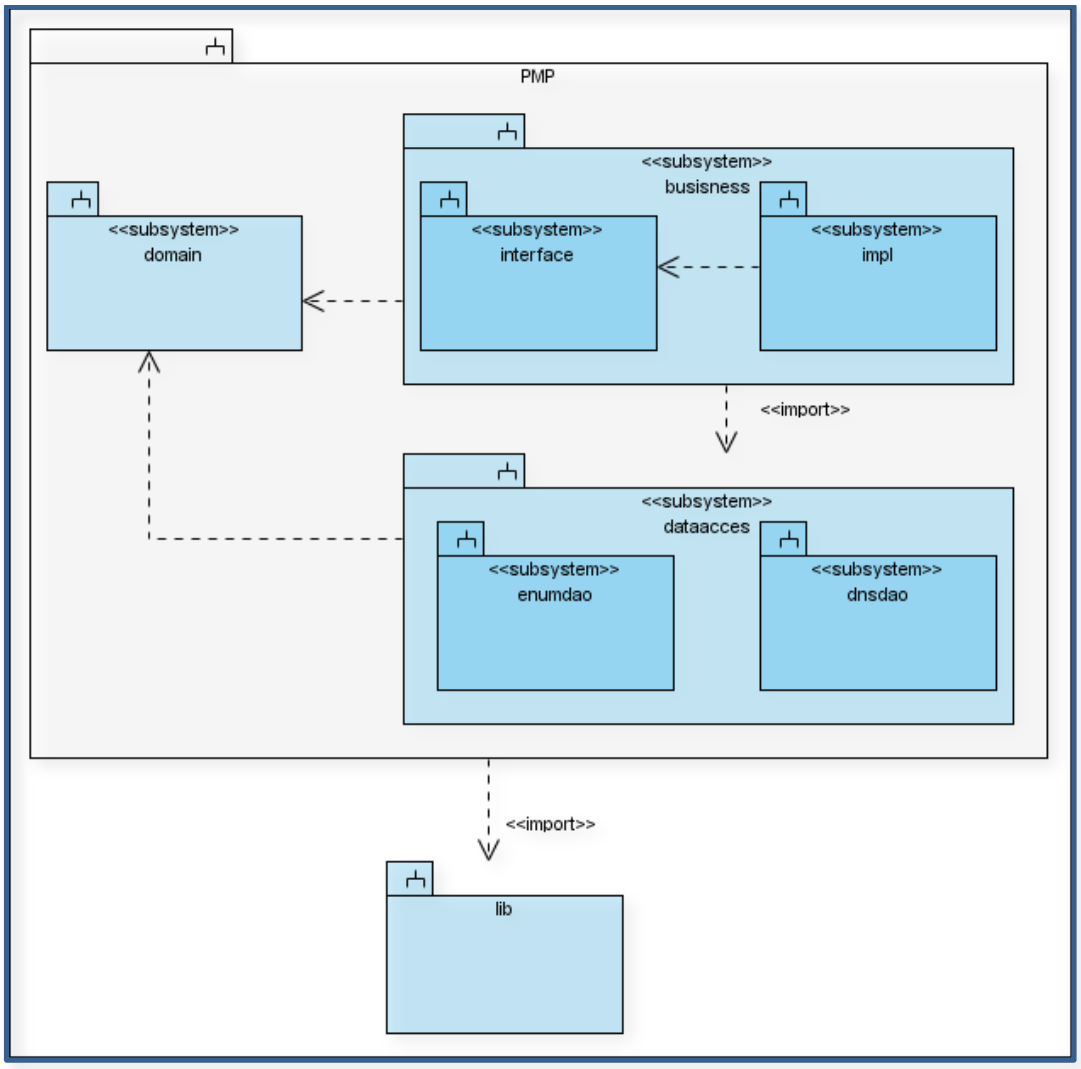

Figura 17: Diagrama de componentes de las capas de Persistencia y Negocio.

El diagrama de componentes del subsistema de la capa del negocio está formado por las interfaces de las clases del negocio y por un subsistema que tiene un componente por cada interfaz del negocio con la implementación de cada una de ellas (Figura 18).

