Universidad de las Ciencias Informáticas
Facultad 2

TELEIDENTIFICADOR PERSONAL: DISEÑO DE LA ARQUITECTURA DE LA PLATAFORMA MANEJADORA DE PETICIONES

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autores: Mayrilis Castillo Suárez
Daymi Sablón Marrero

Tutor: Lic. Darian Horacio Grass Boada

Ciudad de la Habana
Junio/2009

Agradecimientos

A todas las personas que hemos ido a consultar y que de una forma u otra esclarecieron nuestras dudas y contribuyeron a la realización de esta tesis.

A Pavel y Abelito por todo el apoyo que nos brindaron durante este curso tanto desde el punto de vista profesional como en el plano personal, por confiar en nosotras, por soportarnos todo este tiempo y por darnos fuerzas y hacernos reír en los momentos más difíciles.

A Denys por alentarnos a seguir adelante y ayudarnos en cuanto estuvo a su alcance.

A nuestro tutor Darian por guiarnos por el camino correcto y ayudarnos siempre que lo necesitamos.

A nuestros compañeros de estudios que han estado presentes en los buenos y malos momentos.

A todos aquellos que han formado parte de nuestras vidas y que han contribuido de una forma u otra a nuestra formación.

Dedicatoria

Daymí:

A mis padres Nancy y Pedro por ser mis principales guías e inspiración. Por ser tan maravillosos e incondicionales. Por todo sus sacrificios para brindarme todo lo que ha estado a su alcance. Por tanto cariño y comprensión.

A mi hermana Iliana por ser tan buena y cariñosa conmigo. Por constituir mi principal ejemplo a seguir.

A mi tía Nery por ser como una segunda madre para mí. Por estar todo el tiempo pendiente de mis necesidades y bienestar. Por apoyarme en todo momento.

A mi Tito por ser tan especial y brindarme tanto apoyo y amor durante todos estos años de carrera universitaria. Por darme aliento y fuerzas cuando lo necesitaba.

A toda mi familia en general por su cariño y ayuda.

A la familia de mi novio por todo su cariño y apoyo. Por preocuparse tanto por mí.

Mayrilis:

A mi mamá Caridad y a mi papá Agustín por todos los sacrificios que siempre han hecho por mí, por todo su cariño, sus enseñanzas, por siempre haber estado para mí cuando los he necesitado y porque gran parte de lo que soy se los debo a ellos.

A la hermanita más linda que tengo, Maylet, por ayudarme en todo cuanto ha podido y por preocuparse por mí.

A toda mi familia por su apoyo y cariño a lo largo de estos años.

A mis amigas Lumy, Mare y Yely y a mis amigos Lelo y Abdanys, por formar parte de algunos de mis mejores recuerdos y por estar dentro de las personas en quienes más confío y con quienes más cuento.

A todos los profesores que dejaron una huella en mí por sus grandes enseñanzas, tanto en el plano profesional como personal, y que tanto contribuyeron a mi formación.

A mis vecinos que desde pequeña me han ayudado.

Resumen

La Empresa de Telecomunicaciones de Cuba (ETECSA) desea poner en práctica el desarrollo de un novedoso servicio basado en el protocolo ENUM. Con este propósito surge el proyecto TeleIdentificador Personal en el cual se implementarán un conjunto de soluciones informáticas que darán soporte al desarrollo de este servico en el país. Debido a la diversidad tecnológica, la posibilidad de incremento de estas aplicaciones y la continua interacción que requieren las mismas con el repositorio de datos (base de datos y servidor DNS), se implementará un sistema que funcione como una capa de abstracción entre estas aplicaciones y la fuente de datos, denominado Plataforma Manejadora de Peticiones.

En el presente trabajo se describe la arquitectura base sobre la que se desarrollará la Plataforma Manejadora de Peticiones. Para ello se realiza un estudio de los principales aspectos a tener en cuenta para diseñar una arquitectura de software. Se especifican las principales herramientas y tecnologías a utilizar en el desarrollo. Se describe la estructura general de la arquitectura y de forma más detallada se representa a través de un conjunto de vistas que pretenden modelar diferentes aspectos del sistema. Además se proponen mecanismos para tratar aspectos transversales como la seguridad, el tratamiento de excepciones y la auditoría.

Palabras Claves: ENUM, DNS, TeleIdentificador Personal, Plataforma Manejadora de Peticiones, arquitectura de software.

Índice

INTRODUCCIÓN 1
Capitulo1 Fundamentación Teórica 4
1.1 Introducción 4
1.2 ENUM en Cuba 4
1.3 Arquitectura de Software 5
1.3.1 Definición 5
1.3.2 Estilos Arquitectónicos 6
1.3.2.1 Estilo arquitectónico propuesto para la Plataforma Manejadora de Peticiones 12
1.3.3 Patrones y la Arquitectura de software 13
1.3.4 Lenguajes de Descripción de Arquitecturas 14
1.3.5 El Lenguaje Unificado de Modelado 15
1.3.6 Atributos de calidad y Arquitectura de software 17
1.3.7 Importancia de la Arquitectura de Software 18
1.4 Metodología de desarrollo de software 19
1.4.1 Metodología utilizada 21
1.5 Tecnologías para el desarrollo de software 22
1.5.1 Tecnología Java 22
1.5.2 Python 24
1.5.3 Tecnología utilizada 25
1.6 Conclusiones 25
Capítulo 2 Ambiente de Desarrollo 27
2.1 Introducción 27
2.2 Herramientas Horizontales 27
2.3 Herramientas verticales 28
2.4 Frameworks 31
2.5 APIs 36
2.6 Conclusiones 37
Capítulo 3 Diseño de la Arquitectura 38
3.1 Introducción 38
3.2 Objetivos y Restricciones Arquitectónicas 38
3.3 Patrones aplicados 40
3.4 Estructura general de la arquitectura 44
3.5 Vistas arquitectónicas 47
3.5.1 Vista de Casos de Uso 47
3.5.2 Vista Lógica 51
3.5.2.1 Visión general de la arquitectura – Alineamiento de paquetes y capas 51
3.5.2.2 Descripción de los paquetes 53

3.5.3 Vista de Implementación 58
3.5.4 Vista de Despliegue 62
3.5.5 Vista de Datos 65
3.6 Conclusiones 69
Capítulo 4 Aspectos Transversales 70
4.1 Introducción 70
4.2 Tratamiento de excepciones 70
4.3 Seguridad 72
4.3.1 Inyección SQL 76
4.4 Auditoría 79
4.5 Conclusiones 79
CONCLUSIONES GENERALES 81
RECOMENDACIONES 82
Referencias Bibliográficas 83
Glosario de Términos 85
ANEXOS 88

INTRODUCCIÓN

El avance en las Tecnologías de la Informática y las Comunicaciones (TIC) ha cambiado el modelo de integración de las redes de telefonía clásicas y las que ofrecen acceso a diversos sistemas de información. Internet y la red Telefónica, se diseñaron y se construyeron para posibilitar respectivamente la transmisión de datos y voz. Con el paso del tiempo la telefonía se ha digitalizado e Internet ha incorporado aplicaciones que soportan una variedad de formatos audiovisuales permitiendo, sobre la base del protocolo TCP/IP, nuevos horizontes en una red global.

Uno de los desafíos técnicos, 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. En general, se pretende que exista un plan de acceso de abonados integrado mundialmente[1].

Ese proceso de integración ha llevado a definir un nuevo protocolo denominado ENUM (del inglés: Telephone Numbering Mapping), desarrollado por el Grupo de Trabajo en Ingeniería de Internet (sus siglas en inglés IETF[1]). ENUM utiliza los números telefónicos E.164[2] (Plan Internacional de Numeración de las Telecomunicaciones Públicas) y los correlaciona con los identificadores de recursos uniformes URI (del inglés: Uniform Resource Identifier) almacenados en las bases de datos jerárquicas y físicamente distribuidas del Sistema de Nombres de Dominio o DNS (del inglés: Domain Naming System)[1]. El concepto sobre el cual se desarrolló el ENUM es el de “un número para todos los servicios” de las telecomunicaciones.

El desarrollo e implantación de ENUM es un reto difícil tanto a nivel nacional como internacional. Algunos países desarrollados lo han implementado en un período de tiempo no menor de tres años[2]. La Empresa de Telecomunicaciones de Cuba (ETECSA) desea poner en práctica el desarrollo de ENUM, pues constituye la base de nuevos servicios en los cuales no se tiene experiencia práctica. Para ello se necesita

[1] El IETF (*Internet Engineering Task Force*, en castellano Grupo de Trabajo en Ingeniería de Internet) es una organización internacional abierta de normalización, formada básicamente por técnicos en Internet e informática cuya misión es velar porque la arquitectura de la red y los protocolos técnicos que unen a millones de usuarios de todo el mundo funcionen correctamente.

[2] E.164, recomendación de la Unión Internacional de Telecomunicaciones (UIT) que asigna a cada país un código numérico (código de país) usado para las llamadas internacionales.

de un grupo de soluciones informáticas que soporten el desarrollo e implantación del servicio en el país. Dichas soluciones informáticas serán implementadas usando diversas tecnologías y requerirán realizar peticiones continuas a la base de datos y al servidor DNS. Por esta razón se hace necesaria la existencia de un software que funcione como una capa de abstracción entre las aplicaciones y la fuente de datos que permita el acceso de forma estándar a la información.

Tomando como referencia lo expuesto anteriormente, se tiene el siguiente **problema científico:** ¿Cómo estructurar los componentes de un sistema que funcione como una capa de abstracción entre aplicaciones distribuidas y la fuente de datos, que permita la implantación del servicio ENUM de usuario en Cuba? El **objeto de estudio** está enmarcado en las arquitecturas de software para el desarrollo de aplicaciones empresariales. El **campo de acción** del presente trabajo se centra en el modelo arquitectónico aplicable al desarrollo del servicio ENUM de usuario en Cuba. Como **objetivo general** se tiene diseñar una arquitectura robusta y flexible de un sistema que funcione como una capa de abstracción entre aplicaciones distribuidas y la fuente de datos, que permita la implantación del servicio ENUM de usuario en Cuba.

Para dar cumplimiento al objetivo planteado se desarrollaron las siguientes **tareas de investigación**:

- Investigar las diferentes arquitecturas y estilos arquitectónicos a partir de la consulta de bibliografía especializada.
- Definir las herramientas y tecnologías a utilizar para el desarrollo de aplicaciones sobre la plataforma JEE, en la Plataforma Manejadora de Peticiones.
- Desarrollar una propuesta de diseño de las capas lógicas de la aplicación que cumpla con buenas prácticas y patrones de diseño.
- Definir propuesta de estructura de paquetes dentro de la arquitectura de la aplicación.
- Definir mecanismos de seguridad para la Plataforma Manejadora de Peticiones.

Para la realización de las tareas de investigación se emplearon los siguientes **métodos**:

Métodos teóricos:

Analítico – Sintético: para analizar los diferentes aspectos relacionados con la arquitectura de software y extraer los elementos más importantes a tener en cuenta en el modelo arquitectónico propuesto.

Histórico – Lógico: para determinar las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos.

Métodos empíricos:

Entrevista: a personas que han desempeñado el rol de arquitectos en proyectos productivos y que cuentan con experiencia en el desarrollo de arquitecturas de software de aplicaciones empresariales.

El presente trabajo está estructurado en cuatro capítulos:

Capítulo 1: Fundamentación Teórica

En este capítulo se pretende proporcionar los aspectos principales relacionados con la arquitectura de software, partiendo de los conceptos fundamentales para su mejor entendimiento. Análisis de las metodologías y tecnologías candidatas para el desarrollo de software.

Capítulo 2: Ambiente de Desarrollo

Se definen las herramientas y tecnologías a utilizar para el desarrollo de la Plataforma Manejadora de Peticiones.

Capítulo 3: Diseño de la Arquitectura

Se describe la arquitectura base mediante la representación de las vistas propuestas por la metodología de desarrollo de software Rational Unified Process (RUP).

Capítulo 4: Aspectos Transversales

Se proponen mecanismos para tratar aspectos transversales del sistema, como tratamiento de excepciones, seguridad y auditoría.

Capítulo 1
Fundamentación Teórica

1.1 Introducción

En el presente capítulo se realiza una breve descripción de las aplicaciones requeridas por ETECSA para la implantación del servicio ENUM en Cuba. Se hace un estudio de los principales aspectos arquitectónicos como los estilos, patrones, lenguajes de descripción de arquitectura, así como la importancia de la arquitectura en el proceso de desarrollo de software. Se analizan dos de las principales metodologías de desarrollo de software, seleccionándose la metodología a utilizar para en el proyecto y se realiza una caracterización de algunas tecnologías candidatas para la implementación de la plataforma.

1.2 ENUM en Cuba

La Empresa de Telecomunicaciones de Cuba (ETECSA) es una organización que tiene como objeto social la prestación de los servicios públicos de telecomunicaciones en todo el territorio nacional. Esta empresa tiene una alta responsabilidad en el desarrollo socio-económico del país y en especial, en la informatización de la sociedad. Como parte de la estrategia para lograr sus propósitos se encuentra el desarrollo del servicio ENUM de usuario en Cuba.

Para el desarrollo teórico práctico de este novedoso servicio por parte de ETECSA se han dado algunos pasos, destacándose hasta la fecha la confección del documento “Proyecto Desarrollo de ENUM de Usuario en la Intranet de ETECSA” que sirve de guía y posibilitará el despliegue profesional de este servicio por la Empresa [3].

Se han presentado a la Universidad de las Ciencias Informáticas(UCI) los documentos que contienen, cumpliendo el acuerdo de colaboración firmado entre ETECSA y la UCI, las especificaciones de varias lógicas de servicios, que no son más que aplicaciones software para informatizar y poner en práctica el servicio ENUM de usuario en Cuba.

Las aplicaciones software solicitadas son:

1. **“Web desarrollo ENUM Intranet de ETECSA”:** Aplicación Web que a modo de página Web en la Intranet de ETECSA le dará soporte al sistema ENUM y en la que se publicarán: estado del proyecto, noticias, y servicios adicionales[2].
2. **“Web para registrador ENUM”:** Aplicación cliente servidor que será utilizada por el registrador para registrar, validar, revalidar y modificar preferencias del subscriptor[3] ENUM[2].
3. **“Aplicación Cliente ENUM (desktop)”:** Aplicación a instalar en la PC que funcionará como Cliente ENUM (aplicación cliente servidor) y que mostrará al usuario todos los contactos que tenga un subscriptor en la base de datos ENUM[2].
4. **“Locución de contactos de clientes ENUM mediante AGI Asterisk”:** Es una aplicación en lo fundamental para teléfonos PSTN[4] que permitirá a través de locuciones de voz interactuar con la base de datos ENUM obteniendo beneficios similares a un usuario de la Red IP[4].
5. **“Incorporar funcionalidad ENUM en Aplicaciones”:** Desarrollo de módulos o parches de software para incorporar la funcionalidad ENUM en aplicaciones existentes, así como la inclusión de la funcionalidad ENUM en nuevas aplicaciones a desarrollar[5].
6. **“Software Aplicación ENUM en los terminales celulares”:** Consiste en una aplicación que permite visualizar en la pantalla de un celular la aplicación ENUM desktop o traída desde la WEB y consultar la base de datos ENUM para conocer los contactos del subscriptor seleccionado[6].

1.3 Arquitectura de Software

1.3.1 Definición

Debido a la reciente aparición de la disciplina arquitectura de software, existen aún diferentes criterios respecto a su definición.

Una definición reconocida es la de Clements[7]: La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes

[3] Persona que contará con el servicio ENUM.
[4] PSTN (Public Switched Telephone Network): Red pública de telefonía conmutada - es la concentración de las redes públicas mundiales de circuitos conmutados.

según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.

Los autores Bass, Clements y Kazman[8] indican que: ¨La arquitectura de software de un programa o sistema computacional es la estructura o estructuras del sistema, que comprende componentes software, las propiedades extremadamente visibles de esos componentes y las relaciones entre ellos¨.

Según Shaw y Garlan [9]: ¨…la arquitectura de software implica la descripción de los elementos desde los que se construyen los sistemas, las interacciones entre esos elementos, patrones que guían su composición y restricciones en esos patrones¨.

Atendiendo a la recomendación IEEE-1471, arquitectura de software se entenderá como "la organización fundamental de un sistema encarnada en sus componentes, las relaciones de los componentes con cada uno de los otros y con el entorno, y los principios que orientan su diseño y evolución" [10].

Aunque existen diferentes criterios en cuanto a las definiciones de arquitectura de software, todos convergen en que es la estructura de alto nivel del sistema, estructura consistente en componentes y las relaciones entre ellos.

1.3.2 Estilos Arquitectónicos

Los estilos arquitectónicos son uno de los aspectos fundamentales de la disciplina arquitectura de software. Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica donde se conjugan componentes, conectores, configuraciones y restricciones. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales.

Mary Shaw y Paul Clements [11] identifican los estilos arquitectónicos como un conjunto de reglas de diseño que identifican las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo.

En la literatura existente del tema se pueden encontrar diversos catálogos de estilos, clasificados a su vez

en 5 ó 6 clases. A continuación se realiza una breve descripción de algunos de los estilos y variantes más significativos[12].

Estilos de Flujo de Datos

Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Ejemplares de la misma serían las arquitecturas de tubería-filtros y las de proceso secuencial en lote.

Arquitectura de tubería y filtros

Descripción

- Cada componente tiene un conjunto de entradas y un conjunto de salidas.
- Un componente lee entradas y las transforma en salida.

Restricciones:

- Los filtros deben ser independientes.
- No deben compartir estado con otros filtros.
- Los filtros realizan la labor independientemente del flujo de entrada.

Ventajas

- Permite entender el sistema global en términos de la combinación de componentes.
- Soporta de buena manera la reutilización.
- Los filtros son independientes de sus vecinos.
- Facilidad de mantenimiento y mejora
- Facilidad de diagnóstico.
- Soportan la ejecución concurrente

Desventajas

- No aconsejable para cuando se necesita interactividad.
- Problemas de rendimiento ya que los datos se transmiten en forma completa entre filtros.

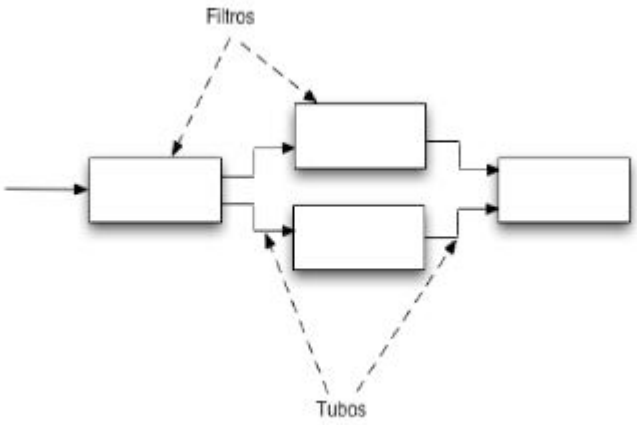

Figura 1: Estilo de tubos y filtros

Estilos Centrados en Datos

Esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

Arquitectura de Pizarra o Repositorio

En este estilo arquitectónico existen dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. Habitualmente un sistema de pizarra se implementa para resolver problemas en los cuales las entidades en forma individual son incapaces de alcanzar una solución, o para los que no existe una solución analítica, o para los que si existe pero no es viable dada la dimensión del espacio de búsqueda. El estilo de pizarra fue desarrollado en el contexto de la inteligencia artificial, donde cada componente o agente tenía información parcial y una nueva fuente podía ser agregada fácilmente.

Estilos de Llamada y Retorno

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas a gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

Modelo-Vista-Controlador

El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- Modelo. El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- Vista. Maneja la visualización de la información.
- Controlador. Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual.

Arquitectura en Capas

Garlan[5] y Shaw[6] definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas.

En un estilo en capas, los conectores se definen mediante los protocolos que determinan las formas de la interacción[13]. Las restricciones topológicas del estilo pueden incluir una limitación, más o menos rigurosa, que exige a cada capa operar sólo con capas adyacentes.

Arquitectura de tres capas:

La calidad tan especial de la arquitectura de tres capas consiste en aislar la lógica de la aplicación y en convertirla en una capa intermedia bien definida y lógica del software[14].

[5] Profesor en la Universidad Carnegie Mellon, donde lleva varios proyectos de investigación y es el Director de Programas de Ingeniería de Software Profesional. Dentro de sus áreas de investigación se encuentra la arquitectura de software.

[6] Profesora de Ciencias de la Computación y decana de los programas profesionales en la Universidad Carnegie-Mellon en Pittsburgh, Pennsylvania. Ha hecho grandes aportes en el campo de la informática y ha escrito varios libros sobre la arquitectura de software.

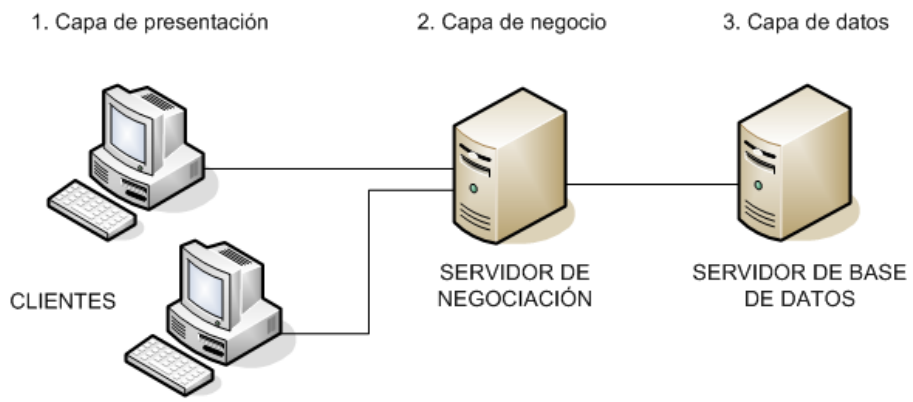

Figura 2: Ejemplo de arquitectura de tres capas.

Más allá de las tres capas: Arquitectura multicapas.

Cuando vemos la arquitectura desde la perspectiva de una descomposición más detallada, podemos prescindir de la designación "arquitectura de tres capas" y hablar en cambio de arquitecturas multicapas. Es posible agregar más capas y descomponer ulteriormente las ya existentes[14].

Resumiendo, las principales características de las arquitecturas en capas son:

- Organizado jerárquicamente en capas, donde cada capa provee servicios a la capa superior y es servido por la capa inferior.
- Los componentes son cada una de las capas.
- Los conectores son los protocolos de interacción entre las capas.

Restricciones:

- La interacción está limitada a las capas adyacentes.

Ventajas:

- Facilita la descomposición del problema en varios niveles de abstracción.
- Soporta fácilmente la evolución del sistema; los cambios sólo afectan a las capas vecinas.
- Se pueden cambiar las implementaciones respetando las interfaces con las capas adyacentes.

Desventajas:

- No todos los sistemas pueden estructurarse en capas.
- A menudo es difícil encontrar la separación en capas adecuada.

Entre los motivos por los cuales se recurre a la arquitectura multicapas se cuentan los siguientes[14]:

- Aislamiento de la lógica de aplicaciones en componentes independientes susceptibles de reutilizarse después en otros sistemas.
- Distribución de las capas en varios nodos físicos de cómputo y en varios procesos. Esto puede mejorar el desempeño, la coordinación y el compartir la información.
- Asignación de los diseñadores para que construyan determinadas capas; por ejemplo, un equipo que trabaje exclusivamente en la capa de presentación. Y así se brinda soporte a los conocimientos especializados en las habilidades de desarrollo y también a la capacidad de realizar actividades simultáneas en equipo.

