Universidad de las Ciencias Informáticas

Facultad 2

Título: Teleidentificador Personal: Desarrollo de la plataforma manejadora de peticiones. Capa de servicios web y módulo de seguridad

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

Autores: Pavel Ernesto Navarro Guerrero
Abel Alejandro Yáñez Moncada

Tutor: Ing. Yunisel Viera Vargas

Ciudad de La Habana, Junio de 2009

"Año 50 del triunfo de la Revolución"

"Déjeme decirle, a riesgo de parecer ridículo, que el revolucionario verdadero está guiado por grandes sentimientos de amor…"

Ernesto Che Guevara de la Serna

Declaración de autoría

Declaramos ser los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Pavel Ernesto Navarro Guerrero	Abel Alejandro Yáñez Moncada	Ing. Yunisel Viera Vargas
_____	_____	_____
Firma del autor	Firma del autor	Firma del tutor

Datos de Contacto:
Autores:
aamoncada@estudiantes.uci.cu
penavarro@estudiantes.uci.cu

Tutor: Ing. Yunisel Viera Vargas.

Síntesis del Tutor: Ingeniero en Ciencias Informáticas, graduado el La Universidad de las Ciencias Informáticas. Con categoría docente de Instructor, ejerce como profesor en la facultad 2 dentro de la misma escuela.

Agradecimientos

Deseamos agradecerle a todas las personas que de una forma u otra han contribuido con el desarrollo de nuestro trabajo de curso: a nuestros amigos, a nuestros compañeros de grupo y de proyecto y a nuestros profesores.

También desearíamos agradecerle muy especialmente a Daymi y Mayrilis, por estar siempre presentes y ayudándonos en todo momento y por el excelente trabajo que realizaron siendo nuestras arquitectas.

A Rammel, Frank y Rasiel por brindarnos ayuda con sus conocimientos y auxiliarnos en todo lo posible para lograr el desarrollo del presente trabajo.

Al máster en Informática José Andrés Yáñez Menéndez por todo el apoyo brindado y la disposición para contribuir en todo lo que hiciera falta para que este trabajo saliera adelante.

A nuestro tutor, el profesor Yunisel Vargas Viera por el apoyo que nos brindo en nuestra investigación.

Al modelo socialista de nuestra revolución, por permitirles a personas sencillas convertir en verdad sus sueños.

Dedicatoria

Pavel:

A mi mamá, por ser la persona más importante para mí, la que siempre ha estado a mi lado, en todos los buenos y malos momentos. Por sus infinitos consejos y su apoyo. Por ser mi amiga y consejera.

A mi papá, por su ejemplo y sus consejos. Por su constante preocupación por mi y por mi desarrollo. Por ser un modelo ejemplar como padre y revolucionario. Por ser este, uno de sus sueños.

A mi novia Dianelys, por ser la personita más especial del mundo y que más quiero. Por su infinita ternura. Por darme toda la felicidad que nunca había imaginado encontrar.

A mis hermanos, por ser tan especiales y diferentes. Por todos los buenos recuerdos que siempre tengo de ellos y por todo lo que me malcriaron cuando niño.

A mi abuela Nena, por ser, sin quererlo, el centro de atención de nuestra familia.

A mi tío Papilando y mi tía Mamitere, por ser para mí, como dos padres. Por el gran cariño que me han hecho tenerles por ser tan especiales.

A mis primos Orly y Armandito, por todo lo que me han enseñado en la vida, por su infinita sabiduría. Por ser como dos hermanos más y por lo bien que siempre me siento cuando comparto con ellos.

A tía Nélida y tío Polo por ser tan queridos por mí y por todos. Por todo lo que me han ayudado en la vida y ser los abuelos que siempre hubiera deseado tener.

A mis amigos, a los que conocí aquí en la UCI y los que vienen conmigo del Pre. A ustedes, que siempre llevaré en el corazón. Por todos los buenos y malos momentos que vivimos juntos. Por todo lo que me han enseñado y lo que nos hemos divertido.

Abel:

A mis padres por estar siempre a mi lado, cuidarme, apoyarme en todo lo que me haga falta y darme el mejor ejemplo que pueda recibir de lo que es una familia.

Al Joe por todo los momentos felices que hemos compartido juntos, y también por los sustos que he pasado con él.

A mi hermanita Nani a la que quiero mucho y que tuvo que soportarme en mi niñez, y por si fuera poco ahora tiene que lidiar con una pandilla.

A mis sobrinitos los cuales son muy originales y la verdad que no son fáciles.

A mi abuela mima la cual quiero muchísimo por su forma de ser, y ser la persona más sencilla que conozco.

A mis abuelos mima lola y pipo que siempre se han preocupado por mí, los quiero mucho y gracias por ser parte de mi vida.

A mi abuela Aba por todo el cariño que siempre me demostró.

A mis tíos Niurka, Tania y Carlitos, por ser uno de los que contribuyeron y me sirvieron de inspiración para estudiar esta carrera y a mi tío Víctor al cual admiro mucho y respeto.

A mis tíos Isa, Gero y Clarita por esa alegría que siempre los caracteriza, por todo el cariño que me tienen, y por estar siempre dispuestos a ayudar en cualquier cosa que haga falta, creo que es difícil tener unos mejores que ellos.

A mis primos por todos los momentos que disfrutamos juntos en la infancia, tanto de un lado como de otro, a todos, gracias.

A todos los amigos que hice aquí en la universidad, que sin dudas es lo mejor que tiene haber cogido la UCI, el haber conocido amigos de todo el país, que sin dudas a pesar de que nos separemos y tomemos caminos distintos, nunca los olvidaré.

A Penny que no por ser la última es la menos importante, por haber compartido conmigo toda mi juventud y demostrarme tanto cariño.

Resumen

En el presente trabajo de diploma se expone el desarrollo de la capa de servicios web y el manejo de la seguridad de la Plataforma Manejadora de Peticiones del Teleidentificador Personal, utilizando las capacidades del framework Spring Web Services 1.5, el Entorno de Desarrollo Integrado Eclipse 3.4, la plataforma Java 2, Enterprise Edition con la JDK 6.0, y la herramienta SoapUI para realizar las pruebas funcionales.

Los componentes implementados en la capa de servicios exponen las funcionalidades de las capas inferiores, específicamente de la capa de negocio, posibilitándole a los desarrolladores de la misma, centrarse en la implementación de las reglas de negocio o requerimientos del sistema.

Palabras Claves

Capa de servicios web, Seguridad.

Índice

Introducción

La Empresa de Telecomunicaciones de Cuba S.A. (ETECSA), atendiendo a su misión institucional y en su afán de mejorar la prestación de los servicios públicos de telecomunicaciones en Cuba, ha decido brindar un novedoso servicio: La correspondencia entre números telefónicos, más conocido como ENUM[1] (del inglés Telephone Number Mapping) que se considera el primer servicio de convergencia entre los servicios suministrados por las redes de conmutación de circuitos y las redes de conmutación de paquetes.

ENUM es un protocolo para unificar el sistema de numeración telefónica E.164 con el sistema de direccionamiento de Internet DNS[2] por medio del uso de un método de búsqueda indirecta para obtener registros NAPTR[3]. Los registros son guardados en una Base de Datos DNS y permite relacionar a cada número telefónico con un único nombre de dominio.

Aunque conceptualmente sencillo, el ENUM trae consigo retos complejos a resolver, por lo cual ETECSA en convenio con la Universidad de las Ciencias Informáticas, ha comenzado el desarrollo del "Proyecto Desarrollo de ENUM de Usuario en la Intranet de ETECSA" que posibilitará el despliegue profesional de este servicio por la Empresa.

---

[1] ENUM del **E**.164 **NU**mber **M**apping y en español mapeo de números telefónicos utilizando el estándar de numeración telefónica E.164 (13).

[2] El **Domain Name System** (DNS) 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, los usos más comunes son la asignación de nombres de dominio a direcciones IP y la localización de los servidores de correo electrónico de cada dominio (13).

[3] Un **Name Authority Pointer** (NAPTR) es un tipo de registro usado en el DNS, el cual es pieza clave de la infraestructura de Internet (13).

ENUM de Usuario sigue el objetivo de desarrollar un identificador personal de telecomunicaciones que permita a cada llamante, conociendo un solo número de la persona con la que desea comunicarse, independientemente del contenido a comunicar (llamada de voz, correo, videoconferencia, etc.), establecer comunicación con el mismo, o sea acceder con este a todos sus servicios de telecomunicaciones y de internet desde cualquier lugar, en cualquier momento y desde cualquier dispositivo electrónico. Es por eso que se le conoce también como Teleidentificador Personal o TIP.

Para implementar el servicio Teleidentificador Personal, se necesita de un conjunto de plataformas y sistemas que permitan gestionar las peticiones de los clientes y acceder a dicho servicio. Estos sistemas están desarrollados en una gran variedad de lenguajes y hacen uso de diversas tecnologías por lo que se hace muy engorroso manejar cada una de dichas peticiones de manera independiente e implementar la misma lógica de negocio para cada una por separado. Además se debe permitir que un futuro nuevas aplicaciones puedan integrarse de forma flexible.

Esto da lugar a una **situación problémica** de que existe la necesidad de crear un sistema que integre las aplicaciones que necesitan interactuar con el servidor DNS, y que brinde un grupo de funcionalidades comunes para que accedan de forma estándar y fiable a la información almacenada de los subscritores, además debe soportar la integración de futuras aplicaciones que complementen el servicio ENUM.

Teniendo como **problema científico**: ¿Cómo integrar las diversas tecnologías y aplicaciones que implementan el servicio ENUM para que accedan al servidor DNS de una manera segura y uniforme?

El objeto de estudio de este trabajo son las capas de servicios web y la seguridad de las aplicaciones empresariales; y el **campo de acción** la capa de servicios web y la seguridad de la plataforma manejadora de peticiones del Teleidentificador Personal.

Persiguiendo como **objetivo general**: Implementar la capa de servicios web y la seguridad de la plataforma manejadora de peticiones, para que permita la conexión desde diversas tecnologías y mantenga en todos los casos una elevada seguridad.

Para lo cual se llevarán a cabo las siguientes **tareas investigativas**:

- Estudiar y analizar los lenguajes y las herramientas vinculadas al desarrollo de una capa de servicios web para evitar en su selección usar simplemente la intuición.
- Estudiar y analizar las tecnologías relacionadas con la seguridad en aplicaciones desarrolladas en capas.
- Estudiar y analizar métodos de prueba para realizar pruebas de aceptación a los servicios web.

Capítulo 1 "Fundamentación teórica", donde se incluyen los resultados del estudio sobre el estado del arte a nivel internacional y nacional sobre el desarrollo de ENUM; se describen los conceptos principales que se van a tratar a lo largo del trabajo así como el análisis de las principales herramientas a utilizar en el desarrollo de la aplicación.

Capítulo 2 "Características del Sistema", Se define el objeto de automatización, se hace una descripción general de la propuesta de sistema y de cómo debe funcionar el negocio, a través de procesos. Se definen las funcionalidades del sistema.

Capítulo 3 "Implementación de la Capa de Servicios": Se describe cómo se implementó la capa de servicios y los elementos más significativos de la misma.

Capítulo 4 "Manejo de la seguridad y Pruebas a los servicios": Se describen como se implementaron los aspectos de seguridad en el TeleIdentificador personal y las pruebas realizadas a los servicios.

CAPITULO 1

Fundamentación Teórica

1.1 Introducción

El desarrollo de este capítulo tiene como objetivo investigar las metodologías, lenguajes y herramientas existentes en la actualidad para implementar el sistema. Así como definir conceptos importantes para entender la investigación.

1.2 ENUM

La red telefónica permite a cualquier usuario, en cualquier parte del mundo, ponerse en contacto con otro si conoce su número, el cual es único y es administrado por la Unión Internacional de Telecomunicaciones (ITU por sus siglas en inglés) según su recomendación E.164 que asigna los prefijos que luego administra cada país.

El estándar E.164 establece un número telefónico con una cadena de cifras decimales que indica, unívocamente, el punto de terminación en la red pública. El número contiene la información necesaria para encaminar la llamada a este punto de terminación.

Para interrelacionar números de teléfonos con direcciones IP[4] es necesario una correspondencia biunívoca, de forma que cualquier persona que disponga de un número de teléfono debería de tener acceso a un registro IP asociado a ese número y viceversa.

El proyecto ENUM (ElectronicNUMber/tElephone NUmber Mapping) surge en un grupo de trabajo de la Internet Engineering Task Force[5] (IETF) el cual, inicialmente, analiza la posibilidad de utilizar nombres de dominio numéricos que permitan interrogar a un servidor DNS para encontrar información asociada a ese número (por ejemplo una página web, un email, un móvil, o una dirección VoIP[6]).

ENUM es un mecanismo para establecer una correspondencia de números E.164 con identificadores de recursos uniformes (URI, del Inglés: Universal Resource Identifier), a los cuales están asociadas aplicaciones de comunicación.

ENUM no es voz sobre IP ni voz sobre Internet, no es telefonía, no es una aplicación en sí mismo. ENUM es una base de datos que se le interroga con un número y responde con una serie de registros. Es un facilitador para desarrollar aplicaciones puesto que permite traducir un número telefónico en un punto de contacto a través de IP.

ENUM no realiza ni procesa llamadas pero da información para que una aplicación o un dispositivo puedan realizar dicha llamada.

Un dispositivo ENUM está en Internet, la interrogación de ENUM se hace en Internet, la base de datos ENUM trabaja y forma parte de la base de datos y de la arquitectura DNS. Solo una vez que el usuario o la aplicación obtienen la información que ENUM le proporciona, podría proceder a realizar una llamada en la que intervenga la red telefónica.

ENUM funciona sobre una arquitectura basada en DNS y protocolos para el mapeo de números E.164 en URI's, de forma que toda la información relacionada con un usuario se centralice en el DNS, almacenada en los registros NAPTR. Estos registros se utilizan para identificar formas disponibles de contactar con un usuario, teniendo presente que para cada uno existen diferentes vías de comunicación.

---

[4] Una dirección **IP** es un número que identifica de manera lógica y jerárquica a una interfaz de un dispositivo (habitualmente una computadora) dentro de una red que utilice el protocolo IP (Internet Protocol) (13).

[5] El **IETF** (Internet Engineering Task Force, en español Grupo de Trabajo en Ingeniería de Internet) es una organización internacional abierta de normalización, que tiene como objetivos el contribuir a la ingeniería de Internet, actuando en diversas áreas, tales como transporte, encaminamiento, seguridad. Fue creada en EE. UU. en 1986 (13).

[6] **Voz sobre Protocolo de Internet**, también llamado Voz sobre IP, VozIP, VoIP (por sus siglas en inglés), es un grupo de recursos que hacen posible que la señal de voz viaje a través de Internet empleando un protocolo IP (Internet Protocol) (13).

1.2.1 Estudio de mercado

La Universidad de las Ciencias Informáticas tiene en su misión informatizar la sociedad cubana y producir software con calidad para generar ingresos monetarios por motivo de ventas al país. Para ello se decidió que cada una de las 10 facultades con las que cuenta se especialice en una línea de desarrollo específica. La facultad 2 y en específico el Polo de Telecomunicaciones se dedica a la producción de aplicaciones y servicios para este sector, donde cuenta con experiencias en el área de la producción de juegos para celulares, servicios de mensajería SMS[7] y MMS[8], plataformas de gestión de contenidos y portales WAP, esto se ha logrado por la vinculación de estudiantes y profesores a proyectos productivos, en especial los trabajos desarrollados en la entidad Procyonsoluciones.

El desarrollo del proyecto TeleIdentificador Personal contribuirá en gran medida a mejorar el estado de las comunicaciones del país. Está dirigido a toda la población y cumple con todas las normas y estándares establecidos para el desarrollo de servicios para las telecomunicaciones.

1.2.2 Desarrollo de ENUM de Usuarios a nivel nacional e internacional

A nivel mundial, la portabilidad numérica en las redes de telecomunicaciones es considerada un factor esencial que contribuye al desarrollo de la competencia de los servicios de telecomunicaciones en la medida que elimina una barrera a la entrada de nuevos operadores y permite la utilización eficiente de la numeración.

El surgimiento del protocolo ENUM permite la conversión de los números telefónicos de la recomendación E164 del ITU-T a nombres de dominio que utilizan las redes IP, cuyos principios han sido establecidos por la Internet Engineering Task Force (IETF) en la recomendación RFC3761[9]. La arquitectura desarrollada en base al plan de numeración internacional de la recomendación E164 y nombres de dominio DNS tiene un enorme potencial comercial ya que permite bajo un número único integrar servicios de voz, video, datos, solucionando de manera natural y ordenada la introducción de la portabilidad del número y de la telefonía IP a nivel nacional. Esto, en todo caso, conlleva a la migración de las redes de telefonía convencional a redes de última generación (NGN del inglés Next Generation Networks).

---

[7] El servicio de mensajes cortos o SMS (Short Message Service) es un servicio disponible en los teléfonos móviles que permite el envío de mensajes cortos entre teléfonos móviles, teléfonos fijos y otros dispositivos de mano.

[8] Multimedia Messaging System (MMS) o sistema de mensajería multimedia es un estándar de mensajería que le permite a los teléfonos móviles enviar y recibir contenidos multimedia, incorporando sonido, video, fotos o cualquier otro contenido disponible en el futuro.

[9] La recomendación RFC 3761 de la IETF establece el uso de los Sistemas de Nombres de Dominio(DNS) para el almacenamiento de los números E.164 (13)

Actualmente existen algunos países que han puesto en marcha el desarrollo del servicio ENUM, a continuación se puede apreciar el estado en que se encuentra cada uno de ellos:

País	E.164	Estado
Austria	43	Explotación
República Checa	420	Explotación
Finlandia	358	Explotación
Alemania	49	Explotación
Irlanda	353	Explotación
Polonia	48	Explotación
Rumanía	40	Explotación
Holanda	31	Explotación
China	86	Prueba
Francia	33	Prueba
Japón	81	Prueba
Liechtenstein	423	Prueba
Australia	61	Detenido
Suecia	46	Detenido

Tabla 1. Estado de desarrollo de Enum en el mundo.

1.3 Plataforma manejadora de peticiones

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

Principales funcionalidades:

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

Estandarizar el acceso a la información: Todo el acceso a los DNS y la BD ENUM se realizará a través de la plataforma, separando así esta responsabilidad de las aplicaciones clientes.

Integrar aplicaciones: La plataforma contará con una capa de servicios que permitirá integrar y comunicar todas las aplicaciones que se desarrollen por separado y que requieran interactuar con la información de los subscriptores almacenada en el BIND DNS y la base de datos ENUM.

1.4 Servicios Web

La World Wide Web Consortium lo define como *"…un sistema de software diseñado para soportar interacción interoperable máquina a máquina sobre una red. Este tiene una interface descrita en un formato procesable por una máquina (específicamente WSDL). Otros sistemas interactúan con el servicio web en una manera prescrita por su descripción usando mensajes SOAP, típicamente enviados usando HTTP[10] con una serialización XML en relación con otros estándares relacionados con la web"* [1]

A mediados de la década de los 90 y con la aparición de Internet y su posterior masificación a niveles jamás pensados, ha existido siempre la necesidad e inquietud por parte de las empresas desarrolladoras de software de buscar o contar con la manera de lograr la integración entre sistemas heterogéneos, o sea tanto a nivel del software como del hardware. Para tal efecto muchas compañías fueron creando de forma individual la mejor manera de lograr esta integración. Muchas empresas comenzaron una loca carrera para generar la mejor tecnología integradora de sistemas, pero a medida que la competencia se hacía cada vez más fuerte, la integración se hacia cada vez más difícil.

Debido a la masificación de Internet a niveles insospechables y al impacto causado por las tecnologías de la información en las últimas dos décadas del siglo pasado, la forma de hacer negocios y la comunicación entre las personas y las empresas cambió de una manera significativa. Bajo este contexto se hacía cada vez mayor la necesidad de integrar y compartir información entre distintas plataformas de software y hardware.

[10] El protocolo de transferencia de hipertexto (HTTP, HyperText Transfer Protocol) es el protocolo usado en cada transacción de la Web (WWW). HTTP define la sintaxis y la semántica que utilizan los elementos software de la arquitectura web (clientes, servidores, proxies) para comunicarse.

Las empresas se percataron que era imposible crear una plataforma integradora de forma individual, así que decidieron atacar el problema de raíz. Para esto decidieron que en vez de crear la mejor plataforma integradora, era mejor buscar un leguaje común de intercambio de información aprovechando los estándares existentes en el mercado.
Bajo este contexto nacen los Servicios Web basados en XML, los cuales son indispensables a la hora de desarrollar un sistema integrador, como el que es objeto de estudio de este trabajo.

1.4.1 Servicios Web y la evolución hacia la Economía Global

Las aplicaciones web actuales ya no son suficientes. El modelo actual de negocio electrónico no facilita la integración de las aplicaciones de Internet con el resto de software de las empresas. Si las compañías quieren extraer el máximo beneficio de Internet, los Sitios Web deben evolucionar. Este es el contexto en el que surgen los Servicios Web. Los servicios web son componentes de software que permiten a los usuarios usar aplicaciones de negocio que comparten datos con otros programas modulares, vía Internet. Son aplicaciones independientes de la plataforma que pueden ser fácilmente publicadas, localizadas e invocadas mediante protocolos web estándar, como XML[11], SOAP[12], UDDI[13] o WSDL[14]. El objetivo final es la creación de un directorio en línea de servicios web, que pueda ser localizado de un modo sencillo y que tenga una alta fiabilidad.