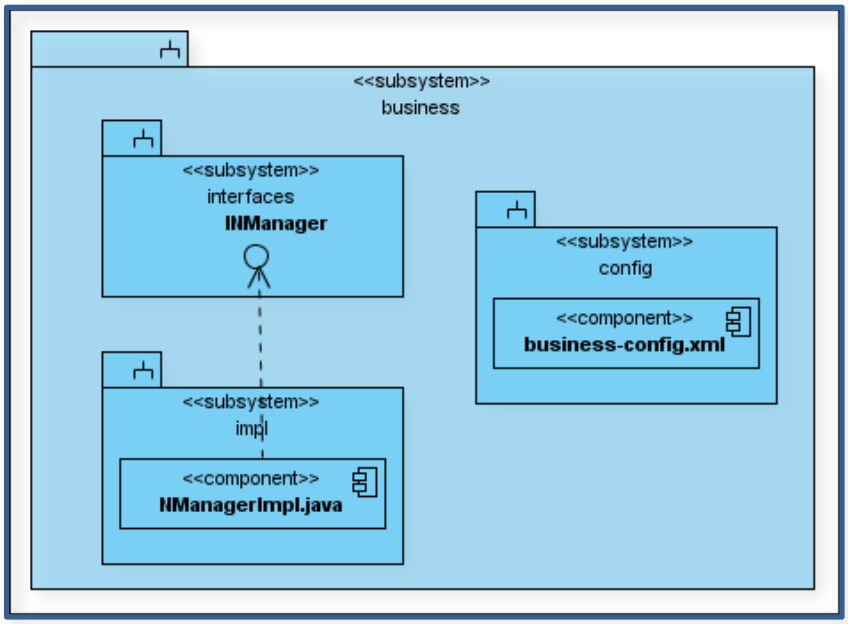

Figura 18: Diagrama de componentes de la Capa de Negocio.

El diagrama de componentes del subsistema de la capa de acceso a datos está formado por las interfaces de las clases de acceso a datos y dos subsistemas: *impl* y *map*. El subsistema *impl* tiene un componente por cada interfaz de acceso a datos con la implementación de cada una de ellas. Dicho subsistema se relaciona con el componente ***IbatisUtil***. El subsistema *map* contiene un componente por cada clase del dominio. (Figura 19)

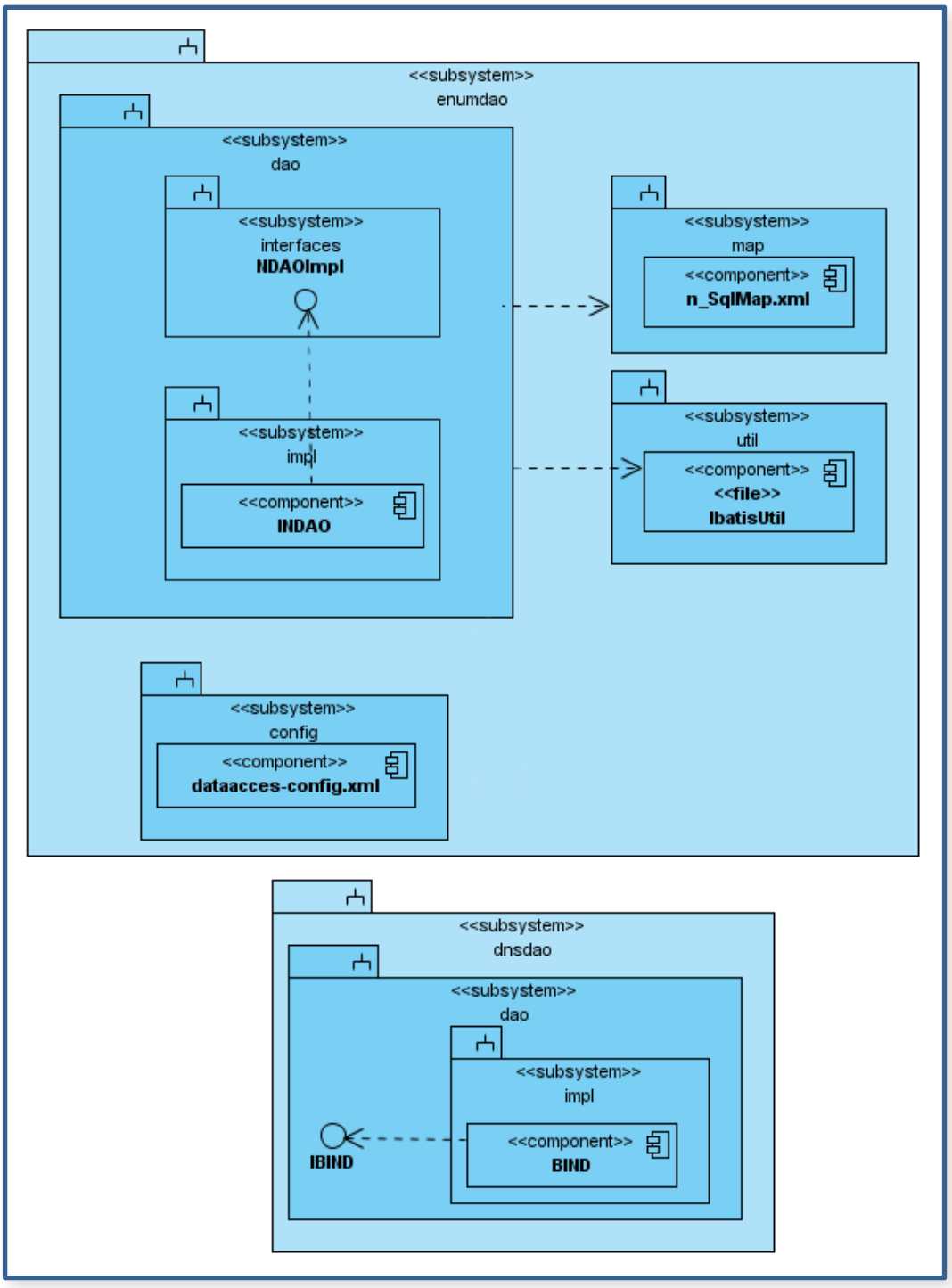

Figura 19: Diagrama de componentes de la Capa de Acceso a Datos.