Estilos de Código Móvil

Esta familia de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. Fuera de las máquinas virtuales y los intérpretes, los otros miembros del conjunto han sido rara vez estudiados desde el punto de vista estilístico.

Arquitectura de Máquinas virtuales

La arquitectura de máquinas virtuales está formada por cuatro componentes:

- Un motor de simulación o interpretación.
- Una memoria que contiene el código a interpretar
- Una representación del estado de la interpretación
- Una representación del estado del programa que se esta simulando

Ventajas:

- Solución software a problemas hardware.

Desventajas:

- No siempre es aplicable
- Reducido a lenguajes de programación

Estilos Peer-to-Peer

Esta familia, también llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar

mensajes a otras entidades, pero no controlarlas directamente. Miembros de la familia son los estilos basados en eventos, en mensajes, en servicios y en recursos.

Arquitectura Orientada a Servicios

Un servicio es una entidad de software que encapsula funcionalidad de negocios y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas. Los componentes del estilo (o sea los servicios) están débilmente acoplados. El servicio puede recibir requerimientos de cualquier origen. La funcionalidad del servicio se puede ampliar o modificar sin rendir cuentas a quienes lo requieran. Los componentes que requieran un servicio pueden descubrirlo y utilizarlo dinámicamente mediante UDDI[7] y sus estándares sucesores.

1.3.2.1 Estilo arquitectónico propuesto para la Plataforma Manejadora de Peticiones

Luego de realizar un análisis de los estilos arquitectónicos más representativos, se propone el estilo Arquitectura en Capas para la Plataforma Manejadora de Peticiones, pues proporciona las siguientes ventajas: primero que nada, el estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los desarrolladores la partición de un problema complejo en una secuencia de pasos incrementales. En segundo lugar, el estilo admite muy naturalmente optimizaciones y refinamientos. En tercer lugar, proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. Además tributa al desarrollo en paralelo del sistema y a la especialización en los conocimientos por parte de los desarrolladores.

[7] UDDI es uno de los estándares básicos de los servicios Web cuyo objetivo es ser accedido por los mensajes SOAP y dar paso a documentos WSDL, en los que se describen los requisitos del protocolo y los formatos del mensaje solicitado para interactuar con los servicios Web del catálogo de registros.

1.3.3 Patrones y la Arquitectura de software

La creación de software es una actividad humana que se viene desarrollando desde hace décadas y por tanto no es extraño que hayan surgido esquemas de soluciones a problemas planteados anteriormente. En este ámbito surge el concepto de patrón, cuya idea proviene de los trabajos del arquitecto Christopher Alexander[8], el cual da la siguiente definición "Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma."[15]. Es decir, un patrón resulta ser una solución a un problema, que se puede aplicar muchas veces, en distintas situaciones. Una definición de patrón adaptada al desarrollo del software es la dada por Richard Gabriel en "A Timeless Way of Hacking"[16], el cual expone que cada patrón es una regla de 3 partes, que expresa una relación entre un contexto, un sistema de fuerzas que ocurren repetidamente en ese contexto y una configuración de software que permite que se resuelvan esa fuerzas.

Construir una arquitectura es tanto una actividad donde desarrollar ideas nuevas como una oportunidad de usar la experiencia acumulada, por lo que los patrones constituyen un medio importante para la construcción de arquitecturas de software de alta calidad. Existen gran cantidad de patrones, los cuales se pueden clasificar según el nivel de abstracción al que se aplican; en Pattern Oriented Software Architecture[17] se dividen en tres niveles:

- Patrones arquitectónicos: los patrones de arquitectura expresan la estructura fundamental de un sistema de software. Cada actividad de desarrollo es gobernada por esta estructura.
- Patrones de diseño: la arquitectura de software consiste habitualmente en pequeñas unidades de arquitectura que se describen por los patrones de diseño. El objetivo de los patrones de diseño es el de capturar buenas prácticas que permitan mejorar la calidad del diseño de un sistema, encapsulando complejidad y haciéndolo más flexible.
- Patrones de idiomas: Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

[8] Arquitecto reconocido por sus diseños destacados de edificios en California, Japón y México. Creó y validó el término lenguaje de patrón, un método estructurado que pone la arquitectura al alcance de personas no especializadas profesionalmente en la materia, y que popularizó en su libro A Pattern Language.

Es importante destacar que esta división no es absoluta, dado que no incluye a otros patrones de amplia aceptación y utilización como los de análisis, integración de aplicaciones, pruebas, entre otros.

1.3.4 Lenguajes de Descripción de Arquitecturas

Un lenguaje de descripción de arquitectura (en inglés Architecture Description Language o ADL) es un lenguaje que proporciona elementos para modelar la arquitectura conceptual de un sistema software[9].
Los lenguajes de descripción de arquitecturas ocupan una parte importante del trabajo arquitectónico desde la fundación de la disciplina. Se trata de un conjunto de propuestas de variado nivel de rigurosidad, casi todas ellas de extracción académica, que fueron surgiendo desde comienzos de la década de 1990 hasta la actualidad, más o menos en contemporaneidad con el proyecto de unificación de los lenguajes de modelado bajo la forma de UML.

La siguiente tabla muestra un listado de los principales ADLs:

ADL	Fecha	Investigador - Organismo	Observaciones
Acme	1995	Monroe & Garlan (CMU), Wile (USC)	Lenguaje de intercambio de ADLs
Aesop	1994	Garlan (CMU)	ADL de propósito general, énfasis en estilos
ArTek	1994	Terry, Hayes-Roth, Erman (Teknowledge, DSSA)	Lenguaje específico de dominio - No es ADL
Armani	1998	Monroe (CMU)	ADL asociado a Acme
C2 SADL	1996	Taylor/Medvidovic (UCI)	ADL específico de estilo
CHAM	1990	Berry / Boudol	Lenguaje de especificación
Darwin	1991	Magee, Dulay, Eisenbach, Kramer	ADL con énfasis en dinámica
Jacal	1997	Kicillof , Yankelevich (Universidad de Buenos Aires)	ADL - Notación de alto nivel para descripción y prototipado
LILEANNA	1993	Tracz (Loral Federal)	Lenguaje de conexión de módulos
MetaH	1993	Binns, Englehart (Honeywell)	ADL específico de dominio

Rapide	1990	Rapide 1990 Luckham (Stanford)	ADL & simulación
SADL	1995	Moriconi, Riemenschneider (SRI)	ADL con énfasis en mapeo de refinamiento
UML	1995	Rumbaugh, Jacobson, Booch (Rational)	Lenguaje genérico de modelado –No es ADL
UniCon	1995	Shaw (CMU)	ADL de propósito general, énfasis en conectores y estilos
Wright	1994	Garlan (CMU)	ADL de propósito general, énfasis en comunicación
xADL	2000	Medvidovic, Taylor (UCI, UCLA)	ADL basado en XML

Tabla 1: Lenguajes de descripción de arquitectura[18].

Aunque el uso de ADLs es la técnica más potente para describir arquitecturas, presenta algunos problemas:

- No han tenido impacto en el ámbito industrial debido fundamentalmente a que no son lo bastante genéricos, no están estandarizados y hay pocas herramientas para soportarlos.
- Debido a sus orígenes en métodos formales son difíciles de comprender y no están incorporados en ningún proceso de desarrollo software.
- Algunos se encuentran especializados solo en un tipo particular de sistemas.
- Requieren una extensa capacitación.
- No son amigables para presentar la arquitectura a personas ajenas a la construcción del software.

1.3.5 El Lenguaje Unificado de Modelado

Las desventajas que presentan los lenguajes de descripción de arquitectura pueden ser superadas si se utiliza un lenguaje de modelado que sea conocido en la industria y que además esté apoyado por herramientas y metodologías de desarrollo, este lenguaje de modelado es UML.

El Lenguaje Unificado de Modelado (UML, del inglés: Unified Modeling Language) es una notación para especificar, visualizar y documentar sistemas de software desde la perspectiva de la orientación a objetos. Su objetivo es representar el conocimiento acerca de los sistemas que se pretenden construir y las decisiones tomadas durante su desarrollo, tanto lo referido a su estructura estática como a su comportamiento dinámico. UML postula un proceso de desarrollo iterativo, incremental, guiado por los casos de uso y centrado en la arquitectura [19]. La representación en UML de un sistema de software consta de cinco vistas o modelos parciales separados, aunque relacionados entre sí:

1. La vista de casos de uso, como la perciben los usuarios, analistas y encargados de las pruebas.
2. La vista de diseño que comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución.
3. La vista de procesos que conforman los hilos y procesos que forman los mecanismos de sincronización y concurrencia.
4. La vista de implementación que incluye los componentes y archivos sobre el sistema físico.
5. La vista de despliegue que comprende los nodos que forma la topología de hardware sobre la que se ejecuta el sistema.

Cada uno de estos modelos representa el sistema por medio de diversos diagramas. Aunque no existe de forma explícita una vista arquitectónica, estas cinco vistas pretenden describir, en su conjunto, la arquitectura del sistema [20].

Este lenguaje tiene una sintaxis y una semántica bien definidas, sirviendo además para todas las etapas de desarrollo. Por otra parte el lenguaje es extensible, de forma que pueden añadirse nuevas construcciones para abordar aspectos del desarrollo de software no previstos inicialmente en la notación.

En cuanto a su capacidad para describir la arquitectura, se puede señalar que UML maneja conceptos utilizados también por la Arquitectura del Software, como son los de interfaz, componente o conexión, y que los mecanismos de extensión disponibles pueden utilizarse para definir otros conceptos no contemplados o para establecer restricciones que definan de forma más precisa la semántica de estos conceptos.

Teniendo en cuenta lo expuesto anteriormente sobre los ADL y UML se decidió utilizar el Lenguaje Unificado de Modelado para la representación de la arquitectura de la Plataforma Manejadora de Peticiones, garantizándose una comunicación sencilla y el fácil entendimiento entre los diferentes

interesados en el proyecto.

1.3.6 Atributos de calidad y Arquitectura de software

Si sólo la funcionalidad de un sistema de software fuese importante a la hora de hacer su diseño, cualquier sistema monolítico podría servir. Sin embargo la realidad es otra. Lograr una arquitectura que satisfaga los requisitos de funcionalidad y calidad no es una tarea fácil. Las decisiones que se tomen en la etapa de diseño de la arquitectura tendrán un impacto decisivo sobre las cualidades del sistema resultante.

Se pueden clasificar los atributos de calidad del software en dos grandes grupos: aquellos que pueden comprobarse durante la ejecución del software y aquellos que no pueden comprobarse durante la ejecución[21].

Atributos de calidad observables durante la ejecución:

- Performance: Es el grado en el cual el sistema o componente lleva a cabo una funcionalidad especifica dada una restricción de velocidad, precisión y el uso eficiente de los recursos.
- Seguridad: La seguridad es una medida de la capacidad del sistema para resistir el uso no autorizado sin dejar de ofrecer sus servicios a los usuarios legítimos.
- Disponibilidad: Es la probabilidad de un sistema o componente de estar operativo cuando se necesite.
- Funcionalidad: Habilidad de un sistema de hacer la tarea para la cual fue creado.
- Usabilidad: Se refiere a cuán fácil es para el usuario dar cumplimiento a una tarea deseada, y al tipo de soporte al usuario, que el sistema proporciona.

Atributos de calidad no visibles durante la ejecución:

- Modificabilidad: Está relacionado con el costo del cambio.
- Portabilidad: Habilidad de un sistema software para ejecutarse en diferentes ambientes (hardware, software, o una combinación de ambos).
- Reusabilidad: Es la capacidad que tiene un sistema para que su estructura o alguna de sus componentes puedan ser usadas en futuras aplicaciones.
- Integrabilidad: Habilidad para hacer que piezas de software desarrolladas separadamente trabajen

correctamente juntas. La integrabilidad depende de: la complejidad de las componentes, mecanismos y protocolos de comunicación, claridad en la asignación de responsabilidades y calidad y completitud de la especificación de las interfaces.

- Testabilidad: Facilidad con la cual el software puede mostrar sus defectos (típicamente a través de pruebas de ejecución). Probabilidad de que, suponiendo que el software tiene al menos un defecto, fallará en la siguiente prueba.

Hacer un adecuado balance entre los diferentes atributos de calidad, es esencial para obtener un sistema que cumpla las expectativas de las distintas partes interesadas.

La arquitectura es fundamental para alcanzar numerosas cualidades de interés en un sistema; estas cualidades deben diseñarse y pueden ser evaluadas a nivel arquitectónico. La arquitectura, por sí misma, es incapaz de alcanzar cualidades pero proporciona la base para alcanzar la calidad[22].

1.3.7 Importancia de la Arquitectura de Software

La arquitectura de software constituye un elemento fundamental en el proceso de desarrollo de software ya que brinda un conjunto de elementos claves para lograr un producto con valor, no solo para los usuarios finales sino también para los desarrolladores.

Entre los principales valores prácticos aportados por la arquitectura de software se encuentran:

- Comunicación mutua. La arquitectura de software representa un alto nivel de abstracción común que la mayoría de los participantes, si no todos, pueden usar como base para crear entendimiento mutuo, formar consenso y comunicarse entre sí. En sus mejores expresiones, la descripción arquitectónica expone las restricciones de alto nivel sobre el diseño del sistema, así como la justificación de decisiones arquitectónicas fundamentales[23].
- Decisiones tempranas de diseño. La arquitectura de software representa la encarnación de las decisiones de diseño más tempranas sobre un sistema, y esos vínculos tempranos tienen un peso fuera de toda proporción en su gravedad individual con respecto al desarrollo restante del sistema, su servicio en el despliegue y su vida de mantenimiento[23].
- Reutilización, o abstracción transferible de un sistema. La arquitectura de software encarna un modelo relativamente pequeño, intelectualmente tratable, de la forma en que un sistema se

estructura y sus componentes se entienden entre sí; este modelo es transferible a través de sistemas; en particular, se puede aplicar a otros sistemas que exhiben requerimientos parecidos y puede promover reutilización en gran escala[23].

- Evolución. La arquitectura de software puede exponer las dimensiones a lo largo de las cuales puede esperarse que evolucione un sistema. Haciendo explícitas estas “paredes” perdurables, quienes mantienen un sistema pueden comprender mejor las ramificaciones de los cambios y estimar con mayor precisión los costos de las modificaciones. Esas delimitaciones ayudan también a establecer mecanismos de conexión que permiten manejar requerimientos cambiantes de interoperabilidad, prototipado y reutilización.
- Análisis. Las descripciones arquitectónicas aportan nuevas oportunidades para el análisis, incluyendo verificaciones de consistencia del sistema, conformidad con las restricciones impuestas por un estilo, conformidad con atributos de calidad, análisis de dependencias y análisis específicos de dominio y negocios.
- Administración. La experiencia demuestra que los proyectos exitosos consideran una arquitectura viable como un logro clave del proceso de desarrollo industrial. La evaluación crítica de una arquitectura conduce típicamente a una comprensión más clara de los requerimientos, las estrategias de implementación y los riegos potenciales.

1.4 Metodología de desarrollo de software

Todo desarrollo de software es riesgoso y difícil de controlar por lo que se hace necesario la adopción de una metodología de desarrollo de software que guíe el trabajo del equipo de desarrollo.

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de software. Habitualmente se utiliza el término “método” para referirse a técnicas, notaciones y guías asociadas, que son aplicables a actividades del proceso de desarrollo.

No existe una metodología de software universal. Las características de cada proyecto exigen que el proceso sea configurable, por lo que actualmente existen un conjunto de metodologías de desarrollo de software con características específicas. Entre las más representativas se encuentran: Rational Unified

Process (RUP) y Extreme Programming (XP).

Proceso Unificado de Desarrollo (RUP)

El proceso unificado de desarrollo (RUP) es una metodología para la ingeniería de software que va más allá del mero análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan el ciclo completo de desarrollo de software. El resultado es un proceso basado en componentes, dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental [24].

Características principales de RUP:

- Dirigido por Casos de Uso

Un caso de uso es un fragmento de funcionalidad del sistema que proporciona un resultado de valor a un usuario. Los casos de uso modelan los requerimientos funcionales del sistema y guían el proceso de desarrollo (diseño, implementación, y prueba). Basándose en los casos de uso los desarrolladores crean una serie de modelos de diseño e implementación que los llevan a cabo. De este modo, no solo inician el proceso de desarrollo sino que le proporcionan un hilo conductor.

- Centrado en la arquitectura

La arquitectura de un sistema software se describe mediante diferentes vistas del sistema en construcción. Incluye los aspectos estáticos y dinámicos más significativos del sistema. Los casos de uso y la arquitectura están profundamente relacionados. Los casos de uso deben encajar en la arquitectura, y a su vez esta debe permitir el desarrollo de todos los casos de uso requeridos, actualmente y en un futuro. El arquitecto desarrolla la forma o arquitectura a partir de la comprensión de un conjunto reducido de casos de usos fundamentales o críticos.

- Iterativo e incremental

Es práctico dividir el esfuerzo de desarrollo de un proyecto de software en partes más pequeñas o mini proyectos. Cada mini proyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos a crecimientos en el producto.

La metodología RUP divide en 4 fases el desarrollo del software:

- Inicio: El objetivo en esta etapa es determinar la visión del proyecto.
- Elaboración: En esta etapa el objetivo es determinar la arquitectura del sistema.
- Construcción: En esta etapa el objetivo es obtener la capacidad operacional inicial.

- Transición: El objetivo es llegar a obtener la versión lista para su instalación en las condiciones reales.

Cada fase se subdivide en iteraciones. En cada iteración se desarrolla en secuencia un conjunto de disciplinas o flujos de trabajos.

Extreme Programing (XP)

Es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizada para proyectos de corto plazo y corto equipo. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

La metodología XP se basa en:

- Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, realizando pruebas de las fallas que pudieran ocurrir. Es como adelantarse a obtener los posibles errores.
- Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.
- Cambiar los requerimientos en base a nuevos descubrimientos.

Lo fundamental en este tipo de metodología es:

- La comunicación entre los usuarios y los desarrolladores.
- La simplicidad al desarrollar y codificar los módulos del sistema.
- La retroalimentación concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

1.4.1 Metodología utilizada

Después de un análisis de las metodologías mencionadas anteriormente se decidió utilizar RUP, pues es una metodología que provee un alto nivel organizativo y aboga por una vasta documentación del proyecto. Para cada iteración, se hace exigente el uso de artefactos que describen desde diferentes perspectivas el

producto, lo que favorece en gran medida el mantenimiento y futuras extensiones que se le puedan realizar al mismo. Esta característica hace que la metodología sea adaptable para proyectos de largo plazo, como es el caso del presente proyecto. Otra de las ventajas que brinda es la existencia de diferentes elementos de planificación (plan de desarrollo, plan de iteración, plan de calidad, etc.) con los que se controla el desarrollo del software y se pueden reconocer previamente problemas y fallos de forma temprana y prevenirlos o corregirlos. Otro de los factores que influyó en al selección de la metodología fue el conocimiento por parte de los desarrolladores de sus especificaciones.

1.5 Tecnologías para el desarrollo de software

1.5.1 Tecnología Java

La tecnología Java es una revolucionaria plataforma informática presentada por Sun Microsystems[9] en 1995 orientada al desarrollo de software que permite realizar toda una variedad de aplicaciones.

El lenguaje de programación Java originalmente llamado "Oak", fue concebido bajo la dirección de James Gosling y Bill Joy. Oak nació para programar pequeños dispositivos electrodomésticos y un poco más adelante se utilizó para ejecutar aplicaciones para televisores. Ninguno de estos productos tuvo éxito comercial. Gosling y Joy se quedaron con una tecnología robusta, eficiente, orientada a objetos, independiente de la arquitectura, pero hasta ese momento, sin ninguna utilidad práctica.

No pasó mucho tiempo, cuando en Sun se dieron cuenta de que todas estas características cubrían a la perfección las necesidades de las aplicaciones de Internet. De esta manera, con unos cuantos retoques, Oak se convirtió en Java.

Cuando Nestcape Navigator anunció que incorporaría la tecnología Java, el nivel de interés sobre el lenguaje creció dramáticamente, debido al número importante de personas que utilizan WWW diariamente. Todo lo anterior se ha conjugado para lograr el éxito actual de Java, que constituye un intento de resolver simultáneamente todos los problemas que se le plantean a los desarrolladores de software por la proliferación de arquitecturas incompatibles, tanto entre las diferentes máquinas como entre los diversos sistemas operativos.

[9] Sun Microsystems es una empresa informática de Silicon Valley, fabricante de semiconductores y software.

La tecnología Java está compuesta por dos partes:

- El lenguaje de programación Java.
- La plataforma Java (que es una plataforma sólo de software y se ejecuta sobre las otras plataformas de hardware/ software).

La plataforma Java tiene 2 componentes:

- La máquina virtual de Java (JVM) que ejecuta el código resultante de la compilación del código fuente, conocido como bytecode[10], en instrucciones nativas de la plataforma destino.
- El Java API (Application Programming Interface), gran colección de componentes de software que proporcionan muchas utilidades para el programador.

Java posee tres tipos de subestándares, cada uno es un mundo de tecnología disponible para desarrollar familias de aplicaciones:

- Java Standard Edition (JSE), anteriormente conocido como Java 2 Standard Edition (J2SE) es una colección de APIs del lenguaje de programación Java útiles para muchos programas de la Plataforma Java.
- Java Micro Edition (JME), anteriormente conocido como Java 2 Micro Edition (J2ME), es una colección de APIs de Java para el desarrollo de software para dispositivos de recursos limitados, como PDA[11], teléfonos móviles y otros aparatos de consumo.
- Java Enterprise Edition (JEE) anteriormente conocido como Java 2 Enterprise Edition (J2EE) provee una especificación de cómo debe construirse una aplicación empresarial.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL[12], de acuerdo con las especificaciones del Java Community Process[13], de tal forma que prácticamente todo el Java de Sun es ahora software libre.

La versatilidad y eficiencia de la tecnología Java, la portabilidad de su plataforma y la seguridad que aporta, la han convertido en la tecnología ideal para su aplicación en redes. Se ha convertido en un

[10] El bytecode es un código intermedio más abstracto que el código máquina.
[11] Es un computador de mano originalmente diseñado como agenda electrónica
[12] GNU GPL es una licencia creada que está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre.
[13] El Proceso de la Comunidad Java es un proceso formalizado el cual permite a las partes interesadas a involucrarse en la definición de futuras versiones y características de la plataforma Java.

recurso inestimable, pues permite a los desarrolladores:

- Desarrollar software en una plataforma y ejecutarlo en prácticamente cualquier otra plataforma.
- Crear programas para que funcionen en un navegador web o como servicios web.
- Desarrollar aplicaciones para servidores como foros en línea, tiendas, encuestas, procesamiento de formularios HTML, etc.
- Desarrollar potentes y eficientes aplicaciones para teléfonos móviles y prácticamente cualquier tipo de dispositivo digital.

1.5.2 Python

Python es un lenguaje de programación creado por Guido Van Rossum a principios de los años 90. Es similar a Perl, pero con una sintaxis muy limpia y que favorece un código legible. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo. Permite escribir programas muy compactos y legibles. Los programas escritos en Python son típicamente mucho más cortos que sus equivalentes en C o C++, por varios motivos:

- Los tipos de datos de alto nivel permiten expresar operaciones complejas en una sola sentencia.
- El agrupamiento de sentencias se realiza mediante sangrado (indentación) en lugar de begin/end o llaves.
- No es necesario declarar los argumentos ni las variables.