Aunque la idea de la programación modular no es nueva, el éxito de esta tecnología reside en que se basa en estándares conocidos en los que ya se tiene una gran confianza, como el XML. Además, el uso de los servicios web aporta ventajas significativas a las empresas. El principal objetivo que se logra, es la interoperabilidad y la integración. Mediante los servicios web, las empresas pueden compartir servicios con sus clientes y sus socios de negocio. Esto ayudará a las compañías a escalar sus negocios, reduciendo el coste en desarrollo y mantenimiento de software, y sacando los productos al mercado con mayor rapidez. La integración de aplicaciones hará posible obtener la información demandada en tiempo real, acelerando el proceso de toma de decisiones. La evolución de Internet hacia los servicios web, mejorará los resultados globales de las empresas, reduciendo sus gastos y guiándolas hacia una mejora progresiva de la calidad. La

[11] **XML**: Siglas en inglés de Extensible Markup Language (lenguaje de marcas), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web. (12)
[12] **SOAP** (siglas de 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 (11)
[13] **UDDI** son las siglas del catálogo de negocios de Internet denominado Universal Description, Discovery and Integration.
[14] **WSDL** son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios Web (14).

adopción de la tecnología de servicios web por la industria es el primer paso hacia una economía global.

1.4.2 Seguridad

Actualmente, los servicios web están siendo ampliamente aceptados por las empresas para el desarrollo de software de uso interno. De este modo, los servicios pueden implementar toda su funcionalidad y permanecer seguros tras el cortafuegos de la compañía. Los desarrollos actuales no ayudan a la cooperación entre las empresas ya que no hay ningún estándar establecido sobre las técnicas de seguridad. Debido a la tecnología que es usada por los servicios web, y en concreto al uso de SOAP, las técnicas de seguridad convencionales que se han venido usando en Internet, ya no son suficientes. Con SOAP, cada mensaje simple que se intercambia realiza múltiples saltos y es ruteado a través de numerosos puntos antes de que alcance su destino final. Es por ello que los servicios web necesitan tecnologías que protejan los mensajes desde el principio hasta el final. Existen un conjunto de técnicas que se pueden usar para garantizar la seguridad a nivel de mensaje, estas son:

- Encriptación XML: Evita que los datos se vean expuestos a lo largo de su recorrido.
- Firma Digital XML: Asocia los datos del mensaje al usuario que emite la firma, de modo que este usuario es el único que puede modificar dichos datos.
- XKMS y los Certificados: XKMS (XML Key Management Specification) define servicios web que se pueden usar para chequear la confianza de un certificado de usuario.
- SAML y la Autorización: SAML (Security Assertion Mark-up Language) hace posible que los servicios web intercambien información de autenticación y autorización entre ellos, de modo que un servicio web confíe en un usuario autenticado por otro servicio.
- Validación de datos: Permite que los servicios web reciban datos dentro de los rangos esperados.

Además, también hay técnicas que permiten mantener la seguridad a otros niveles. La seguridad en UDDI permite autentificar todas las entidades que toman parte en la publicación de un servicio web: proveedor, agente y consumidor del servicio. De este modo, nadie podrá registrar servicios en el papel de un proveedor o hacer uso de ellos sin contar con los permisos adecuados.

1.4.2.1 Autenticación y autorización

La **autenticación** es el proceso por el que se comprueba la identidad de alguien o algo, para ver si es lo que dice ser. Ese "alguien" o "algo" se denomina principal. La autenticación requiere pruebas de identidad, denominadas credenciales. Por ejemplo, una aplicación cliente puede presentar una contraseña como sus credenciales. Si la aplicación cliente presenta las credenciales correctas, se asume que es quien dice ser.

Una vez que se ha autenticado la identidad de un principal, deben tomarse decisiones sobre la **autorización**. El acceso se determina comparando la información del principal con información de control de acceso, como listas de control de acceso (ACL del Ingles Access Control List). Es posible que los clientes tengan distintos grados de acceso. Por ejemplo, algunos clientes pueden tener acceso total a los servicios Web XML, mientras que otros estarían autorizados sólo a ciertas operaciones. A algunos clientes se les permitirá un acceso total a todos los datos, mientras que a otros sólo se les permitirá acceso a un subconjunto de los datos y otros tendrán acceso de sólo lectura.

Un modo sencillo de implementar servicios Web XML consiste en aprovechar las características de autenticación del protocolo que se utilice para intercambiar mensajes. Para la mayoría de los servicios Web XML, esto significa aprovechar las características de autenticación disponibles en HTTP.

- **Básica:** utilizada para identificación no segura o poco segura de clientes, ya que el nombre de usuario y la contraseña se envían como texto codificado en base 64, que puede ser fácilmente descodificado.
- **Básica sobre SSL[15]:** igual que la autenticación básica, excepto que el canal de comunicación está cifrado y protege de ese modo el nombre de usuario y la contraseña. Una buena opción para entornos en Internet; sin embargo, el uso de SSL influye negativamente en el rendimiento.

[15] Secure Sockets Layer -Protocolo de Capa de Conexión Segura- (SSL) y Transport Layer Security -Seguridad de la Capa de Transporte- (TLS), su sucesor, son protocolos criptográficos que proporcionan comunicaciones seguras por una red, comúnmente Internet.

- **Implícita:** utiliza algoritmos hash[16] para transmitir las credenciales del cliente de forma segura. Sin embargo, es posible que no sea compatible con todas las herramientas de desarrollo para generar clientes de servicios Web XML.
- **Certificados de cliente a través de SSL:** requiere que cada cliente obtenga un certificado. Los certificados se asignan a las cuentas de usuario, que son utilizadas para autorizar el acceso a los servicios Web XML. Se trata de una solución viable para entornos en Internet, aunque el uso de certificados digitales no está muy extendido actualmente. Es posible que no sea compatible con todas las herramientas de desarrollo para generar clientes de servicios Web XML. Sólo está disponible en conexiones SSL, de modo que el rendimiento puede verse afectado.

Desde la perspectiva de alguien que intenta implementar servicios Web XML, una de las ventajas de utilizar estos mecanismos de autenticación es que no se necesita escribir nuevo código. IIS/ISA Server completa todo el proceso de autenticación y autorización ACL antes de llamar a los servicios Web XML. Sin embargo, al implementar el cliente será necesario realizar algunas tareas adicionales. La aplicación cliente necesitará responder a las peticiones de autenticación y credenciales del servidor.

1.4.2.2 Creación de conexiones seguras

Uno de los métodos más sencillos para aumentar la seguridad de los servicios Web XML consiste en asegurarse de que la conexión entre el cliente de servicios Web XML y el servidor es segura. Esto se puede llevar a cabo mediante varias técnicas, dependiendo de la extensión de la red y el perfil de actividades de las interacciones. Tres de las técnicas más comunes y accesibles son: reglas basadas en un servidor de seguridad, SSL (Secure Sockets Layer) y redes privadas virtuales (VPN).
Si se sabe con exactitud qué equipos van a tener acceso a los servicios Web XML, puede utilizar reglas de servidor de seguridad para restringir el acceso a equipos con direcciones IP conocidas. Esta técnica resulta particularmente útil si desea restringir el acceso a equipos en una red privada virtual y no desea mantener el contenido de los mensajes en secreto (cifrados).

Se puede utilizar SSL para establecer conexiones seguras en redes que no son de confianza (como Internet). SSL cifra y descifra mensajes enviados entre el cliente y el servidor. Al cifrar los

[16] En informática, Hash se refiere a una función o método para generar claves o llaves que representen de manera casi unívoca a un documento, registro, archivo, etc., resumir o identificar un dato a través de la probabilidad, utilizando una función hash o algoritmo hash. Un hash es el resultado de dicha función o algoritmo.

datos, se evita que los mensajes sean leídos por terceros durante la transmisión. SSL cifra el mensaje del cliente y a continuación lo envía al servidor. Una vez que el servidor recibe el mensaje, SSL lo descifra y comprueba que procede del remitente correcto (proceso que se denomina autenticación). El servidor y el cliente, o ambos, pueden tener certificados que se utilizan como parte del proceso de autenticación y que proporcionan capacidades de autenticación además del cifrado de la conexión. Aunque es un método muy eficaz para crear comunicaciones seguras, SSL presenta un coste en rendimiento que debe tenerse en cuenta.

Una red privada virtual es la extensión de una red privada que se conecta a través de redes compartidas o públicas, como Internet. Las redes virtuales privadas permiten enviar datos entre dos equipos en una conexión segura. Si bien su funcionamiento es similar a SSL, una red virtual segura es una conexión punto a punto establecida a largo plazo. De este modo, mejora la eficacia y permite la comunicación con servicios Web XML, pero requiere que se establezca una conexión duradera a largo plazo para obtener un rendimiento mayor.

1.4.3 Calidad

Actualmente ya existen en el mercado algunas herramientas específicamente diseñadas para medir la calidad de los servicios web, pero sigue siendo necesaria una estandarización sobre este tema. Los resultados sobre la calidad de diferentes servicios web, servirán como parámetro de comparación y ayudarán al consumidor a decantarse por un servicio u otro. Para que un servicio web se ejecute con corrección y satisfaga las expectativas creadas, a parte del precio, habrá que tener en cuenta una serie de parámetros como por ejemplo, que los resultados obtenidos del mismo sean los esperados o que el entorno de uso sea amigable. Otro elemento a tener en cuenta es la integración. Aunque teóricamente los servicios web proporcionan conectividad con cualquier software de un modo transparente, cada proveedor de servicios puede adoptar soluciones diferentes que resultan más o menos adecuadas para el consumidor. Analizando la escalabilidad se comprobará el grado de modularidad y flexibilidad del servicio. Por último, también sería interesante analizar las características que ofrece el proveedor de servicios web. Actualmente no hay definidos estándares sobre este tema, pero la mayoría de las empresas ya está demandando algún tipo de acuerdo o contrato con los proveedores, de modo que se pueda garantizar la calidad y la fiabilidad de los servicios por los que se paga.

1.4.4 Estandarización

Los servicios web están basados en el estándar XML, que ha sido universalmente aceptado. Pero la situación para el resto de protocolos es bien distinta. La mayor parte de ellos se encuentran todavía en desarrollo y pueden ser objeto de cambios. Esa es la razón por la que la mayoría de las empresas están esperando a que estos protocolos sean más universales antes de profundizar en esta tecnología. Actualmente, ni SOAP, ni WSDL, ni UDDI han sido oficialmente reconocidos por ningún organismo de estandarización. SOAP es el único que en este momento está en consideración por el World Wide Web Consortium y se encuentra cercano a la estandarización. SOAP y WSDL están siendo ampliamente usados, pero de momento UDDI no ha tenido el mismo éxito. El principal motivo es que las técnicas de seguridad son todavía muy inmaduras y las compañías prefieren hacer uso de registros privados para dar soporte a intercambios privados de servicios web. Recientemente, algunas de las empresas más importantes en el desarrollo de Negocio Electrónico como IBM, Intel, Microsoft u Oracle, han creado el WS-I: organización para la Interoperabilidad de los Servicios Web. El objetivo de dicha organización es la promoción de la estandarización de los servicios web de modo que se fomente la cooperación e interoperabilidad entre las compañías y mercados.

1.4.5 Servicios Web hoy

En la actualidad hay un gran auge del desarrollo de aplicaciones distribuidas que se comunican enviando mensajes SOAP.

El trabajo sobre las plataformas de servicios Web continuará en el futuro, y aparecerán mejoras en tres áreas fundamentales. En primer lugar, se añadirán servicios de más alto nivel. Todo el mundo está de acuerdo en que debe existir un modo estándar de asegurar servicios Web, rutear mensajes, garantizar una entrega fiable de mensajes, definir semántica transaccional, etc. Estas características de propósito general se expanden más allá de los dominios del problema y no hay ninguna razón por la que cada desarrollador de servicios Web deba implementarlas individualmente. Microsoft, IBM y otros están realizando mucho trabajo en estas áreas. La iniciativa Global XML Web Service Architecture (GXA) define un conjunto de especificaciones sobre cómo implementar estos servicios de infraestructura en términos de mensajes SOAP (por ejemplo, de un modo neutral respecto del protocolo de transporte.

En segundo lugar, seguirán estandarizándose especificaciones. El ciclo de vida de las especificaciones de servicios Web típicamente progresa desde una propuesta hasta un estándar específico. Con SOAP 1.2 y WSDL 1.2 las peculiaridades finales están siendo elaboradas de las especificaciones SOAP y WSDL y algunas de las especificaciones de servicios de mayor nivel,

como WS-Security, ya están en el proceso de estandarización. Las empresas siguen proponiendo nuevas especificaciones como añadidas a las plataformas de servicios Web y la industria en su conjunto necesitará acordar cuáles adoptar. Esas especificaciones necesitarán a continuación ser estandarizadas.

En tercer lugar, los kits de herramientas y marcos de trabajo seguirán mejorando. Además de servicios de más alto nivel como la seguridad y los objetos adjuntos, se añadirá soporte para protocolos de transporte alternativos como SMTP[17] o TCP[18]. De modo más importante, los modelos de programación migrarán desde los servicios de tipo RPC[19] hacia servicios centrados en documentos, para promover un acoplamiento débil. Todos estos cambios ocurrirán en paralelo, mientras los desarrolladores continúan desarrollando e implantando sistemas basados en servicios Web.

1.4.6 Conceptos e ideas generales acerca de los Servicios Web

"Los web services son componentes software que permiten a los usuarios usar aplicaciones de negocio que comparten datos con otros programas modulares, vía Internet. Son aplicaciones independientes de la plataforma que pueden ser fácilmente publicadas, localizadas e invocadas mediante protocolos web estándar, como XML, SOAP, UDDI o WSDL. El objetivo final es la creación de un directorio de online de web services, que pueda ser localizado de un modo sencillo y que tenga una alta fiabilidad." (1)

"Los servicios web son la revolución informática de la nueva generación de aplicaciones que trabajan colaborativamente en las cuales el software está distribuido en diferentes servidores." (2)

"Los servicios XML Web son los bloques de construcción de la computación distribuida en el Internet. Usted puede crear soluciones al usar los múltiples servicios de XML Web desde varias fuentes que trabajan en conjunto-independientemente de dónde residan o cómo fueron implementadas." (2)

---

[17] Simple Mail Transfer Protocol (SMTP) Protocolo Simple de Transferencia de Correo, es un protocolo de red basado en texto utilizado para el intercambio de mensajes de correo electrónico entre computadoras u otros dispositivos (PDA's, teléfonos móviles, etc.).

[18] TCP (Transmission-Control-Protocol, en español Protocolo de Control de Transmisión) es uno de los protocolos fundamentales en Internet. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron.

[19] El RPC (del inglés Remote Procedure Call, Llamada a Procedimiento Remoto) es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos

1.5 Principales Conceptos

1.5.1 XML (eXtensible Markup Language)

El lenguaje de marcas ampliable XML (siglas en inglés de Extensible Markup Language), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML[20] y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML[21], SVG[22], MathML[23].

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

1.5.2 WSDL (Web Services Description Language)

El lenguaje de descripción de los servicios web WSDL (son las siglas de Web Services Description Language), un formato XML que se utiliza para describir servicios Web (algunas personas lo leen como wisdel). La versión 1.0 fue la primera recomendación por parte del W3C y la versión 1.1 no alcanzó nunca tal estatus. La versión 2.0 se convirtió en la recomendación actual por parte de dicha entidad.

---

[20] SGML son las siglas de Standard Generalized Markup Language o "Lenguaje de Marcado Generalizado". Consiste en un sistema para la organización y etiquetado de documentos. La Organización Internacional de Estándares (ISO) normalizó este lenguaje en 1986.

[21] XHTML, acrónimo en inglés de eXtensible Hypertext Markup Language (lenguaje extensible de marcado de hipertexto), es el lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas web.

[22] Scalable Vector Graphics (SVG) es un lenguaje para describir gráficos vectoriales bidimensionales, tanto estáticos como animados (estos últimos con ayuda de SMIL), en XML.

[23] El MathML o Mathematical Markup Language es un lenguaje de marcado basado en XML, cuyo objetivo es expresar notación matemática de forma que distintas máquinas puedan entenderla, para su uso en combinación con XHTML en páginas web, y para intercambio de información entre programas de tipo matemático en general.

WSDL describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

Así, WSDL se usa a menudo en combinación con SOAP y XML Schema. Un programa cliente que se conecta a un servicio web puede leer el WSDL para determinar que funciones están disponibles en el servidor. Los tipos de datos especiales se incluyen en el archivo WSDL en forma de XML Schema. El cliente puede usar SOAP para hacer la llamada a una de las funciones listadas en el WSDL.

1.5.3 SOAP (Simple Object Access Protocol)

Son las siglas de Simple Object Access Protocol. Este protocolo deriva de un protocolo creado por David Winer[24], XML-RPC en 1998. Con este protocolo se pedían realizar RPC o remote procedure calls, es decir, se puede bien en cliente o servidor realizar peticiones mediante http a un servidor web. Los mensajes debían tener un formato determinado empleando XML para encapsular los parámetros de la petición. Con el paso del tiempo el proyecto iniciado por David Winer interesó a importantes multinacionales entre las que se encuentran IBM y Microsoft y de este interés por XML-RPC se desarrollo SOAP.

En el núcleo de los servicios Web se encuentra el protocolo simple de acceso a datos SOAP, que proporciona un mecanismo estándar de empaquetar mensajes. SOAP ha recibido gran atención debido a que facilita una comunicación del estilo RPC entre un cliente y un servidor remoto. Pero existen multitud de protocolos creados para facilitar la comunicación entre aplicaciones, incluyendo RPC de Sun, DCE de Microsoft, RMI de Java y ORPC de CORBA. ¿Por qué se presta tanta atención a SOAP?

Una de las razones principales es que SOAP ha recibido un increíble apoyo por parte de la industria. SOAP es el primer protocolo de su tipo que ha sido aceptado prácticamente por todas las grandes compañías de software del mundo. Compañías que en raras ocasiones cooperan entre sí están ofreciendo su apoyo a este protocolo. Algunas de las mayores Compañías que soportan SOAP son Microsoft, IBM, SUN Microsystems, SAP y Ariba.

[24] **Dave Winer** es un desarrollador y empresario de software de Berkeley, California. Pionero en las áreas de RSS (RSS es una familia de formatos de fuentes web codificados en XML) y XML.

Algunas de las Ventajas de SOAP son:

- **No está asociado con ningún lenguaje:** los desarrolladores involucrados en nuevos proyectos pueden elegir desarrollar con el último y mejor lenguaje de programación que exista pero los desarrolladores responsables de mantener antiguas aflicciones heredadas podrían no poder hacer esta elección sobre el lenguaje de programación que utilizan. SOAP no especifica una API[25], por lo que la implementación de la API se deja al lenguaje de programación, como en Java, y la plataforma como Microsoft .Net.
- **No se encuentra fuertemente asociado a ningún protocolo de transporte:** La especificación de SOAP no describe como se deberían asociar los mensajes de SOAP con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.
- **No está atado a ninguna infraestructura de objeto distribuido:** La mayoría de los sistemas de objetos distribuidos se pueden extender, y ya lo están alguno de ellos para que admitan SOAP.
- **Aprovecha los estándares existentes en la industria:** Los principales contribuyentes a la especificación SOAP evitaron, intencionadamente, reinventar las cosas. Optaron por extender los estándares existentes para que coincidieran con sus necesidades. Y como ya se ha mencionado SOAP no define un medio de trasporte de los mensajes; los mensajes de SOAP se pueden asociar a los protocolos de transporte existentes como HTTP y SMTP.
- **Permite la interoperabilidad entre múltiples entornos:** SOAP se desarrolló sobre los estándares existentes de la industria, por lo que las aplicaciones que se ejecuten en plataformas con dicho estándares pueden comunicarse mediante mensaje SOAP con aplicaciones que se ejecuten en otras plataformas.

1.5.4 UDDI (Universal Description Discovery and Integration)

UDDI son las siglas del catálogo de negocios de Internet denominado Universal Description, Discovery and Integration. El registro en el catálogo se hace en XML. UDDI es una iniciativa industrial abierta (sufragada por la OASIS) entroncada en el contexto de los servicios Web. El registro de un negocio en UDDI tiene tres partes:

[25] API: Una interfaz de programación de aplicaciones o API (del inglés Application Programming Interface) es el conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Páginas blancas - dirección, contacto y otros identificadores conocidos.

Páginas amarillas - categorización industrial basada en taxonomías.

Páginas verdes - información técnica sobre los servicios que aportan las propias empresas.

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.

1.5.5 Arquitectura en capas

Garlan[26] y Shaw[27] en su famoso articulo *"An Introduction to Software Architecture"* (1) en el anuario de la facultad de Ciencias de la Computación de la Universidad de Carnegie Mellon, 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.

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

Una especialización muy usada de la arquitectura en capas es la arquitectura de tres capas donde se observan muy bien delimitadas las responsabilidades de cada funcionalidad en la aplicación. Una capa superior interactúa con una capa inferior mediante interfaces que definen las

[26] David Garlan es profesor en la escuela de Ciencias de la Computación en la universidad de Carnegie Mellon, donde dirige diversos proyectos y es director de Professional Software Engineering Programs.
[27] Mary Shaw es miembro de la facultad de ingeniería de software de la escuela de Ciencias de la Computación de la Universidad de Carnegie Mellon.

funcionalidades que la misma debe brindar. Las capas de la aplicación pueden residir tanto en el mismo nodo físico como en nodos separados (Figura 1).