Conclusiones

Al concluir este capítulo, se definieron los elementos y la estructura que debe tener la capa de persistencia y de negocio, tanto desde el entorno de desarrollo como la estructura lógica presentada a través de diagramas de paquete.

Se modelaron los diseño de clases y el modelo de datos propuestos, ya que son los artefactos que proporcionan la entrada para las actividades de implementación y las clases entidades para manejar los datos persistentes.

Se realizaron y mostraron las técnicas fundamentales para configurar iBatis, y por consiguiente, lograr transacciones lógicas entre la aplicación y la base de datos. Además se generaron los objetos de negocio y archivos de mapeo necesarios para la implementación, así como también, la definición y especificación de elementos, atributos y propiedades de éstos que optimizan el trabajo y la persistencia de objetos a través de las diferentes capas arquitectónicas.

Capítulo 4

Pruebas Automatizadas

Las aplicaciones de software han crecido en complejidad y tamaño, y por consiguiente, también en costos. Hoy en día es crucial evaluar la calidad de lo construido de modo que se minimice el costo de su reparación.

En este capítulo se validan, comprueban o evalúan los componentes implementados en la capa de persistencia que permiten el almacenamiento y recuperación de objetos en las entidades o tablas que conforman la base de datos, además de comprobar su correcto funcionamiento al integrarse con la lógica de la capa de negocio.

4.1 Pruebas de Unidad

Cuando se implementan componentes de software, resulta recomendable no dar por finalizada la implementación sin estar seguros del correcto funcionamiento de todos los módulos, certificándose que cada uno de los mismos funcione correctamente por separado.

Una prueba de unidad pretende probar cada función en un archivo de programa simple (una clase o método en terminología de lenguajes orientados a objetos). Las librerías de pruebas de unidad formalizan este trabajo al proporcionar clases para pruebas.

La prueba de unidad ayuda a que el módulo se haga independiente. Esto quiere decir que un módulo que tiene una prueba de unidad se puede probar independientemente del resto del sistema.

El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas mediante el cumplimiento de un contrato escrito que el segmento de código debe satisfacer.

Mantener automatizado un conjunto amplio de pruebas permite reducir el tiempo que se tarda un equipo de desarrolladores en verificar la corrección del código y depurar errores, estos, dado que se tienen pruebas unitarias que pueden desenmascararlos, se encuentran más acotados y son más fáciles de localizar, además de facilitar que el programador cambie el código para mejorar su estructura (lo que se ha dado en llamar refactorización[20]), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.

Las pruebas unitarias además facilitan la integración ya que permiten llegar a la fase de integración con un alto grado de seguridad de que el código está funcionando correctamente y sirven incluso como documentación del código ya que se puede observar cómo utilizarlo.

Las pruebas deben ser diseñadas para descubrir fallos y no para demostrar que el software funciona. Siendo más razonable diseñar pruebas de aquellas partes en donde la probabilidad de fallo es mayor. No debería requerirse una intervención manual. Esto es especialmente útil para integración continua. Una buena práctica resultaría llevar paralelamente el proceso de desarrollo y el proceso de evaluación o comprobación de los elementos de implementación que se van generando. Encontrar errores es una tarea desafiante para cualquier desarrollador, pero existen herramientas que proporcionan una manera sencilla, rápida y elegante para escribir pruebas y validarlas automáticamente. JUnit es precisamente, una de estas herramientas, es un framework que se ha convertido en el estándar universal para realizar pruebas de unidad en Java.

JUnit permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces, JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente. (21)

JUnit tiene una manera muy peculiar para visualizar los reportes de las pruebas realizadas:

[20] **Refactorización:** (del inglés Refactoring) Técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

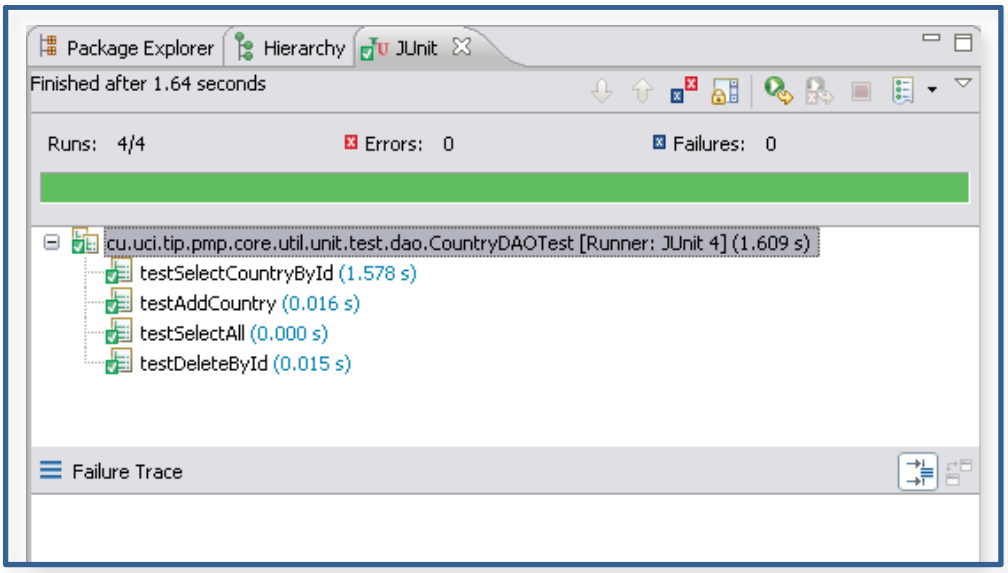

Figura 20: Visualización de una prueba satisfactoria con JUnit.