En los últimos años el lenguaje se ha hecho muy popular, gracias a varias razones como:

- La cantidad de librerías que contiene, tipos de datos y funciones incorporadas en el propio lenguaje, que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero.
- La sencillez y velocidad con la que se crean los programas. Un programa en Python puede tener de 3 a 5 veces menos líneas de código que su equivalente en Java o C.
- La cantidad de plataformas en las que se puede desarrollar, como Unix, Windows, OS/2, Mac, entre otras.
- Además, Python es software libre.

1.5.3 Tecnología utilizada

Las dos tecnologías mencionadas anteriormente al igual que otras, permiten la implementación de la Plataforma Manejadora de Peticiones. Decidir cuál es la tecnología ideal para el desarrollo de una aplicación no resulta una tarea trivial. Decir que la implementación realizada utilizando una tecnología específica sería de mayor calidad y menor tiempo de desarrollo, estaría basado en suposiciones y no en pruebas concretas, pues como dijese el Ing. Medardo Rodríguez[14] un criterio como ese debe estar fundamentado por un estudio realizado por especialistas, basados en experiencias prácticas y estadísticas resultantes de estas experiencias.

Se propone la utilización de la tecnología Java para la implementación de la Plataforma Manejadora de Peticiones, pues es una tecnología robusta, flexible y probada. Consta de una amplia documentación y de una comunidad especializada de millones de desarrolladores. Además ofrece grandes prestaciones para el acceso a los servidores DNS.

Específicamente se desarrollará sobre la plataforma JEE, pues la misma define un estándar para desarrollar aplicaciones empresariales. Incluye las APIs del estándar JSE e incorpora soporte completo para Enterprise JavaBeans, servlets, Java Server Pages y varias tecnologías de servicios web. Esto permite crear aplicaciones empresariales portables entre plataformas y escalables, a la vez que integrables con tecnologías anteriores.

1.6 Conclusiones

En este capítulo se han examinado diferentes aspectos a tener en cuenta en el diseño de arquitecturas de software tales como estilos arquitectónicos, patrones, atributos de calidad de software (enfatizando el papel que juegan los mismos para lograr el desarrollo de arquitecturas robustas) y lenguajes de descripción de arquitecturas, entre otros elementos.

Luego de un análisis de los estilos arquitectónicos más representativos, los lenguajes de descripción de arquitectura, y las metodologías de desarrollo de software, se propuso el estilo Arquitectura en Capas, el Lenguaje Unificado de Modelado (UML) y el Proceso Unificado de Desarrollo de software (RUP) para el

[14] Consultor de la informatización de la prensa cubana. Activista del Software Libre. Casi siempre tiene proyectos de cursos de teoría de programación en universidades, dentro de las que se encuentra la UCI.

desarrollo de la Plataforma Manejadora de Peticiones; y debido a su versatilidad y eficiencia, la portabilidad de su plataforma y la seguridad que aporta, entre otros elementos, se seleccionó la tecnología Java para la implementación de la misma.

Capítulo 2
Ambiente de Desarrollo

2.1 Introducción

En el presente capítulo se realiza una descripción de las herramientas y tecnologías utilizadas en el desarrollo de la Plataforma Manejadora de Peticiones, especificando en cada caso su versión.

2.2 Herramientas Horizontales

Control de versiones:

Subversion 1.5.1

Subversion, también conocido como SVN, es un sistema de control de versiones, libre y de código abierto que se distribuye bajo una licencia de tipo Apache[15]. Es un sistema de gestión de archivos y directorios, cuya principal característica es que mantiene la historia de los cambios y modificaciones que se han realizado sobre ellos a lo largo del tiempo. Un árbol de los archivos se guarda en un repositorio central. El repositorio es como un servidor de ficheros ordinario, salvo que recuerda todos los cambios que se han hecho a sus ficheros y directorios. Se trata de un sistema general que puede ser utilizado para manejar cualquier colección de ficheros [25].

Gestión de proyecto:

Trac 0.11

Trac es una herramienta de código abierto con una interfaz web simple que integra herramientas para comunicación, gestión, seguimiento de proyecto, y gestión de la configuración. Permite la inserción de errores encontrados en el proyecto a través de órdenes de trabajo llamadas “tickets”. Se integra con el

[15] La licencia Apache es una licencia de software libre creada por la Apache Software Foundation (ASF). Permite el uso y distribución del código fuente para software libre y software propietario.

sistema de control de versiones Subversion, permitiendo chequear desde este ambiente todas las actualizaciones realizadas, así como mantener el control de acceso y la asignación de tareas usando los usuarios que ya se poseen para el control de versiones[26].

DotProject 2.0

DotProject es una herramienta que permite gestionar las distintas fases y tareas que componen un proyecto. Permite la gestión y planificación de proyectos en entornos colaborativos, así como la asignación de recursos a los mismos. Contiene vista de eventos y tareas en calendario, filtrado por varios elementos. Permite la visualización y generación de informes sobre los proyectos[27].

El grupo que desarrolla DotProject basa su espíritu de trabajo en los siguientes puntos:

• Proveer a los usuarios de funcionalidad orientada a la Gestión de Proyectos.

• Construir una herramienta con una interfaz de usuario simple, claro y consistente.

• Ser de código abierto, libre acceso y utilización.

2.3 Herramientas verticales

Herramientas de modelado:

Visual Paradigm Suite 2.3

Es una herramienta visual para el modelado UML. Está diseñada para una amplia gama de usuarios entre los que se incluyen ingenieros de software, analistas de sistemas, analistas de negocio, arquitectos y desarrolladores. Está orientada a la creación de diseños usando el paradigma de programación orientada a objetos. Con esta herramienta se puede generar código para los lenguajes PHP, JAVA y C# y para los gestores de base de datos DB2, Informix, SQL Server, MYSQL, Oracle y PostgreSql. Es multiplataforma a diferencia de otras herramientas de modelado como el Rational Rose.

ER/Studio 7.5.2

ER/Studio es una herramienta de modelado de datos fácil de usar para el diseño y construcción de bases de datos a nivel físico y lógico. La creación de diagramas es clara y rápida, teniendo la posibilidad de realizar diagramas con desempeño rápido. Esta herramienta ofrece las siguientes características: Ambiente de diseño orientado al modelo, soporte del ciclo de vida de bases de datos completas, administración del modelo empresarial, soporte para integración y almacenes de datos, y diseño de base

datos con calidad.

Entorno de desarrollo integrado (IDE):

Eclipse 3.4

Eclipse es un entorno de desarrollo integrado de código abierto y multiplataforma. Es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. La definición que da el proyecto Eclipse acerca de su software es: "una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular"[28]. Constituye una plataforma universal para integrar herramientas de desarrollo, con una arquitectura abierta y basada en plug-ins. La arquitectura de plug-ins permite integrar diversos lenguajes sobre un mismo IDE e introducir otras aplicaciones accesorias.

Servidor Web:

Apache Tomcat 6.0

Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat es un servidor web con soporte de servlets y JSPs. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Dado que fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual.

Se analizó como otra alternativa el servidor de aplicaciones JBoss, pues es un servidor de aplicación libre por excelencia, pero como todo servidor de aplicación su pieza fundamental es el contenedor de EJBs y en la arquitectura del sistema no está concebido el uso de los mismos. Además los servidores de aplicaciones son más difíciles de configurar, administrar y consumen más recursos que un contenedor web.

Sistema Gestor de Bases de Datos:

PostgreSQL 8.3

PostgreSQL es un Sistema Gestor de Bases de Datos Relacionales Orientadas a objetos, liberado bajo la

licencia BSD[16] o Berkeley Software Distribution. PostgreSQL es uno de los gestores de bases de datos de código abierto más avanzado hoy en día; soporta casi toda la sintaxis SQL y ofrece muchas características modernas[29]:

- consultas complejas (complex queries)
- claves foráneas (foreign keys)
- disparadores (triggers)
- vistas (views)
- integridad transaccional (transactional integrity)
- control de concurrencia multiversión (multiversion concurrency control)

Además, PostgreSQL puede ser ampliado por el usuario de muchas formas, por ejemplo, mediante la adición de nuevos:

- tipos de datos (data types)
- funciones (functions)
- operadores (operators)

Cuenda también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, perl, tcl y python) y debido a la licencia open source, puede ser usado, modificado y distribuido con cualquier propósito, ya sea privado, comercial, o académico.

Pruebas:

SoapUI 2.0.2

SoapUI es una aplicación de escritorio, libre y de código abierto que entre sus principales funcionalidades permite inspeccionar, invocar y desarrollar servicios web. Brinda la opción de hacer pruebas funcionales y de carga a los servicios.

SoapUI está especialmente dirigida a desarrolladores, consumidores o probadores de servicios web. Las pruebas funcionales y de carga a los servicios pueden ser realizadas ambas de forma interactiva o automática. Puede ser integrado con el IDE Eclipse mediante un plugin[30].

[16] Licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Pertenece al grupo de licencias de software Libre.

2.4 Frameworks

Spring 2.5.5

Spring es un framework de aplicación de código abierto que ayuda a hacer el desarrollo en JEE mucho más fácil y más productivo[31]. Ofrece servicios que pueden ser usados en un gran rango de entornos, a diferencia de otros frameworks como Struts, Spring se puede utilizar en cualquier aplicación no solo en aplicaciones web. Es un framework ligero por el mínimo impacto que tiene en las aplicaciones. Constituye una alternativa al uso de componentes EJB y de servidores de aplicaciones, pues provee servicios similares a estos como es el caso de manejos y la gestión de transacciones declarativamente, disminuyendo el tiempo y el esfuerzo en el desarrollo de las aplicaciones. El framework está basado en la técnica de Inversión de Control (IoC) o inyección de dependencia, la cual hace externa la creación y el manejo de dependencias de los componentes, permitiendo reducir el acoplamiento entre los objetos. Posee soporte para la programación orientada a aspectos (AOP), que permite separar la lógica de aplicación de los aspectos trasversales (como la auditoría y el manejo de transacciones)[32]. Otra de sus características importantes es la fácil integración que permite con otros frameworks como Hibernate, Spring Security, iBatis, entre otros.

Algunos de los principales valores de Spring que expone Rod Johnson, en su libro: “Professional Java Development with the Spring Framework” [31] se pueden resumir en:

- Provee un consistente modelo de programación, útil en cualquier ambiente: Muchas aplicaciones web simplemente no necesitan ser ejecutadas sobre un servidor pesado de aplicaciones, sino que es mejor ejecutarlas sobre un servidor web, como Tomcat o Jetty.
- 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 probar. La dependencia sobre las APIs de Spring normalmente serán en forma de interfaces que son fáciles para simular.
- Promueve la selección arquitectónica: Mientras Spring provee una columna vertebral 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.
- No reinventa la rueda: Spring no introduce una solución propia en áreas tales como ORM donde ya existen muy buenas soluciones, Hibernate, iBatis, JDO, OBJ, TopLink. Tampoco implementa su propia solución de abstracción para poner en bitácoras (logging), pool de conexiones, coordinador de transacciones distribuidas, protocolos remotos u otros sistemas de servicios que son ofrecidos.

Spring Web Services 1.5.5

Spring Web Services (Spring-WS) es un producto de la comunidad de Spring centrada en la creación de servicios web. Spring Web Services sigue el enfoque de la creación del contrato primero del servicio, tributando a una mayor flexibilidad de los servicios web. Este enfoque constituye una buena práctica a la hora de desarrollar servicios web pues permite un bajo acoplamiento entre el contrato y la implementación del servicio.

El framework tiene soporte para varias tecnologías y APIs de manejo de contenidos de los mansajes XML. Brinda soporte al estándar Web Service Security permitiendo firmar mensajes SOAP, encriptar y desencriptar estos mensajes, así como la autenticación a través de los mismos. El producto se basa en Spring, lo que significa que pueden usarse los conceptos de Spring tales como la inyección de dependencia. Spring-WS usa los contextos de aplicaciones de Spring para toda la configuración, y su arquitectura se asemeja a la de Spring-MVC[33].

iBATIS 2.3.0.677

iBatis es un framework que facilita la implementación de la capa de persistencia utilizada en las aplicaciones. Es conocido como un *data mapper*, lo que significa que mueve los datos entre los objetos y la base de datos, manteniéndolos independientes.

Se divide en dos elementos[34]:

- iBatis Data Mapper (SQL Maps)**:** proporciona un modo simple y flexible de mover los datos entre los objetos Java y la base de datos relacional. SQL Maps reduce considerablemente la cantidad de código que normalmente se necesita para acceder a una base de datos relacional. Este framework mapea clases a sentencias SQL usando un descriptor XML muy simple. De este modo, se tiene toda la potencia de SQL sin una línea de código JDBC[17].
- iBatis Data Access Objects (DAO): Implementación realizada por el framework del patrón DAO. iBatis DAO es una capa de abstracción que oculta los detalles de la capa de persistencia y proporciona un API común para el resto de la aplicación. El uso de DAOs permite configurar una aplicación dinámicamente para usar distintos mecanismos de persistencia.

Dentro de las características de iBatis se encuentran [35]:

- Simplicidad: iBatis es ampliamente considerado como uno de los frameworks de persistencia más simples disponibles en la actualidad. La simplicidad es la esencia de los objetivos de diseño del framework. Esta simplicidad está dada por la sólida base sobre la cual se construye iBatis: JDBC y SQL. iBatis es fácil para los desarrolladores de Java, pues trabaja de manera semejante a JDBC, pero con mucho menos código y un número de beneficios arquitectónicos que JDBC no tiene. Es además fácil de entender por desarrolladores con conocimientos previos y experiencia en el uso de SQL.
- Productividad: La finalidad primordial de cualquier buen framework es hacer a los desarrolladores más productivos; meta que es alcanzada exitosamente por el framework iBatis. En un caso de estudio presentado por un grupo de usuarios de Java en Italia donde la capa de acceso a datos completa se migró desde JDBC a iBatis, se encontró que el último redujo la cantidad de código en la capa de persistencia a un significativo 62 por ciento. Esta facilidad unida a una curva de aprendizaje baja, tributan a una mayor productividad de los desarrolladores disminuyéndose el tiempo de desarrollo de los proyectos.

[17] JDBC (Java Database Connectivity): API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java.

- 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. iBatis posibilita separar el trabajo del programador de SQL y el programador Java manteniendo la consistencia que se requiere.
- Separación de incumbencias: iBatis gestiona todos los recursos relacionados con la persistencia como las conexiones a la base de datos, los Prepared Staments[18] y los ResultSet[19]. Proporciona una interfaz independiente de la base de datos y APIs que ayudan al resto de la aplicación a permanecer independiente de cualquier recurso de persistencia. Con iBatis siempre se trabaja con los objetos específicos del dominio de la aplicación y nunca directamente con los ResultSet.
- Software Libre: iBatis es software libre, de código abierto, desarrollado por la Apache Software Foundation.
- IBatis brinda soporte a muchas características de herramientas ORM como almacenamiento en caché, carga retardada (lazy loading), generación de código en tiempo de ejecución, herencia y un manejo avanzado de transacciones.
- Las bases de datos que gestiona iBatis deben ser exclusivamente relacionales.

Spring Security 2.0.3

Spring Security es un framework de seguridad para las aplicaciones basadas en Spring. Proporciona una solución completa a la seguridad al implementar servicios de autenticación y autorización a recursos, tanto a nivel de peticiones web como a nivel de invocación de métodos. Sobre la base de Spring, Spring Security aprovecha al máximo las técnicas de inyección de dependencias y orientación a aspectos [36].

Acegi Security es ahora Spring Security, el proyecto de seguridad oficial para las aplicaciones de Spring. Spring Security 2.0.0 es la primera versión pública bajo el nuevo nombre; se basa en los sólidos cimientos de Acegi Security, añadiendo nuevas características dentro de las que se encuentran [37]:

- Sintaxis de configuraciones más simple (Las antiguas configuraciones podrían requerir cientos de líneas de XML).

[18] Es una técnica que permite separar de una manera segura los datos pasados en una consulta SQL de la consulta en sí. Se mantiene la estructura de la consulta y se cambia solamente el dato que se requiera.
[19] Conjunto de registros de una base de datos, así como la meta-información acerca de la consulta.

- Soporte a OpenID[20].
- Mayor soporte a Web Service Security a través de la versión 1.5 de Spring Web Service.

Al asegurar las aplicaciones web, Spring Security utiliza filtros de servlet que interceptan las peticiones a servlet para realizar la autenticación y hacer cumplir la seguridad. Spring Security emplea un mecanismo único para declarar filtros de servlet que posibilita inyectarlos con sus dependencias, utilizando la inyección de dependencia de Spring. También puede aplicar seguridad a un nivel más bajo, al asegurar la invocación a métodos. Cuando asegura métodos Spring Security usa Spring AOP, aplicando aspectos que aseguran que el usuario tiene los permisos requeridos para invocar los métodos seguros. En cualquier caso, si se necesita únicamente seguridad a nivel de peticiones web o si se requiere seguridad a nivel de método, Spring Security emplea cinco componentes para garantizar la seguridad como se muestra en la siguiente figura:

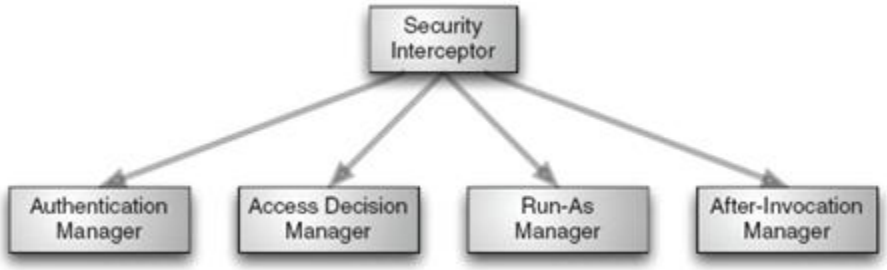

Figura 3: Elementos fundamentales de Spring Security

- Security Interceptor: intercepta el acceso a los recursos para garantizar la seguridad. Realmente, no aplica las reglas de seguridad, en su lugar delega esta responsabilidad a otros manejadores (representados en la parte inferior de la figura anterior).
- Authentication manager: Es el encargado del proceso de autenticación.
- Access decision manager: Decide si se está autorizado a utilizar determinado recurso teniendo en cuenta la información de autenticación y los atributos de seguridad que han sido asociados al recurso seguro.
- Run-as manager: Puede ser usado para reemplazar una autenticación por otra que permita acceder a los objetos seguros que están en lo profundo de la aplicación.

[20] OpenID es un sistema de identificación digital descentralizado, con el que un usuario puede identificarse en una página web a través de una URL y puede ser verificado por cualquier servidor que soporte el protocolo.

- After Invocation Manager: Garantiza la seguridad después que el recurso seguro ha sido accedido.

Otras ventajas ofrecidas por Spring Security son:

- Soporta diversos proveedores de autenticación (autenticación contra una base de datos, contra un servidor LDAP[21], usando un certificado X.509, entre otros) que pueden utilizarse combinados de manera que si falla la autenticación contra un proveedor, se pueda intentar con otro.
- Soporta diversas formas de autenticación (el usuario se puede autenticar mediante una ventana de login en el navegador usando http basic authentication; o mediante un formulario en una página de login; o a través de un certificado X.509, entre otros).
- Provee configuración de protección del canal, permitiendo redireccionar hacia un canal adecuado (HTTP o HTTPS) en dependencia de la solicitud.

JUnit

Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas.
JUnit es un conjunto de clases que 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 es un framework que automatiza la ejecución de pruebas unitarias al software en entornos Java.

2.5 APIs

dnsjava

El dnsjava es un API que posibilita el acceso a servidores DNS en Java. Soporta todos los tipos de registros DNS definidos. Puede ser utilizada para consultas, transferencias de zona y actualizaciones dinámicas. Ofrece tanto un alto como un bajo nivel de acceso a los DNS. Las funciones de alto nivel realizan consultas a los registros retornando las respuestas o el motivo del fallo. Las funciones de bajo nivel permiten la manipulación directa de los mensajes y registros de DNS. El dnsjava es en resumen, una herramienta simple para hacer búsquedas de DNS (DNS lookups) y actualizaciones dinámicas.

[21] Es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio.

2.6 Conclusiones

Mediante la descripción de las herramientas y tecnologías a utilizar en el desarrollo del sistema se pueden apreciar los beneficios que aportan cada una de ellas. En el caso de las herramientas horizontales permiten gestionar y controlar el desarrollo del sistema. Las herramientas y tecnologías verticales, así como los frameworks facilitan el trabajo de los desarrolladores.

Capítulo 3
Diseño de la Arquitectura

3.1 Introducción

En el presente capítulo se proporciona una visión arquitectónica general del sistema y luego se detalla, mediante una serie de vistas arquitectónicas para representar diferentes aspectos del sistema: una vista del modelo de casos de uso, una lógica, una del modelo de despliegue, una del modelo de implementación y una del modelo de datos.

3.2 Objetivos y Restricciones Arquitectónicas

En esta sección de realiza una descripción de los requerimientos y objetivos del software que tienen un impacto significativo en la arquitectura.

Seguridad

Los datos a acceder y modificar mediante los servicios web expuestos por la plataforma constituyen datos generalmente personales y de vital importancia para el correcto funcionamiento del servicio ENUM de usuario en Cuba, por lo que se debe garantizar la seguridad de los mismos y restringir el acceso a estos servicios web.

Usabilidad

Puesto que la interacción con la plataforma será a través del consumo de los servicios web se necesitan conocimientos intermedios de programación y en el manejo de servicios web.

Fiabilidad

Disponibilidad

El sistema tiene que estar disponible un 99.99 % del tiempo, ya que gestiona servicios de telecomunicaciones.

Exactitud

Toda la información que se gestiona a través del sistema es de carácter personal por lo que debe ser debidamente verificada, ser 100 % exacta y real.

Errores

Se identifican 2 tipos de errores significativos: errores menores (tiempo excedido, error en algún parámetro) y errores críticos (pérdida de conexión con la BD, DNS).

Portabilidad

El sistema deberá correr sobre cualquier Sistema Operativo.

Soporte

1. Se debe generar un documento detallado que explique el funcionamiento del sistema.
2. Se debe cumplir con las pautas de codificación establecidas para el proyecto TeleIdentificador Personal (ver Anexo 1).
3. 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.

Restricciones del diseño e implementación

1. Lenguaje de programación: Java
2. Se utilizarán las tecnologías que brindan los frameworks definidos para cada una de las capas del sistema:
    - Para la capa de servicios web: Framework Spring Web Service.
    - Para la capa del negocio: Framework Spring.
    - Para la capa de Acceso a Datos: Framework iBatis.

Interfaces de Comunicación

1. La comunicación con el sistema se hará a través del protocolo SOAP.
2. La comunicación con la BD: TCP/IP.
3. La comunicación con el DNS: TCP/IP.

Hardware

Los servidores (servidor donde se alojará la aplicación, servidor de base de datos y servidor DNS) deben cumplir los siguientes requerimientos:

- Se requiere tarjeta de red.
- Se requiere que tengan al menos 2 GB de RAM.
- Procesador 3.0 GHz o superior.

Software

Para el servidor donde se desplegará la Plataforma Manejadora de Peticiones:

- Máquina Virtual de Java versión 1.6
- Servidor Web Apache Tomcat versión 6.0.

Para los servidores de Base de Datos y DNS:

- Sistema Gestor de Base de Datos PostgreSQL versión 8.3.