1.5.6 Framework

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

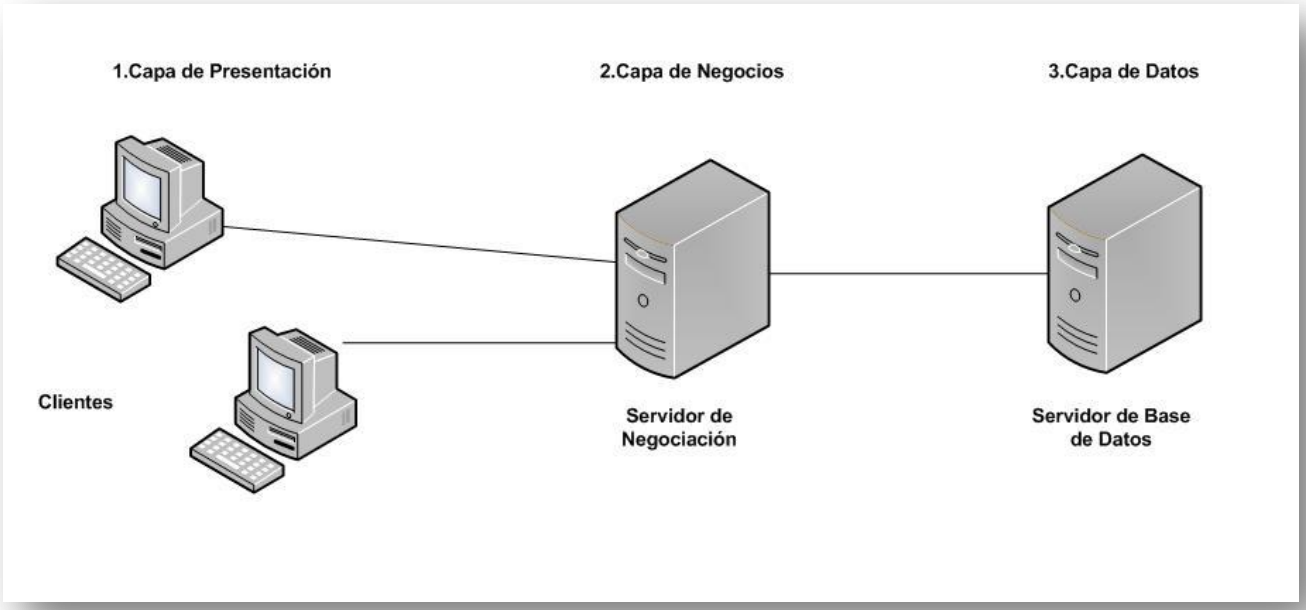

Figura 1: Arquitectura en Capas

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

No hay una definición común para los frameworks, pero la mayoría tiene un tema común: la reutilización. Una definición ampliamente aceptada es dada por R. E. Johnson, y B. Foote en 1988 en su publicación *"Designing Reusable Classes"* (4)*:*

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

Otro punto de vista, según R. E. Johnson y V. F. Russo en *"Reusing Object-Oriented Designs"* (3) incluye:

"Una clase abstracta es un diseño para un objeto simple. Un framework es el diseño de un conjunto de objetos que colaboran para llevar a cabo un conjunto de responsabilidades. De esta manera los frameworks son diseños a escalas más grandes que las clases abstractas. Los frameworks son una forma de reutilizar el diseño a un nivel superior." (5)

1.6 Tecnologías Utilizadas

1.6.1 Plataforma Java

La plataforma Java es el nombre de un entorno o plataforma de computación originaria de la compañía Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java u otros lenguajes que compilen a bytecode y un conjunto de herramientas de desarrollo. En este caso, la plataforma no es un hardware específico o un sistema operativo, sino más bien una máquina virtual encargada de la ejecución de aplicaciones, y un conjunto de librerías estándar que ofrecen funcionalidad común.

La plataforma Java incluye:

Plataforma Java, Edición Estándar (Java Platform, Standard Edition), o Java SE (antes J2SE)

Plataforma Java, Edición Empresarial (Java Platform, Enterprise Edition), o Java EE (antes J2EE)

Plataforma Java, Edición Micro (Java Platform, Micro Edition), o Java ME (antes J2ME)

La plataforma Java se ejecuta sobre otra plataforma hardware/software. Esta posee dos componentes fundamentales:

1. La Máquina Virtual de Java (JVM): La JVM es la encargada de ejecutar los programas escritos en el lenguaje de programación Java y es quien permite que las aplicaciones desarrolladas en este lenguaje puedan ser ejecutadas en diversos sistemas, con arquitecturas diferentes.

2. El API Java es un conjunto de clases ya desarrolladas que ofrecen amplias posibilidades al programador.

Hoy en día esta plataforma ha evolucionado en concordancia con el avance tecnológico y se ha convertido en una de las plataformas de programación más usadas por los desarrolladores. Su principal ventaja es que su entorno de desarrollo es independiente de la plataforma sobre la que se trabaje, es decir, sus aplicaciones son funcionales independientemente del sistema operativo sobre el que estén operando.

1.6.1.1 Lenguaje JAVA

Java es un lenguaje de programación surgido en 1991. Los creadores de Java se basaron en C++, pero eliminaron la mayoría de sus complejidades. Es toda una tecnología orientada al desarrollo de software con el cual podemos realizar cualquier tipo de programa. Hoy en día, la tecnología Java ha cobrado mucha importancia en el ámbito de Internet gracias a su plataforma J2EE. Pero Java no se queda ahí, ya que en la industria para dispositivos móviles también hay una gran acogida para este lenguaje.

La tecnología Java está compuesta básicamente por 2 elementos: el lenguaje Java y su plataforma. Con plataforma nos referimos a la máquina virtual de Java (Java Virtual Machine).

Una de las principales características que favoreció el crecimiento y difusión del lenguaje Java es su capacidad de que el código funcione sobre cualquier plataforma de software y hardware. Esto significa que el mismo programa escrito para Linux puede ser ejecutado en Windows sin ningún problema. Además es un lenguaje orientado a objetos que resuelve los problemas en la complejidad de los sistemas.

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

Orientado a Objetos: Basado en C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Soporta las tres características propias de la orientación a objetos: encapsulación, herencia y polimorfismo.

Distribuido: Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp. La característica de ser distribuido es debido a que proporciona las librerías y herramientas para que los programas puedan distribuirse, es decir, que se corran en varias máquinas, interactuando.

Interpretado: Se traduce el código fuente a un código intermedio (bytecode), que es interpretado por La Maquina Virtual de Java, lo cual permite que se pueda ejecutar en cualquier sistema operativo.

Robusto: Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo.

Seguro: No se permite el acceso ilegal a memoria ya que no se trabaja con punteros.

Portabilidad: Una de las principales características de Java es su portabilidad. En lugar de compilarse a código nativo de la máquina, los programas de Java se traducen al formato bytecode que es el mismo en cualquier sistema operativo. Estos bytecodes son procesados directamente por la Máquina Virtual de Java que es dependiente de la máquina en uso. La portabilidad radica en que se pueden programar las aplicaciones solo una vez y ejecutar en cualquier plataforma. Por ejemplo, cuando se compila un programa Java en una plataforma Windows/Intel, se obtiene la misma salida compilada (o los mismos bytecodes) que en un sistema Macinstosh o Unix

Altas prestaciones: No se pierde tiempo optimizando código que no se ejecutará.

Multihilo: Permite la ejecución de varias tareas a la vez.

Dinámico: No conecta todos los módulos que comprende una aplicación hasta el tiempo de ejecución.

1.6.2 Frameworks Utilizados

1.6.2.1 Spring

Spring Framework es un framework de aplicación de código abierto que ayuda a hacer el desarrollo en JEE[28] mucho más fácil. Ayuda a estructurar aplicaciones completas en una manera consistente y productiva para crear arquitecturas coherentes (6).

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

Presenta varios módulos de los cuales los principales son:

- **Contenedor de Inversión de Control:** permite una sofisticada administración de configuración para POJOs[29] y trabaja con otras partes de Spring para proveer servicios como la administración de la configuración.
- **Una abstracción de acceso a datos:** Spring anima a un consistente acercamiento arquitectónico para acceso a datos, y posee una abstracción única y poderosa para implementarlo. Presenta una rica jerarquía de excepciones de acceso a datos, independiente de cualquier framework de persistencia en particular.
- **Simplificación de JDBC[30]:** presenta una capa de abstracción sobre JDBC que es significativamente mucho más simple y menos propensa a errores para usar que JDBC puro cuando se necesita usar acceso a través de SQL a bases de datos relacionales.
- **Administración de transacciones:** presenta una abstracción de transacciones que puede mover transacciones globales JTA[31] (manipuladas por un servidor de aplicaciones) o transacciones locales usando alguna API de acceso a datos. Estas abstracciones

[28] JEE: Java Enterprise Edition o Java EE (anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE hasta la versión 1.4), es una plataforma de programación—parte de la Plataforma Java—para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de N niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

[29] POJO: Es el acrónimo de Plain Old Java Object. Clase del lenguaje de programación Java. Este nombre se les da a las clases que no son de algún tipo especial (EJBs, Java Beans, etcétera) y no cumplen ningún otro rol ni implementan alguna interfaz especial.

[30] JDBC: Es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

[31] JTA (del inglés Java Transaction API - API para transacciones en Java) es parte de Java EE APIs, JTA establece una serie de Interfaces java entre el manejador de transacciones y las partes involucradas en el sistema de transacciones distribuidas: el servidor de aplicaciones, el manejador de recursos y las aplicaciones transaccionales, JTA fue desarrollado por SUN.

presentan un consistente modelo de programación en una variedad de ambientes y es la base de la administración de transacciones programáticas y declarativas usadas por Spring.

- **Framework Web MVC[32]:** Spring presenta un framework web MVC basado en peticiones. Éste tiene un acercamiento al framework de Struts pero el framework web de Spring es más flexible, y se integra discretamente al contenedor de Inversión de control de Spring. Todos los demás rasgos de Spring se pueden usar con otros frameworks tales como Struts, JSF, WebWork, Tapestry, etcétera.

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

- **Ayuda a promover la reusabilidad de código:** Ayuda a evitar la necesidad de tomar decisiones importantes y duras, como es si una aplicación alguna vez usará JTA o JNDI[33]; las abstracciones de Spring permitirán desarrollar el código en un diferente ambiente si tú alguna vez lo necesitas.
- **Facilita el diseño Orientado a Objetos (OO) en aplicaciones JEE:** Se debería preguntar uno: ¿Cómo una aplicación JEE, escrita en Java – un lenguaje OO – no es OO? En realidad muchas aplicaciones JEE no merecen el nombre OO. Con Spring es más fácil eliminar los impedimentos del diseño OO en aplicaciones JEE tradicionales haciendo el código más coherente, con más bajo acoplamiento y más reusable.
- **Facilita buenas prácticas de programación,** tales como la programación a interfaces: El uso de un contenedor de Inversión de Control reduce grandemente la complejidad del código a interfaces, más que a clases. El uso de los objetos a través de estas interfaces protege los requerimientos, los cuales pudieran cambiar en el desarrollo de la aplicación.
- **Permite la extracción de valores de configuración desde el código java a archivos XML o archivos de propiedades:** Mientras que algunos valores de configuración pudieran ser programados en Java, todas las aplicaciones de mediana y alta complejidad necesitan algunas configuraciones externalizadas del código fuente, para permitir la administración sin recompilar las clases nuevamente.

[32] Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

[33] La Interfaz de Nombrado y Directorio Java (JNDI) es una Interfaz de Programación de Aplicaciones (API) para servicios de directorio. Esto permite a los clientes descubrir y buscar objetos y nombres a través de un nombre y, como todas las APIs de Java que hacen de interfaz con sistemas host, es independiente de la implementación subyacente.

- **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.

1.6.2.2 Spring Web Services

Spring Web Services (Spring-WS) es un producto de la comunidad Spring enfocado a la creación de aplicaciones basadas en el uso de servicios web. Spring Web Services tiene como objetivo facilitar el desarrollo de servicios SOAP contract-first[34], permitiendo la creación de servicios web flexible usando una de las tantas maneras de manipular contenido XML. Este producto es basado en el propio Spring, lo que significa que se pueden utilizar conceptos de Spring como la inyección de dependencias.

1.6.2.3 Spring Security

Spring Security es un framework JAVA/JAVA EE que provee autenticación avanzada, autorización y otros elementos de seguridad para aplicaciones empresariales usando Spring. El proyecto fue iniciado a finales del 2003 bajo el nombre de 'Acegi Security' por Ben Alex[35], el cual fue publicado bajo la licencia de Apache en Marzo del 2004. Subsecuentemente, Acegi fue incorporado en el entorno de Spring como Spring Security, como un subproyecto oficial de Spring. El primer lanzamiento público bajo el nuevo nombre de Spring Security 2.0.0 fue en abril del 2008, con soporte comercial y adiestramiento disponible desde Spring Source.

1.7 Patrones de Diseño Utilizados

[34] Cuando se crea un Servicio Web, hay dos estilos de desarrollo: *Contract Last* and *Contract First*. Cuando se utiliza el enfoque contract-last, se empieza con el código, y luego con el contrato del Servicio (WSDL) siendo generado desde el mismo código. Cuando se usa contract-first, se empieza por el contrato WSDL, y se usa el lenguaje de programación para implementar dicho contrato.

[35] El Dr. Ben Alex es el Ingeniero de Software principal de Spring, ha trabajado profesionalmente en el mundo del software desde 1995. Fue fundador del proyecto Spring Security en el 2003 y aun se mantiene al frente del mismo.

El Framework Spring ha surtido de vida al desarrollo en Java. Luego del lanzamiento del libro de Rod Johnson *Expert One-on-One J2EE Design and Development*, en el 2002 (6) y su eventual evolución en el framework Spring, el entorno Java ha tenido un faro de esperanza El núcleo del framework Spring es un contenedor ligero basado en la Inversión del Control que es capaz de añadir nuevas funcionalidades a objetos Java declarativamente. Spring hace un uso extensivo de una excelente metodología de programación -AOP (Aspect-Oriented Programming) para proveer de estos servicios a sus componentes. Dentro del ámbito del contenedor de Spring, los componentes son llamados Beans. El framework Spring en si mismo incorpora varios patrones de diseño, incluidos los patrones orientados a objetos de GoF[36] (Gang of Four) y los Java EE propios del núcleo de la Sun. Mediante el uso de Spring, se aplican las mejores prácticas de la industria para implementar y diseñar aplicaciones.

1.7.1 Inyección de Dependencias/ Inversión del Control

Es común escuchar hablar de los términos Inversión del Control e Inyección de Dependencias indistintamente, pero de hecho no son la misma cosa. Inversión del Control es un término mucho más general, y puede ser expresado de muchas maneras. Inyección de Dependencias es simplemente un ejemplo concreto de Inversión del Control.

Originalmente a Inyección de Dependencias se le llamó: Inversión del control. Pero en un artículo escrito a principios del 2004, Martin Fowler[37] preguntó qué aspecto del control estaba siendo invertido, y concluyó que era la adquisición de dependencias la que estaba siendo invertida. Basado en esto sugirió la frase *"inyección de dependencia"* como un término que describía mejor lo que se estaba ocurriendo realmente. Cualquier aplicación está compuesta por más de una clase, que colaboran entre ellas para llevar a cabo una lógica determinada. Tradicionalmente un objeto es responsable de obtener las referencias a los objetos con los que colabora (sus dependencias). Esto puede llevar a un código altamente acoplado y difícil de probar.

Cuando se aplica la *inyección de dependencias*, a los objetos le son dadas sus dependencias en el momento en que son creados por alguna entidad externa que coordina cada objeto en el sistema. En otras palabras, las dependencias son inyectadas a los objetos. Entonces, la *inyección de*

[36] Gang of Four es el nombre con el que se conoce comúnmente a los autores del libro "Design Patterns: Elements of reusable OO software" (17), referencia en el campo del diseño orientado a objetos. Del cual sus autores son Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides.

[37] Martin Fowler es autor de numerosos libros y auditor internacional en el desarrollo de software, especialista en análisis y diseño orientado a objetos, UML, patrones y metodologías de desarrollo ágiles como extreme programing.

dependencias implica una inversión de la responsabilidad con respecto a cómo los objetos obtienen las referencias a los objetos con los que colaboran.
El beneficio más importante de la *inyección de dependencias* es el desacople. Si un objeto solo conoce sus dependencias por sus interfaces, no por su implementación ni cómo fueron instanciadas, entonces las dependencias pueden ser cambiadas con implementaciones diferentes sin que el objeto dependiente conozca la diferencia.

1.7.2 Patrón Instancia Única (Singleton)

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.

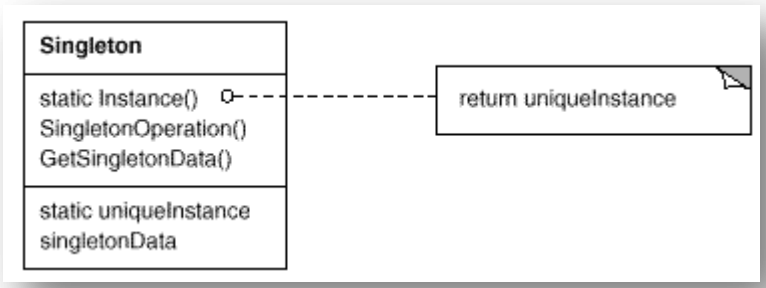

Figura 2: Patrón Singleton

1.7.3 Patrón Validador de Mensaje (Message Validator)

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.

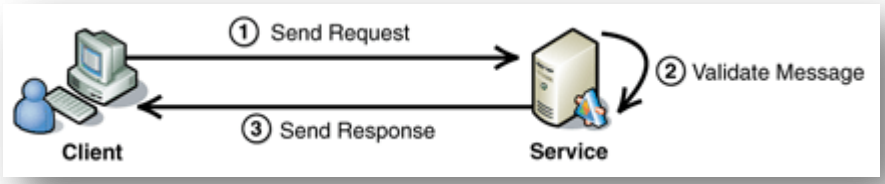

Figura 3: La validación del mensaje se produce en el servicio web

1.8 Programación Orientada a Aspectos

La Programación Orientada a Aspectos (AOP) es una nueva metodología para complementar la tradicional Orientada a Objetos (OOP). La principal motivación de AOP no es sustituir a la OOP. De hecho, AOP se usa en conjunto con OOP. En el mundo de la Programación Orientada a Objetos, las aplicaciones son organizadas en clases e interfaces. Estos elementos son buenos para implementar los requerimientos de la lógica de negocios, pero no las incumbencias transversales (por ejemplo: las funciones o requerimientos que se extienden a través de múltiples módulos de una aplicación). Las incumbencias transversales son muy comunes en las aplicaciones empresariales. Los ejemplos típicos lo constituyen la auditoria, la validación y la administración de transacciones.

Aunque la inyección de dependencias hace posible enlazar todos los componentes de una aplicación con un bajo acople, la programación orientada a aspectos (AOP) permite agrupar

funcionalidades que son usadas en toda la aplicación en componentes, los cuales pueden ser reutilizados.

Los sistemas de software están compuestos por muchos componentes, cada uno con ciertas funcionalidades. Sin embargo, a menudo estos componentes también poseen responsabilidades adicionales más allá de sus verdaderas funcionalidades. Servicios del sistema como auditorías, administración de transacciones, y seguridad a menudo se encuentran en componentes cuya verdadera responsabilidad es otra. Estos servicios a menudo involucran a múltiples componentes del sistema, lo que provoca un aumento en el nivel de complejidad del código.

El código que implementa este tipo de servicios se encuentra duplicado a través de múltiples componentes. Esto conlleva a que si se necesita cambiar la lógica con que se realizan estas tareas, se necesita cambiarla en todos estos componentes. Y aunque se haya abstraído esta lógica a una clase o un conjunto de clases para que el impacto en los componentes sea menor, como una llamada a un método, esa llamada se encontrará duplicada en múltiples lugares.

La programación orientada a aspectos hace posible modular estos servicios y luego aplicarlos declarativamente a los componentes que deben afectar. Esto nos permite tener componentes más cohesivos y que se centren en sus propias funcionalidades, ignorando por completo cualquier servicio del sistema en el que pueda estar involucrado.

1.9 Herramientas a Utilizar

1.9.1 Eclipse IDE

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

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Estos proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Java, C#, Visual Basic, Object Pascal, etcétera.

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma. Esta plataforma, típicamente, ha sido usada para desarrollar entornos de desarrollo integrados.

En cuanto a las aplicaciones clientes, eclipse provee al programador con frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc. Por ejemplo, GEF (Graphic Editing Framework - Framework para la edición gráfica) es un plugin[38] de Eclipse para el desarrollo de editores visuales que pueden ir desde procesadores de texto hasta editores de diagramas UML, (GUI). Dado que los editores realizados con GEF "viven" dentro de Eclipse, además de poder ser usados conjuntamente con otros plugins, hacen uso de su interfaz gráfica personalizable y profesional.

Características

La versión actual de Eclipse dispone de las siguientes características:

- ✓ Editor de texto
- ✓ Resaltado de sintaxis
- ✓ Compilación en tiempo real
- ✓ Pruebas unitarias con JUnit
- ✓ Control de versiones con CVS (del inglés: Concurrent Versions System)
- ✓ Asistentes (wizards): para creación de proyectos, clases, tests.
- ✓ Refactorización

Asimismo, a través de "plugins" libremente disponibles es posible añadir control de versiones con Subversion.

1.9.2 Apache Tomcat

Apache Tomcat (también llamado Jakarta Tomcat o Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Sun Microsystems.

Tomcat es un servidor web con soporte de servlets y JSPs. Tomcat no es un servidor de aplicaciones, como JBoss o JOnAS. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache.

[38] Un complemento (o plug-in en inglés) es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.

Tomcat puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos 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 Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

1.9.3 Subversion

Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias. Es de software libre desarrollado bajo una licencia de tipo Apache/BSD y se le conoce también como “svn” por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.

CAPITULO 2

Descripción y Características del Sistema

2.1 Introducción

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

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

Se exponen conceptos que serán manejados tanto en la capa de servicios como en el manejo de la seguridad, además de los elementos organizacionales, todo con el fin de lograr un mejor entendimiento de la estructura de la Plataforma, así como las reglas que deben satisfacerse para cumplir los objetivos y funcionalidades del Teleidentificador Personal.

2.2 Descripción de la Plataforma Manejadora de Peticiones

El proyecto Teleidentificador Personal es un desarrollo en conjunto entre la Universidad de las Ciencias Informáticas y la Empresa de Telecomunicaciones de

Cuba, su futura implantación en el país no solo marcará un hito en las telecomunicaciones, sino que al prestar este servicio de avanzada se le proveerán soluciones efectivas a las cuestiones de cómo aunar sus contactos que hoy se manifiestan en la población.

TIP tiene para su desarrollo varios módulos entre ellos la “Plataforma Manejadora de Peticiones” (PMP) (Figura 4), la cual permite la integración de diversas aplicaciones donde todas las peticiones realizadas por los usuarios son tramitadas por la PMP, garantizando así centralizar toda la información y el control del acceso a los DNS, estableciendo un modo estandarizado del acceso a los datos.