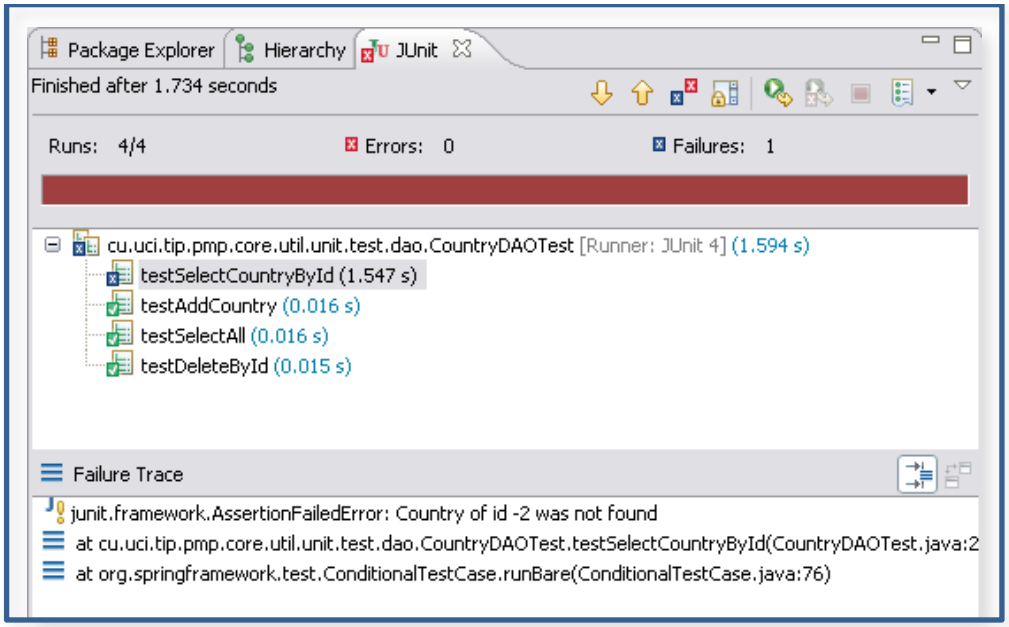

Figura 21: Visualización de una prueba fallida con JUnit.

JUnit es también un medio de controlar las pruebas de regresión[21], necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación. (21)

En versiones anteriores de JUnit no existían caracteres especiales, llamados también anotaciones, en la versión 4 del framework se han incluido para intentar simplificar más la labor del programador. Se trata de palabras claves que se colocan delante de los métodos y que indican a las librerías JUnit instrucciones concretas: (22)

```
public class TestClass {

      @BeforeClass                                    (1)
      public static void setUpClass () throws Exception {
            // Inicialización general de variables...
      }

      @AfterClass                                     (2)
      public static void tearDownClass () throws Exception {
            // Liberación de recursos...
      }

      @Before                                         (3)
      public void setUp () {
            // Inicialización de variables antes de cada test
      }

      @After                                          (4)
      public void tearDown () {
            // Tareas a realizar después de cada test
      }

      @Test                                           (5)
      public void testSomeMethod () {
            // Se crea el entorno necesario para la prueba
            AssertNotNull (someObject);               (6)

            AssertEquals (expected, actual);
      }                                               (7)
}
```

[21] **Pruebas de regresión**: pruebas de software que intentan descubrir las causas de nuevos errores (bugs), carencias de funcionalidad, o divergencias funcionales con respecto al comportamiento esperado del software, inducidos por cambios recientemente realizados en partes de la aplicación que anteriormente al citado cambio no eran propensas a este tipo de error.

1. *@BeforeClass:* Sólo puede haber un método con este marcador. Este método será invocado una sola vez, al principio del lanzamiento de todas las pruebas. Se suele utilizar para inicializar los atributos comunes a todas las pruebas o para realizar acciones que tardan un tiempo considerarse en ejecutarse.
2. *@AfterClass:* Sólo puede haber un método con este marcador. Este método será invocado una sola vez cuando finalicen todas las pruebas.
3. *@Before:* Indica que el siguiente método se debe ejecutar antes de cada test.
4. *@After:* Indica que el siguiente método se debe ejecutar después de cada test.
5. *@Test:* Indica a JUnit que se trata de un método de Test. En versiones anteriores de JUnit los métodos tenían que tener un nombre con la siguiente estructura: "testXXX". Con esta notación colocada delante de los métodos se pude elegir el nombre libremente además permite sustituir la herencia de TestCase.
6. *AssertNotNull:* Comprueba que el objeto indicado no es nulo.
7. *AssertEquals:* Realiza la comprobación entre 2 valores. Devuelve una excepción si no se produce el resultado esperado.