3.3 Patrones aplicados

Inversión de Control

Problema

Cuando un componente necesita un recurso externo, como una referencia a otro componente, la forma más directa de acceso es a través de una búsqueda. El inconveniente de esta búsqueda es que el componente necesita saber acerca de la recuperación de recursos, a pesar de haber utilizado un servicio de localización para encapsular la lógica de búsqueda [38].

Solución

Una alternativa al problema en cuestión es aplicar Inversión de Control (IoC, del inglés Inversion of Control). La idea de este principio es invertir la dirección de la recuperación de recursos. En una búsqueda tradicional, los componentes buscan los recursos realizando peticiones a un contenedor y el contenedor devuelve debidamente los recursos en cuestión. Con la Inversión de Control el contenedor entrega activamente los recursos gestionados a sus componentes. Un componente sólo tiene que elegir una forma de aceptar los recursos [38].

Inversión de control es un principio que generalmente siguen los frameworks, en lugar de tener las clases de la aplicación el control sobre la creación de objetos, le asigna esta responsabilidad a un agente externo como un contenedor. Spring proporciona un contenedor que maneja el ciclo de vida de los objetos y resuelve las dependencias entre dichos objetos.

Se suele confundir los términos de Inversión de Control e Inyección de Dependencia, pero no son lo mismo. Mientras que la Inversión de Control es un principio de diseño general, la Inyección de Dependencia es un patrón de diseño concreto que encarna este principio. Como la Inyección de Dependencia es la más típica realización de la Inversión de Control, comúnmente estos términos se usan indistintamente[36].

Patrón Inyección de Dependencia

Contexto / Problema

Cualquier aplicación no trivial se compone de dos o más clases que colaboran entre sí para realizar alguna lógica de negocio. Tradicionalmente, cada objeto es responsable de obtener sus propias referencias de los objetos con los que colabora (y sus dependencias). Pero esto da lugar a un alto acoplamiento entre las clases y a que el código sea difícil de probar[36].

Solución

El patrón Inyección de Dependencias (DI, del inglés Dependency Inyection), es una técnica que busca facilitar la resolución de dependencias entre objetos. Consiste en inyectar objetos a una clase en lugar de ser la propia clase quien cree el objeto. La forma habitual de implementar este patrón es mediante un contenedor. El contenedor inyecta a cada objeto los objetos necesarios según las relaciones plasmadas en un fichero de configuración. Típicamente este contenedor es implementado por un framework externo a la aplicación (como Spring).

La arquitectura de Spring está basada en este patrón, Rob Harrop[22] lo define como: “Es la manera de externalizar la creación y el manejo de las dependencias de los componentes”.

Patrón Instancia Única (Singleton) [39]

Contexto

Varios clientes distintos precisan referenciar a un mismo elemento y es necesario que no haya más de una instancia de ese elemento.

Problema

¿Cómo lograr disponer globalmente de una instancia de una clase y a la vez garantizar que solamente una instancia de dicha clase sea creada?

Solución

Crear en la clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado).

El patrón Singleton provee una única instancia global gracias a que:

- La propia clase es responsable de crear la única instancia.
- Permite el acceso global a dicha instancia mediante un método de clase.
- Declara el constructor de clase como privado para que no sea instanciable directamente.

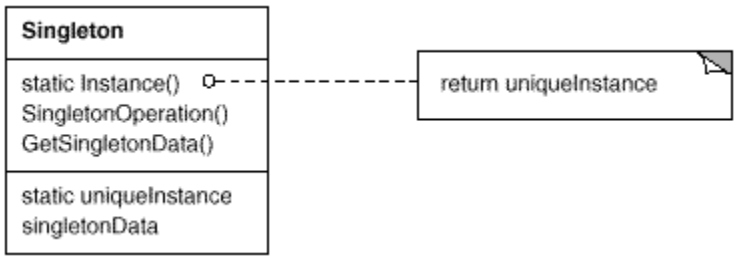

Figura 4: Patrón Singleton

Patrón DAO (Data Access Object)

Contexto

El acceso al almacenamiento persistente varía en gran medida dependiendo del tipo de almacenamiento (bases de datos relacionales, bases de datos orientadas a objetos, ficheros planos, etc.).

Problema

Comúnmente las aplicaciones necesitan utilizar datos persistentes. El acceso a estos datos varía en dependencia del mecanismo de almacenamiento. Cuando los componentes de negocio necesitan acceder a una fuente de datos, pueden utilizar el API apropiado para conseguir la conectividad y manipular la misma. Pero introducir el código de conectividad y de acceso a datos dentro de estos componentes genera un fuerte acoplamiento entre los mismos y el repositorio de datos. Dichas dependencias de código en los componentes hace difícil y tedioso migrar la aplicación de un tipo de fuente de datos a otra.

[22] Desarrollador del núcleo del framework Spring.

Solución

Utilizar un Data Access Object para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con el repositorio de datos para obtener y almacenar la información. Los componentes de negocio que tratan con el DAO utilizan una interfaz simple expuesta por este para sus clientes. El DAO oculta completamente los detalles de implementación de la fuente de almacenamiento a sus clientes. Como la interfaz expuesta por la clase de acceso a dato no cambia cuando cambia la implementación de la fuente de datos subyacente, este patrón permite al sistema adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a los componentes de negocio. Esencialmente, el DAO actúa como un adaptador entre las clases del negocio y la fuente de datos, desacoplando la lógica de negocio de la lógica de acceso a datos, de manera que se pueda cambiar la fuente de datos fácilmente.

Patrón Validador de Mensaje (Message Validator) [40]

Contexto

Un servicio web interactúa con otras aplicaciones a través de la red. Los datos esperados por el servicio pueden estar incorrectamente estructurados o haber sido transmitidos con fines maliciosos. También existe el riesgo de ataques de inyección cuando los datos de los mensajes entrantes son manipulados para incluir sintaxis adicional.

Problema

¿Cómo proteger los servicios web de contenido malicioso o incorrecto?

Cualquiera de las siguientes condiciones justifica el uso de la solución descrita en este patrón:

- **Contenido malicioso supone un riesgo para el servicio web**. Un atacante puede insertar sintaxis en el mensaje entrante para ocasionar que el servicio web que procese los datos recibidos, se comporte de una manera no deseada. El atacante puede hacer esto a través de ataques de inyección, tales como inyección de XML e inyección de SQL.
- **Existe el riesgo de que los atacantes eludan las técnicas de validación usadas por las aplicaciones clientes, mediante el uso de clientes alternativos o mediante la modificación de los datos después que han abandonado el cliente**. Los servicios web deben ser diseñados para ser autónomos y realizar su propia validación de entradas en lugar de confiar en la validación que es realizada en la aplicación cliente.

La siguiente condición es una razón adicional para utilizar la solución:

- **Un atacante puede usar un mensaje mal formado o de gran tamaño para lanzar un ataque de denegación de servicio**. Los ataques de denegación de servicio ocurren como resultado del incremento desproporcionado en el uso de los recursos tales como tiempo de uso del CPU, uso de memoria, o conexiones a la base de datos.

Solución

Asumir que todos los datos de entrada son maliciosos hasta que no se demuestre lo contrario, y usar validación de los mensajes para protegerse en contra de los ataques tales como inyección de SQL, desbordamientos de búfer, entre otros. Se valida el contenido del mensaje XML contra un esquema XML (XSD) para asegurar que esté bien formado y en consonancia con lo que el servicio web espera para procesar. La validación lógica también mide los mensajes contra ciertos criterios: examinando el tamaño del mensaje, su contenido, y los conjuntos de caracteres que se utilizan. Cualquier mensaje que no cumpla los criterios es rechazado.

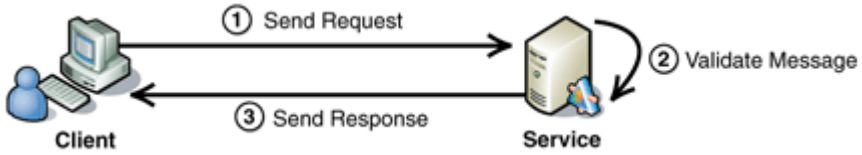

Figura 5: La validación del mensaje se produce en el servicio web

3.4 Estructura general de la arquitectura

Como se explicaba en el capítulo 1 la arquitectura de software es la estructura de alto nivel del sistema en la que se describen los principales componentes y las relaciones entre ellos. Por otra parte los estilos existentes catalogan las estructuras posibles de un software y definen la organización arquitectónica donde se conjugan estos componentes y sus conectores.

El estilo arquitectónico utilizado en la arquitectura de la Plataforma Manejadora de Peticiones es el estilo en capas. La plataforma consta de 3 capas:

- Capa de Servicios Web: Esta capa es la responsable de publicar los servicios web y atender las peticiones realizadas a estos servicios por parte de las aplicaciones clientes. Las clases

fundamentales de esta capa son las llamadas “punto final” o “endpoint”[23] y son las que manejan dichas peticiones e invocan las funcionalidades implementadas en las clases de servicios de la capa de Negocio.

- Capa de Negocio: En esta capa se encuentran las clases responsables de implementar la lógica de la aplicación, es decir las funcionalidades que se expondrán como servicios web de la plataforma. Cada servicio cuenta con una interfaz que contiene los métodos o funcionalidades que brinda y su respectiva clase de implementación.
- Capa de Acceso a Datos: En la implementación de esta capa se usó el patrón DAO (explicado en la sección 3.3). Contiene las clases responsables de la comunicación con el repositorio de datos, ya sea la base de datos o el servidor DNS. Existe una interfaz de acceso a datos (que define los métodos necesarios para la comunicación con la base de datos) por cada clase del dominio existente, con su respectiva clase de implementación.

Interacción entre las Capas

La interacción entre las capas de la aplicación fluye en un solo sentido como expone el estilo arquitectónico en capas, cada capa brinda servicios a la capa superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

La comunicación entre las capas se realiza utilizando el patrón Inyección de Dependencia (explicado en la sección 3.3), donde las clases de cada capa no son las responsables de crear las instancias de las clases de la capa inferior que necesitan, sino que les son inyectadas en ficheros de configuración. Cada clase de acceso a datos y del negocio cuenta con una interfaz donde expone las funcionalidades que brinda a la capa superior; de modo que las dependencias se establezcan con estas interfaces, permitiendo de esta forma el desacople entre las capas.

[23] Los endpoints son el concepto central de Spring-WS soportados del lado del servidor. Proveen el acceso al comportamiento de la aplicación, interpretan los mensajes XML e invocan a los métodos de la capa de negocio.

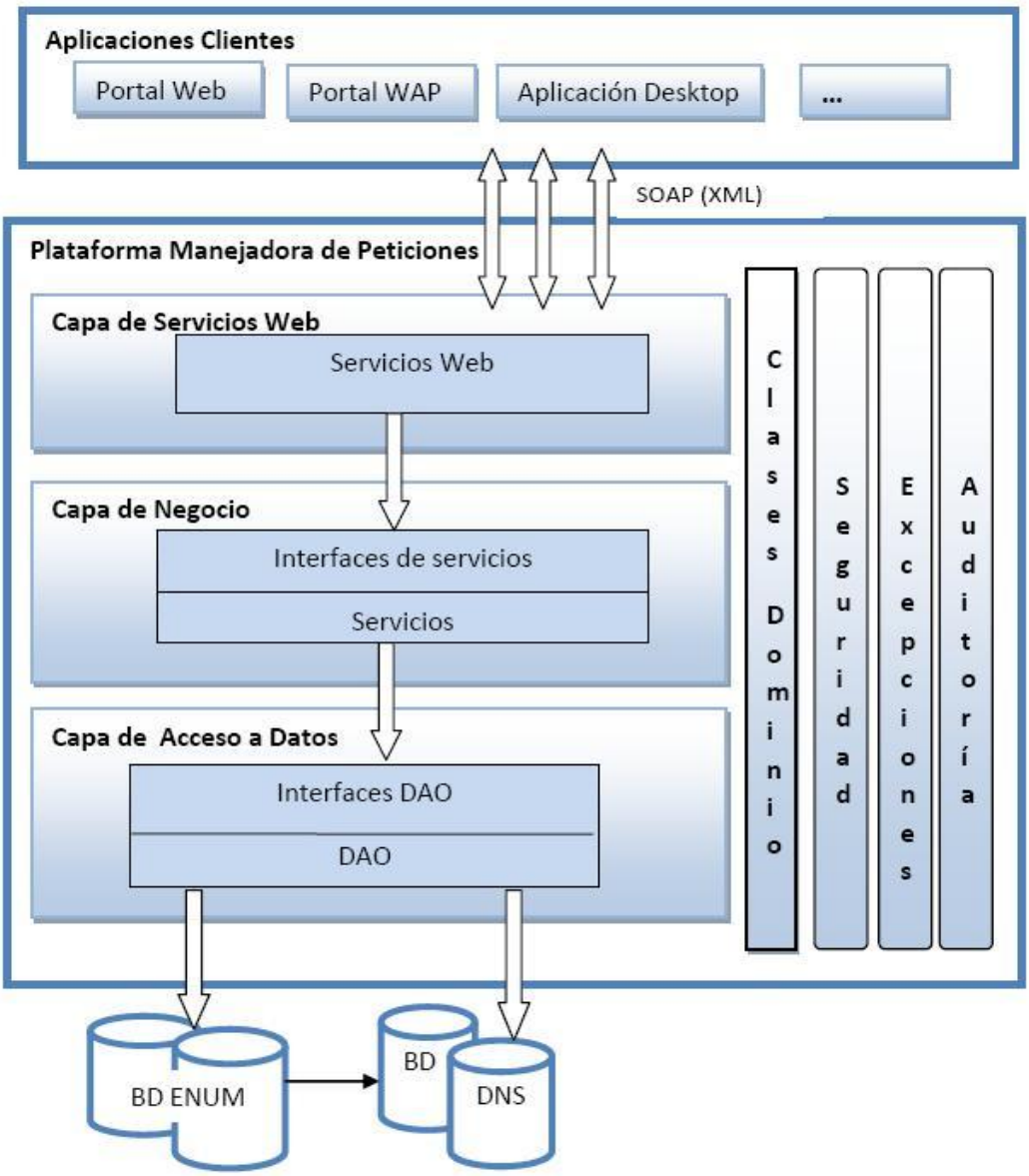

Figura 6: Estructura general de la arquitectura de la Plataforma Manejadora de Peticiones

Como se puede observar en la figura 6 existen aspectos que no se encuentran ubicados en ninguna capa específica, pues son comunes (clases del dominio) o constituyen aspectos trasversales (seguridad, manejo de excepciones y auditoría) que pueden manejarse en cada una de las capas.

3.5 Vistas arquitectónicas

Esencialmente, las vistas de la arquitectura se pueden ver como abstracciones o simplificaciones de los modelos construidos en las que se hacen más visibles las características importantes, y se dejan de lado los detalles.

En esta sección se representa la arquitectura mediante una serie de vistas arquitectónicas:

- Vista de Caso de Uso
- Vista Lógica
- Vista de Implementación
- Vista de Despliegue
- Vista de Datos

3.5.1 Vista de Casos de Uso

La vista de casos de uso muestra un subconjunto arquitectónicamente significativo del modelo de casos de uso del sistema.

En esta sección se representan los casos de usos arquitectónicamente significativos así como una breve descripción de cada uno de ellos. Es necesario aclarar que en la Plataforma Manejadora de Peticiones los casos de usos no contarán con una interfaz visual, pues serán expuestos como servicios web y los actores son aplicaciones software.

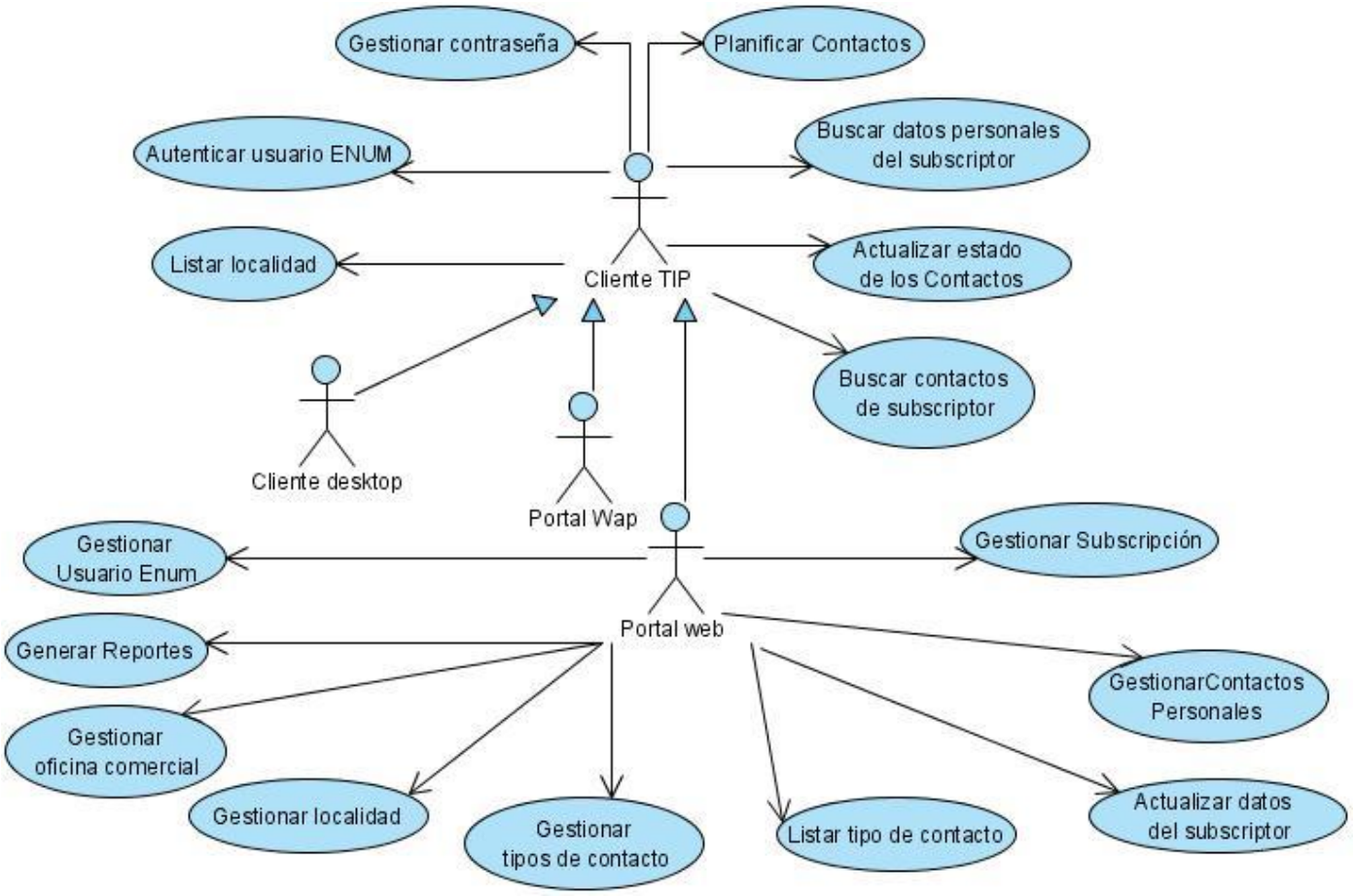

Figura 7: Vista de casos de usos

Descripción de los Casos de Uso:

Caso de uso: 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.

Caso de uso: 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.

Caso de uso: 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.

Caso de uso: 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.

Caso de uso: 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.

Caso de uso: 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.

Caso de uso: 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 los datos del subscriptor requeridos. Para modificar el estado de una subscripción se enviará el identificador de la subscripción y el nuevo estado; y en caso que se desee eliminarla, el identificador de la subscripción.

Caso de uso: 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.

Caso de uso: 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.

Caso de uso: Generar reportes

El caso de uso permite a las aplicaciones clientes generar reportes sobre la cantidad de subscripciones gestionadas por un registrador; cantidad y listado de registradores asociados a una oficina comercial; así como la cantidad de consultas realizadas a los contactos de un subscriptor y la cantidad de peticiones realizadas a los contactos de los subscriptores, en un intervalo de tiempo especificado. También posibilita listar las trazas de auditoría realizadas sobre las tablas de la base de datos.

Caso de uso: 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.

Caso de uso: 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; para listar las provincias es necesario el código del país. En el caso de listar los países no se hace necesario el envío de ninguna información.

Caso de uso: Gestionar tipos de contacto

El caso de uso le permite a las aplicaciones clientes adicionar, modificar o eliminar tipos o subtipos de contactos. Para adicionar o actualizar un nuevo tipo o subtipo de contacto, se debe enviar a la plataforma los datos del tipo o subtipo en cuestión, y para eliminarlo, el identificador del mismo.

Caso de uso: Listar tipos de contactos

El caso de uso permite a las aplicaciones clientes listar los tipos y subtipos de contactos que existen.
Dado un tipo se pueden listar los subtipos de contactos asociados al mismo y viceversa.

Caso de uso: Gestionar oficina comercial

El caso de uso le permite a las aplicaciones clientes adicionar, modificar o eliminar oficinas comerciales. Para adicionar y actualizar una oficina se debe enviar a la plataforma los datos de la misma; para eliminarla, el identificador.

Caso de uso: Gestionar usuario ENUM

El caso de uso permite a las aplicaciones clientes adicionar, actualizar y eliminar a los usuarios ENUM exceptuando a los subscriptores, que son gestionados mediante el caso de uso Gestionar subscripción. 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.

3.5.2 Vista Lógica

En esta sección se representan las clases más importantes, su organización en paquetes y la organización de estos paquetes en capas, destacando los elementos del diseño que son arquitectónicamente significativos.

3.5.2.1 Visión general de la arquitectura – Alineamiento de paquetes y capas.

Como se explicaba en el epígrafe 3.4 la Plataforma Manejadora de Peticiones cuenta con tres capas fundamentales: la capa de servicios web, la capa de negocio y la capa de acceso a datos. En la siguiente figura se muestra el diseño general de los principales paquetes en cada una de ellas.