Dentro de las funcionalidades que brinda se destacan:

- ✓ *Gestionar las peticiones de los clientes:* Permite darle respuesta a las consultas hechas por los usuarios del servicio con la calidad y la rapidez requerida. El cliente tendrá la posibilidad de mostrar sus contactos, modificar la preferencia de los mismos y realizar búsquedas personalizadas.
- ✓ *Estandarizar el acceso a la información:* Todo el accedo a los DNS y la BD ENUM se realizará a través de la plataforma, separando así, esta responsabilidad de las aplicaciones clientes.
- ✓ *Integrar aplicaciones:* La plataforma cuenta con una capa de servicios que permite integrar y comunicar todas las aplicaciones que se desarrollen por separado y que requiera interactuar con la información de los subscriptores almacenada en el BIND [39]DNS y la base de datos ENUM.

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

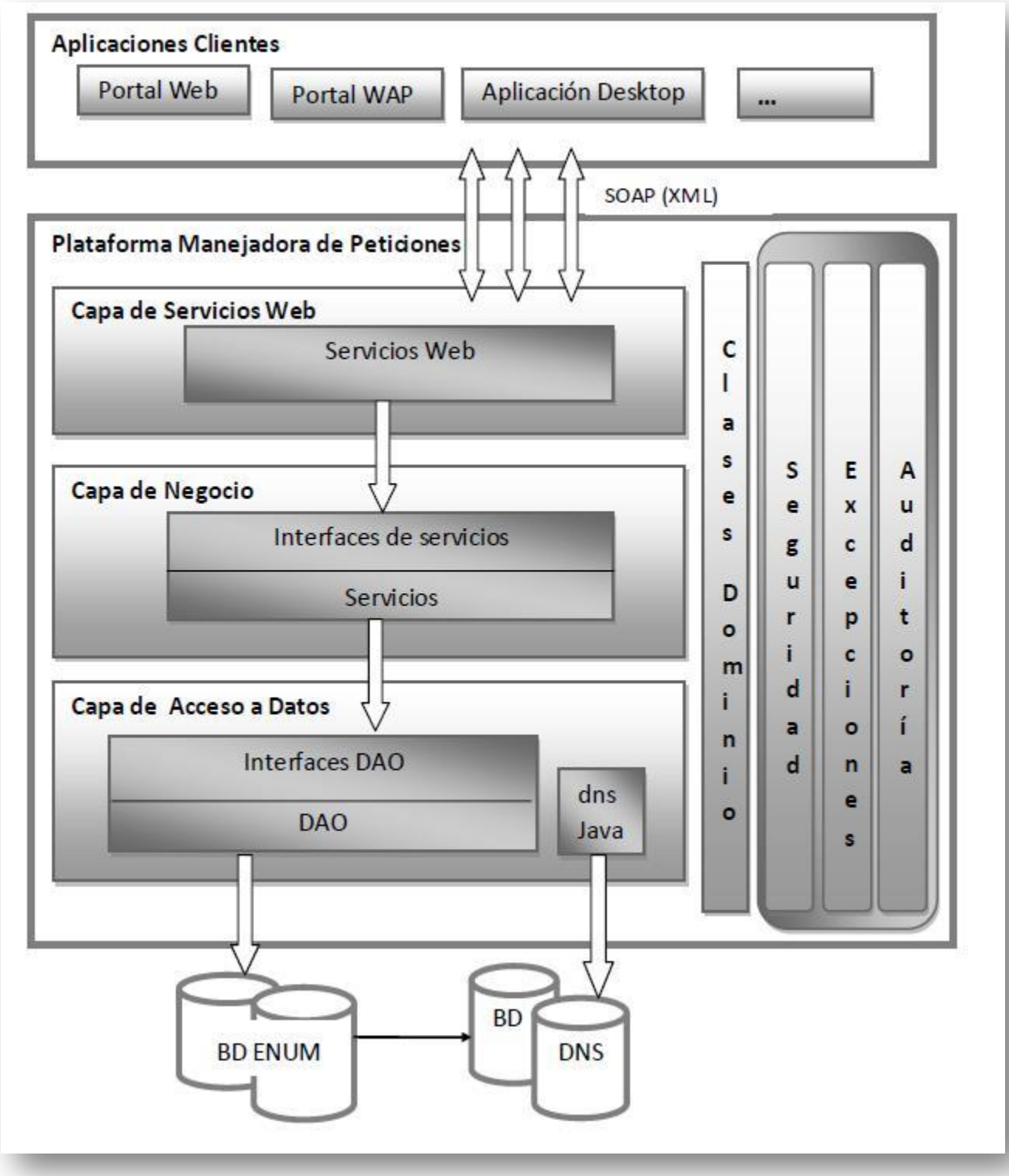

Figura 4: Estructura de la Plataforma Manejadora de Peticiones.

.

2.3 Principales Casos de Usos de la Plataforma Manejadora de Peticiones

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

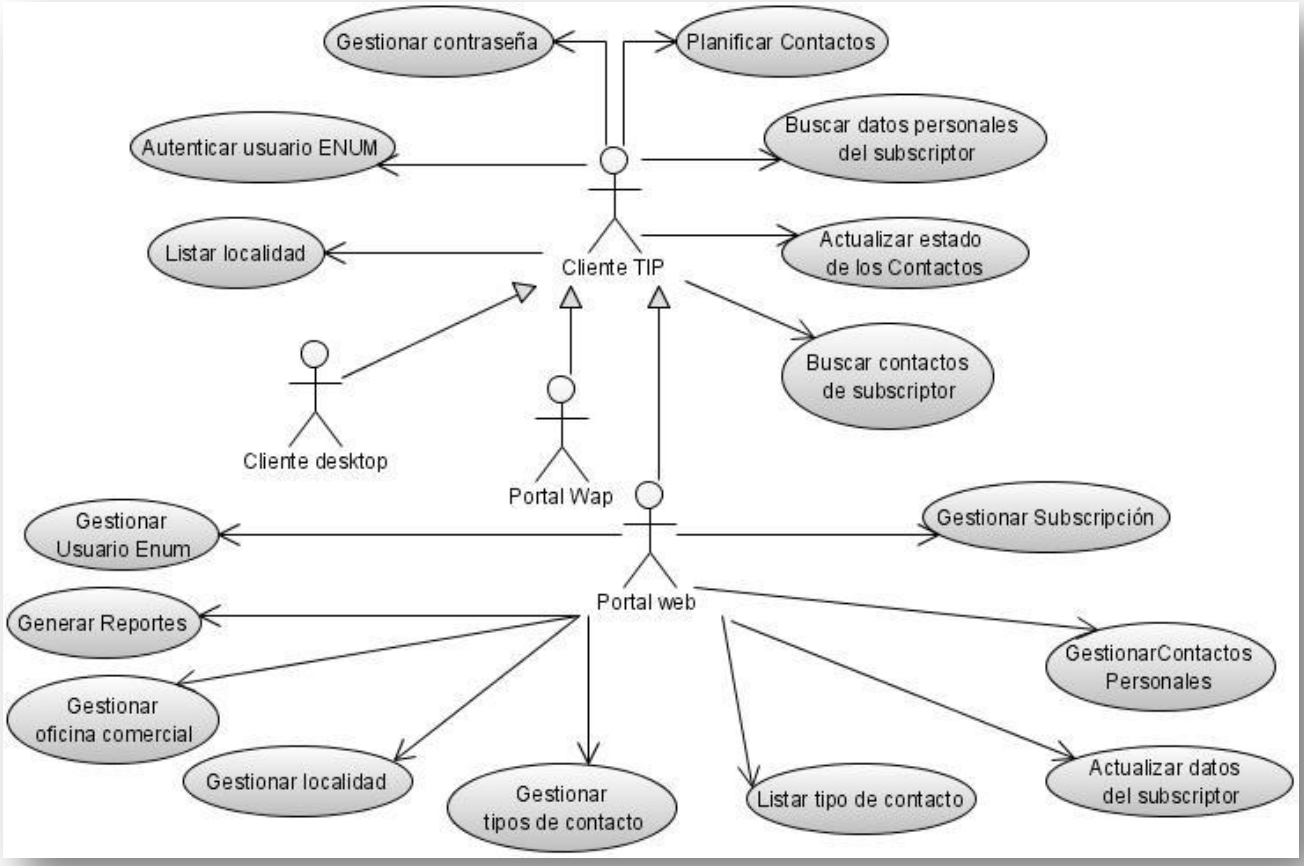

Figura 5: Principales Casos de Uso

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 su estado. Para adicionar una subscripción se debe enviar a la plataforma el usuario del registrador y los datos del subscriptor requeridos. Para modificar el estado de una subscripción se

enviará el identificador de la subscripción, el usuario del registrador y el nuevo estado; y en caso que se desee eliminarla, el identificador de la subscripción y el usuario del registrador. En cualquiera de los tres casos, se enviará a la plataforma la descripción de la acción realizada (opcional).

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 subscriptores 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 operaciones efectuadas sobre determinada subscripción, así como el registrador asociado a cada operación.

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. En cualquiera de los otros dos casos 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 de contactos. Para adicionar un nuevo contacto se debe enviar a la plataforma el número TIP del subscriptor y el contacto, para actualizarlo es necesario el contacto, y para eliminarlo el identificador del mismo.

Caso de uso: Listar tipos de contactos

El caso de uso permite a las aplicaciones clientes listar los tipos de contactos que existen.

Caso de uso: Gestionar oficina comercial

El caso de uso le permite a las aplicaciones clientes adicionar, modificar o eliminar oficinas comerciales. Para adicionar una nueva oficina se debe enviar a la plataforma el identificador del municipio al cual pertenece y la oficina; para actualizarla es necesaria la oficina, y para eliminarla el identificador de la misma.

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.

2.4 Modelo de Dominio

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

2.4.1 Conceptos del Modelo del Dominio

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

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

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

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

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

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

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

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

Los conceptos que anteriormente se precisaron son los que se relacionan entre sí conformando la naturaleza del problema a analizar. En la figura 6 se muestra el Modelo de Dominio que se definió:

2.5 Requisitos Funcionales

El propósito fundamental de la descripción de los requisitos funcionales es guiar el desarrollo hacia el sistema correcto. Los requerimientos funcionales definen las funciones que el sistema debe ser capaz de realizar, son el punto de partida para identificar qué debe hacer el sistema.

Deben comprenderlo tanto los desarrolladores como los usuarios.
Los requerimientos funcionales que debe cumplir el sistema a desarrollar se listan a continuación.

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

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

RF2.1: Cambiar contraseña.

RF2.2: Solicitar una nueva contraseña.

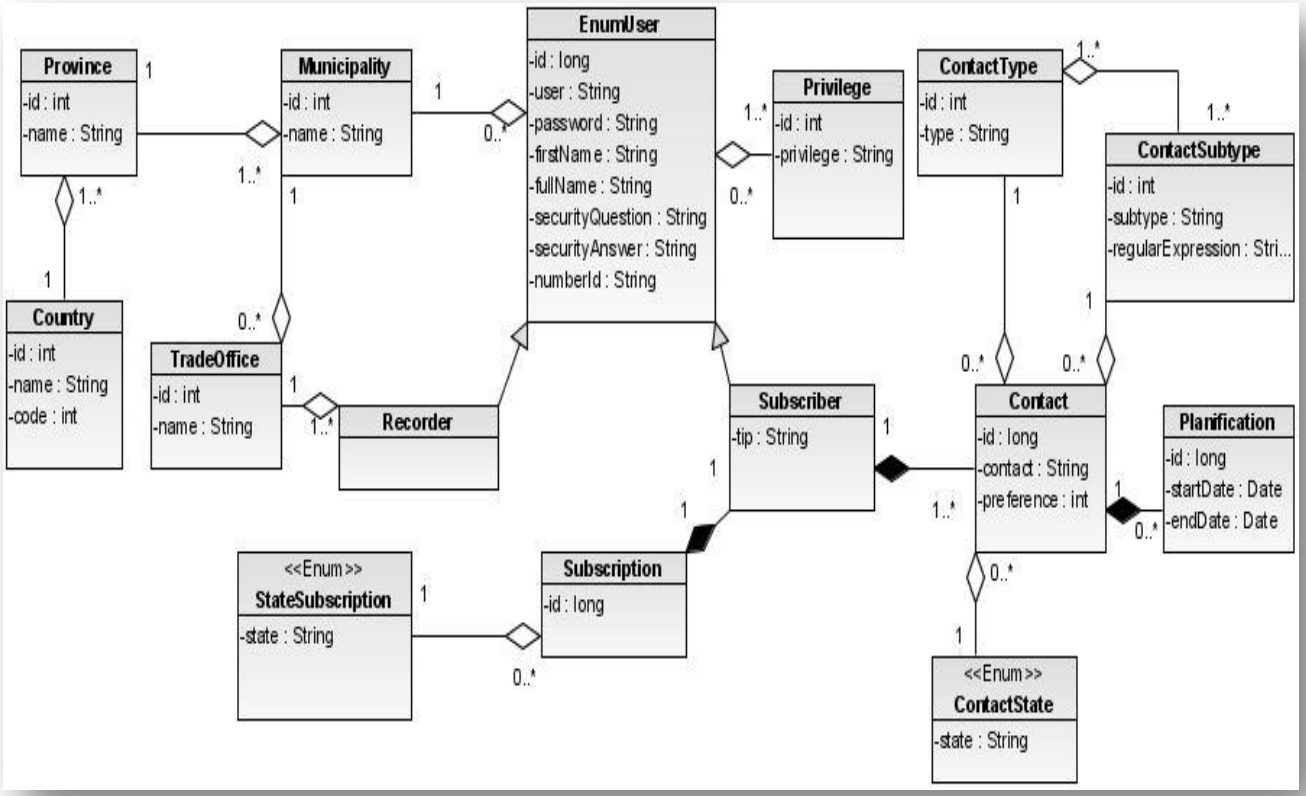

Figura 6: Diagrama de clases del Dominio

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

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

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

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

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

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

RF8.1 Registrar subscripción.

RF8.2 Actualizar subscripción.

RF8.3 Eliminar subscripción.

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

RF9.1 Insertar nuevo contacto.

RF9.2 Modificar contacto.

RF9.3 Eliminar contacto.

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

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

RF11.1 Insertar nuevo tipo de contacto.

RF11.2 Modificar tipo de contacto.

RF11.3 Eliminar tipo de contacto.

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

RF12.1 Insertar nueva oficina comercial.

RF12.2 Modificar oficina comercial.

RF12.3 Eliminar oficina comercial.

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

2.6 Requisitos no Funcionales

2.6.1 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.

2.6.2 Seguridad

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

2.6.3 Fiabilidad

a) El sistema tiene que estar disponible un 99.99 % del tiempo, ya que gestiona telecomunicaciones.
b) Toda la información que gestiona el sistema es de carácter personal y está debidamente verificada por lo que debe de ser 100 % exacta y real.
c) Se identifican 2 tipos de errores significativos: errores menores (tiempo excedido, error en algún parámetro) errores críticos (perdida de conexión con la BD, DNS).

2.6.4 Portabilidad

El sistema deberá correr sobre cualquier Sistema Operativo.

2.6.5 Soporte

a) Se debe de generar un documento detallado que explique el funcionamiento del sistema.
b) Se debe cumplir con las pautas de codificación establecidas para el proyecto TeleIdentificador Personal.

2.6.6 Restricciones del diseño e implementación

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

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

1. Se debe documentar cada una de las funcionalidades del sistema.
2. Documentación del código, siguiendo el formato especificado en el estándar de codificación.
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.

2.6.8 Interfaces de Comunicación

a) La comunicación con el sistema se hará a través del protocolo SOAP.
b) La comunicación con la BD TCP/IP.
c) La comunicación con el DNS TCP/IP.

2.6.9 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 1 GB de RAM o superior.
- Procesador 3.0 GHz o superior.

2.6.10 Software

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

- Maquina 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.

2.6.11 Estándares Aplicables

a) El estándar de codificación se realizó basado en el estándar de SUM Microsystem.

2.7 Reglas de Negocio

Las reglas de negocio describen políticas que deben cumplirse o condiciones que deben satisfacerse para alcanzar los objetivos misionales, por lo que regulan algún aspecto del negocio. En el sistema se han definido las reglas del negocio que se especifican a continuación:

2.7.1 Reglas de estructura

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

2.7.2 Reglas de derivación

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

2.7.3 Reglas de acción

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

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

2.8 Pautas de Codificación

Ver Anexo 1.

2.9 Conclusiones

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

CAPITULO 3

Implementación de la Capa de Servicios

3.1 Introducción

Los autores del presente trabajo decidieron en conjunto, que dado lo novedoso del uso de Spring WS para el desarrollo de servicios Web y lo importante que fue su uso en la implementación de la capa de servicios, describir en el desarrollo de este capítulo, las principales características del mismo y la forma en que fue implementado.

A lo largo de este capítulo se hace una descripción de la implementación de la capa de servicios con Spring WS, las características de la misma de forma general así como los componentes que la componen para lograr una mayor claridad en la descripción de la solución propuesta al problema planteado en el diseño teórico de la investigación.

3.2 Creación de Servicios contract-first con Spring WS

Existen dos enfoques para desarrollar servicios web, en ambos los servicios web son publicados mediante un contrato, que es descrito mediante WSDL (Web Service Definition Language). Su clasificación está dada en si el contrato se define al principio o al final.

El primero de ellos *contract-last* es generalmente el más popular por una razón simple, es más fácil de desarrollar. La mayoría de los programadores no están dispuestos a lidiar con WSDL, SOAP y esquemas XML (XSD). Mediante esta vía no hay necesidad de manipular complejos ficheros WSDL y XSD. Simplemente se implementa una lógica determinada en código java por ejemplo, y se utiliza un framework para exponer dicha lógica como un servicio web.

Cuando un servicio web se desarrolla de esta forma su contrato termina siendo un reflejo de la lógica interna de la aplicación. Esto trae consigo que cualquier cambio que se necesite hacer en la aplicación implica cambios en el contrato del servicio y finalmente significan cambios en los clientes que están consumiendo el servicio web.

Esto lleva al clásico problema del versionado de servicios web. Es mucho más fácil cambiar el contrato del servicio que cambiar los clientes que consumen ese servicio. Si el servicio web tiene mil clientes y se cambia el contrato entonces estarán inhabilitados de utilizar el servicio hasta que realicen los cambios necesarios para consumir el nuevo servicio. Esto sin embargo pudiera dificultar el mantenimiento y hacerlo más costoso, ya que se tendrá que mantener varias versiones del mismo servicio.

Una mejor solución es evitar cambiar el contrato del servicio. Y cuando el contrato deba ser cambiado, los cambios no deberían afectar la compatibilidad con versiones anteriores. Pero esto puede ser muy difícil de lograr cuando el contrato es generado automáticamente.

El contrato es tratado como un efecto secundario, solo necesario para la comunicación entre el cliente y el servidor. Esto también implica que los desarrolladores no tienen control sobre la estructura del fichero WSDL y de los datos transmitidos en el mensaje SOAP. Por ejemplo, no es posible especificar qué parámetros deben ser atributos de los elementos del mensaje, y cuáles deberían ser valores. Debido a que la *carga útil* del XML no puede ser controlada, existe la posibilidad de sobrecarga de tráfico en el intercambio de mensajes, ya que esta auto-generación a menudo crea más XML de lo necesario.