Para que una prueba unitaria sea buena esta debe cumplir con un conjunto de características, haciendo especial énfasis en las que a continuación se presentan: (22)

- ✓ *Completas*: deben cubrir la mayor cantidad de código.
- ✓ *Repetibles o Reutilizables*: no se deben crear pruebas que sólo puedan ser ejecutadas una sola vez.
- ✓ *Independientes*: la ejecución de una prueba no debe afectar a la ejecución de otra.
- ✓ *Profesionales*: las pruebas deben ser consideradas igual que el código, con la misma profesionalidad y documentación.

La capa de acceso a datos ocupa un lugar cimero dentro del correcto funcionamiento de una aplicación, para el funcionamiento adecuado de las capas superiores es necesario que las funcionalidades que brinda el acceso a datos se encuentre lo mejor depuradas posible para que no se propaguen los posibles errores hacia niveles superiores. Con tal objetivo se diseñaron casos de pruebas que permitieron verificar que los

métodos implementados en el acceso a datos de la Plataforma Manejadora de Peticiones funcionan correctamente:

Ejemplos de Pruebas de Unidad para CountryDAOImpl

Caso de prueba: Listar países.

```
public class CountryDAOTest extends BaseDaoTestCase {

...

        @Test
        public void selectAll() {
                String query = "select * from tb_ncountry ";
                List<Map<String, Object>> list = simpleJdbcTemplate.
                                                        queryForList(query);
                System.out.println ("Listado de países existentes:");
                for (Map<String, Object> map: list) {
                        System.out.println (map.get ("country_name"));
                }
        }
```

(1) Al anotar este método como test se comprobó su correcto funcionamiento y se verifica que realmente se obtengan los resultados que se esperan, en este caso un listado completo de los países existentes en la correspondiente tabla de la base de datos.

pk_country_id	co	country_name
14	25	Angola
1	53	Cuba
12	41	Francia
13	33	Mozambique
2	58	Venezuela

Figura 22: Tabla tb_ncountry en la base de datos.

Resultado de la prueba:

```
Listado de países existentes:
Venezuela
Cuba
Francia
Mozambique
Angola
```

```
Argentina
```

Caso de prueba: Seleccionar país por ID. (Caso satisfactorio)

```
@Before                                                    1
@Timed (millis = 500)
public void init() {
      simpleJdbcTemplate.update ("insert into tb_ncountry (pk_country_id,
      country_code, country_name) values (?,?,?)",
      3, 21, "Argentina");
}

@Test
public void selectCountryById() {
      String query = "select country_name from tb_ncountry where pk_country_id
      =?";
      String name = (String) simpleJdbcTemplate.queryForObject (query,
                  String.class, 3);
      assertEquals ("Argentina", name);                    2
}
```

1 Antes de iniciar el caso de prueba se adiciona en la base de datos un nuevo país, con la restricción de que la operación se concrete en un lazo de tiempo inferior al pasado por valor a la anotación.

2 Una vez guardado el nuevo dato, se procede a comprobar si el nombre del país del código que se provee coincide con el valor esperado. En caso que la operación demore más de 500 milisegundos o no coincidan los parámetros el caso de prueba lanza una excepción.

Caso de prueba: Seleccionar país por ID. (Caso fallido)

```
@Before
@Timed (millis = 500)
public void init() {
      simpleJdbcTemplate.update ("insert into tb_ncountry (pk_country_id,
      country_code, country_name) values (?,?,?)",
      3, 21, "Argentina");
}

@Test

public void selectCountryById() {

      String query = "select country_name from tb_ncountry where pk_country_id
      =?";
```

```
            String name = (String) simpleJdbcTemplate.queryForObject (query,
                  String.class, 3);
            assertEquals ("Cuba", name);
      }
```

Resultado de la prueba:

En este caso el valor retornado (Argentina) no coincide con el valor esperado (Cuba).

Es importante destacar que las pruebas unitarias no descubrirán todos los errores del código. Por definición, sólo prueban las unidades por sí solas. Por lo tanto, no descubrirán errores de integración, problemas de rendimiento y otros problemas que afectan a todo el sistema en su conjunto. Las pruebas unitarias sólo son efectivas si se usan en conjunto con otras pruebas de software.

4.2 Pruebas de Integración

El objetivo principal de las pruebas de integración es detectar las fallas de interacción entre las distintas clases que componen el sistema. En relación a este tema, Pressman menciona lo siguiente:

"Los datos se pueden perder en una interfaz; un módulo puede tener un efecto adverso e inadvertido sobre otro; las subfunciones, cuando se combinan, pueden no producir la función principal deseada; la imprecisión aceptada individualmente puede crecer hasta niveles inaceptables; y las estructuras de datos globales pueden presentar problemas." (23)

Debido a que cada clase probada unitariamente se inserta de manera progresiva dentro de la estructura, además de que a la par se van siguiendo los lineamientos dictados por el diseño, las pruebas de integración son realmente el mecanismo para comprobar el correcto ensamblaje del sistema completo.