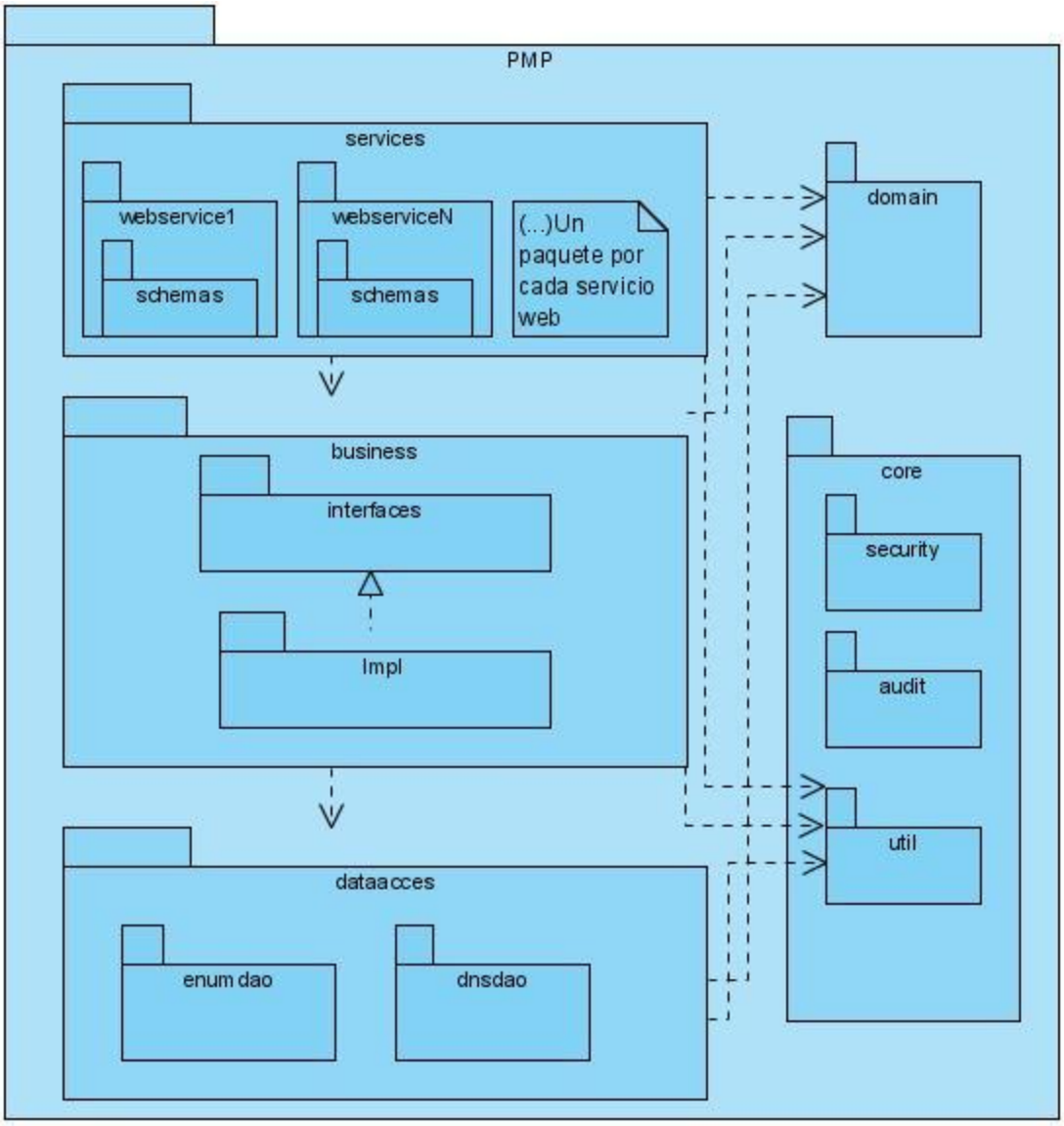

Figura 8: Estructura general de paquetes del diseño de la Plataforma Manejadora de Peticiones

La figura 9 muestra la estructura de paquetes propuesta, atendiendo a la organización de los paquetes del diseño anteriormente mostrado. Esta contiene un paquete raíz con el nombre cu.uci.tip.pmp, y dentro, la jerarquía de paquetes organizativos de la plataforma.

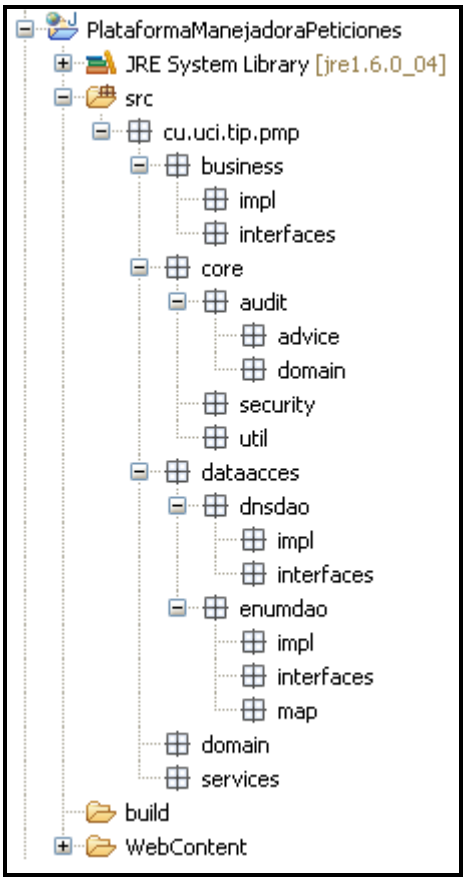

Figura 9: Estructura de paquetes de la Plataforma Manejadora de Peticiones

3.5.2.2 Descripción de los paquetes

cu.uci.tip.pmp.services: Dentro de este paquete se encuentran los paquetes y clases necesarias para manejar las peticiones a los servicios web expuestos por la plataforma.

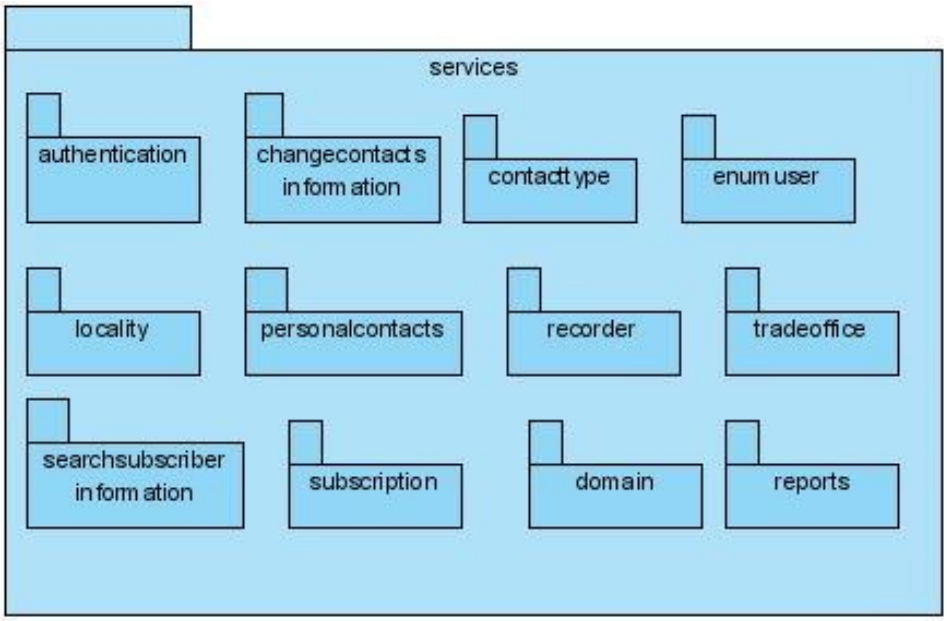

Figura 10: Diagrama de paquetes de la capa de servicios.

Para cada servicio web existe un paquete que contiene:

- Un fichero de definición del esquema XML (XSD por sus siglas XML Schema Definition) que especifica la estructura de los datos XML necesaria para consumir el servicio web.
- Un endpoint por cada método que contenga el servicio web, que recibe y procesa los mensajes enviados al servicio.
- Un paquete con el nombre schema, que contiene las clases (generadas a partir del fichero XSD usando el API JAXB[24]) necesarias para mapear los XML a objetos Java y viceversa.

La siguiente figura muestra el diagrama de clases de un servicio web cuyos elementos están contenidos dentro del paquete correspondiente a este servicio. Las clases endpoint heredan de *AbstractMarshallingPayloadEndpoint,* que es una clase brindada por Spring WS para mapear el contenido de los mensajes XML a objetos java y viceversa.

[24] JAXB (Java Architecture for XML Binding) proporciona una manera de traducir XML a objetos Java. Dado un esquema, el compilador JAXB genera un conjunto de clases de Java que contienen todo el código para analizar los documentos XML basados en el esquema.

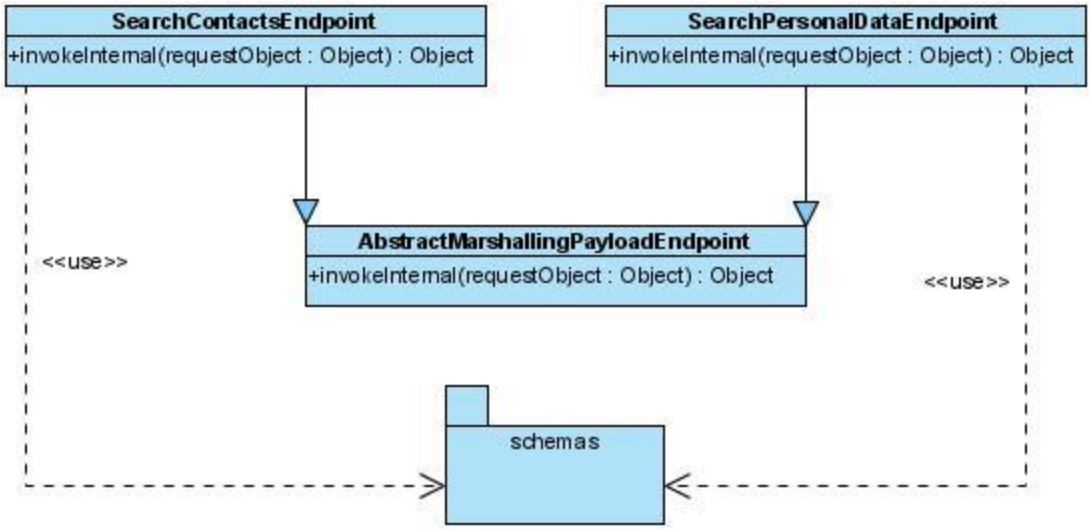

Figura 11: Diagrama de clases del servicio Cambiar Información de contactos.

cu.uci.tip.pmp.business: Representa la capa de negocio y contiene las clases que implementan los servicios brindados por la plataforma. Incluye dos paquetes, uno que tiene las interfaces de las clases del negocio y otro que contiene las implementaciones concretas de dichas clases.

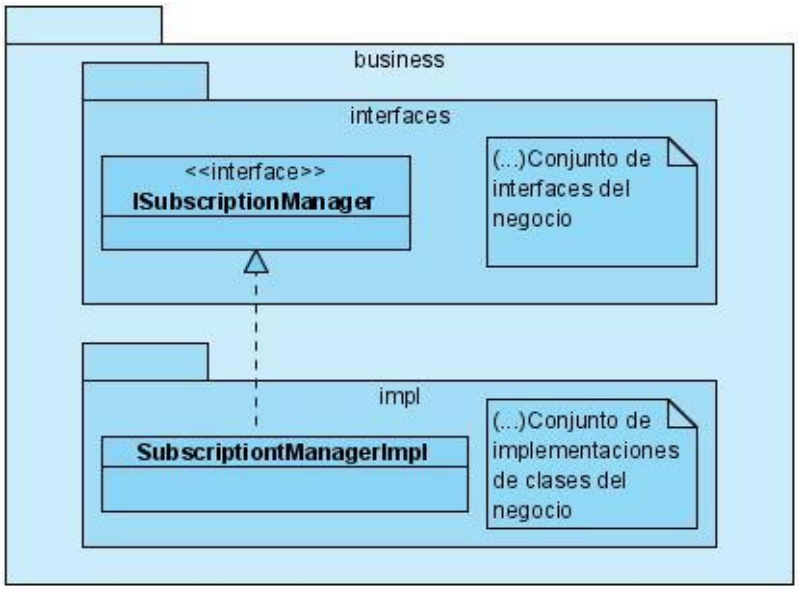

Figura 12: Clases y paquetes de la capa de negocio

cu.uci.tip.pmp.dataacces: Este paquete representa la capa de acceso a datos y encapsula dos paquetes fundamentales: enumdao y dnsdao. El paquete enumdao contiene un paquete llamado map con los ficheros donde se mapean parámetros y resultados de sentencias SQL a las clases del dominio, un paquete con las interfaces de las clases de acceso a datos o DAO (Data Access Object) y otro con las implementaciones concretas de las mismas. Dentro del paquete dnsdao, se encuentran las clases que interactúan directamente con el servidor DNS, encargadas solamente de realizarle consultas al mismo.

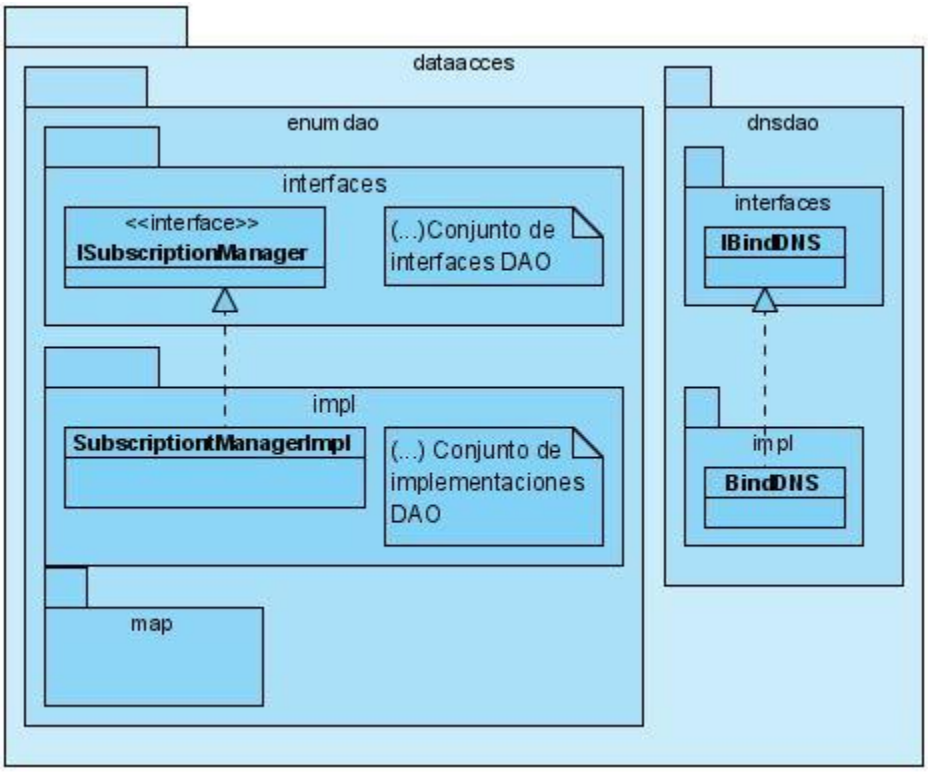

Figura 13: Clases y paquetes de la capa de acceso a datos.

cu.uci.tip.pmp.domain: En este paquete se encuentran las clases que representan los conceptos del dominio y que pueden ser utilizadas en cualquier capa de la aplicación.

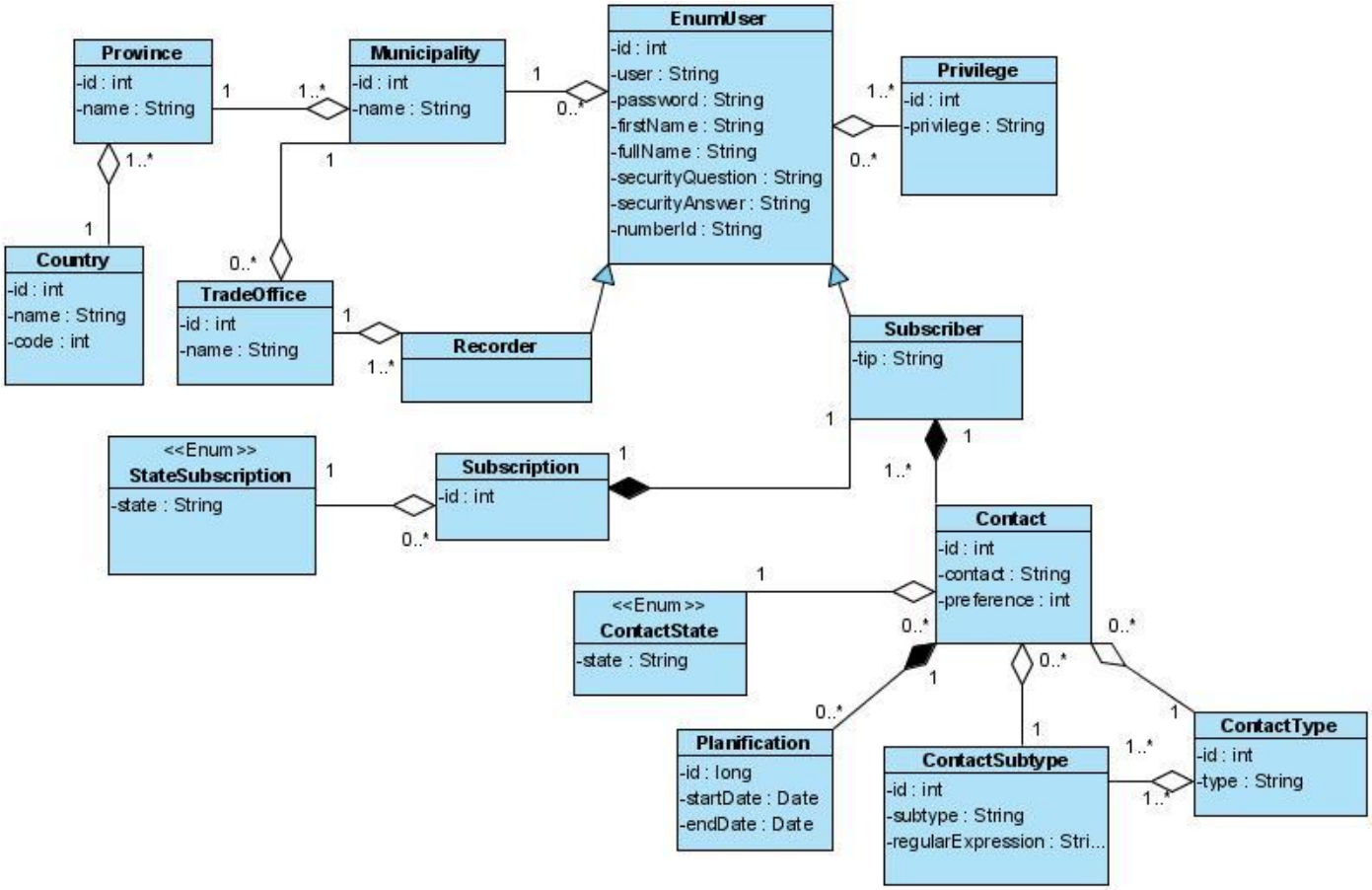

Figura 14: Diagrama de clases del dominio

cu.uci.tip.pmp.core.security: Contiene los ficheros donde se definen las políticas de seguridad necesarias para cada servicio web brindado por la plataforma.

cu.uci.tip.pmp.core.audit: Paquete que contiene las clases y ficheros de configuración necesarios para registrar trazas de las operaciones de gestión efectuadas sobre las tablas de la base de datos.

cu.uci.tip.pmp.core.util: Paquete que contiene clases auxiliares y clases necesarias para el manejo de las excepciones.

3.5.3 Vista de Implementación

El objetivo de la vista de implementación es capturar las decisiones arquitectónicas tomadas para la implementación. Normalmente, la vista de implementación contiene:

- Una enumeración de los subsistemas del modelo de implementación.
- Diagramas de componentes que ilustran cómo se organizan los subsistemas en capas y jerarquías.
- Las dependencias de importación entre subsistemas.

Existe una traza directa entre el diseño y la implementación; es por ello que la estructura de la vista de implementación es muy similar a la estructura de la vista lógica, explicada con anterioridad. La forma exacta en que se crea esta traza depende de cómo van a ser estructurados y modularizados los ficheros de código fuente, dado el lenguaje de programación que se esté usando. Para el caso específico de la Plataforma Manejadora de Peticiones (figura 15), por cada clase del diseño debe existir un componente (fichero con extensión .java) donde se implemente la misma y los subsistemas de implementación representan los paquetes del modelo de diseño. Estos subsistemas contendrán a la vez otros subsistemas y componentes.

En la figura 16 se representan los componentes constituyentes de un servicio en la capa de servicios web. Esta capa contiene un subsistema para cada servicio web, los cuales siguen la misma estructura respecto a los componentes que contienen: un fichero por cada clase endpoint, un fichero con el esquema del servicio y un subsistema que contiene un fichero por cada clase generada a partir del esquema del servicio.

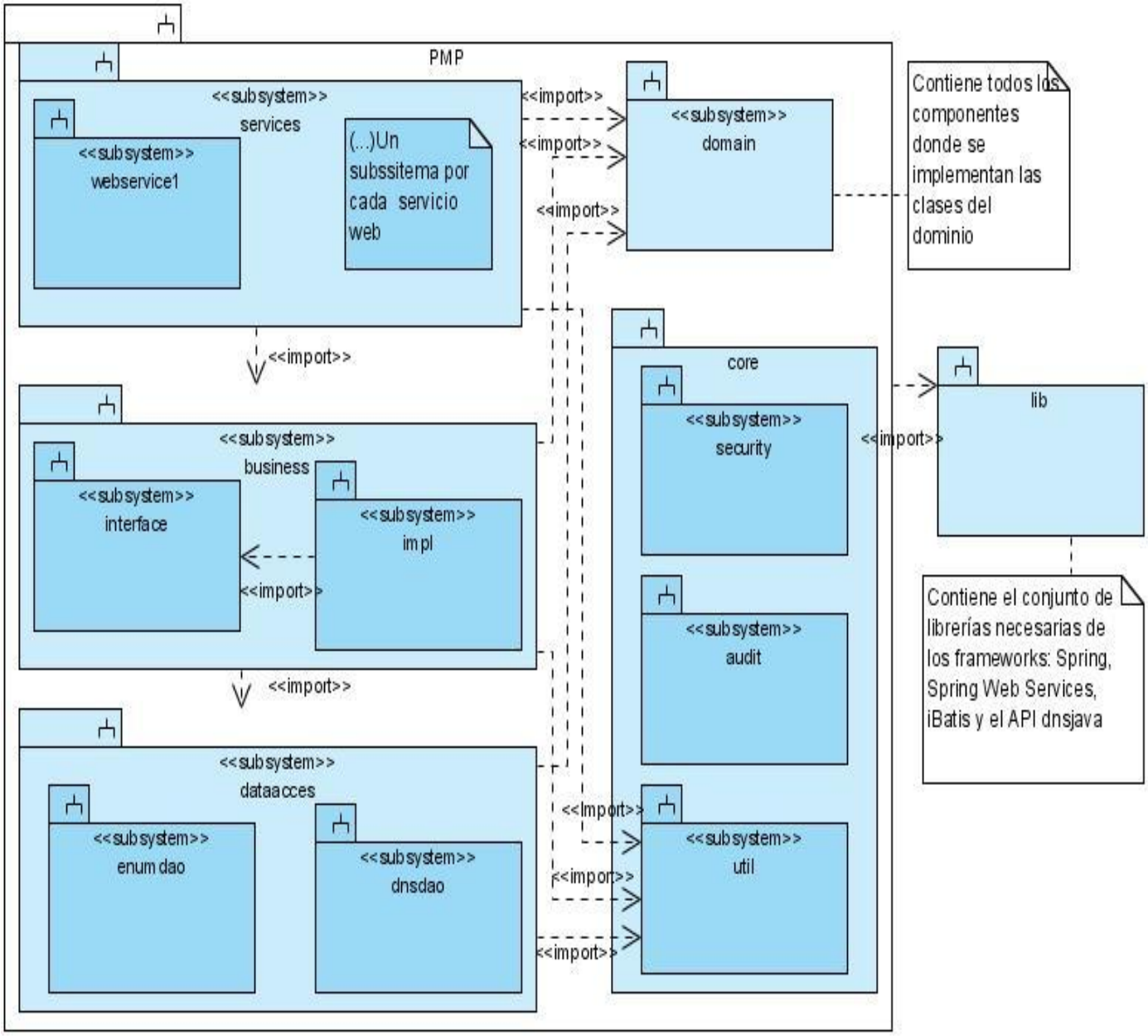

Figura 15: Diagrama de componentes general

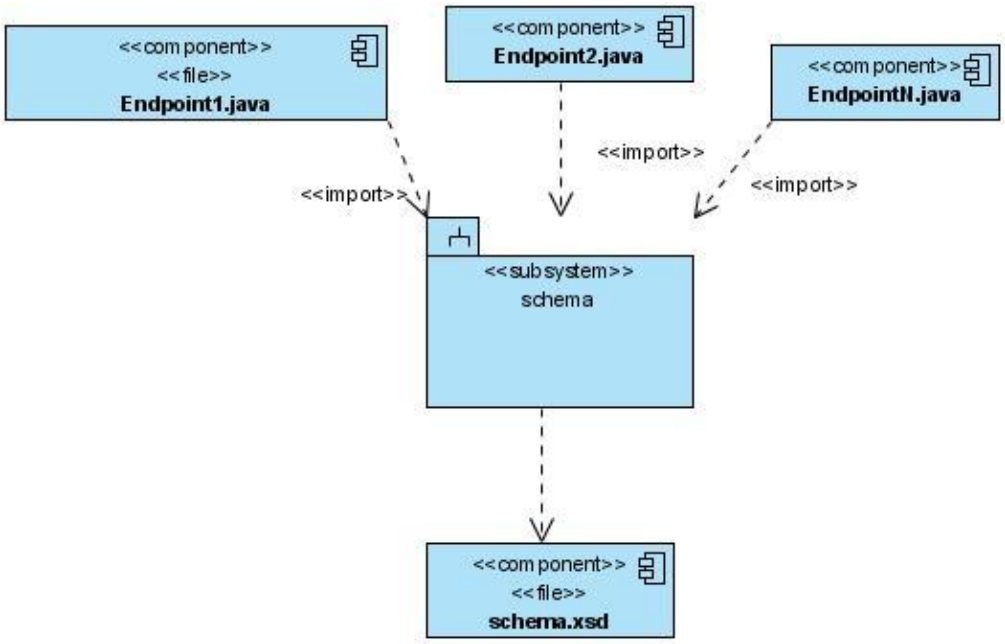

Figura 16: Diagrama de componentes de un servicio web.