Por ejemplo, en vez de un XML auto generado como este:

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
   <first_name>Kurt</first_name>
   <last_name>Cobain</last_name>
</person>
```

Los desarrolladores bien pueden optimizarlo así:

```
<?xml version="1.0" encoding="UTF-8"?>
<person fn="Kurt" ln="Cobain" />
```

Resumiendo, el problema con este enfoque *contract-last* es que el artefacto más importante del servicio web, el contrato, es tratado como algo secundario. Se preocupa principalmente en *cómo* el servicio debe estar implementado y no en *qué* debe hacer.

La solución a los problemas que trae consigo *contract-last*, es sencillamente, invertir este enfoque, o sea, crear el contrato primero y luego decidir cómo ha de ser implementado. Cuando se hace esto, se conoce como *contract-first.*

El contrato utilizando esta vía se realiza sin tener mucho en cuenta cómo será la lógica de la aplicación. Esto es un enfoque pragmático, porque hace énfasis en lo que se espera del servicio y no en cómo será implementado.

Spring Web Services es un nuevo y prometedor subproyecto de Spring que se centra en la construcción de servicios web mediante contract-first.

Los tres pasos fundamentales para la creación de un servicio web con Spring Web Services son:

- ✓ **Definir el contrato del servicio:** Esto implica diseñar los mensajes XML que se procesaran en el servicio web, de forma general se crean esquemas XML a partir de los cuales se generan los WSDL.
- ✓ **Implementar el *Endpoint* del servicio:** Se crean las clases que recibirán y procesaran los mensajes enviados al servicio Web.
- ✓ **Configurar el *Endpoint* y la infraestructura de Spring-WS:** Se conectará el Endpoint con un grupo de beans de Spring-WS, que trabajaran en conjunto.

3.2.1 Definir el contrato de los Servicios Web

El contracto de un Servicio consta de dos partes: el *contrato de datos* (data contract) y el *contrato del servicio* (service contract), también conocido como contrato de las operaciones. Ambos se definen mediante XML para mantener la independencia del lenguaje y la plataforma.

Contrato de Datos: Describe los tipos de datos complejos, así como los mensajes de consulta (request) y respuesta (response) del servicio. Se define mediante XSD.

Contrato del Servicio: Describe las operaciones del servicio. Un servicio puede tener múltiples operaciones. Se define mediante WSDL.

En Spring-WS se le facilita gran parte del trabajo al desarrollador ya que el contrato del servicio puede ser generado automáticamente. Pero se debe crear el contrato de datos.

Para crear el contrato de datos existen poderosas herramientas para crear los archivos XSD, el entorno de desarrollo Eclipse es una muestra de ello.

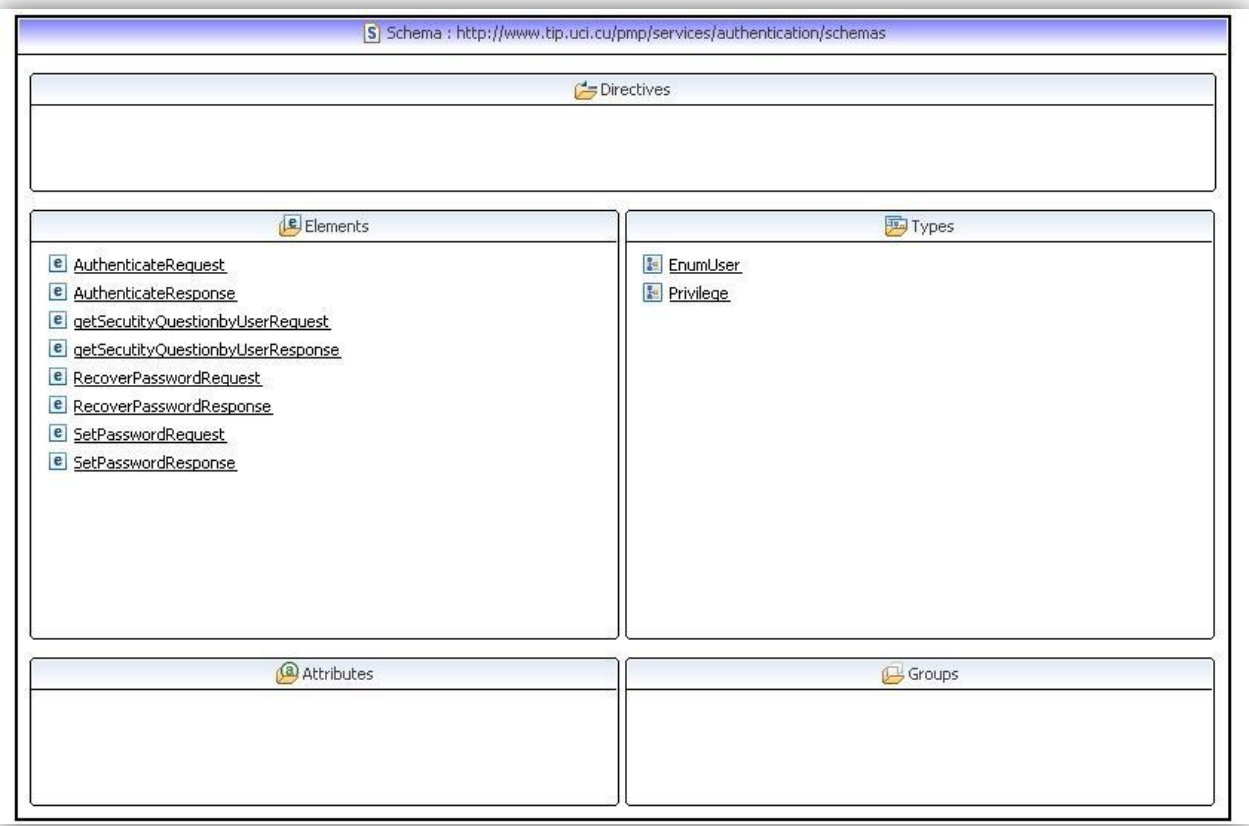

Figura 7: Creación de un XSD mediante el entorno de desarrollo Eclipse.

A continuación mostramos un fragmento de un contrato de datos del servicio *authentication* y de contrato del servicio a través del WSDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.tip.uci.cu/pmp/services/authentication/schemas"
        elementFormDefault="qualified"
        xmlns:schemas="http://www.tip.uci.cu/pmp/services/authentication/schemas">

        <element name="AuthenticateRequest">
                <complexType>
                        <attribute name="user" type="string" />
                        <attribute name="password" type="string" />
                </complexType>
        </element>
</schema>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<wsdl:definitionsxmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:sch="http://www.tip.uci.cu/pmp/services/authentication/schemas"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.tip.uci.cu/pmp/services/authentication/schemas"
   targetNamespace="http://www.tip.uci.cu/pmp/services/authentication/schemas">
<wsdl:types>
<schemaxmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:schemas="http://www.tip.uci.cu/pmp/services/authentication/schemas"
   elementFormDefault="qualified"
   targetNamespace="http://www.tip.uci.cu/pmp/services/authentication/schemas">

<element name="AuthenticateRequest">
<complexType>
<attribute name="user" type="string" />
<attribute name="password" type="string" />
</complexType>
</element></wsdl:definitions>
```

3.2.2 Manejando los Mensajes XML mediante Endpoints

El concepto de Endpoint se puede asociar a un controlador en las aplicaciones Web que implementa el MVC, sin embargo no son lo mismo. El controlador Web se encarga de atender las peticiones HTTP, en cambio el Endpoint se encarga de recibir los mensajes XML del cliente, basado en el contexto del mensaje, hace llamadas a los objetos internos de la aplicación y retorna el resultado en forma de otro mensaje XML. La siguiente figura ilustra cómo los mensajes del cliente interactúan con el Endpoint.

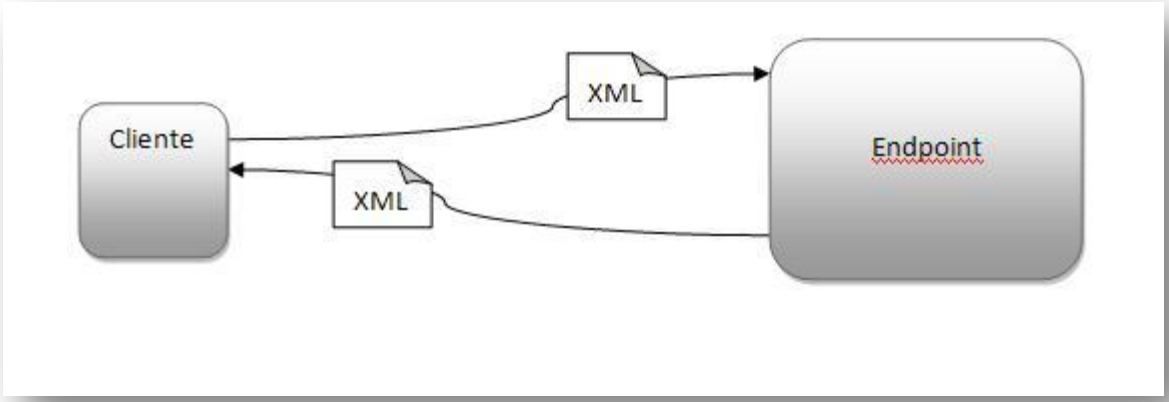

Figura 8: Proceso de manejo de mensajes de los endpoints.

En fin los Enpoints son la implementación de los servicios en Spring-WS. Tomando un enfoque centrado en los mensajes, los endpoint procesan los mensajes XML que le llegan y produce un XML como respuesta.

Spring WS ofrece varias clases endpoints abstractas para procesar los mensajes XML, haciendo uso de diversas tecnologías o APIs de procesamiento de XML. Estas clases están localizadas en el paquete org.springframework.ws.server.endpoint

Tecnología API	Clase Endpoint
DOM	AbstractDomPayloadEndpoint
JDOM	AbstractJDomPayloadEndpoint
dom4j	AbstractDom4jPayloadEndpoint
XOM	AbstractXomPayloadEndpoint
SAX	AbstractSaxPayloadEndpoint
stAX basado en eventos	AbstractStaxEventPayloadEndpoint
stAX por flujos	AbstractStaxStreamPayloadEndpoint
XML Marshalling	AbstractMarshallingPayloadEndpoint

Tabla 2: Clases Enpoint para las diferentes tecnologías de procesamiento de XML.

3.2.3 Serializando los mensajes XML

Los desarrolladores de la capa de servicios de la PMP, se inclinaron por el desarrollo de los Endpoints haciendo uso de la tecnología XML Marshaling para serializar[40] los mensajes XML.
Utilizando XML Marshaling se evita tener que lidiar directamente con los XML y parsearlos directamente con APIs de bajo nivel, para extraer la información que portan, lo que lo convierte en un proceso sumamente trabajoso, además que conspira en contra de la claridad y legibilidad del código. En lugar de esto se trabaja directamente con objetos, lo que agiliza en gran medida el proceso de desarrollo.
Un endpoint basado en serialización, trabaja con un *unmarshaller*[41] que convierte el XML entrante en un POJO. Una vez que el endpoint ha terminado simplemente retorna otro POJO y un marshaller lo convierte a un mensaje XML que es retornado al cliente, como se puede observar en el siguiente diagrama.

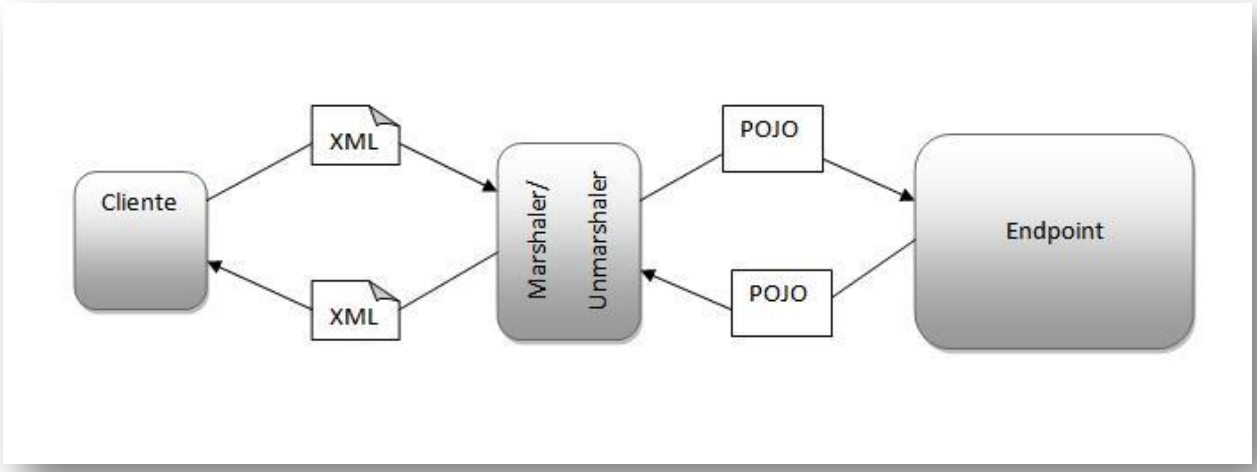

Figura 9: Proceso de mapeo de XML a objetos mediante un Marshalés/Unmarshaller.

Para ilustrar lo que facilita este proceso mostraremos la clase *SearchContactsEndPoint* del servicio *searchsubscriberinformación*, primeramente heredando de la clase abstracta *AbstractJDomPayloadEndpoint*, y luego haciendo uso de las facilidades que brinda el Marshaling con la clase abstracta *AbstractMarshallingPayloadEndpoint*.

```
public class SearchContactsJDomEndpoint
                    extends AbstractJDomPayloadEndpoint
                        implements InitializingBean{

    private Namespace namespace;
```

[40] Serialización (Marshalling en Ingles): Es el proceso de transformar las representaciones en memoria de un objeto a un formato apropiado para ser almacenado o transmitido como XML.
[41] Unmarshaller: Es básicamente lo contrario del marshaller, o sea convierte de un formato como XML a un objeto.

```
        private XPath tipXPath;

        @Override
        protected Element invokeInternal(Element arg0) throws Exception {

                String tip = tipXPath.valueOf(arg0);

                List<Contact> contacts = FactoryManager.getInstance()
                                .getContactManager().searchContactsByTip(tip);


                return createResponse(contacts);
        }
//.....
                private Element createResponse(List<Contact> contacts){

                Element responseElement = new Element("SearchContactResponse", namespace);
                for (int i = 0; i < contacts.size(); i++) {

                        Element element = new Element("contact", namespace);
                        element.setAttribute("state", contacts.get(i).getState());
                        element.setAttribute("contact", contacts.get(i).getContact());
                        element.setAttribute("type", contacts.get(i).getType());
                        element.setAttribute("order",
                String.valueOf(contacts.get(i).getOrder()));
                        element.setAttribute("preference",
                String.valueOf(contacts.get(i).getOrder()));
                        responseElement.addContent(element);
                  }

        return responseElement; } }
```

Como se puede observar, a parte del uso de atributos de tipo *NameSpace* y *XPath*, para el parseo de XML, se ve en el método *createResponse* lo engorroso que resulta hacer el parseo del XML

para cada uno de sus elementos, debido a que el método invokeInternal recibe como parámetro un Element XML y retorna otro Element XML.

Ahora se verá como quedaría la clase *SearchContactsEndpoint* con el Marshaling.

```
public class SearchContactsEndpoint extends AbstractMarshallingPayloadEndpoint {

    @Override
    protected Object invokeInternal(Object requestObject) throws Exception {

        SearchContactsRequest request = (SearchContactsRequest) requestObject;

        List<Contact> contacts = manager.searchContactsByTip(request.getTip());

        SearchContactsResponse response = ConversionUtils.getInstance()
                .convertListTo(contacts, SearchContactsResponse.class);

        return response;
    }

}
```

Obsérvese que primeramente lo que recibe el método *invokeInternal* es un objeto, luego de castearlo a un objeto de tipo *SearchContactsRequest* llamado request, se le puede pasar directamente al método de la interface *getContactManager*, lo que devuelve el método *getTip*. De la misma manera se crea el objeto de respuesta *SearchContactsResponse* response, que será finalmente retornado.

La llave para la conversión de objetos a mensajes XML es un mapeo objeto-XML (OXM[42]). Spring-OXM es un subproyecto de Spring-WS que provee una capa de abstracción sobre otras soluciones OXM populares como JAXB y Castor XML. Los elementos principales de Spring-OXM son sus interfaces Marshaller y Unmarshaller. La implementación de Marshaller genera elementos XML a partir de objetos Java, y por el contrario Unmarshaller construye objetos Java a partir de elementos XML.

AbstractMarshallingPayloadEndpoint toma ventajas de los marshallers y los unmarshallers de Spring-OXM para procesar mensajes, y el solo se encarga de realizar dichos procesos, evitando

[42] OXM del ingles: object-XML mapping

que el desarrollador cree sus propias implementaciones de las interfaces mencionadas. Spring-OXM ya trae consigo una serie de implementaciones listadas en la siguiente tabla.

Solución OXM	Spring OXM marshaller
Castor XML	org.springframework.oxm.castor.CastorMarshaller
JAXB v1	org.springframework.oxm.jaxb.Jaxb1Marshaller
JAXB v2	org.springframework.oxm.jaxb.Jaxb2Marshaller
JiBX	org.springframework.oxm.jibx.JibxMarshaller
XMLBeans	org.springframework.oxm.xmlbeans.XmlBeansMarshaller
XStream	org.springframework.oxm.xstream.XStreamMarshaller

Tabla 3: Soluciones propuestas por Spring-OXM para la transformación de objetos a Xml y viceversa.

El escoger la solución OXM es puramente una cuestión de gusto, ya que todas tienen puntos a favor y en contra. Por lo cual los autores del presente trabajo se decidieron por JAXB v2 sin un motivo en especial.

3.2.4 Configurando la infraestructura de Spring-WS

El último paso (y el más importante) para la implementación de un servicio web con Spring-WS es configurar todo la infraestructura de Spring, o sea llega la parte de la configuración de los beans[43]. A continuación se explicara de forma general el proceso, con ejemplos de servicios de la plataforma.

Spring-WS está basado en Spring MVC[44], en el cual todas las peticiones son manejadas por el *DispatcherServlet*, un servlet[45] especial que envía las peticiones a las clases controladoras que procesaran las peticiones. Igualmente Spring WS presenta un *MessageDispatcherServlet*, una

[43] Como se explico en el capitulo 1 toda la infraestructura de Spring se basa en el manejo de los componentes (beans), que hace el contenedor de Spring mediante la Inversión del Control.

[44] Spring MVC es un subproyecto de Spring para el desarrollo de aplicaciones Web basado en el patrón arquitectónico Modelo-Vista-Controlador.

[45] Servlet: 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.

subclase de *DispatcherServlet* que conoce como despachar[46] las peticiones SOAP a los endpoints de Spring-WS.

El *MessageDispatcherServlet* es configurado en el web.xml de las aplicaciones Web, dentro de los elementos <servlet> y <servlet-mapping>.

```
<servlet>
   <servlet-name>services</servlet-name>
        <servlet-class>
              org.springframework.ws.transport.http.MessageDispatcherServlet
        </servlet-class>
   <init-param>
    <param-name>transformWsdlLocations</param-name>
    <param-value>true</param-value>
   </init-param>
 </servlet>
 <servlet-mapping>
   <servlet-name>services</servlet-name>
   <url-pattern>/services/*</url-pattern>
 </servlet-mapping>
```

Cabe decir que el *MessageDispatcherServlet* es el punto de entrada de Spring-WS, pero aun faltan una serie de beans auxiliares que colaboran en el contexto de Spring.

El primero de estos beans es el encargado de hacer el mapeo de las peticiones hacia los endpoints. *PayloadRootQNameEndpointMapping* es el encargado de esta tarea de mapear los mensajes SOAP examinando el *QName* (Nombre calificado) del contenido del mensaje y luego buscando el endpoint adecuado en de la lista de mapeos (configurados a través de la propiedad *endpointMap*). Este bean está definido en el archivo *WebContent/WEB-INF/services-servlet.xml*.

[46] Despachar es un término típico que se utiliza para describir el proceso de manejo de peticiones que realizan los Dispacher (Despachadores en español).

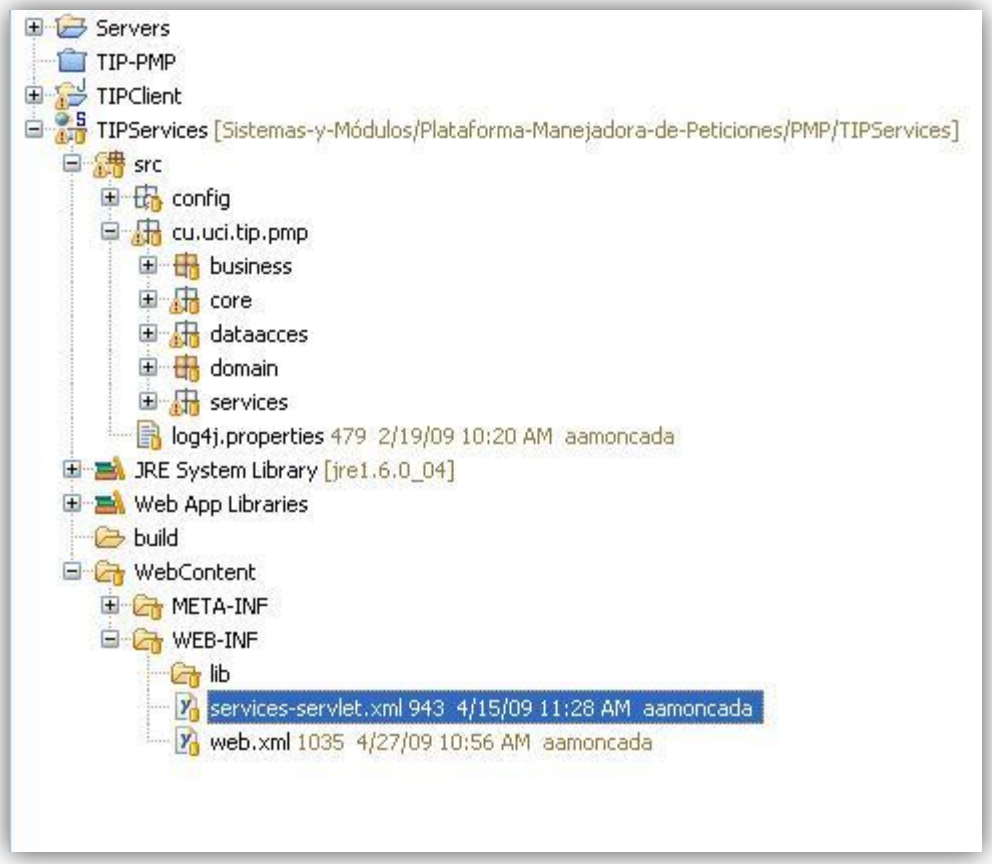

Figura 10: Ubicación del archivo web.xml y services-servlet.xml

```
<bean id="payloadMapping"          class="org.springframework.ws.server.endpoint.mapping.
PayloadRootQNameEndpointMapping">
        <property name="endpointMap">
                <props>
                        <prop  key="{http://www.tip.uci.cu/pmp/services/searchsubscriberinformation/
                        schemas}SearchContactsRequest"> searchContactsEndpoint</prop>
                        <!-- Resto de los endpoints -->
                        </props>
                </property>
        </bean>
```

Como se puede apreciar en el contenido del services-servlet se encuentran mapeados cada uno de las peticiones hacia cada uno de los endpoints que las atenderán.

A continuación se deberá crear un bean por cada endpoint de la capa de servicios de la plataforma, todos definido en el archivo *src/cu/uci/tip/pmp/services/config/endpoints.xml*. A continuación se muestra un ejemplo:

```
<bean id="searchContactsEndpoint"
    class="cu.uci.tip.pmp.services.searchsubscriberinformation.SearchContactsEndpoint">
            <property name="marshaller"  ref="searchsubscriberinformationmarshaller" />
            <property name="unmarshaller" ref="searchsubscriberinformationmarshaller" />
</bean>
```

Obsérvese que cada bean de los enpoints presenta dos propiedades marshaller y unmarshaller. Como se explico anteriormente los endpoints desarrollados en la PMP, heredan de la clase abstracta *AbstractMarshallingPayloadEndpoint*, para realizar el proceso de conversión de los XML a objetos. Por lo tanto a continuación se deberá definir también estos beans que harán marshalling y unmarshalling a cada uno de los objetos y de las peticiones XML que entren y salgan de los endpoints. Estos se encuentran definidos en *src/cu/uci/tip/pmp/services/config/marshallers.xml*. El siguiente es el bean del marshaller del servicio autenticación.

```
    <bean id="authenticationmarshaller"
    class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
            <property name="contextPath"
                    value="cu.uci.tip.pmp.services.authentication.schemas" />
    </bean>
```

Es válido aclarar algo de gran importancia, y es la propiedad contextPath de cada uno de los marshallers. En el contextPath se guardara la referencia al paquete schemas que contendrá cada una de las clases que el marshaller mapeara a elementos XML y viceversa. Es por eso que en el siguiente epígrafe que se explicará la estructura general de la capa de servicios, se podrá observar la distribución de un paquete schema por cada servicio.