4.2.1 Integrando JUnit con Spring

Una de las tareas más costosas en versiones anteriores liberadas de Spring era el desarrollo de pruebas, en ocasiones, se tenían casos de prueba que deberían ser rápidos de ejecutar, pero cada vez que se ejecutaba un test se debía cargar el contexto de Spring, y si este contexto pertenecía a un sistema muy grande con muchos beans, la ejecución de los test podía ser una tarea que llevara mucho tiempo en realizarse. (22)

Desde la versión 2.5, Spring Framework proporciona un marco de desarrollo y creación de pruebas que facilitan la realización de pruebas de unidad e integración basado en anotaciones y con un alto grado de abstracción en cuanto al framework escogido para desarrollar las pruebas. (22)

Además, Spring TestContext Framework proporciona una infraestructura que no solo facilita la carga de contextos de Spring, sino que soluciona problemas relacionados con la inyección de dependencias y el manejo automático de transacciones contra la base de datos. (22)

Spring TestContext Framework ofrece clases de apoyo para la gestión del contexto de la aplicación, de esta forma las clases de pruebas definidas simplemente extienden de la clase definida, según el framework de prueba escogido. (22)

Framework de prueba	Clases de Soporte
JUnit 3.8	AbstractJUnit38SpringContextTests
JUnit 4.4	AbstractJUnit4SpringContextTests
TestNG	AbstractTestNGSpringContextTests

Tabla 5: Clases de apoyo para la gestión del contexto.

Para el manejo de transacciones, el soporte de Spring para pruebas ofrece clases de apoyo según el framework escogido para implementar las clases de prueba, garantizando que los cambias previos realizados por un método de prueba no alteren el resultado del próximo y evitando que valores erróneos permanezcan almacenados en la base de datos una vez finalizadas las pruebas. (22)

Framework de prueba	Clases de Soporte
JUnit 3.8	AbstractTransactionalJUnit38SpringContextTests
JUnit 4.4	AbstractTransactionalJUnit4SpringContextTests
TestNG	AbstractTransactionalTestNGSpringContextTests

Tabla 6: Clases de apoyo para el manejo de las transacciones.

JUnit 4.4 y Spring TestContext ofrecen un conjunto de anotaciones tanto a nivel de métodos como de clases que facilitan la implementación de las pruebas:

```
@RunWith (SpringJUnit4ClassRunner.class)
```
1

```
@ContextConfiguration (locations = {"conf/spring-context.xml"})

@Transactional
public class TestClass {

      @Autowired
      private SomeService service;

      ...
}
```

1 *@RunWith (SpringJUnit4ClassRunner.class):* La anotación especifica que el framework empleado para el desarrollo de los test es JUnit 4.4.

2 *@ContextConfiguration (locations = {"conf/spring-context.xml"}):* Esta anotación indica donde están los ficheros de configuración de Spring. Si hubiese más de uno, se debe especificar dentro del atributo *locations* separado por comas. Este contexto se cargará una única vez por todos los test que se ejecuten dentro de la clase TestClass. Se hace automáticamente, y no como antes, que se cargaba expresamente el contexto con código de la creación del programador.

3 *@Transactional:* Se indica que el siguiente test, debe ser marcado con una transacción, y lo que es más importante, si todo va bien, hacer rollback al final del método, para asegurarse que cada test deja la base de datos en un estado inerte. Ningún cambio realizado por las operaciones realizadas en los test, se transmite a la base de datos.

4 *@Autowired:* La anotación extrae del contexto de Spring un objeto del tipo que anota, en este caso SomeService.

Spring, como parte del soporte que ofrece para pruebas provee su propio set de anotaciones para simplificar la creación de pruebas. Sin embargo, son solo compatibles con JUnit (3.8 y 4.4): (22)

- ✓ *@Repeat*: Indica que el método de prueba tiene que ejecutarse tantas veces como se especifique en el valor de la anotación.
- ✓ *@Timed*: Indica que el método de prueba debe de completarse en un periodo de tiempo específico (en milisegundos). En caso contrario la prueba falla.

- ✓ *@IfProfileValue*: Esto indica que un método de prueba sólo puede ejecutarse en un entorno de prueba específico. Este método de prueba sólo se ejecutará cuando el valor real coincide con el valor especificado.
- ✓ *@ExpectedException*: Esta anotación tiene el mismo efecto tanto en JUnit 4.4 como en TestNG como soporte de excepciones.

Existen tendencias a integrar los módulos de una aplicación por anticipado, esto desenlaza un conjunto de errores que hace difícil la corrección del código puesto que es complicado aislar las causas al tener delante el programa entero en toda su extensión. Con el objetivo de evitar una situación similar a la planteada, una vez que han sido diseñados y ejecutados un conjunto de pruebas para verificar el correcto funcionamiento de los métodos implementados en el acceso a datos es necesario garantizar que al integrarse estas funcionalidades a las presentes en la capa de negocio funcionen como una unidad integra, para lo cual se diseñaron los siguientes casos de prueba:

Ejemplos de Pruebas de Integración para ProvinceManagerImpl:

Caso de prueba: Seleccionar provincias dado un ID de país.

```
@Test
@Timed (millis = 500)                                    1

public void selectProvinceByCountryID () throws Exception{
      List<Province> list = provinceManagerImpl.
      getProvincesByCountryID (1);

      assertEquals (super.countRowsInTable ("tb_nprovince"),    2
            list.size ());
      for (Province province: list)
            System.out.println (province.getName ());
}
```

1 El caso de prueba verifica que realmente se obtengan los resultados que se esperan, en este caso un listado completo de las provincias existentes en la correspondiente tabla de la base de datos. El método debe ejecutarse en un lapso de tiempo inferior al pasado por valor a la anotación.