El diagrama de componentes del subsistema de la capa del negocio representado a continuación está formado por dos subsistemas, uno que contiene las interfaces de las clases del negocio y otro con un fichero por cada clase del negocio con la implementación concreta de cada una de ellas. El subsistema cuenta además con un componente (un fichero con extensión .xml) donde se realizan las configuraciones correspondientes a la capa de negocio.

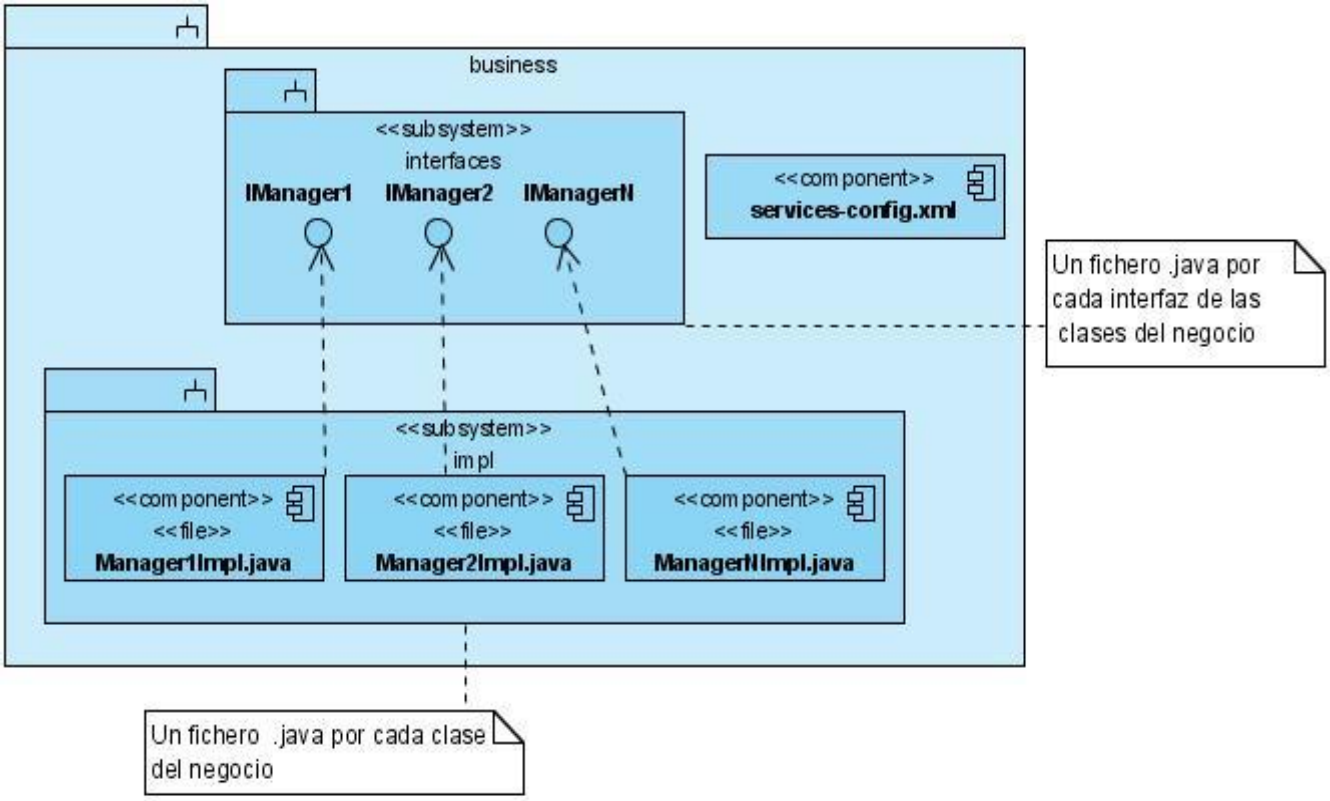

Figura 17: Diagrama de componentes de la capa de negocio

En el subsistema de implementación de la capa de acceso a datos al igual que el subsistema de la capa de negocio, hay un subsistema por cada paquete definido en el diseño y un componente por cada clase contenida en estos paquetes. El subsistema map contiene un conjunto de ficheros (con extensión .xml) donde se mapean parámetros y resultados de sentencias SQL a las clases del dominio. Además contiene el componente dataacces-config.xml con las configuraciones necesarias para la conexión a la base de datos y otras configuraciones requeridas en esta capa.

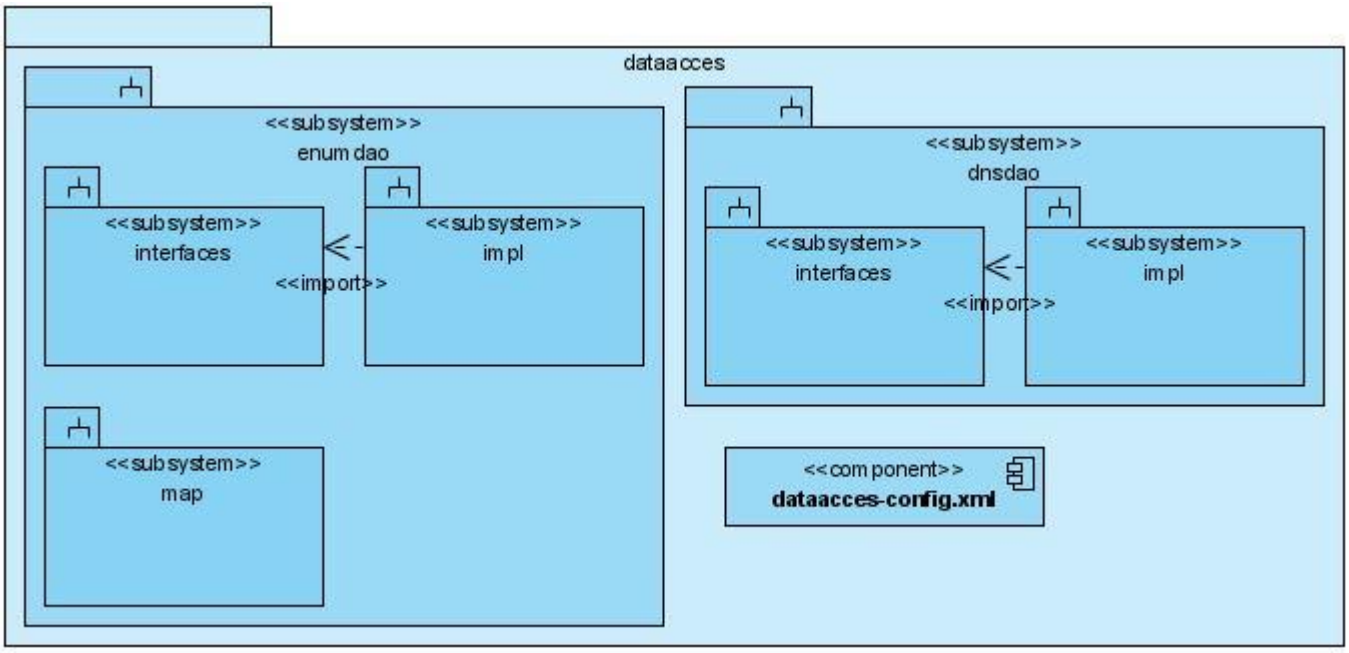

Figura 18: Diagrama de componentes de la capa de acceso a datos

3.5.4 Vista de Despliegue

En esta vista se representa una propuesta de la distribución física del sistema representada por nodos y las interconexiones entre ellos. Estos nodos constituyen elementos de hardware donde se ejecutan los componentes software necesarios para el funcionamiento del sistema. Para cada nodo se muestran los requerimientos de hardware y software requeridos para desplegar el sistema (especificados en la sección 3.2. Objetivos y Restricciones Arquitectónicas).

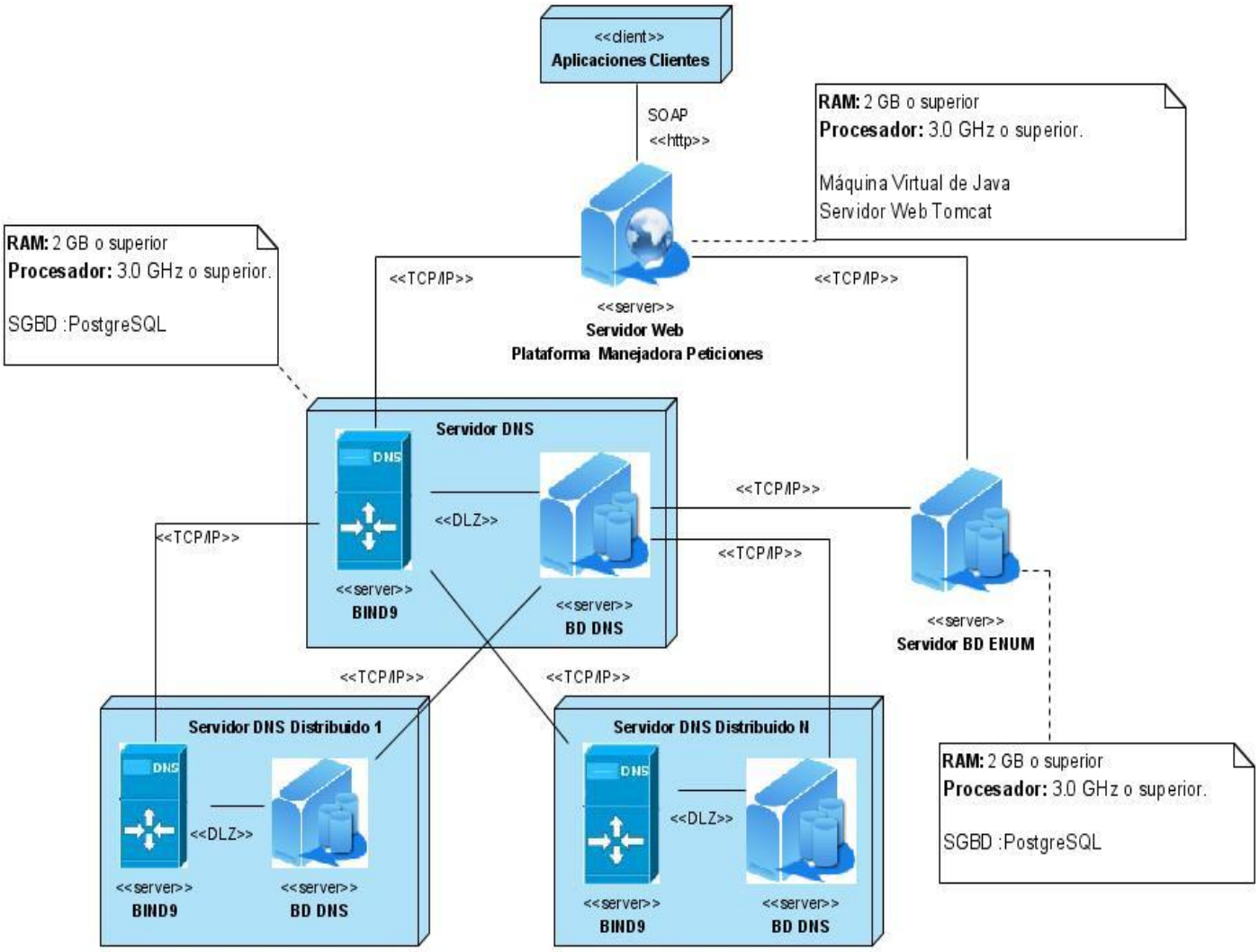

Figura 19: Diagrama de despliegue

Descripción de los nodos físicos:

- **Servidor web Plataforma Manejadora de Peticiones:** En esta máquina se mantendrá corriendo el servidor web Tomcat donde se publicará la Plataforma Manejadora de Peticiones.
- **Servidor DNS:** En este nodo radicará el servidor DNS BIND central o primario y la base de datos asociada al mismo, que contendrá la información asociada a los contactos personales de los subscriptores, así como otros datos administrativos y de registros propios de los servidores DNS. La comunicación entre el servidor DNS BIND y su base de datos asociada, se realizará mediante el

driver DLZ.

Existirá una jerarquía de servidores DNS distribuidos por zonas (cuyas estructuras serán iguales a la del servidor DNS primario explicado con anterioridad), que contendrán los datos correspondientes a su zona. La Plataforma Manejadora de Peticiones solo realizará consultas al servidor DNS primario, el cual si no contiene los datos solicitados los buscará por la jerarquía de servidores DNS inferiores o superiores a este.

- **Servidor BD ENUM:** En este nodo se alojará la base de datos ENUM, que contendrá los datos asociados al servicio TeleIdentificador Personal, los cuales serán gestionados a través de la Plataforma Manejadora de Peticiones. La base de datos ENUM se comunicará con la base de datos del servidor DNS primario para realizar las operaciones referentes a la gestión de los contactos. Esta última se comunicará con las bases de datos correspondientes a los servidores DNS de su nivel inmediatamente inferior, y así sucesivamente.
- **Aplicaciones Clientes:** Este nodo representa a las aplicaciones que consumirán los servicios expuestos por la Plataforma Manejadora de Peticiones. Aunque el despliegue de las mismas está fuera del ámbito de este documento, se representan para enfatizar que la comunicación con la plataforma se realizará a través de servicios web por medio de mensajes SOAP (Simple Object Access Protocol) utilizando como protocolo de transporte HTTP.

Descripción de las Conexiones:

- **TCP/IP (Transmition Control Protocol/Internet Protocol)**: Es un conjunto o familia de protocolos desarrollados para permitir a computadoras cooperativas y heterogéneas compartir recursos a través de una red.
- **HTTP (HyperText Transfer Protocol):** Es un protocolo usado en la WWW para la transferencia de hipertexto que sigue el esquema petición-respuesta entre un cliente y un servidor. Define la sintaxis y la semántica que utilizan los elementos de la arquitectura web para comunicarse.
- **SOAP (Simple Object Access Protocol):** Es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML; permitiendo la comunicación y la interoperabilidad entre diversas aplicaciones desarrolladas bajo tecnologías diferentes.

- **DLZ (Dynamically Loadable Zones):** Es un driver que en un principio surge como un parche para dotar al BIND9 con la capacidad de actualizar sus zonas sin necesidad de perder tiempo en recargas o reinicio del sistema, permitiéndole además del uso de los enfoques de almacenamiento mediante ficheros de texto, el almacenamiento de esta misma información pero en un servidor de base de datos. DLZ es una funcionalidad propia del BIND9 que puede ser activada para que las peticiones que se le hacen al DNS sean traducidas en consultas SQL y transferidas al sistema gestor de base de datos elegido como contenedor de la información que antes se tenía en un fichero de zona, luego el mismo DLZ crea la respuesta con el resultado de esta consulta y se la envía a la aplicación cliente que solicitó el servicio del DNS.

3.5.5 Vista de Datos

En esta sección se representan los modelos de datos de la base de datos ENUM y la base de datos asociada al servidor DNS primario. En la base de datos ENUM van a persistir todos los datos asociados al servicio TeleIdentificador Personal, los cuales serán gestionados a través de la Plataforma Manejadora de Peticiones.

La base de datos del servidor DNS contendrá la información asociada a los contactos personales de los subscriptores, así como otros datos administrativos y de registros propios de los servidores DNS, que garantizan el correcto funcionamiento de los mismos.

En la base de datos ENUM existirán dispararadores (triggers[25]) para insertar los datos referentes a los contactos de los subscritores en la base de datos del DNS primario, y funciones para la actualización y eliminación de dichos datos.

[25] Un trigger o un disparador en una base de datos es un evento que se ejecuta cuando se cumple una condición establecida al realizar una operación.

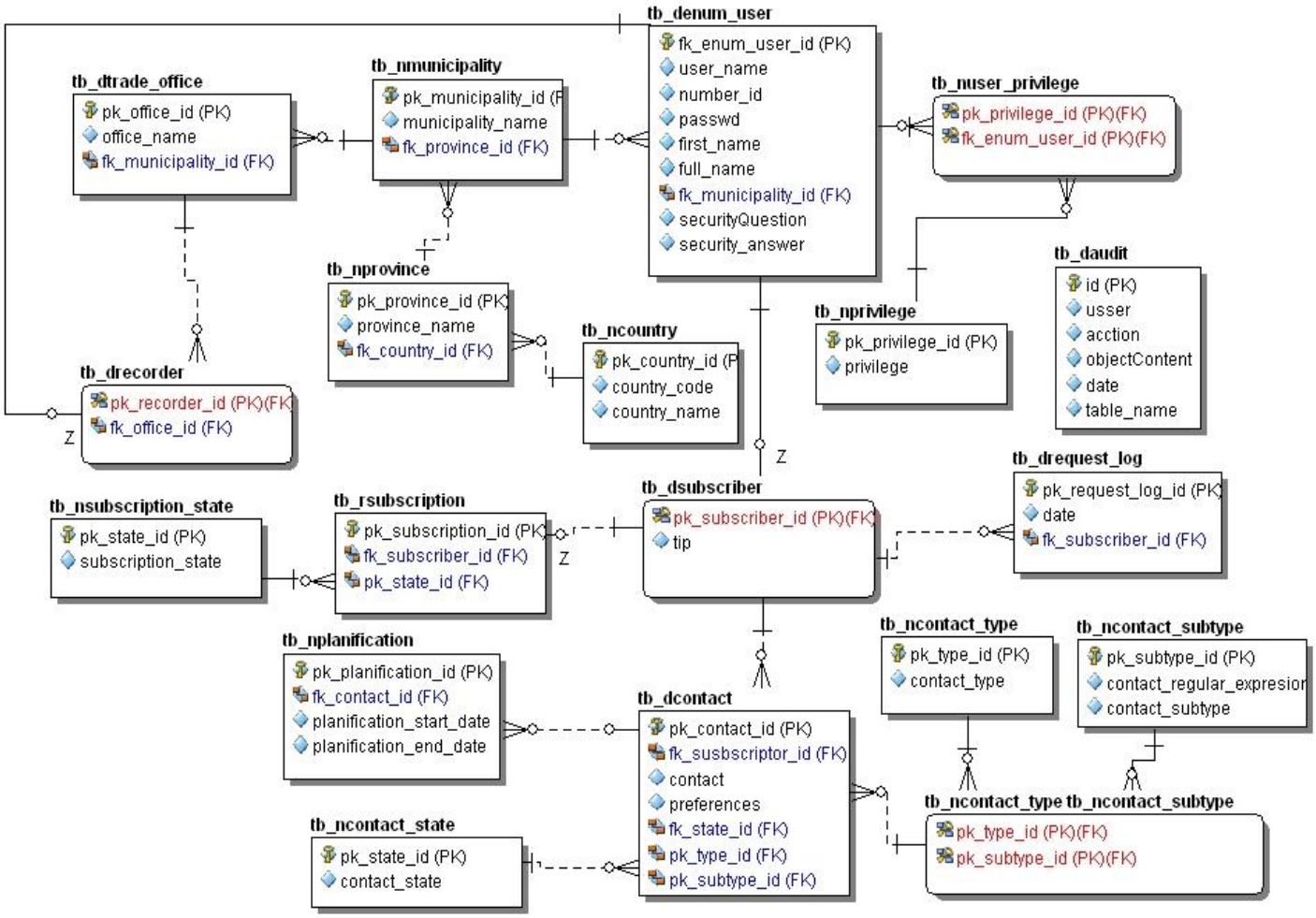

Figura 20: Modelo lógico de la base de datos ENUM

Descripción de las tablas:

Nombre de la tabla	Descripción
tb_dtrade_office (oficina comercial)	Tabla que contiene los datos referentes a las oficinas comerciales.
tb_nmunicipality (municipio)	Tabla que contiene los datos referentes a los municipios.
tb_nprovince	Tabla que contiene los datos referentes a las provincias.

(provincia)	
tb_ncountry (país)	Tabla que contiene los datos referentes a los países: nombre, identificador y código de país (el código asignado al país según el plan internacional de numeración para las telecomunicaciones).
tb_denum_user (usuario ENUM)	Tabla que almacena información referente a los usuarios ENUM (administrador de la plataforma, registrador, subscriptor, auditor)
tb_nuser_privilege (usuario_ privilegios)	Tabla que asocia un privilegio a un usuario ENUM (contiene el identificador del usuario y del privilegio).
tb_nprivilege (privilegios)	Nomenclador que contiene los privilegios que pueden ser asociados a los usuarios ENUM.
tb_drecorder (registrador)	Tabla que contiene información específica de los registradores, que son las personas que tienen la responsabilidad de gestionar las subscripciones al servicio TIP.
tb_dsubscriber (subscriptor)	Tabla que contiene información específica de los subscriptores que son las personas que poseen el servicio TIP.
tb_drequest_log (logs de subscriptor)	Tabla donde se almacena un registro (fecha, identificador del subscriptor), por cada consulta realizada a los contactos de un subscriptor.
tb_dcontact (contacto)	Tabla donde se almacenan los contactos de los subscriptores. Medios a través de los cuales se puede establecer comunicación, como teléfono, correo electrónico, beeper, entre otros.
tb_ncontact_state (estado del contacto)	Nomenclador que contiene los posibles estados en que se pueda encontrar un contacto: verificación, activo y oculto.
tb_ncontact_type (tipo de contacto)	Tabla que contiene los tipos de contactos que pueden existir.
tb_ncontact_subtype (subtipo de contacto)	Tabla que contiene los protocolos o subtipos de contactos que pueden existir.
tb_ncontact_type tb_ncontact_subtype (tipo_contacto_subtipo_contacto)	Tabla que surge como resultado de la relación de muchos a muchos entre la tabla tb_ncontact_type y tb_ncontact_subtype.
tb_dplanification (planificación)	Tabla donde se almacenan las planificaciones correspondientes a cada contacto. Una planificación es un intervalo de tiempo durante el cual el contacto permanecerá oculto en el servidor DNS.
tb_dsubscription	Tabla donde se gestiona la información referente a las subscripciones

(subscripción)	(identificador del registrador y subscriptor).
tb_nsubscription_state (estado de subscripción)	Nomenclador donde se almacenan los posibles estados en los que se puede encontrar una subscripción (verificación, activa y fuera de servicio).
tb_daudit	Tabla donde se almacenarán las trazas de auditoría referentes a las acciones de gestión sobre los datos de las tablas de la base de datos y los intentos fallidos de autenticación.