Para finalizar queda por definir cada uno de los beans que se encargaran de la creación de los WSDL. Como se explico anteriormente el WSDL es generado automáticamente en Spring-WS, pero debemos definir un bean de tipo *DynamicWsdl11Definition*, que se encargara de crear el WSDL a partir del esquema XML (XSD).

El proceso de creación del WSDL es descrito mediante el siguiente esquema:

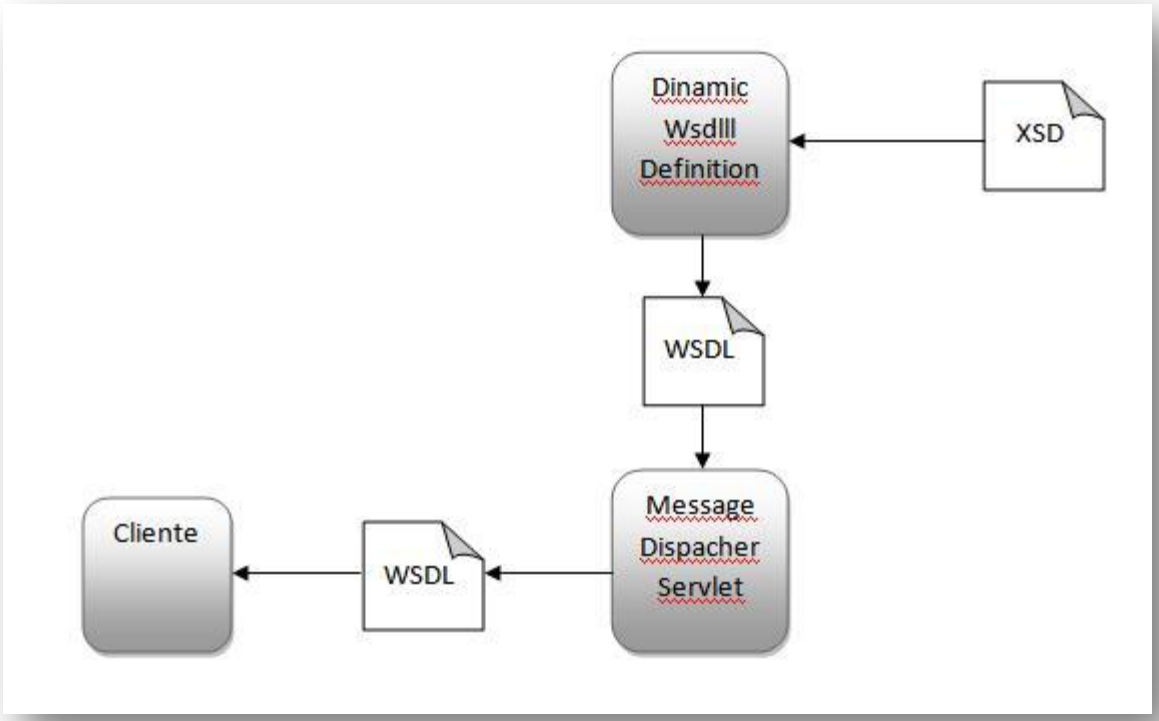

Figura 11: Proceso de creación dinámica del WSDL.

Estos beans aparecen declarados en el archivo *src/cu/uci/tip/pmp/services/config/wsdls.xml*. El siguiente es el bean para generar el wsdl del servicio autenticación.

```
<bean id="authentication"
        class="org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition">
        <property name="schema">
            <bean class="org.springframework.xml.xsd.SimpleXsdSchema">
                <property name="xsd"
                    value="/WEB-INF/classes/cu/uci/tip/pmp/
                    services/authentication/authentication.xsd" />
            </bean>
        </property>
        <property name="portTypeName" value="Authentication" />
        <property name="locationUri" value="/services/" />
</bean>
```

Luego de analizado el proceso de creación de servicios de la capa de servicios de la PMP mediante Spring-WS, se puede pasar a ver la estructura general de la misma así como lo principales elementos que la componen.

3.3 Elementos y Estructuración de la Capa de Servicios

Esta es una vista general de la Plataforma Manejadora de Peticiones:

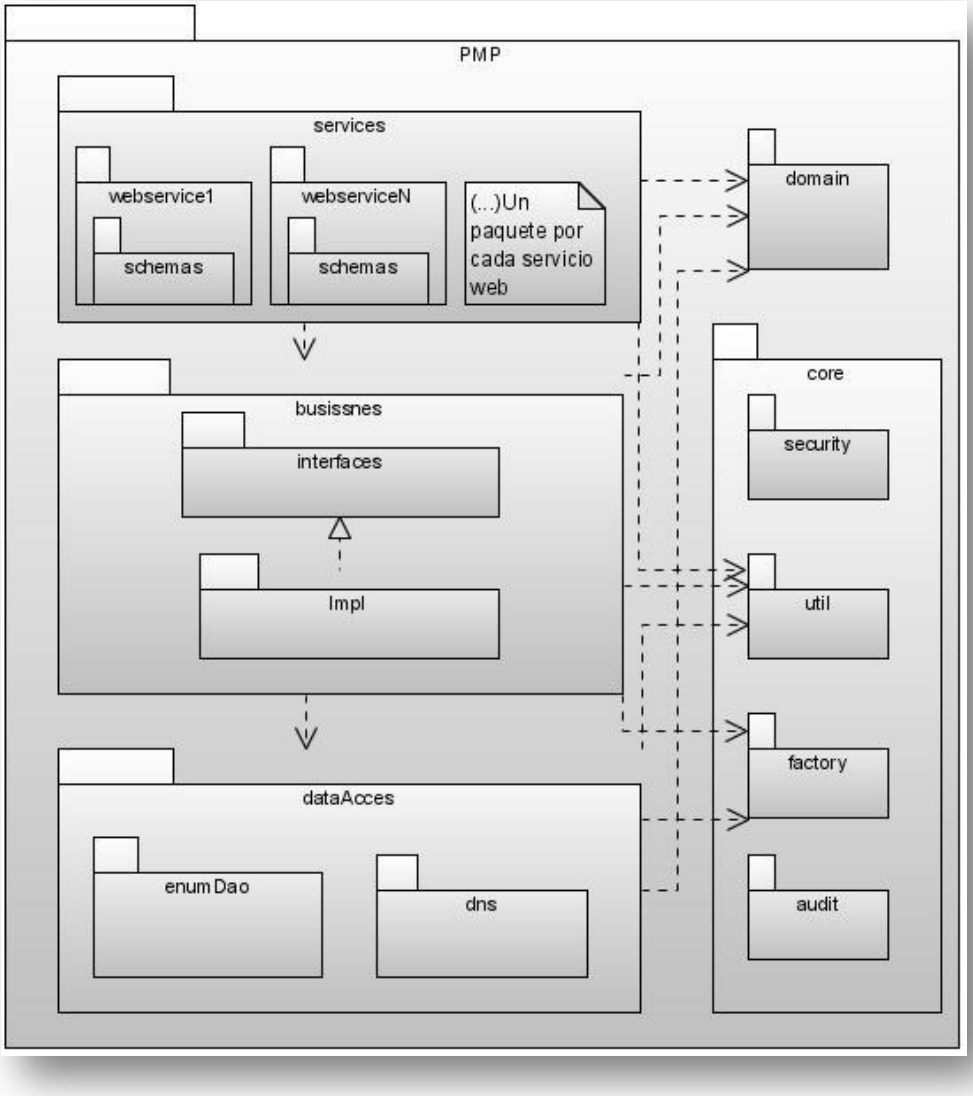

Figura 12: Vista Lógica Arquitectónica de la PMP.

La capa de servicios encapsula el conjunto de clases y componentes responsables de la creación y la publicación de los servicios web mediante Spring WS que brinda la plataforma. Incluyendo la

integración de dicha capa con las capas inferiores mediante las interfaces que proveen los objetos necesarios con la capacidad de resolver el problema en cada situación.

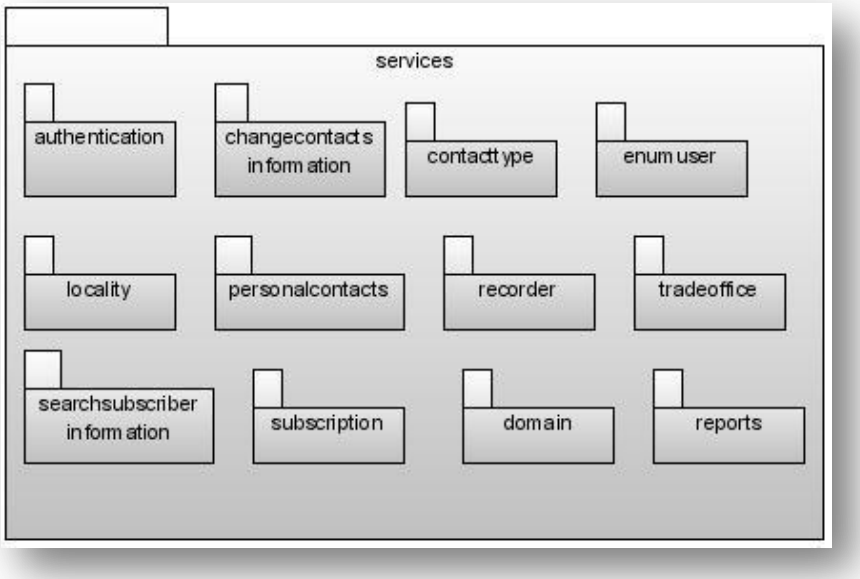

Figura 13: Detalle de la vista lógica de la capa de servicios.

Desde el entorno de desarrollo dichos módulos se muestran por paquetes de la siguiente forma:

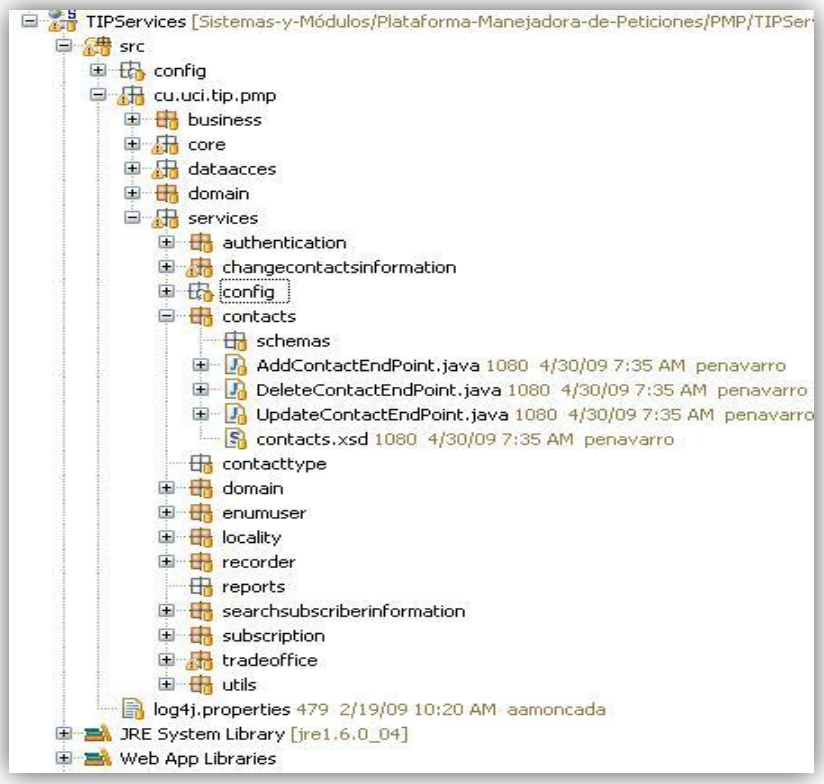

Figura 16: Vista de la capa de servicios desde el entorno de desarrollo

Los servicios se organizan en paquetes, cada uno contendrá todas las clases y archivos asociados a un servicio.

Dentro de cada paquete se definen los endpoints asociados al servicio, un fichero xsd (esquema XML) donde se establecerá los datos del contrato de cada servicio para luego generar el WSDL y un paquete schema para los marshallers como se explico en el epígrafe anterior.

3.3.1 Descripción de las clases por paquetes de la capa de servicios

A continuación se describirán cada uno de las clases endpoints de la capa de servicios de la plataforma por paquetes.

a) Paquete *cu.uci.pmp.services.authentication:*

Nombre:	`public class` **`AuthenticateEndPoint`** `extends` **`AbstractMarshallingPayloadEndpoint`**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**`private`** ***`manager`*** : **`IPasswordManager`**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**`public`** **`invokeInternal`** (**`Object requestObject`**): **`Object`**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	`public class GetSecurityQuestionByUserEndpoint` **`extends`** `AbstractMarshallingPayloadEndpoint`
Tipo de clase:	**Control**
Atributos:	
Nombre:	**`private`** ***`manager`*** : **`IPasswordManager`**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**`public`** **`invokeInternal`** (**`Object requestObject`**) : **`Object`**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	`public class` **`RecoverPasswordEndPoint`** `extends` **`AbstractMarshallingPayloadEndpoint`**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**`private`** ***`manager`*** : **`IPasswordManager`**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**`public`** **`invokeInternal`** (**`Object requestObject`**) : **`Object`**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	`public class` **`SetPasswordEndPoint`** `extends` **`AbstractMarshallingPayloadEndpoint`**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**`private`** ***`manager`*** : **`IPasswordManager`**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**`public`** **`invokeInternal`** (**`Object requestObject`**) : **`Object`**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

b) Paquete *cu.uci.pmp.services.changecontactsinformation:*

Nombre:	`public class PlanContactbyIntervalEndPoint` **`extends`** **`AbstractMarshallingPayloadEndpoint`**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**`private`** ***`manager`*** : **`IPlanificationContactManager`**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**`public`** **`invokeInternal`** (**`Object requestObject`**) : **`Object`**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	public class SetContactsPreferenceEndPoint **extends** **AbstractMarshallingPayloadEndpoint**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**private *manager*** : **IPlanificationContactManager**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**public invokeInternal** (**Object requestObject**) : **Object**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	public class SetContactStateEndPoint **extends** **AbstractMarshallingPayloadEndpoint**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**private *manager*** : **IPlanificationContactManager**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**public invokeInternal** (**Object requestObject**) : **Object**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

c) Paquete *cu.uci.pmp.services.contacts:*

Nombre:	public class AddContactEndPoint **extends** **AbstractMarshallingPayloadEndpoint**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**private *manager*** : **IContactManager**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**public invokeInternal** (**Object requestObject**) : **Object**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	public class DeleteContactEndPoint extends AbstractMarshallingPayloadEndpoint
Tipo de clase:	Control
Atributos:	
Nombre:	private *manager* : IContactManager
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	public invokeInternal (Object requestObject) : Object
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	public class UpdateContactEndPoint extends AbstractMarshallingPayloadEndpoint
Tipo de clase:	Control
Atributos:	
Nombre:	private *manager* : IContactManager
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	public invokeInternal (Object requestObject) : Object
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

d) Paquete *cu.uci.pmp.services.domain:*

Nombre:	public class getDomainEndPoint extends AbstractMarshallingPayloadEndpoint
Tipo de clase:	Control
Atributos:	
Nombre:	private *manager* : IDomainManager
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	public invokeInternal (Object requestObject) : Object
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	public class UpdateDomainEndPoint extends AbstractMarshallingPayloadEndpoint
Tipo de clase:	Control
Atributos:	

Nombre:	**private** ***manager*** : **IDomainManager**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**public invokeInternal** (**Object requestObject**) : **Object**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

e) Paquete *cu.uci.pmp.services.enumuser:*

Nombre:	public class AddEnumUserEndpoint **extends** **AbstractMarshallingPayloadEndpoint**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**private** ***manager*** : **IEnumUserManager**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**public invokeInternal** (**Object requestObject**) : **Object**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	public class GetEnumUserbyIdEndpoint **extends** **AbstractMarshallingPayloadEndpoint**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**private** ***manager*** : **IEnumUserManager**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**public invokeInternal** (**Object requestObject**) : **Object**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	public class GetEnumUserbyUserEndpoint **extends** **AbstractMarshallingPayloadEndpoint**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**private** ***manager*** : **IEnumUserManager**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	

Nombre:	**public invokeInternal** (**Object requestObject**) : **Object**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	public class RemoveEnumUserEndpoint **extends** **AbstractMarshallingPayloadEndpoint**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**private *manager*** : **IEnumUserManager**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**public invokeInternal** (**Object requestObject**) : **Object**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

Nombre:	public class UpdateEnumUserEndpoint **extends** **AbstractMarshallingPayloadEndpoint**
Tipo de clase:	**Control**
Atributos:	
Nombre:	**private *manager*** : **IEnumUserManager**
Descripción:	*Encapsula la interface del negocio que será seteada mediante el bean de Spring*
Métodos:	
Nombre:	**public invokeInternal** (**Object requestObject**) : **Object**
Descripción:	*Método encargado de recibir y responder a las peticiones que llegan al Endpoint*

3.3.2 Paquete *utils* y mapeo de objetos mediante la librería Dozer

Cada uno de los servicios utiliza un grupo de clases necesarias para llevar a cabo la serialización de los objetos que se transformarán en XML y viceversa, las cuales se encuentran en el paquete *schema*. Estas clases en apariencia son muy similares a las de la capa de dominio, lo que solo contienen los datos necesarios para cada servicio en específico. Ahora bien, para interactuar con las capas inferiores es necesario hacerlo mediante las clases del dominio, por lo que es necesario transformar estos objetos de los paquetes *schema* a objetos del dominio. Realizar este proceso paso a paso, es decir, atributo por atributo, se torna muy engorroso y demora el proceso de desarrollo. Sin embargo existen librerías para facilitar este proceso, una de estas se denomina Dozer.

Dozer es un mapeador de objetos java que recursivamente copia los datos de un objeto a otro. A menudo estos objetos suelen ser tipos de datos complejos. Dozer soporta mapeo de propiedades simples, tipos complejos, mapeo bidireccional, mapeo implícito y explícito, además de mapeo recursivo. Esto incluye mapeo de colecciones que a su vez requiere mapear sus elementos.

Dozer no solo soporta mapeo mediante nombres de atributos, sino que también convierte automáticamente entre tipos. La mayoría de los escenarios de conversión ya están previstos, no obstante permite especificar conversiones personalizadas a través de XML.

El mapeador es usado en cualquier instante que se necesite tomar un tipo de objeto y mapearlo hacia otro objeto de otro tipo. La mayoría de los mapeos de los campos pueden ser realizados automáticamente mediante reflexión, pero cualquier mapeo personalizado debe ser prescrito mediante formato XML. El mapeo es bidireccional por lo que solo se debe definir una relación entre clases. Si algunas de las propiedades de ambos objetos coinciden en el nombre no se necesita especificar el mapeo explícitamente para estas propiedades.

Para llevar a cabo el proceso de mapeo mediante el Dozer se utilizaron un conjunto clases para auxiliares. A continuación se listan las mismas:

Nombre:	`public class ConversionUtils`
Atributos:	
Nombre:	`private static instance: ConversionUtils`
Descripción:	*Contiene la instancia de la clase*[47]*.*
Nombre:	`private mapper: Mapper`
Descripción:	*Contiene el mapeador del Dozer*
Métodos:	

[47] Esta clase implementa el patrón Singleton el cual permite tener solamente una sola instancia de dicha clase. En el campo *instance* esta almacenada la misma.

Nombre:	`private static synchronized` **`createInstance()`** : `void`
Descripción:	*Método encargado de crear la instancia de la clase.*
Nombre:	`public static` **`getInstance()`**:**`ConversionUtils`**
Descripción:	*Método encargado de retornar la instancia de la clase*
Nombre:	`public` **`convertTo(Object source, Object destination)`**: `void`
Descripción:	*Método encargado de convertir el objeto origen al objeto destino*
Nombre:	`public` **`<T> convertTo(Object source, Class<T> destinationClass): T`**
Descripción:	*Método encargado de convertir el objeto origen a un objeto de la Clase destino que es retornado.*
Nombre:	`public` **`<T,C> convertListTo(List<T> source, Class<C> destinationClass): C`**
Descripción:	*Método encargado de convertir una lista origen a un objeto de la Clase destino que es retornado.*

Nombre:	**`public class`** `ListContainer<T>`
Atributos:	
Nombre:	`private` **`elements: List<T>`**
Descripción:	*Contiene la lista de Elementos Genéricos*
Métodos:	
Nombre:	`public` **`getElements()`**:**`List<T>`**
Descripción:	*Método encargado de retornar la lista de genéricos.*
Nombre:	`public` **`setElements(List<T> elements)`**:`void`
Descripción:	*Método encargado de setear la lista de genéricos.*

Nombre:	**`public class`** `MunicipalityCustomConverter`
Métodos:	
Nombre:	`public` **`convert()`**:**`Object`**
Descripción:	*Método encargado de realizar la conversión.*

Nombre:	**`public class`** `PrivilegeCustomConverter`
Métodos:	

Nombre:	`public` **`convert()`**:**`Object`**
Descripción:	*Método encargado de realizar la conversión.*

La clase *ConversionUtils* brinda un conjunto de funcionalidades que facilitan el trabajo con el Dozer en la capa de servicios, así mismo la clase *ListContainer* es genérica, es una especie de comodín y contiene una lista de elementos; se usa en el caso que se desea convertir una lista de elementos en una clase determinada, este escenario se repite a menudo, con este objetivo se diseño dicha clase.

Las clases *MunicipalityCustomConverter* y *PrivilegeCustomConverter* implementan la interfaz *org.dozer.CustomConverter*, se encuentran en el paquete *utils.converters*, estas son usadas para realizar mapeos personalizados entre dos objetos, en este caso son usadas para convertir un objeto de tipo *Integer* a *Municipality* y viceversa , al igual que de tipo *Integer* a *Privilege* y viceversa.