2 Este método se puede utilizar para comprobar de una forma sencilla y rápida el número de filas en una tabla, una vez realizada esta operación se procede a listar las provincias por nombre que pertenezcan al país cuyo código es 1.

pk_p	province_name	fk_count
4	Matanzas	1
5	Villa Clara	1
6	Cienfuegos	1
7	Ciudad Habana	1
8	Ciego de Avila	1
9	Camagüey	1
10	Las Tunas	1
11	Holguín	1
12	Granma	1
13	Santiago de Cuba	1
14	Guantánamo	1
15	Isla de la Juventud	1
16	Sancti Spíritus	1
47	La Habana	1
48	Pinar del Río	1

Figura 23: Tabla tb_nprovince en la base de datos.

Resultado de la prueba:

```
Listado de provincias para el país cuyo código es 1:
Matanzas
Villa Clara
Cienfuegos
Ciego de Ávila
Camagüey
Las Tunas
Holguín
Granma
Santiago de Cuba
Guantánamo
Isla de la Juventud
Pinar del Río
La Habana
Sancti Spíritus
Ciudad Habana
```

En relación a la planificación de las pruebas de integración, es importante tomar en cuenta varios aspectos, tales como el grado de complejidad de las clases, su importancia funcional con respecto a las especificaciones del sistema, la cantidad de módulos que usan o dependen de cada clase, y las

estadísticas de error en la fase de pruebas unitarias. En este último caso, el objetivo es tratar de integrar primero las clases que incurrieron en mayor número de fallos, ya que, aún cuando los errores en las pruebas unitarias hayan sido solventados, son las clases más susceptibles de contener errores en la integración porque pudieron cambiar algunas condiciones de entrada y/o salida que no fueron reportadas a tiempo.

Conclusiones

En este capítulo se expuso la validación de la solución propuesta en el capítulo anterior, probando los métodos implementados con la utilización del framework JUnit 4.4 y Spring TestContext Framework, que evidencian el correcto funcionamiento de los componentes para la persistencia de objetos y su integración con la lógica del negocio.

Conclusiones

El mundo de las Telecomunicaciones está en constante desarrollo, las redes IP con amplias funcionalidades ha sobrepasado a las viejas redes de circuitos para mejorar la calidad de vida y el crecimiento tecnológica de los países.

En el caso de Cuba, país en vías de desarrollo, la implantación del primer servicio de integración entre las redes de circuito y las redes de paquete, el *ENUM de Usuarios*, constituye un paso de avance en la evolución de las telecomunicaciones.

Con este trabajo se implementan dos capas de vital importancia en el funcionamiento de la Plataforma Manejadora de Peticiones, la Capa de Acceso a Datos y la Capa de Negocio, encargadas de manejar las peticiones, persistencia y recuperación de datos a los servidores DNS y Base de Datos de las aplicaciones a las cuales la Plataforma le brinda servicios.

Siendo lo anterior expuesto, el principal objetivo de nuestro trabajo y quedando complacido el cliente, concluye este estudio y Trabajo de Diploma donde se profundizó en el conocimiento de las tecnologías utilizadas, aportando una gran experiencia para la implementación de futuras aplicaciones de este tipo, se comprendió el funcionamiento del ENUM y se realizó pruebas a los distintos componentes implementados, demostrando la veracidad del sistema.

Recomendaciones

Con el objetivo de mejorar y agregarle nuevas funcionalidades a las capas de Acceso de Datos y Negocio se recomienda:

- ✓ Refinar e incluir funcionalidades que permitan brindar nuevos servicios a los usuarios.

Bibliografía

1. **ETECSA.** ETECSA - Empresa de Telecomunicaciones de Cuba S.A. *ETECSA - Información corporativa.* [Online] http://www.etecsa.cu/infocorporativa.asp?codigo=19&padre=19.

2. **Pérez, Marcos.** AHCIET. *Impacto del ENUM en las redes y los servicios.* [Online] Junio 2008. http://www.ahciet.net/actualidad/revista/r.aspx?ids=10739&ids2=21829.

3. **Vázquez, Adolfo.** IDG.es. *Enum: todo en uno.* [Online] http://www.idg.es/comunicaciones/articulo.asp?id=135297.

4. **Enum Trial México.** Enum Trial México. [Online] http://www.enum.org.mx/?q=node/14.

5. **Enum.at.** Enum.at. [Online] http://enum.at/.

6. **David Gallardo, Robert McGovern.** *Eclipse In Action: A Guide for Web Developers.* 2003.

7. **Kaisler, Stephen H.** *Software Paradigms.* 2005.

8. **Begin, Clinton, Goodin, Brandon and Meadors, Larry.** *iBATIS in Action.* s.l. : Manning Publications, 2007.

9. **Johonson, Rod.** *Professional Java Development with the Spring Framework.* 2005.