Tabla 2: Descripción de las tablas de la base de datos ENUM

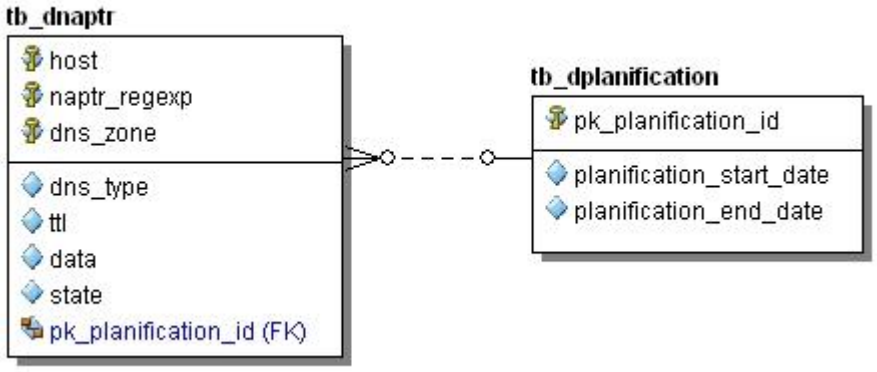

Figura 21: Modelo lógico de la base de datos del servidor DNS

Descripción de las tablas:

Nombre de la tabla	Descripción
tb_dnaptr (registros naptr)	Tabla donde se almacenan los registros NAPTR del servidor DNS. Cada registro contiene: host (número ENUM), naptr_regexp (expresión regular + contacto), zone (código de la zona), data (campo que contiene el orden en que los registros de recursos de tipo NAPTR han de ser procesados, la preferencia del contacto y el tipo de servicio), dns_type (tipo de registro dns), ttl (tiempo de vida en que se almacena el registro en la caché), state (estado del contacto)
tb_dplanification (planificación)	Tabla donde se almacenan las planificaciones correspondientes a cada contacto. Una planificación es un intervalo de tiempo durante el cual el contacto permanecerá oculto en el servidor DNS.

Tabla 3: Descripción de las tablas de las base de datos asociadas a los servidores DNS.

3.6 Conclusiones

En el presente capítulo se han descrito los patrones aplicados en la solución propuesta. Se ha proporcionado una visión general de la arquitectura de la Plataforma Manejadora de Peticiones, mediante varias vistas arquitectónicas propuestas por la metodología de desarrollo utilizada. La descripción de la arquitectura propuesta en este capítulo sirve como medio de comunicación entre todos los interesados en el proyecto, respecto a las decisiones significativas para la arquitectura que se llevan a cabo en el mismo.

Capítulo 4

Aspectos Transversales

4.1 Introducción

En el presente capítulo se definen los mecanismos transversales reflejados en la estructura general de la arquitectura de la Plataforma Manejadora de Peticiones (sección 3.4). Se analizan y dan solución específicamente al tratamiento de excepciones, la seguridad y la auditoría.

4.2 Tratamiento de excepciones

Las excepciones son situaciones anómalas que ocurren durante el tiempo de ejecución de un programa y requieren un tratamiento especial.

Si se consigue dominar su programación, la calidad de las aplicaciones que se desarrollen aumentará considerablemente. La correcta programación de excepciones significa diseñar los algoritmos pensando únicamente en la forma habitual en la que deben ejecutarse, manejando las situaciones extraordinarias aparte. De esta manera se consigue un diseño mucho más estructurado, legible, robusto y fácil de mantener.

En la Plataforma Manejadora de Peticiones el tratamiento de excepciones se realizará utilizando las facilidades que brindan el lenguaje de programación y los frameworks utilizados.

El lenguaje de programación Java, como otros lenguajes, incorpora mecanismos para el manejo de excepciones como parte de sus estructuras de control (try - catch - finally, throw y throws); permitiendo lanzar y capturar excepciones para su adecuado tratamiento en las instrucciones de código donde sea necesario, separando el código para el manejo de errores de la lógica de aplicación del programa. Además el lenguaje y los frameworks utilizados brindan un conjunto de clases de excepciones predefinidas. El programador también puede definir sus propias clases excepciones aportando siempre información útil cuando sean capturadas.

En el caso de los servicios web si se lanza una excepción en el curso del procesamiento de un mensaje, debe ser enviada a los clientes del servicio mediante un fallo SOAP (SOAP Fault[26]), que es un elemento que se puede incluir dentro del cuerpo del mensaje SOAP para notificar el fallo ocurrido. Por tal motivo se necesita convertir las excepciones lanzadas por el servicio web a fallos SOAP.

Spring Web Service proporciona una clase llamada *SoapFaultMappingExceptionResolver* que permite tomar el nombre de la clase de cualquier excepción que pueda ser lanzada y mapearla a un fallo SOAP. Como se muestra en la figura 22 esta clase manejará cualquier excepción no capturada que se produzca durante el curso de la manipulación de un mensaje mapeándola a un fallo SOAP apropiado que se enviará al cliente.

Figura 22: SoapFaultMappingExceptionResolver mapea cualquier excepción de Java lanzada desde un Endpoint a un fallo SOAP para ser enviada al cliente.

Las asignaciones de las clases excepciones a fallos SOAP son realizadas a través de la propiedad *exceptionMappings*, en la que se especifica el nombre de las clases excepciones y el mensaje descriptivo correspondiente, que permite enviar información personalizada del fallo al cliente. Se debe tener en cuenta que los mensajes de las excepciones a ser enviados a los clientes deben ser presentados de forma tal que se facilite su interpretación y procesado; pero sin exponer detalles internos de implementación del servicio, pues esto supone un riesgo de seguridad para el propio servicio y aporta información no útil de cara al consumidor del mismo.

[26] El mecanismo estándar para la definición de los errores dentro de un Servicio Web es el SOAP Fault.
<soap:Fault>
Cualquier tipo de fallo que se produzca será notificado en esta sección. La cual esta contenida dentro del cuerpo del mensaje.
</soap:Fault>

4.3 Seguridad

La seguridad es uno de los aspectos más importantes a tener en cuenta al desarrollar aplicaciones empresariales, sobre todo si se expone información o funcionalidades de los objetos y reglas del negocio a través servicios web. El bien más preciado por cualquier institución es la información, de ahí que se han desarrollado protocolos y mecanismos adecuados, para preservar su seguridad. Se puede hablar en este sentido de cuatro aspectos básicos de seguridad: integridad, confidencialidad, disponibilidad y el no repudio.

- Integridad: La información sólo puede ser modificada por quien está autorizado y de manera controlada.
- Confidencialidad: La información o los activos informáticos son accedidos sólo por las personas autorizadas.
- Disponibilidad: Los activos informáticos son accedidos por las personas autorizadas en el momento requerido.
- No repudio: El uso y/o modificación de la información por parte de un usuario debe ser irrefutable, es decir, que el usuario no puede negar dicha acción.

En la Plataforma Manejadora de Peticiones garantizar la seguridad de los servicios es de vital importancia. Se hace necesario restringir el acceso al consumo de los mismos a los usuarios autorizados y evitar el acceso o alteración por parte de terceros a la información que viaja en los mensajes SOAP. Se analizaron dos alternativas para garantizar la seguridad de los servicios web expuestos por la plataforma: la seguridad a nivel de transporte (https) y la seguridad a nivel de mensajes (basada en el estándar Web Service Security).

Seguridad a nivel de transporte

La seguridad en la capa de transporte es proporcionada por los mecanismos de transportes usados para transmitir la información a través del enlace entre clientes y proveedores, por lo tanto se basa en HTTP seguro (HTTPS). La seguridad en el transporte es un mecanismo de seguridad punto apunto que puede usarse para autenticación, integridad del mensaje y confidencialidad[41].

Las ventajas de usar seguridad en la capa de transporte son:

- Es relativamente simple.

- No requiere que las partes en comunicación entiendan los conceptos de seguridad a nivel de XML.

Las desventajas:

- Altamente acoplado al protocolo de transporte.
- No es flexible, pues no se puede proteger solo una porción del mensaje, sino que tiene que ser aplicado a todo el mensaje.
- No es una solución extremo a extremo (end-to-end), simplemente punto a punto (point-to-point). Si un mensaje debe pasar a través de varios puntos para llegar a su destino, cada punto intermedio debe reenviarlo a través de una nueva conexión SSL (ver figura 23).

Seguridad a nivel de mensaje

En la seguridad a nivel de mensaje, la información de seguridad es contenida en el mensaje SOAP, por lo que difiere de la de a nivel de transporte en que se encapsulan las credenciales de seguridad así como cualquier otro mecanismo de protección (firma o cifrado) en cada mensaje. La aplicación directa de seguridad al mensaje, modificando su contenido, permite que el mensaje seguro sea autónomo con respecto a los aspectos de seguridad.

Las ventajas de la seguridad a nivel de mensaje son:

- La seguridad se queda con el mensaje a través de todos los saltos y después de que el mensaje llega a su destino, por lo que puede usarse en conjunción con intermediarios a través de múltiples saltos.
- Tiene una fina granularidad. Se puede aplicar seguridad a diferentes porciones del mensaje, resultando más flexible puesto que se pueden firmar o cifrar partes del mensaje en lugar del mensaje completo.
- Compatibilidad con varios transportes. La seguridad de mensaje protege el mensaje sin tener en cuenta el transporte que se utilice para transmitir el mensaje; el contexto de seguridad se incrusta directamente dentro del mensaje.

Desventajas:

- Es relativamente complejo, pues requiere implementación de mecanismos de seguridad a nivel de XML.

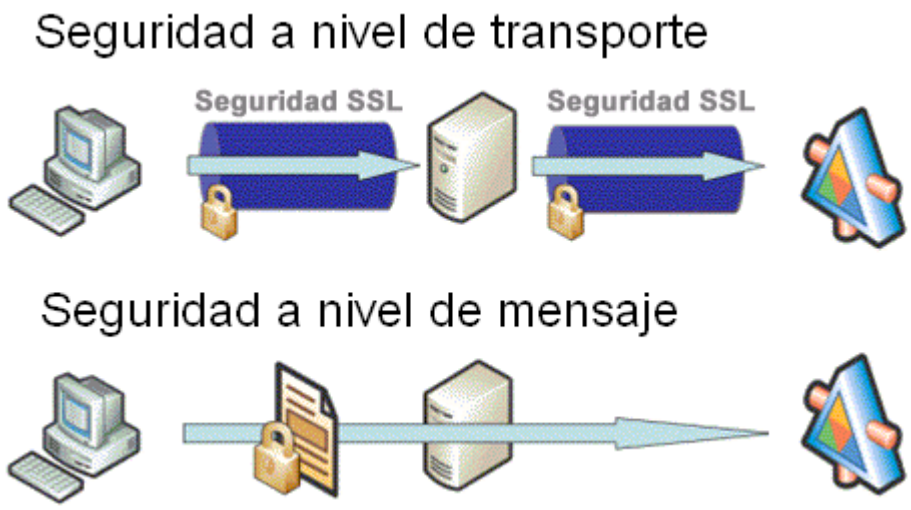

Figura 23: Seguridad a nivel de transporte frente a seguridad a nivel de mensaje.

WS-Security (Seguridad en Servicios Web) es un estándar que suministra un medio para aplicar seguridad a los Servicios Web. Esta especificación define una serie de extensiones para el protocolo SOAP, que permiten el intercambio entre cliente y servidor, de identificadores (“Tokens“) de seguridad. Este intercambio de Tokens, junto con los mecanismos asociados de encriptación y firmado, permiten asegurar la autentificación, integridad y confidencialidad de las operaciones realizadas.

Se decidió utilizar el estándar Web Service Security para garantizar la seguridad a nivel de mensaje de los servicios brindados por la Plataforma Manejadora de Peticiones. Utilizar la seguridad de nivel de mensaje en lugar de seguridad de nivel de transporte tiene las siguientes ventajas[42]:

- Seguridad global. Un transporte seguro, como capa de sockets seguros (SSL) sólo funciona cuando la comunicación es de punto a punto. Si el mensaje se dirige a uno o más intermediarios de SOAP antes de alcanzar el receptor final, el propio mensaje no está protegido cuando un intermediario lo lee desde la conexión. Dado que el modo de seguridad funciona directamente con el mensaje y protege el XML en él, la seguridad permanece con el mensaje sin tener en cuenta cuántos intermediarios están implicados con el mensaje antes de alcanzar el receptor final. Esto habilita el verdadero escenario de seguridad global.
- Compatibilidad con varios transportes. Puede enviar mensajes seguros sobre muchos transportes diferentes (SMTP, FTP y TCP). Con seguridad de nivel de transporte, toda la información de

seguridad es delimitada a una conexión de transporte determinada única y no está disponible desde el propio contenido del mensaje. La seguridad de mensaje protege el mensaje sin tener en cuenta el transporte que se utiliza para transmitir el mensaje y el contexto de seguridad se incrusta directamente dentro del mensaje.

- Compatibilidad con un amplio conjunto de credenciales y notificaciones. La seguridad del mensaje está basada en la especificación WS-Security, que proporciona un marco extensible capaz de transmitir cualquier tipo de notificación dentro del mensaje SOAP. A diferencia de la seguridad de transporte, el conjunto de mecanismos de autenticación, o las notificaciones, que puede utilizar no están limitados por las funciones del transporte.

El framework utilizado para exponer los servicios web, Spring Web Service, brinda soporte a dicho estándar específicamente en tres áreas: autenticación, firma digital y encriptación; se integra además con Spring Security, lo que posibilita el uso de configuraciones existentes de Spring Security para servicios web, garantizándose entre otros elementos la autenticación y la autorización .

En la Plataforma Manejadora de Peticiones existen servicios que solamente pueden ser consumidos por usuarios autorizados. Para restringir el acceso a los mismos se requiere que se trasmitan las credenciales (usuario y contraseña encriptada) de los usuarios que invoquen estos servicios, en la cabecera de los mensajes SOAP.

Se propone la encriptación, ya sea total o parcial, de los mensajes cuya información requiera la garantía de confidencialidad; así como el uso de firmas digitales[27] en los mensajes SOAP enviados y recibidos por la plataforma para garantizar la integridad de los datos transmitidos.

Cada aplicación que consuma los servicios requerirá un certificado digital[28]. Estos certificados deben ser firmados por una Autoridad Certificadora[29] de confianza del servicio. Los certificados serán creados con la herramienta keytool contenida en el JDK y la Autoridad Certificadora con la herramienta OpenSSL, que consiste en un robusto paquete de herramientas de administración y librerías relacionadas con la

[27] Una firma digital es un mecanismo criptográfico que permite certificar la identidad del emisor de un mensaje, así como la no alteración de sus datos durante la transmisión.
[28] Un Certificado Digital es un documento digital mediante el cual un tercero confiable (una autoridad de certificación) garantiza la vinculación entre la identidad de un sujeto o entidad y su clave pública.
[29] Organismo independiente que emite certificados digitales que atestiguan la identidad de los propietarios de los mismos.

criptografía.

Para validar los elementos de seguridad anteriormente mencionados se utilizará las facilidades que brinda Spring Web Service, específicamente la clase *XwsSecurityInterceptor,* que es un interceptor de endpoint cuya funcionalidad es validar los elementos de seguridad contenidos en la cabecera de los mensajes SOAP. Este interceptor necesita para operar un fichero de políticas de seguridad, en el que se especificarán los elementos de seguridad requeridos en los mensajes recibidos y los elementos a añadir a los mensajes a enviar. El *XwsSecurityInterceptor* delega a un conjunto de manejadores las responsabilidades de validar las credenciales de los usuarios y recuperar certificados digitales para operaciones criptográficas (firma digital y cifrado de mensajes).

Para realizar el proceso de autenticación se utilizarán las clases que brinda Spring Web Service que permiten la integración con el framework Spring Security, específicamente *SpringDigestPasswordValidationCallbackHandler*. Este manejador requiere un Spring Security *UserDetailService*[30] para recuperar la contraseña del usuario especificado en el token.

Spring Security toma ventaja del soporte de Spring AOP para proveer seguridad declarativa a nivel de método. Dicha facilidad será utilizada para garantizar la autorización a los recursos en la Plataforma Manejadora de Peticiones. Se utilizará específicamente la implementación que brinda el framework con las clases *org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator* y *org.springframework.security.intercept.method.MethodSecurityInterceptor* que trabajan en conjunto para interceptar las invocaciones a los métodos asegurados y verificar que se tienen los privilegios requeridos para invocar a dichos métodos.

4.3.1 Inyección SQL

La inyección SQL es una vulnerabilidad informática en el nivel de la validación de las entradas a la base de datos de una aplicación. Una inyección SQL sucede cuando se inserta o inyecta un código SQL invasor dentro de otro código SQL para alterar su funcionamiento normal, y hacer que se ejecute maliciosamente el código invasor en la base de datos. Constituye un problema de seguridad informática que debe ser

[30] La propiedad userDetailsService es usada para identificar el bean que será usado para recuperar la información del usuario de la base de datos.

prevenido en las aplicaciones. La vulnerabilidad puede ocurrir cuando un programa arma descuidadamente una sentencia SQL, con parámetros dados por el usuario, para luego hacer una consulta a una base de datos. Dentro de los parámetros dados por el usuario podría venir el código SQL inyectado.

Algunas medidas para evitar este tipo de ataque son:

- Filtrar los valores entrados por los usuarios, con el fin de validar si son correctos, para ello se debe tener en cuanta los siguientes aspectos:
    - Limitar, siempre que se pueda, el tamaño del valor ingresado. De esta manera, se limita la cantidad de código SQL que un atacante puede inyectar. En especial si se trata de una consulta anidada.
    - No permitir todos los caracteres que puedan llegar a ser usados como parte de una inyección SQL: comilla simple (’), punto y coma (;) y caracteres de comentario (/* */ y –).
    - Filtrar con expresiones regulares todo lo que ingrese el usuario, para evitar que inserte valores que no corresponden al formato esperado.
- Realizar las consultas de la aplicación, utilizando un usuario con la menor cantidad de privilegios posible dentro de la base de datos. De modo que si un atacante, pudiera saltarse de algún modo la verificación de los valores e inyectar código en alguna las consultas, los daños se reduzcan al no tener suficientes privilegios.
- Usar Declaraciones Preparadas (Prepared Statements[31]): Esta es una muy buena solución a utilizar para evitar este tipo de ataques, pues prácticamente son inmunes a las inyecciones SQL.

Con el fin de poner en práctica las medidas anteriormente mencionadas, en la Plataforma Manejadora de Peticiones, se propone realizar las siguientes acciones:

1. Aplicar el patrón Validador de Mensaje (explicado en la sección 3.3), filtrando los datos enviados por las aplicaciones clientes a través de la validación de los mismos contra los esquemas XML (XML Schema) definidos para los servicios web. En la definición del esquema de cada servicio se debe especificar el formato necesario para cada uno de los parámetros requeridos por el servicio;

[31] Una declaración preparada es una consulta especial en la que se mantiene la estructura y se cambia solamente el dato que se requiera.

restringiendo mediante el uso de expresiones regulares, los valores válidos aceptados, así como la longitud máxima o mínima de los mismos.

El framework Spring Web Service posibilita el uso de esquemas para la validación de los mensajes XML, salientes y entrantes. Con este propósito provee la clase *PayloadValidatingInterceptor*, que retorna al cliente un error de validación en caso que el mensaje no se corresponda con el esquema especificado.

2. Se establecerá un usuario en la base de datos con los permisos mínimos requeridos para realizar las operaciones de inserción, actualización, eliminación y consulta sobre las tablas. La conexión de la plataforma a la base de datos se realizará utilizando dicho usuario.

3. Utilizar declaraciones preparadas (Prepared Statements) para realizar consultas a la base de datos. iBatis permite concatenar variables directamente en declaraciones SQL utilizando caracteres $, abriendo la puerta a la inyección de SQL .Las declaraciones que usan explícitamente la sustitución de cadenas en la sintaxis SQL se encuentran en situación de riesgo en iBatis. La siguiente declaración constituye un ejemplo de lo anterior :

```
<select id="getItems" parameterClass="MyClass" resultClass="items">
      SELECT * FROM items WHERE owner = $userName$
 </select>
```

iBatis facilita la protección de la aplicación de ataques de inyecciones SQL mediante el uso de declaraciones preparadas (Prepare Statements) que no son susceptibles a este tipo de ataques [35]. iBatis Data Maps permite especificar parámetros dinámicos en SQL y son típicamente definidos por el uso de caracteres #, por ejemplo:

```
<select id="getItems" parameterClass="MyClass" resultClass="items">
      SELECT * FROM items WHERE owner = #userName#
 </select>
```

Los caracteres # alrededor del nombre de la variable indican que iBatis creará una consulta parametrizada con dicha variable.

4.4 Auditoría

Se define como auditoría de sistemas de información al conjunto de procedimientos y técnicas para evaluar y controlar total o parcialmente un sistema informático, con el fin de proteger sus activos y recursos, verificar si sus actividades se desarrollan eficientemente y de acuerdo con la normativa informática y general existentes en cada empresa, y para conseguir la eficacia exigida en el marco de la organización correspondiente[43]. En situaciones en que los datos sean críticos, se debe contar con el riesgo de acciones malintencionadas o de errores involuntarios, que igual pueden causar inconsistencias o falta de veracidad de la información. Para estos casos es interesante mantener un archivo de auditoría (logs[32]), donde son registradas todas las operaciones realizadas por los usuarios que interactúan con la información. Estos logs pueden ser consultados para conocer los daños causados y/o identificar a los responsables de las operaciones irregulares.

En la Plataforma Manejadora de Peticiones se registrarán las operaciones de gestión tales como inserción, modificación y eliminación efectuadas sobre las tablas de la base de datos. De cada operación se almacenará el usuario que la realizó, el nombre de la tabla que modifica, la fecha y el contenido de la tupla afectada. Se registrarán además, los intentos fallidos de autenticación, almacenándose la fecha, el usuario y la contraseña enviada.

Para realizar el proceso de auditoría se utilizará programación orientada a aspectos, que posibilita definir la funcionalidad común en un lugar, y definir declarativamente cómo y dónde esta funcionalidad será aplicada, sin tener que modificar la clase a la que se le aplicará esta nueva característica. Los comportamientos comunes pueden ser modularizados en objetos especiales denominados aspectos.
Como parte del soporte de Spring a la programación orientada a aspectos se encuentra la clase *org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator* que posibilita listar las clases a las que les será aplicado un determinado aspecto.

4.5 Conclusiones

En el presente capítulo se proponen mecanismos para garantizar el tratamiento de excepciones, la

[32] Ficheros informáticos conteniendo detalles de cambios a registros de datos, que pueden usarse en el caso de que se precise la recuperación del sistema.

seguridad de la información gestionada por la plataforma y la auditoría del sistema. Estos dos últimos elementos serán implementados aprovechando las facilidades que brinda la programación orientada a aspectos, evitando la dispersión del código y garantizando que las implementaciones resulten más comprensibles.