3.3.3 Paquete *config*

En el paquete *config* se encuentran los ficheros de configuración de forma general de la capa de servicios.

En el fichero endpoints.xml están definidos los beans de cada uno de los endpoints.

En el fichero marshallers.xml están definidos los beans de cada uno de los marshallers.

En el fichero wsdl.xml están definidos los beans de cada uno de los wsdl.

En el fichero services-config.xml se encuentran las configuraciones en general de los servicios, incluyendo las anteriores y otras como la del Dozer.

3.4 Conclusiones del Capítulo

Durante el presente capítulo se pudo apreciar de forma general la manera en que fue implementada la capa de servicios del Teleidentificador personal, las características de la misma de forma general así como los componentes que la componen.

CAPITULO 4

Manejo de la Seguridad en el TIP y Pruebas a los Servicios

4.1 Introducción

En el presente capítulo se trata uno de los temas más importantes a la hora de desarrollar la plataforma de servicios del Teleidentificador Personal, la Seguridad.

Durante el mismo se expone como se manejaron los aspectos de la seguridad en el TIP mediante del novedoso enfoque que propone Spring, haciendo uso de un paradigma de programación bastante reciente, la Programación Orientada a Aspectos. También se pueden observar varios puntos de importancia a la hora de implementar la seguridad en un sistema empresarial de forma general.

En los epígrafes finales del capítulo se describen como fueran realizadas las pruebas a los servicios web, para de esta forma dar complemento a una de las fases más importantes en las etapas de desarrollo de software.

4.2 Seguridad

La seguridad es uno de los aspectos más importantes a tener en cuenta al desarrollar aplicaciones empresariales, sobre todo si 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 y 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 solo 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 Services Security).

4.2.1 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 la seguridad en la capa de transporte se basa en HTTP seguro (HTTPS) usando Secure Sockets Layer (SSL). La seguridad en el transporte es un mecanismo de seguridad punto a punto que puede usarse para autenticación, integridad del mensaje y confidencialidad.
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.
El mensaje es solo protegido mientras son transportados, no cuando son emitidos y/o recibidos por los participantes. La protección es eliminada automáticamente por el punto final cuando este recibe 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.

4.2.2 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:

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.

Requiere implementación de mecanismos de seguridad de nivel XML.

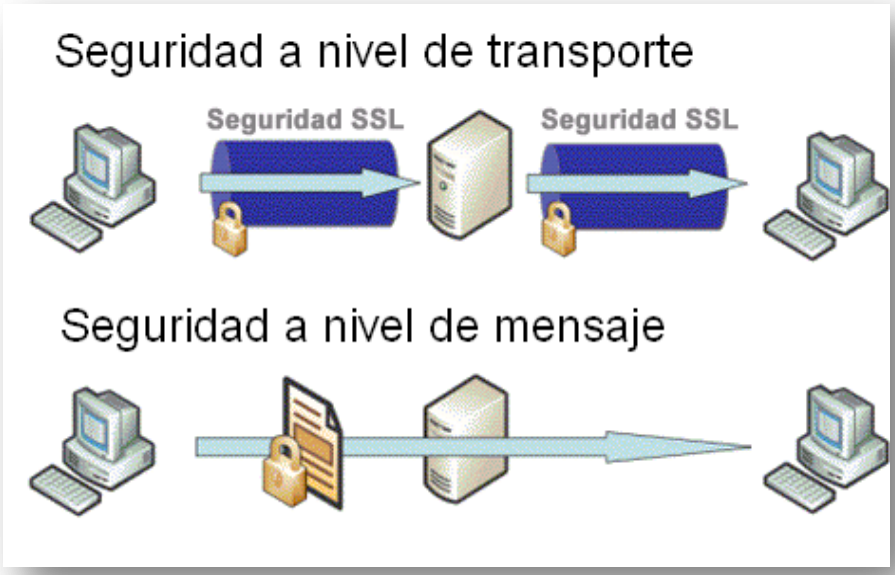

Figura 14: Seguridad en el nivel del protocolo frente a seguridad en el nivel de mensajes

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 Services 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:

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 usted 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 Services, 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 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 encriptarán los mensajes cuya información requiera la garantía de confidencialidad; así como el uso de firmas digitales[48] en los mensajes SOAP enviados y recibidos por la Plataforma Manejadora de Peticiones para garantizar la integridad de los datos transmitidos.

Cada aplicación que consuma los servicios requerirá un certificado digital[49]. Estos certificados deben ser firmados por una entidad certificadora de confianza del servicio. Los certificados serán creados con la herramienta keytool contenida en el JDK y la entidad certificante con la herramienta OpenSSL, que consiste en un robusto paquete de herramientas de administración y librerías relacionadas con la criptografía.

4.2.3 Trabajando con XwsSecurityInterceptor

Para validar los elementos de seguridad anteriormente mencionados se utilizará las facilidades que brinda Spring Web Services, específicamente la clase XwsSecurityInterceptor, que es un interceptor de Endpoints que está basado en el paquete de seguridad de XML y Servicios Web de Sun Microsystems (XWSS) cuya funcionalidad es validar los tokens de seguridad contenidos en la cabecera de los mensajes SOAP. Este interceptor requiere para operar de 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 enviados. El XwsSecurityInterceptor delega a un conjunto de manejadores responsabilidades como la obtención de certificados, llaves privadas, la validación de las credenciales de los usuarios, etc.

Primeramente se agrega este interceptor de seguridad a nuestro mapeo de endpoints, cada petición será interceptada y validada de acorde a las políticas de seguridad que se especificarán más adelante.

```
<bean id="payloadMapping"
        class="org.springframework.ws.server.endpoint.mapping.PayloadRootQNameEndpointMapping">
```

[48] 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.

[49] Un Certificado Digital es un documento digital mediante el cual un tercero confiable (una autoridad certificante) garantiza la vinculación entre la identidad de un sujeto o entidad y su clave pública

```
        <property name="endpointMap">
                <props>
                        <prop
                        key="{http://www.tip.uci.cu/pmp/services/searchsubscriberinformation/schemas}SearchPersonalDataRequest">
                                searchPersonalDataEndpoint</prop>
                        <prop
                        key="{http://www.tip.uci.cu/pmp/services/locality/schemas}getMunicipalitiesByProvinceIdRequest">
                                getMunicipalitiesByProvinceIdEndpoint</prop>
                        <prop
                        key="{http://www.tip.uci.cu/pmp/services/tradeoffice/schemas}ListTradeOfficeByMunicipalyRequest">
                                listTradeOfficeByMunicipalityEndpoint</prop>
                        <prop
                        key="{http://www.tip.uci.cu/pmp/services/contacttype/schemas}ListContactTypeRequest">
                                listContactTypeEndpoint</prop>
                        ...
                </props>
        </property>
        <property name="interceptors">
                <list>
                        <ref bean="xwsSecurityInterceptor" />
                </list>
        </property>
</bean>

<bean id="xwsSecurityInterceptor"
class="org.springframework.ws.soap.security.xwss.XwsSecurityInterceptor">
<property name="policyConfiguration" value="classpath:securityPolicy.xml" />
...
</bean>
```

Como se menciona anteriormente se va a requerir que todos los mensajes vengan con un certificado, el cual contiene información acerca de la aplicación cliente que está tratando de consumir el servicio web en cuestión, además la mayoría de los servicios web requieren de las credenciales de los usuarios para así llevar a cabo la autorización. Adicionalmente algunos mensajes que contienen información confidencial requieren de encriptación.

Ahora se tienen que especificar las políticas de seguridad que se van a aplicar, las cuales le indican al interceptor qué restricciones requieren los mensajes entrantes y si se necesita (como es el caso) para los mensajes salientes también.

Cada mensaje entrante que no cumpla con esta política será rechazado por el interceptor.

Para el caso que se requiere que los mensajes sean firmados digitalmente:

```
<xwss:SecurityConfiguration  xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
        ...
```

```
        <xwss:RequireSignature requireTimestamp="true"/>
        <xwss:Sign>
        <xwss:X509Token certificateAlias="wsserver"/>
        </xwss:Sign>
        ...
</xwss:SecurityConfiguration>
```

Como se puede notar, también se especifica que se agregue un estampado de tiempo para que la fecha y la hora en que se firmó el mensaje sean incluidas en la firma. Adicionalmente se firman los mensajes salientes para garantizar la integridad de los mismos, se le especifica el alias asociado a la llave privada para poder realizar esta operación.

Hasta el momento, de acorde con las políticas de seguridad aplicadas, solo se aceptan mensajes firmados y por lo tanto con el certificado digital del firmante. Sin embargo cualquier cliente puede consumir el servicio web simplemente creando un par arbitrario de llave pública y privada, y crear un certificado auto firmado (en este caso) y consumir el servicio sin problemas. Por lo que se necesita un mecanismo para identificar si un supuesto cliente es en realidad un cliente aprobado de la plataforma. Para esto lo que se hace es firmar el certificado del cliente por una Entidad Certificante (EC) en la cual se confía.

Resumiendo, se confiará en todos los clientes de los cuales se reciban mensajes que contengan un certificado firmado por la EC en la cual se confía.

El siguiente paso es indicarle a Spring WS dónde encontrar dicha EC, para eso se debe crear un almacén de confianza que no es más que un almacén de llaves de java común y corriente en el cual se importará el certificado de la EC en cuestión. Como se dijo anteriormente este interceptor delega responsabilidades en un conjunto de manejadores que son los encargados de realizar la operación requerida. Se le agregará entonces uno de estos manejadores para llevar a cabo la verificación de dichos certificados.

```
<bean id="xwsSecurityInterceptor"
        class="org.springframework.ws.soap.security.xwss.XwsSecurityInterceptor">
        <property name="policyConfiguration" value="classpath:securityPolicy.xml" />
                <property name="callbackHandlers">
                        <list>
                                <ref bean="keystoreCallbackHandler" />
                        </list>
                </property>
</bean>

<bean id="keystoreCallbackHandler"
        class="org.springframework.ws.soap.security.xwss.callback.KeyStoreCallbackHandler">
        <property name="keyStore" ref="keyStore" />
        <property name="privateKeyPassword" value="serverpruebaprivada" />
        <property name="trustStore" ref="trustStore" />
</bean>
```

```
<bean id="keyStore"
        class="org.springframework.ws.soap.security.support.KeyStoreFactoryBean">
        <property name="location" value="classpath:keyStore.jks" />
        <property name="password" value="serverprueba" />
</bean>

<bean id="trustStore"
        class="org.springframework.ws.soap.security.support.KeyStoreFactoryBean">
        <property name="location" value="classpath:truststore.jks" />
        <property name="password" value="truststorepass" />
</bean>
```

Se usa el KeyStoreCallbackHandler y mediante la propiedad trustStore se le indica el almacén de confianza donde se tendrá la EC en cuestión. Se configura además las propiedades keyStore y privateKeyPassword (contraseña para acceder a la llave privada) para que puedan ser firmados los mensajes salientes.

Para lograr la autenticación como se dijo anteriormente viaja en la cabecera del mensaje el usuario y la contraseña del cliente que desea realizar una operación determinada, las políticas de seguridad para lograr esto quedarían así:

```
<xwss:SecurityConfiguration  xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
        ...
        <xwss:RequireUsernameToken passwordDigestRequired="true"/>
        ...
</xwss:SecurityConfiguration>
```

Adicionalmente se especifica que la contraseña viaje encriptada.

Para complementar las políticas de seguridad aplicadas y lograr este objetivo se usa un manejador llamado SpringDigestPasswordValidationCallbackHandler como se muestra a continuación:

```
<bean id="xwsSecurityInterceptor"
        class="org.springframework.ws.soap.security.xwss.XwsSecurityInterceptor">
        <property name="policyConfiguration" value="classpath:securityPolicy.xml" />
        <property name="callbackHandlers">
                <list>
                        ...
                        <ref bean="authenticationHandler" />
                </list>
        </property>
</bean>

<bean id="authenticationHandler"
        class="org.springframework.ws.soap.security.xwss.callback.SpringDigestPasswordValidatio
        nCallbackHandler">
        <property name="userDetailsService" ref="userDetailsService" />
</bean>
```

```
<bean id="userDetailsService"
	class="cu.uci.tip.pmp.business.impl.UserDetailsServiceManagerImpl">
</bean>
```

Este manejador requiere de un UserDetailService de Spring Security para operar. Usa este servicio para obtener la contraseña del usuario que viaja en el mensaje. Ya que el usuario viaja con una encriptación de un solo sentido, se compara este usuario encriptado del mensaje con el que obtiene de este servicio.

Para llevar a cabo la autorización se utiliza Spring Security que hace uso del soporte de Spring en cuanto a la Programación Orientada a Aspectos para aplicar seguridad a nivel de método. Esto se logra configurando un proxy el cual intercepta las llamadas a métodos y le pasa el control a un interceptor de seguridad. Para esto se utiliza la clase BeanNameAutoProxyCreator:

```
<bean id="autoProxyCreator"
	class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
	<property name="interceptorNames">
		<list>
			<value>methodSecurityInterceptor</value>
		</list>
	</property>
	<property name="beanNames">
		<list>
			<value>searchContactsManager</value>
		</list>
	</property>
</bean>

<bean id="methodSecurityInterceptor"
	class="org.springframework.security.intercept.method.aopalliance.MethodSecurityIntercepto
r">
	<property name="authenticationManager" ref="authenticationManager" />
	<property name="accessDecisionManager" ref="accessDecisionManager" />
	<property name="objectDefinitionSource">
		<value>
			cu.uci.tip.core.pmp.bussines.SearchContactsManager.*=ROLE_ADMIN
		</value>
	</property>
</bean>
```

Se configura la propiedad beanNames en la cual se indican los beans que se quieren asegurar con un interceptor llamado MethodSecurityInterceptor al cual se le configura la propiedad objectDefinitionSource, la cual asocia patrones de métodos con privilegios necesarios para invocar el método correspondiente. También se configuran las propiedades authenticationManager y accessDecisionManager mediante la cual se determina si el usuario tiene privilegios suficientes para realizar la operación solicitada.

A este administrador de autenticación se le configura la propiedad providers con una implementación de la interfaz AuthenticationProvider llamada DaoAuthenticationProvider que como bien quiere decir, es un proveedor de autenticación que obtiene los detalles del usuario de una implementación de UserDetailsService.

```
<bean id="authenticationManager"
        class="org.springframework.security.providers.ProviderManager">
        <property name="providers">
                <ref bean="daoAuthenticationProvider" />
        </property>
</bean>

<bean id="daoAuthenticationProvider"
        class="org.springframework.security.providers.dao.DaoAuthenticationProvider">
        <property name="userDetailsService" ref="userDetailsService" />
</bean>

<bean id="userDetailsService"
        class="cu.uci.tip.pmp.business.impl.UserDetailsServiceManagerImpl">
</bean>

<bean id="accessDecisionManager"
        class="org.springframework.security.vote.AffirmativeBased">
        <property name="allowIfAllAbstainDecisions" value="false" />
        <property name="decisionVoters">
                <list>
                        <bean class="org.springframework.security.vote.RoleVoter" />
                </list>
        </property>
</bean>
```

El administrador de decisión de acceso es una implementación de la interfaz AccessDecisionManager llamada AffirmativeBased la cual concede acceso al recurso solicitado si alguno de los votantes da una respuesta afirmativa. En este caso se utiliza un votante basado en roles.

En el caso de los mensajes que requieren de encriptación se aplican las políticas de seguridad siguientes:

```
<xwss:SecurityConfiguration
        xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
        <xwss:RequireEncryption>
                <xwss:X509Token certificateAlias="wsserver" />
                <xwss:EncryptionTarget type="xpath" value="//SOAP-ENV:Body/*"/>
        </xwss:RequireEncryption>
        <xwss:Encrypt >
                <xwss:X509Token certificateAlias=""/>
                <xwss:EncryptionTarget type="xpath" value="//SOAP-ENV:Body/*"/>
        </xwss:Encrypt>
        <xwss:Sign>
    <xwss:X509Token certificateAlias="wsserver"/>
```

```
    </xwss:Sign>
</xwss:SecurityConfiguration>
```

Se desencriptan los mensajes entrantes con la llave privada del servidor, a través del alias (wsserver) vinculado a esta llave, que se encuentra en el almacén de llaves. Los mensajes salientes requieren ser encriptados también, para esto se utiliza la llave pública vinculada al certificado incluido en la petición del cliente, por este motivo no se especifica ningún alias a la hora de asegurar la respuesta. La encriptación no se lleva a cabo sobre el mensaje completo, sino sobre los elementos que viajan dentro del cuerpo del mensaje, para así poder ser mapeados a su endpoint correspondiente.

La entidad certificante en la cual se confía se crea usando OpenSSL, que es un robusto paquete de herramientas de administración y librerías relacionadas con la criptografía. Para generar los almacenes de llaves de java (keystore), se utiliza la herramienta keytool, que se puede encontrar tanto en el JDK como en el JRE de Sun Microsystems. Es una utilidad de gestión de claves y certificados, puede utilizarse para administrar pares de claves públicas y privadas y los certificados asociados, también se puede utilizar para generar peticiones de certificado, de manera que se pueden utilizar autoridades de certificación para firmar los certificados.

4.3 Pruebas a los Servicios Web del Teleidentificador Personal

Una de los procesos claves en el desarrollo de software son las pruebas. Las pruebas aseguran la calidad del software que se crea. Existen varios tipos de prueba, como lo son pruebas de unidad, de integración, funcionales, pruebas al sistema, de rendimiento o de aceptación, e incluso otras más. Las pruebas se pueden efectuar de forma manual o automática.

Las pruebas aseguran muchas veces que las aplicaciones de software que se desarrollan cumplan con lo que desea el usuario final, de ahí la importancia que tiene el flujo de prueba en el proceso de desarrollo de software.

A continuación se podrá observar como se validan, comprueban o evalúan los servicios web implementados en la capa de servicios del Teleidentificador personal, de forma tal que permitan su uso desde las aplicaciones clientes que los utilizan.

4.3.1 Modelo de Pruebas

Durante la disciplina de prueba es objetivo verificar que el sistema cumpla con los requerimientos de los clientes y usuarios finales para lograr una mayor calidad, el principal objetivo es realizar y evaluar las pruebas como se describen en el modelo de pruebas.

El modelo de pruebas describe principalmente cómo se prueban los componentes ejecutables en el modelo de implementación. El modelo de pruebas es una colección de casos de prueba, procedimientos de prueba y componentes de prueba.

4.3.2 Herramienta SoapUI

Las pruebas a los servicios del TeleIdentificador Personal fueron realizadas mediante la herramienta SoapUI.

SoapUI es una aplicación de escritorio, libre y de código abierto que entre sus principales funcionalidades permite inspeccionar, invocar y desarrollar servicios web. Además brinda la opción de hacer pruebas funcionales, de carga y conformidad a los servicios. Permite simular servicios para hacer las pruebas. SoapUI puede también ser integrado con el IDE Eclipse mediante un plugin.

SoapUI está especialmente dirigida a desarrolladores, consumidores o probadores de servicios web. No importa desde que plataforma, ya sea Java, .Net, etcétera. Las pruebas funcionales y de carga a los servicios pueden ser realizadas ambas de forma interactiva o automática.

4.3.3 Realización de Casos de Prueba mediante SoapUI

En SoapUI las pruebas funcionales se pueden usar para validar la funcionalidad requerida para cada una de las invocaciones a un servicio en específico (prueba de unidad) o una secuencia de llamadas (prueba de integración). Además, se le pueden personalizar las pruebas en la forma que se desee, como por ejemplo usando alguna lógica de un flujo de prueba en específico.

4.3.4 Resultados de las pruebas a los servicios del TeleIdentificador personal

Existen dos métodos fundamentales de pruebas: el primero *Prueba de Caja Negra* que no son más que el conjunto de pruebas que se llevan sobre la interfaz de usuario y se centran en los requisitos funcionales del software. El segundo *Prueba de Caja Blanca* que realiza un seguimiento del código fuente según se van ejecutando los casos de prueba de manera que se puede determinar de manera correcta las instrucciones en las que existen errores.

A pesar de que los servicios Web no implementan como tal una interfaz de usuario, con la ayuda de la herramienta SoapUI, se le pudieron realizar las pruebas de caja negra a los mismos. A continuación se describen algunos casos de prueba. Un caso de prueba específica una forma de probar el sistema, incluyendo la entrada o resultado con la que se ha de probar y las condiciones

bajo las que ha de probarse. Para comprobar el funcionamiento del sistema se realizaron diferentes casos de prueba, a continuación se muestra un caso de prueba de caja negra.

Prueba de Caja negra

Servicio Web: Autenticación

Caso de prueba: CPR1.1 Autenticar

Descripción del flujo:

1. El usuario introduce el nombre de usuario y la contraseña.

Clase Válida	Clase no Válida	Resultado Esperado	Resultado
El usuario introduce su nombre de usuario (“aamoncada”) y su contraseña (“mestalla”).		Se le devuelve al usuario un elemento *user* con todos los datos correspondientes. (Figura 1)	Satisfactorio.
	El usuario introduce su nombre de usuario (“aamoncada”), pero no su contraseña.	Se muestra un mensaje de advertencia “Error al autenticarse”.	Satisfactorio (Figura 2).

Tabla 3: Caso de Prueba Autenticar del Servicio Autenticación.

Prueba de Caja negra

Servicio Web: Localidad

Caso de prueba: CPR1.2 Obtener Municipios dado el id de la provincia.

Descripción del flujo:

1. El usuario introduce el id de la provincia.

Clase Válida	Clase no Válida	Resultado Esperado	Resultado
El usuario introduce el id de la provincia Camagüey (9).		Se le devuelve al usuario un listado con todos los datos correspondientes a los municipios. (Figura 3)	Satisfactorio.

	El usuario introduce texto no valido (“dasdasd”).	Se muestra un mensaje de advertencia “Ocurrió un error”.	Satisfactorio

Tabla 4: Caso de Prueba Obtener Municipios dado el id de la provincia.