10. **PostgreSQL.** PostgreSQL 8.3 Documentation. [Online] http://www.postgresql.org/docs/8.3/static/intro-whatis.html.

11. **Pimentel González, Luis Alberto, Pérez Rivero, Iósev and Haro Pérez, Madelín.** *ArBaWeb: Arquitectura Base sobre la Web.* 2007.

12. **Hernández, Octavio.** *Acceso a datos de próxima generación.* [Online] diciembre 2006. http://msdn.microsoft.com/es-es/library/aa730866(VS.80).aspx.

13. **Jacobson, Ivar, Booch, Grady and Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 1999.

14. **Martínez Juan, Francisco Javier.** *Guía de construcción de software en Java con Patrones de Diseño.* Oviedo : s.n.

15. **Kayal, Dhrubojyoti.** *Pro Java™ EE Spring Patterns: Best Practices and Design Strategies Implementing Java™ EE Patterns with the Spring Framework.* s.l. : Apress, 2008.

16. **Alur, Deepak, Crupi, John and Malks, Dan.** *Core J2EE™ Patterns: Best Practices and Design Strategies.* s.l. : Prentice Hall PTR, 2003.

17. **S. Pressman, R.** *Ingeniería de Software: Un enfoque práctico.* s.l. : Mc Graw Hill, 1993.

18. **Massol, V. and Husted, T.** *JUnit in Action.* s.l. : Manning Publications, 2004.

19. **Mak, G.** *Spring Recipes.* s.l. : Apress, 2008.

20. **Garlan, David and Shaw, Mary.** *An introduction to software architecture.* s.l. : CMU Software Engineering Institute, 1994.

21. **Armstrong, Damon.** Simple-Talk. *.NET Application Architecture: the Data Access Layer.* [Online] 2006. http://www.simple-talk.com/dotnet/.net-framework/.net-application-architecture-the-data-access-layer/.

22. **Begin, Clinton.** *iBatis Data Mapper 2.0 Developer Guide.* [pdf] November, 2006.

23. **Armas, Yanisleidys and Vázquez, Edisbel.** *Diseño e implementación del acceso a datos del proyecto TeleBanca.* 2007.

24. **Miguel Pérez.** Asociación Española de Usuarios de Internet. *¿Convergencia o colisión entre Internet y Telefonía?* [Online] http://miguelperezsubias.aui.es/index.php?body=asoc_v1article_socio&id_article=653.

25. **Walls, Craig and Breidenbach, Ryan.** *Spring in Action.* s.l. : Manning Publications Co., 2008. Second Edition.

Glosario de términos

Aislamiento: Es la propiedad que garantiza que las transacciones no entren en conflicto unas con otras.

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

Conmutación de circuitos: Método de transmisión de datos que consiste en configurar una serie de nodos intermedios para comunicar el nodo remitente al nodo receptor. En tal situación, la línea de comunicación se puede comparar con un canal de comunicación dedicado. Es el método usado particularmente por la Red Telefónica Pública Conmutada (RTPC). Al reservar una línea telefónica entre dos hablantes, la red puede garantizar que la transferencia de datos tenga el mejor rendimiento posible.

Conmutación de paquetes: Consiste en dividir la información en paquetes de datos; los nodos intermedios transmiten estos paquetes de datos por separado y los vuelven a montar cuando llegan al destinatario final.

Consistencia: Está relacionado con las restricciones que ponen los esquemas de base de datos para asegurar la integridad. La coherencia requiere que antes o después de una transacción la base de datos se encuentre en un estado coherente. Una base de datos se encuentra en un estado coherente cuando las restricciones de integridad, llaves foráneas y llaves únicas se cumplen.

Durabilidad: Define que después de ejecutada una transacción los datos deben ser seguros. Incluso si se produce un fallo del sistema después de la operación, los datos deben ser seguros.

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

Herramientas de Mapeo objeto relacional (Object-Relational mapping): Es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos. En la práctica esto crea una base de datos orientada a objetos virtual, por sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). Hay paquetes comerciales y de uso libre disponibles que desarrollan el mapeo relacional de objetos, aunque algunos programadores prefieren crear sus propias herramientas ORM.

Mapeo: Acción que permite que una aplicación pueda acceder a un determinado elemento de manera dinámica. Cuando se mapea un archivo, se dice que una aplicación puede acceder a ese elemento. Se utiliza principalmente en programación.

Open-Source: Código abierto (del inglés open source) es el término con el que se conoce al software distribuido y desarrollado libremente.

Protocolo: Se conoce como protocolo de comunicaciones o protocolo de red a un conjunto de reglas que especifican el intercambio de datos u órdenes durante la comunicación entre sistemas.

RDBMS (Relational Database Management System): Una base de datos relacional está compuesta de muchas relaciones en forma de listas de dos dimensiones y columnas que tienen filas. La forma en que la información está presentada al usuario y al programador es conocida como la vista lógica de la base de datos. La información guardada en el disco de la computadora es llamada vista interna.

Transacciones: Son grupos de operaciones que deben dar la apariencia de ser ejecutadas secuencialmente como una unidad. La definición de 'correcta' en una transacción se refiere al cumplimiento de las propiedades ACID.