CONCLUSIONES GENERALES

La importancia de representar de forma explícita la arquitectura de los sistemas de software es evidente. En primer lugar, estas representaciones elevan el nivel de abstracción, facilitando la comprensión de los sistemas de software. En segundo lugar, se puede aplicar a otros sistemas que exhiben requerimientos parecidos y promover la reutilización en gran escala.

En el presente trabajo se ha diseñado y descrito la arquitectura base de la Plataforma Manejadora de Peticiones, cumpliéndose de esta forma con el objetivo trazado para la presente investigación. Dentro de los principales aspectos tratados y resultados obtenidos se pueden listar los siguientes:

- Se realizó un estudio de los principales estilos arquitectónicos, leguajes de descripción de arquitectura, así como los atributos de calidad a tener en cuenta para el diseño de la misma.
- Se definieron las herramientas y tecnologías a utilizar para el desarrollo de la Plataforma Manejadora de Peticiones.
- Se proporcionó una visión general arquitectónica del sistema y luego se detalló, mediante una serie de vistas arquitectónicas para representar diferentes aspectos del mismo.
- Se desarrolló una arquitectura de tres capas garantizándose un fácil mantenimiento, desarrollo en paralelo del sistema y la especialización de las actividades en el proceso de desarrollo.
- Se diseñaron las capas lógicas de la aplicación y se especificó la estructura interna de cada una y la comunicación entre las mismas, teniendo en cuenta buenas prácticas y patrones de diseño.
- Se definieron mecanismos de seguridad basados en estándares para servicios web, garantizándose la integridad y confidencialidad de los datos manejados por los servicios publicados en la Plataforma Manejadora de Peticiones.
- Se definió un mecanismo de auditoría que posibilita registrar información útil sobre las operaciones de gestión tales como inserción, modificación y eliminación efectuadas sobre las tablas de la Base de Datos, para su posterior análisis y monitoreo.

RECOMENDACIONES

Se recomienda:

- Aplicar métodos de evaluación de la arquitectura para identificar las posibles debilidades.
- Mantener un constante refinamiento de la arquitectura a lo largo del ciclo de desarrollo.
- Analizar y aplicar técnicas de almacenamiento en caché para mejorar el rendimiento del sistema.

Referencias Bibliográficas

1. Pérez, M., *Impacto del ENUM en las redes y los servicios*, in *Revista de Telecomunicaciones*. 2008.
2. Nelson Prieto Rivero, A.A.L., *Requisitos para desarrollar el SW "Web Desarrollo ENUM Intranet ETECSA, Web Registrador ENUM y Cliente ENUM"*. 2008.
3. Rivero, N.P., *Requisitos para Incorporar funcionalidad ENUM en Aplicaciones por la UCI*. 2008.
4. Espinosa, A.B., *Requisitos para desarrollar el SW "Locución de Contactos Cliente ENUM mediante AGI Asterisk"* 2008.
5. Nelson Prieto Rivero, A.B.E., *Requisitos para Incorporar funcionalidad ENUM en Aplicaciones por la UCI.* 2008.
6. Martínez, A.C., *Requisitos para desarrollar el software Aplicación ENUM en los terminales celulares.* 2008.
7. Clements, P., *A Survey of Architecture Description Languages*. 1996, Alemania.
8. Bass, L.C., Paul; Kazman, Rick, *Software Architecture in Practice*. 1998: Addison Wesley.
9. Shaw , M.G., David *Software Architecture: Perspectives on an Emerging Discipline*. 1996.
10. IEEE (2000) *IEEE Recommended practice for architectural description of software-intensive systems*. 23.
11. Shaw, M.C., Paul, *A field guide to boxology: Preliminary classification of architectural styles for software systems*. 1997.
12. Carlos Reynoso, N.K. (2004) *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft*.
13. Garlan, D.S., Mary, *An Introduction to Software Architecture*. 1994.
14. Larman, C., *UML y Patrones. Introducción al análisis y diseño orientado a objetos*

1999: Addison Wesley.

15. *Alexander, C.*, *A Pattern Language*. 1977.
16. Group, H. *Home of the Patterns Library*. enero 2009]; Available from: http://hillside.net.
17. Buschmann, F.M., Regine ; Rohnert, Hans ;Sommerlad, Peter ;Stal, Michael *Pattern-Oriented Software Architecture* Vol. 1. 1996: John'Wiley & Sons Ltd.
18. Carlos Reynoso , N.K. (2004) *Lenguajes de descripción de arquitectura*.
19. Jacobson, G.B.J.R.I., *El Lenguaje Unificado de Modelado*. 1999: Addison-Wesley iberoamericana
20. Kruchten, P., *Architecture blueprints-the "4+1" view model of software architecture.* . 1995.
21. Bastarrica, M.C., *Atributos de Calidad y Arquitectura del Software*
22. Len Bass, P.C., Rick Kazman, *Software Architecture in Practice*. Vol. Second Edition. 2003: Addison Wesley.
23. Clements , P.N., Linda *Software architecture: An executive overview.* 1996.
24. Jacobson, I.B., Grady ; Rumbaugh, Jame, *El Proceso Unificado de Desarrollo de Software*. 2000: Addison-Wesley.
25. Ben Collins-Sussman , B.W.F., C. Michael Pilato, *Control de versiones con Subversion.* 2004.
26. Mastrapa, Y.R. *La gestión de errores en el desarrollo de software*.
27. AbartiaTeam. *Gestión de Proyectos con dotProject*. Available from: http://www.abartiateam.com/dotproject.
28. Foundation, E. 2009; Available from: http://www.eclipse.org/.
29. Group, T.P.G.D. *PostgreSQL 8.3.7 Documentation*. [cited 2009; Available from: http://www.postgresql.org/docs/8.3/interactive/intro-whatis.html.

30. Eviware. *SoapUI*. Available from: http://www.soapui.org/eclipse/index.html.
31. Jonson, R., *Professional Java Development with the Spring*. 2005: Wiley Publishing.
32. Walls, C.B., Ryan *Spring in Action*. 2005: Manning Publications.
33. Poutsma, A.E., Rick ; Abed ,Tareq *Spring Web Services - Reference.* 2005 -2007.
34. *iBatis*. Available from: http://ibatis.apache.org/overview.html.
35. Meadors, C.B.B.G.L., *iBATIS in Action*. 2007: Manning Publications Co.
36. Breidenbach, C.W.w.R., *Spring in Action Second Edition*. 2008.
37. Spring Security 2.0.0 Released; Available from: http://www.acegisecurity.org.
38. Mak, G., *Spring Recipes: A Problem-Solution Approach* 2008: Apress.
39. Gamma, J.M.V.R.H.R.J.E., *Design Patterns: Elements of Reusable Object-Oriented Software* 1995: AddisonWesley Professional Computing Series.
40. Center, M.p.p.D. *msdn*. 2005 [cited 2008; Available from: http://msdn.microsoft.com/en-us/library/aa480600.aspx.
41. Jendrock, J.B.D.C.I.E.S.F.K.H.E., *The Java™ EE 5 Tutorial.* 2006.
42. msdn. *Seguridad de los mensajes en WCF*. Available from: http://msdn.microsoft.com/es-es/library/ms733137.aspx.
43. Iturmendi, J.J.A., *Auditoría informática en la empresa* 1994: Paraninfo

Glosario de Términos

AGI: Es la sigla para Asterisk Gateway Interface. Es el puerto del que se sirve Asterisk para conectarse con otras aplicaciones.

AOP (Aspect-Oriented Programming): La Programación Orientada a Aspectos es un nuevo modelo de entender la programación, tanto su estructura como el flujo de control. Se centra en aspectos, entendiendo por tal un servicio que es transversal, es decir, que es utilizado en gran cantidad de clases de la aplicación.

API (Application Programming Interface): Una interfaz de programación de aplicaciones es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

ASTERISK: El Asterisk es un software completo en PBX, actúa en el Linux y provee todas las configuraciones que esperas de un PBX y más. Asterisk hace VoIP en tres protocolos y puede inter operar con equipos de telefonía estándar básicas usando un hardware relativamente sin costo.

BIND (Berkeley Internet Name Domain): Es el servidor de DNS más comúnmente usado en Internet. Una nueva versión de BIND (BIND 9) fue escrita desde cero en parte para superar las dificultades arquitectónicas presentes anteriormente para auditar el código en las primeras versiones de BIND, y también para incorporar DNSSEC (DNS Security Extensions).

DNS (Domain Name System): Es una base de datos distribuida y jerárquica que almacena información asociada a nombres de dominio en redes como Internet. Aunque como base de datos el DNS es capaz de asociar diferentes tipos de información a cada nombre.

EJB (Enterprise JavaBeans): Es un interface de programación desarrollada por la empresa Sun Microsystems que define una arquitectura de componentes para realizar sistemas cliente/servidor multicapa.

Framework: En el desarrollo de software, es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado.

IDE (Integrated Development Environment): Un entorno de desarrollo integrado es un programa compuesto por un conjunto de herramientas para un programador. 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.

IEEE (The Institute of Electrical and Electronics Engineers): El Instituto de Ingenieros Eléctricos y Electrónicos es una asociación técnico-profesional mundial dedicada a la estandarización.

JDK (Java Development Kit): Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java.

JSP (JavaServer Pages): Es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.

Keytool: Es una herramienta para el manejo de certificados digitales y claves.

NAPTR (Name Authority Pointer): Es un tipo de registro de recursos utilizados en el DNS. Los punteros NAPTR proveen un mecanismo general, flexible extensible y estándar para pasar nuevos tipos de información.

ORM (Object-Relational Mapping): El mapeo objeto-relacional 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 relacional.

Plug-in: Es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande.

POJO (Plain Old Java Object): Es una sigla creada por Martin Fowler, Rebecca Parsons y Josh MacKenzie y utilizada por programadores lenguaje de programación Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.

PSTN (Public Switched Telephone Network): Red pública de telefonía conmutada - es la concentración de las redes públicas mundiales de circuitos conmutados, al igual que Internet es la concentración de redes públicas mundiales de paquetes conmutados basados en IP.

Servlets: Son objetos que corren dentro del contexto de un contenedor de servlets (ej. Tomcat) y extienden su funcionalidad. La palabra servlet deriva de otra anterior, applet, que se refería a pequeños programas escritos en Java que se ejecutan en el contexto de un navegador web. Por contraposición, un servlet es un programa que se ejecuta en un servidor.

SOAP (Simple Object Access Protocol): Protocolo para el intercambio de mensajes que utiliza XML y es independiente de la plataforma o sistema operativo, utilizado para codificar información en los diálogos de los Servicios Web.

SSL (Secure Sockets Layer): Protocolo de comunicación de datos para transmitir documentos privados a través del Internet. SSL utiliza un sistema criptográfico que emplea dos llaves para encriptar los datos, una llave pública que puede ser compartida y otra privada o secreta conocida solo por el receptor del mensaje. Por convención las direcciones (URLS) de sitios que utilizan SSL empiezan con https en lugar de http.

URI (Uniform Resource Identifier): Los Identificadores de Recursos Uniformes o URI pueden definirse como un puntero único a las direcciones de Internet. Las URI son cadenas de caracteres que identifican recursos como documentos, imágenes, archivos, bases de datos, direcciones de correo electrónico u otros recursos o servicios en un formato estructurado común.

XML (Extensible Markup Language): Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XSD (XML Schema Definition): Recomendación de la World Wide Web Consortium (W3C) que especifica la manera de describir formalmente los elementos de un documento XML. En general, un esquema es una representación abstracta de las características de un objeto y relación con otros objetos.

ANEXOS

Anexo 1: Pautas de Codificación

En presente anexo constituye un resumen del estándar de codificación del proyecto TeleIdentificador Personal, exponiendo los principales elementos a tener en cuenta en el desarrollo de la Plataforma Manejadora de Peticiones.

1. Consideraciones Generales.

Con el objetivo de ajustarse a los estándares internacionales se establece utilizar el idioma inglés para todos los elementos que intervienen en este proceso de codificación, dígase nombre de clases, variables, constantes, paquetes, comentarios, ficheros, entre otros.

2. Convenio de Nombres

Los nombres deben ser cortos y descriptivos, facilitando el entendimiento del código generado. Se deben utilizar comentarios en todos los casos que sean normados y en caso de que el programador lo considere necesario. Para los nombres se establecen las siguientes reglas:

Tipo	Regla	Ejemplo
paquetes	Se establece que la estructura de paquetes para todo el código que se genere debe comenzar con “cu.uci”	package cu.uci.data; package cu.uci.xml.shema;
paquetes	Los nombres de los paquetes comenzarán con minúscula en caso de utilizar palabras compuestas se utiliza minúscula.	package cu.uci.systemdata;
clases	Los nombres de las clases serán un sustantivo y comenzaran con mayúscula, se utiliza mayúscula en las distintas palabras de los nombres compuestos. Evitar “_”	class Computer; class PriorityQueue; class System_Map; //Evitar

clases	Mantener nombres simples y evitar abreviaturas o acrónimos a no ser que sean ampliamente conocidos. Ej. URL.	class IPFSystem; //Evitar
		class URLMap;

ficheros	Los ficheros que no se consideren fuente deben comenzar con minúscula. Se deben utilizar nombres simples.	readme.txt build.xml
interfaces	Las interfaces seguirán las mismas reglas que las clases. Comenzando en todos los casos con el prefijo I.	interface IComparable; interface ISerializable;

métodos	Los métodos deben ser verbos, comenzando con minúscula y en casos de utilizar nombres compuestos se utiliza mayúscula para las palabras internas. Evitar “_”.	run();
		runFast();
		getBackground();
		show_UserName(); //Evitar

métodos	En el caso de que el método retorne un “bool” se debe utilizar el prefijo “is” y mantener regla de nombre de los métodos.	bool isempty();
		bool isleaf();
		bool ishumanBeing();

variables	Las variables deben tener nombres cortos y alegóricos a su contenido, comenzando siempre con minúscula y utilizando mayúsculas en los nombres compuestos.	int i; string userName;
constantes	Las constantes deben ser escritas con letras mayúsculas y separando las palabras compuestas por un “_”	int MIN_WIDTH = 8; int MAX_HEIGHT = 10;

3. Alineación.

Se deben utilizar 4 espacios como unidad de alineación en ves de utilizar la tecla TAB (8 espacios).

3.1 Longitud de la Línea.

Evitar utilizar líneas con más de 80 caracteres de longitud, estas no son bien manejadas por algunas herramientas y terminales.

3.2 Líneas plegadas.

Cuando una expresión no cabe en una línea simple debido a su extensión debemos dividirla en más de una línea, siguiendo las siguientes precisiones:

- Dividir después de una coma.
- Dividir después de un operador.
- Alinear la nueva línea al inicio de la expresión en el mismo nivel que la línea anterior.
- Si las reglas anteriores hacen que el código sea confuso, o son confusas de codificar simplemente utilice 8 espacios como alineación.

4. Comentarios.

En los programas de Java existen dos tipos de comentarios: comentarios de implementación (Doc Comment) y comentarios de documentación (Imp Comment). Los Comentarios de documentación en Java, están delimitados por /**……*/, estos pueden ser extraídos a ficheros HTML utilizando la herramienta “javadoc”.

Los comentarios deben ser usados para dar una visión general del código y proveer información adicional que no está fácilmente entendible en el código fuente en sí.

Los comentarios no deben ser encerrados en grandes cajas dibujadas con asteriscos u otros caracteres, y

no deben nunca incluir caracteres especiales.

4.1 Formato de los comentarios en la implementación.

Los programas pueden tener 4 estilos de comentarios: bloque, línea simple, de seguimiento y de fin de línea.

4.1.1 Bloque de comentarios.

Los bloques de comentarios son utilizados para proveer descripciones de ficheros, métodos, estructuras de datos y algoritmos. Estos deben ser utilizados al inicio de cada fichero y al inicio de cada método. Pueden ser utilizados en otras partes por ejemplo dentro de los métodos, los bloques de comentarios dentro de los métodos deben tener la misma alineación que el código que describe.

Un bloque de comentario debe ser precedido por una línea en blanco para separarlo del resto del código. Los bloques de comentario tienen solo un asterisco al inicio de cada línea exceptuando la primera.

Ejemplo:

```
/*
*   Este es un bloque de comentarios.
*/
```

4.1.2 Comentarios de Línea Simple.

Los comentarios cortos pueden aparecer en una sola línea alineada al mismo nivel que el código que lo sigue, además debe estar precedido de una línea en blanco. Si un comentario no puede ponerse en una línea simple entonces debe utilizarse un bloque de comentario.

Ejemplo:

```
if (condition) {
    /* Handle the condition. */
    ...
}
```

4.1.3 Comentarios de Seguimiento.

Comentarios muy cortos pueden aparecer al final de la línea de código que describen, pero deben ser alejados lo suficiente para separarlo de las sentencias. Si más de un comentario de seguimiento aparece en un trozo de código deben tener la misma alineación. Evitar el estilo de comentar cada línea de código que se usa en lenguaje ensamblador.

```
if (a == 2) {
   return TRUE;              /* special case */
} else {
   return isprime(a);        /* works only for odd a */
}
```

4.1.4 Comentarios de fin de línea.

El delimitador de comentario // puede convertir en comentario una línea completa o una parte de una línea. No debe ser usado para hacer comentarios de varias líneas consecutivas; sin embargo, puede usarse en líneas consecutivas para comentar secciones de código.

```
if (foo > 1) {
// Hacer algo.
...
}
else {
return false;      // Explicar aquí por que.
}
```

4.2 Comentarios de la documentación.

Los comentarios de documentación describen clases Java, interfaces, constructores, métodos y atributos. Cada comentario de documentación se encierra con los delimitadores de comentarios /**...*/, con un comentario por clase, interface o miembro (método o atributo). Este comentario debe aparecer justo antes de la declaración:

```
/**
* La clase Ejemplo ofrece...
*/
public class Ejemplo {...
```

5. Declaraciones.

5.1 Número de declaraciones por línea.

Se recomienda una declaración por línea, ya que facilita los comentarios. En otras palabras, se prefiere:

```
int  nivel;         // nivel de indentación
int  tam;           // tamaño de la tabla
```

antes que:

```
int  level, size;
```

No debe poner diferentes tipos en la misma línea.

```
int foo, fooarray[];        //ERROR !
```

5.2 Colocación.

Poner las declaraciones solo al principio de los bloques (un bloque es cualquier código encerrado por llaves "{" y "}".) No esperar al primer uso para declararlas; puede confundir a programadores no pres avisados y limitar la portabilidad del código dentro de su ámbito de visibilidad.

```
void myMethod () {
      int int1 = 0;            // comienzo del bloque del método

      if (condition) {
          int int2 = 0;       // comienzo del bloque del "if"
          ...
      }
}
```

La excepción de la regla son los índices de bucles for, que se pueden declarar en la sentencia for:

```
for (int i = 0; i < maximoVueltas; i++) {...}
...
}
}
```

Evitar las declaraciones locales que ocultan declaraciones de niveles superiores. Por ejemplo, no declarar la misma variable en un bloque interno:

```
int cuenta;
...
miMetodo() {
    if (condicion) {
        int cuenta = 0;    // EVITAR!
        ...
    }
    ...
}
```

5.3 Declaración de Clases e Interfaces.

Al codificar clases e interfaces de Java, se siguen las siguientes reglas de formato:

- Ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros.
- La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración.
- La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{"

```
class Ejemplo extends Object {
    int ivar1;
    int ivar2;

    Ejemplo (int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int metodoVacio() {}
    ...
}
```

6. Sentencias.

6.1 Sentencias simples.

Cada línea debe contener solo una sentencia:

```
object.Method();
```

6.2 Sentencias compuestas.

Deberán estar encerradas en un par de llaves:

```
{
        objeto.Method1();
        objeto.Method2();
        …
}
```

6.3 Sentencias de RETURN.

Las sentencias de RETURN no deben contener paréntesis exteriores innecesarios:

Incorrecto:
return (n * (n + 1) / 2);

Correcto:
return n * (n + 1) / 2;

6.4 Sentencias de condiciones simples y anidadas.

6.4.1 IF, IF ELSE, IF ELSE IF ELSE.

```
if (condition) {
    DoSomething();
    ...
}
if (condition) {
    DoSomething();
    ...
} else {
    DoSomethingOther();
    ...
}
if (condition) {
    DoSomething();
    ...
} else if (condition) {
    DoSomethingOther();
    ...
} else {
    DoSomethingOtherAgain();
    ...
}
```

6.4.2 SWITCH CASE

```
switch (condition) {
    case A:
        ...
        break;
    case B: {
        ...
        } break;
    default:
        ...;
}
```

6.5 Sentencias de Lazo.

6.5.1 FOR.

Una sentencia for debe tener la siguiente forma:

```
for (inicialización; condicion; actualización) {
sentencias;
}
```

Al usar el operador coma en la clausula de inicialización o actualización de una sentencia for, evitar la complejidad de usar más de tres variables. Si se necesita, usar sentencias separadas antes de bucle for (para la clausula de inicialización) o al final del bucle (para la clausula de actualización).

6.5.2 WHILE.

Una sentencia while debe tener la siguiente forma:

```
while (condicion) {
sentencias;
}
```

6.5.3 DO WHILE.

Una sentencia do-while debe tener la siguiente forma:

```
do {
sentencias;
    } while (condicion);
```

6.5.4 FOREACH.

Se deben utilizar las llaves aunque sólo se haga una única acción en el ciclo.

```
for (lista : iterador)
{
...
}
```

6.6 Sentencias TRY CATCH

Una sentencia try-catch debe tener la siguiente forma:

```
try {
sentencias;
} catch (ExceptionClass e) {
```

```
sentencias;
}
```

Una sentencia try-catch puede ir seguida de un finally, cuya ejecución se ejecutará independientemente de que el bloque try se halla completado con éxito o no.

```
try {
sentencias;
} catch (ExceptionClass e) {
sentencias;
} finally {
sentencias;
}
```

7. Espacios en blanco.

7.1 Espacios

Se deben usar espacios en blanco en las siguientes circunstancias:

- Una palabra clave del lenguaje seguida por un paréntesis debe separarse por un espacio. Ejemplo:

  ```
  while (true) {
  ...
  }
  ```

- Debe aparecer un espacio en blanco después de cada coma en las listas de argumentos.
- Todos los operadores binarios excepto el operador(.) se deben separar de sus operandos con espacios en blanco. Los espacios en blanco no deben separar los operadores unarios, incremento ("++") y decremento ("--") de sus operandos. Ejemplo:

  ```
  a += c + d;
  a = (a + b) / (c * d);
  while (d++ == s++) {
  n++;
  }
  ```

- Las expresiones en una sentencia for se deben separar con espacios en blanco. Ejemplo:

  ```
  for (expr1; expr2; expr3)
  ```

- Los "Cast"s deben ir seguidos de un espacio en blanco Ejemplos:

  ```
  miMetodo((byte) unNumero, (Object) x);
  miMetodo((int) (cp + 5), ((int) (i + 3))
  + 1);
  ```

7.2 Líneas en blanco.

Las líneas en blanco mejoran la facilidad de lectura separando secciones de código que están lógicamente relacionadas.

Se deben usar siempre dos líneas en blanco en las siguientes circunstancias:

- Entre las secciones de un fichero fuente
- Entre las definiciones de clases e interfaces.

Se debe usar siempre una línea en blanco en las siguientes circunstancias:

- Entre métodos
- Entre las variables locales de un método y su primera sentencia
- Antes de un comentario de o de un comentario de una línea
- Entre las distintas secciones lógicas de un método para facilitar la lectura.