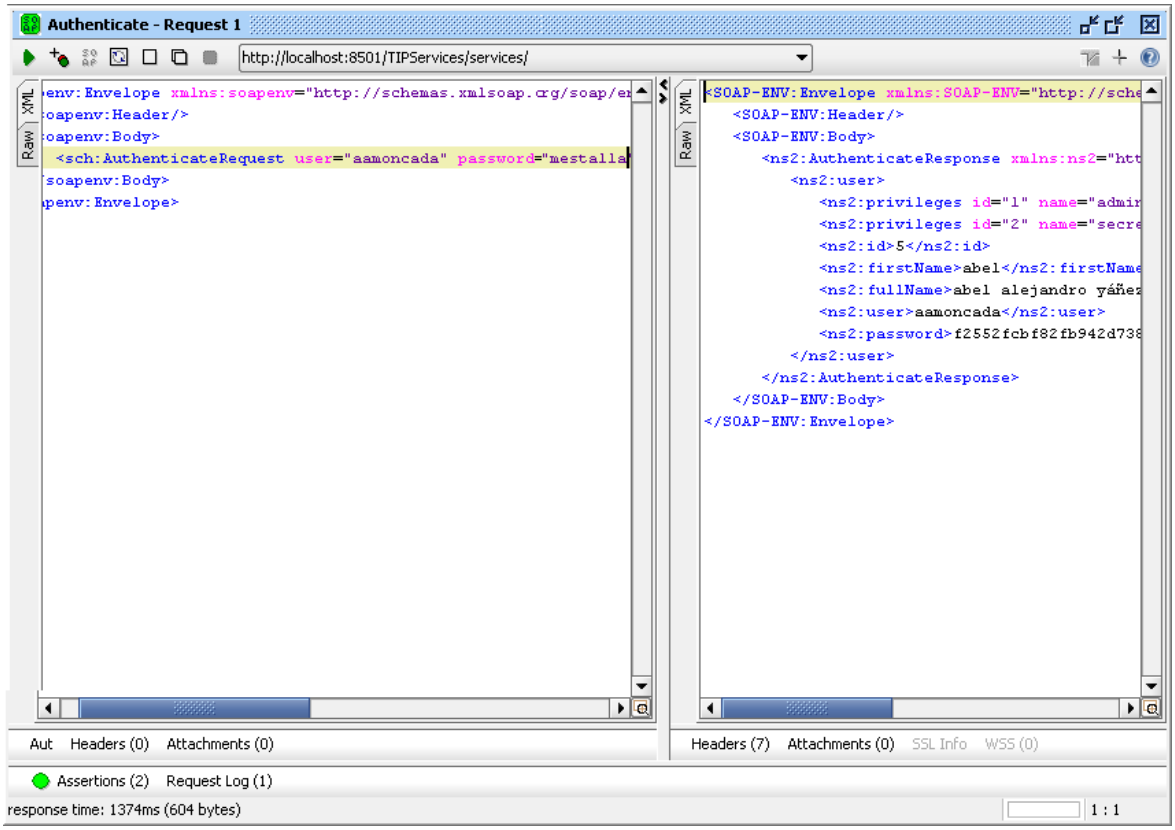

Figura 14: Resultado del caso de prueba autenticar.

4.3.5 Prueba de estrés a los servicios

A partir del SoapUI 1.5 se incluye una funcionalidad simple y poderosa conocida como *Prueba de Carga de Soap* que permite validar el rendimiento de un servicio bajo distintos escenarios de carga, o sea pruebas de estrés. Además permite realizar validaciones funcionales a los servicios para asegurar que no fallen durante las peticiones que se les realicen. Por último permite realizar varias llamadas simultáneamente para ver si no se afectan unas a otras.

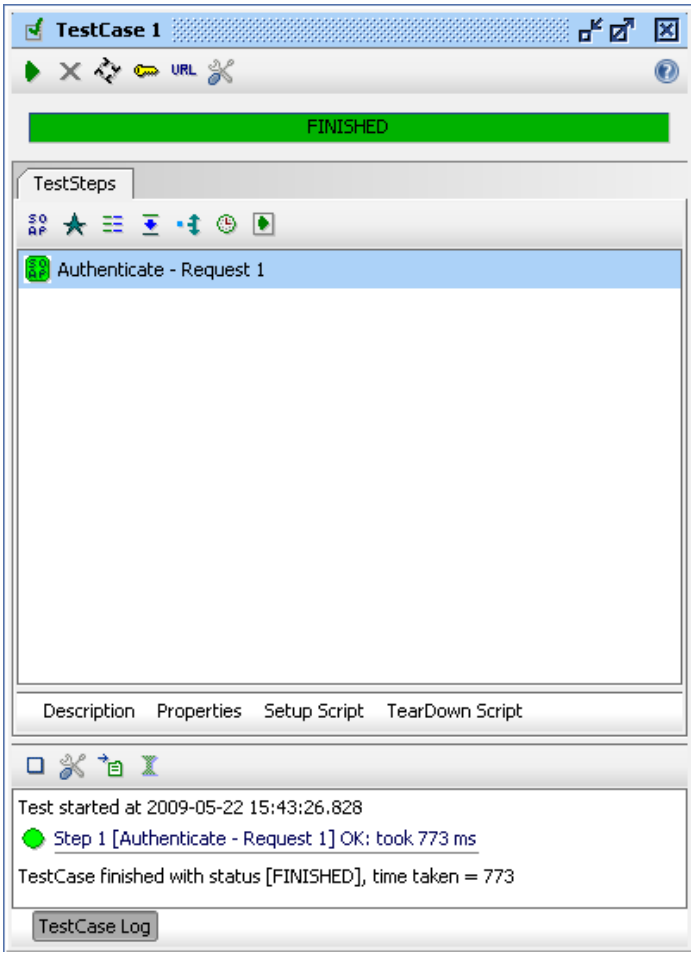

Figura 15: Resultado de la prueba al servicio autenticación con SoapUI

A los servicios del TeleIdentificador personal se le realizaron prueba de carga para ver como respondían bajo determinadas circunstancias como: gran cantidad de peticiones simultaneas a diferentes servicios y por diferentes clientes y respuestas ante fallos. A continuación se describe una de estas pruebas.

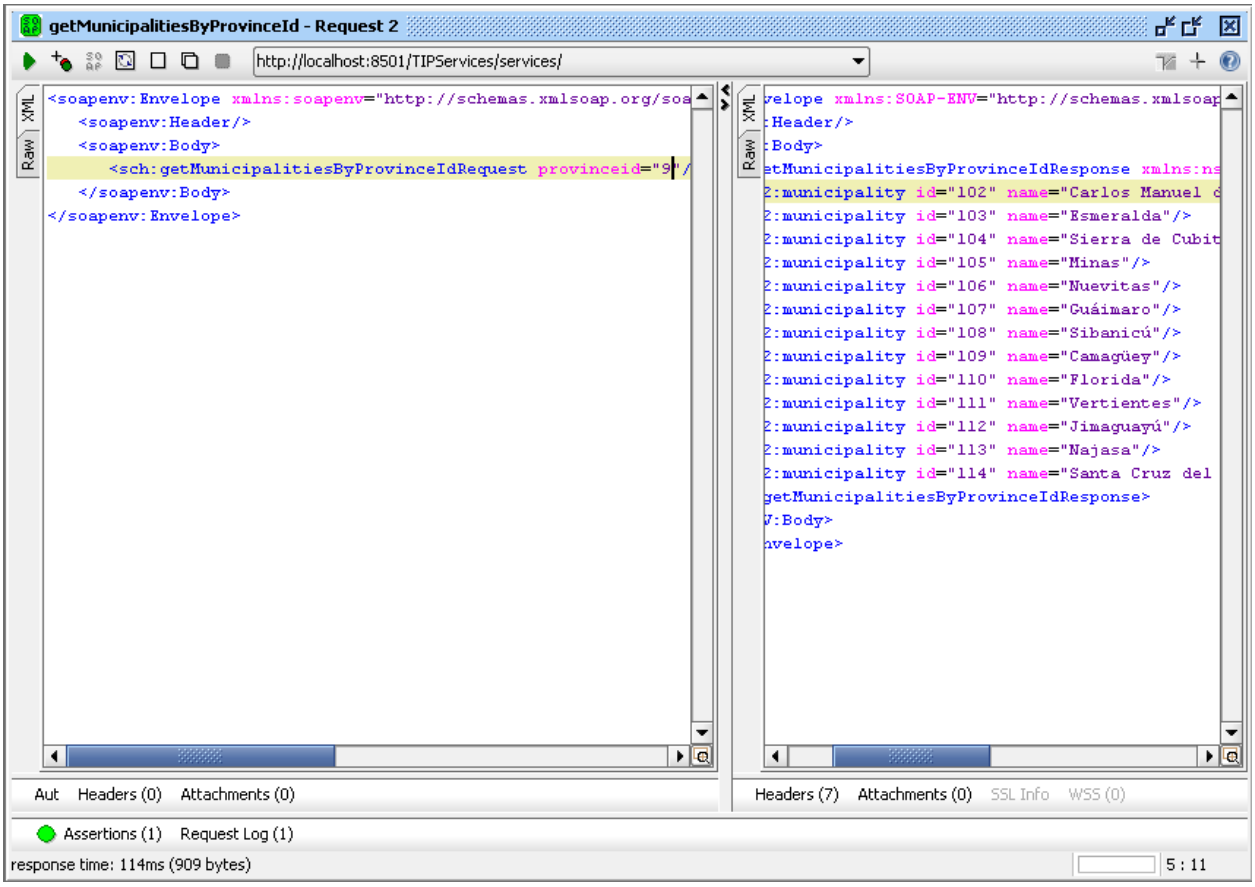

Figura 16: Resultado del caso de prueba Obtener Municipios dado el id de la provincia.

Prueba de Carga

Servicios Web:

- Autenticar
- Localidad
- Búsqueda de Información del Subscriptor
- Usuario Enum
- Tipos de Contactos.

Casos de prueba:

- CPR1.1 Autenticar
- CPR1.2 Obtener Municipios dado el id de la provincia
- CPR1.3 Obtener Provincias dado el id del país.
- CPR1.4 Búsqueda de datos personales.
- CPR1.5 Obtener Usuario Enum por usuario.
- CPR1.6 Listar Tipos de contactos.

Parámetros para la prueba:

- Tiempo limite de respuesta: 30 segundos
- Cantidad de hilos simultáneos: 5

En la siguiente imagen se muestran los resultados de la prueba:

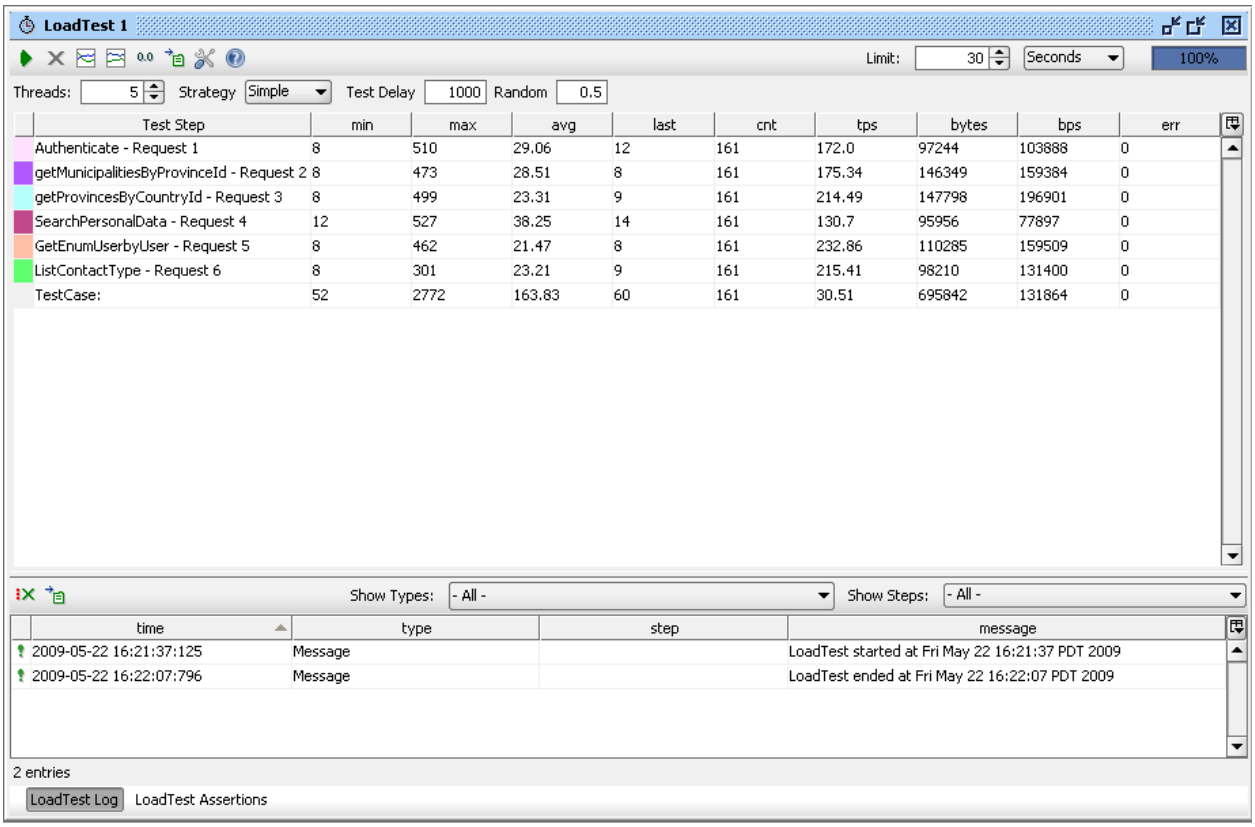

Figura 17: Prueba de carga de Soap a los servicios web.

4.4 Conclusiones

La seguridad en un sistema empresarial es uno de los puntos de vital importancia a la hora de llevar a cabo el desarrollo del mismo. Un sistema que sea desarrollado con totalidad y cumpla con todos los requisitos funcionales, pero que no cuente con los mecanismos necesarios para mantener la confidencialidad de los datos que maneja, así como resguardada la información de sus clientes no encuentra ningún tipo de aceptación en el mercado. Es por esta razón que durante el desarrollo del TeleIdentificador Personal se le presto especial atención a los aspectos relacionados con la seguridad como se explico a lo largo de este capitulo.

La fase de prueba en un ciclo de desarrollo de software puede ser considerada un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación. Un buen desarrollo de pruebas permite que el software cumpla con la calidad requerida antes de ser entregado al cliente y que logre satisfacer sus necesidades.

Conclusiones

Enum es, sin lugar a dudas, un servicio que mejorará en gran medida las telecomunicaciones en el país. Además el uso de un TeleIdentificador personal agilizará muchos procesos de negocio y permitirá en un futuro no muy lejano ampliar estos servicios a otras esferas como la Salud o la Educación. Enum es un novedoso servicio que ETECSA pone en manos de nuestro pueblo para facilitar sus actividades y mejorar su nivel de vida.

Desarrollar la Plataforma Manejadora de Peticiones haciendo uso de una capa de servicios permite la fácil interacción de las aplicaciones clientes con el DNS debido a la flexibilidad que ofrecen los Servicios Web. Se permite que las diversas aplicaciones clientes del servicio ENUM puedan establecer la comunicación con el DNS sin importar en que lenguaje de programación estén implementadas o de que tecnología hagan uso, no obstante, se debieron tomar todas las medidas necesarias para mantener la integridad de la información que se manejaba. La implementación de la seguridad haciendo uso de un nuevo paradigma de programación como es la Programación Orientada a Aspectos, que aparece como un complemento de la Programación Orientada a Objetos para el manejo de elementos verticales al negocio de la aplicación como son la seguridad o la auditoria, permite lograr mayor elegancia y claridad en el código, debido a que mediante la misma ya las clases no incluyen código que no tenga que ver implícitamente con su lógica de negocio.

El desarrollo de la capa de servicios mediante Spring WS permite un novedoso y potente enfoque del desarrollo de Servicios Web en Java. Con la idea de que “el contrato es lo primero” Spring WS mejora muchos elementos que otros frameworks no han tenido en cuenta como el versionado de servicios y el desacople entre la lógica del servicio y su contrato.

Bibliografía

1. *XML. Donde empiezan los Servicios Web.* **Martínez Martínez, E. y Olguín Espinoza, J. M.** Mexico DF : s.n., 2004.
2. *Los servicios web son la revolución informática de la nueva generación de aplicaciones.* **Breña Moral, Juan Antonio.** 2002.
3. **Garlan, D. y Shaw, M.** *An Introduction to Software Architecture.* s.l. : Carnegie Mellon University, 1994 .
4. **R. E. Johnson, y B. Foote.** *"Designing Reusable Classes".* 1988.
5. **Russo, R. E. Johnson y V. F.** *"Reusing Object-Oriented Designs".*
6. **Johnson, Rod.** *Professional Java development with the Spring framework* . s.l. : Wrox, 2005 .
7. —. *Expert One-on-One J2EE Design and Development.* s.l. : Wrox Press, 2003.
8. **Morales Machuca, Carlos Andrés.** *Estado del Arte: Servicios Web.* Colombia : Universidad Nacional.
9. **W3C Consortium.** Web Services Architecture. [Online] Febrero 11, 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#whatis.
10. **W3C Recommendation.** SOAP Version 1.2 Part 0: Primer (Second Edition). [Online] Abril 27, 2007. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.
11. **W3C Consortium.** SOAP. [Online] http://www.w3.org/TR/soap/.
12. —. El estándar XML. [Online] http://www.w3.org/TR/REC-xml.
13. **The Internet Engineering Task Force.** IETF. [Online] http://www.ietf.org/.
14. **W3C Consortium.** WSDL . [Online] http://www.w3.org/TR/wsdl.
15. **Mak, Gary.** *Spring Recipes: A Problem-Solution Approach.* s.l. : Apress, 2008.
16. **Walls, Craig.** *Spring in Action.* s.l. : Manning Publications Co., 2008.
17. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : AddisonWesley Professional Computing Series, 1995.
18. **Ivar Jacobson, Grady Booch, James Rumbaugh.** *El Lenguaje Unificado de Modelado.* s.l. : Addison-Wesley iberoamericana , 1999.
19. **Jacobson, I, Grady, B and Rumbaugh, J.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison-Wesley, 2000.
20. **C., Benjamín González.** http://www.desarrolloweb.com. *El Futuro de los Servicios Web XML.* [Online] http://www.desarrolloweb.com/articulos/1664.php.
21. *Impacto del ENUM en las redes y los servicios.* **Peréz, Marcos.** s.l. : Revista de Telecomunicaciones, 2008.

ENUM (Telephone Numbering Mapping): Protocolo estandarizado por la IETF y que permite la convergencia entre las redes basadas en el protocolo IP como el Internet y las redes telefónicas. ENUM permite almacenar la información de contacto para un número telefónico y poder utilizar un número telefónico convencional como llave para acceder servicios tales como voz sobre IP.

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.

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.

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

Broadcast (en castellano difusión): Es un modo de transmisión de información donde un nodo emisor envía información a una multitud de nodos receptores de manera simultánea, sin necesidad de reproducir la misma transmisión nodo por nodo.

Bytecode: El bytecode es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un fichero binario que contiene un programa ejecutable similar a un módulo objeto, que es un fichero binario producido por el compilador cuyo contenido es el código objeto o código máquina.

API (Application Programming Interface): Una interfaz de programación de aplicaciones es el conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Java Community Process: El Proceso de la Comunidad Java, o Java Community Process, establecido en 1998, 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.

Wiki: Sitio web que permite la edición de sus contenidos por parte de las personas que acceden a él, con ciertas restricciones mínimas.

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.

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

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.

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

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.

XML (Extensible Markup Language): Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y 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.

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.

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.

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.

Certificado X.509: Certificado Digital que define un formato estándar de certificado para certificados de clave pública y la certificación de validación. Como parte del protocolo X.509, los certificados están asociados a cada usuario y son creados por una Autoridad de Certificados (ó Certificate Authority, ó CA) confiable (trusted).

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.

Número TIP (TeleIdentificador Personal): Número ENUM asociado a un subscriptor.

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.

Autoridad certificadora: Organismo independiente que emite certificados digitales que atestiguan la identidad de los propietarios de los mismos.

Keytool: Es una herramienta para el manejo de certificados y claves. Permite a los usuarios administrar sus propios pares de claves pública/privada y asociar certificados para su uso en auto-autenticación o la integridad de los datos y servicios de autenticación, utilizando firma digital. Almacena las claves y certificados en un almacén de claves llamado keystore.

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

OpenSSL: Consiste en un robusto paquete de herramientas de administración y librerías relacionadas con la criptografía, que suministran funciones criptográficas. Esta herramienta ayuda al sistema a implementar el Secure Sockets Layer (SSL), así como otros protocolos relacionados con la seguridad. También permite crear certificados digitales.

Anexos

Anexo 1: Pautas de Codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continuada un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y, posteriormente, se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

1. Consideraciones Generales

Con el objetivo de ajustarse a los estándares internacionales se establece utilizar el idioma ingles 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	package cu.uci.systemdata;

	compuestas se utiliza minúscula.	
clases	Los nombres de las clases serán un sustantivo y comenzarán con mayúscula, se utiliza mayúscula en las distintas palabras de los nombres compuestos.	class Computer; class PriorityQueue; class System_Map; //Evitar

clases	Mantener nombres simples y evitar abreviaturas o acrónimos a no ser que sean ampliamente conocidos.	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.	run();
		runFast();
		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 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).

4. Longitud de la Línea

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

5. Líneas plegadas

Cuando una expresión no entra en una línea simple debido a su extensión se debe dividir 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.

6. 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.

7. 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.

8. 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.
 */
```

9. 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. */
   ...
}
```

10. 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 */
}
```

11. 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.
        }
}
```

12. 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 { … }
```

13. Declaraciones

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!
```

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 pre 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;

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

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() {
            return 0;
      }
      }
```

14. Sentencias

Sentencias simples

Cada línea debe contener solo una sentencia:

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

Sentencias compuestas

Deberán estar encerradas en un par de llaves:

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

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;
```

15. Sentencias de condiciones simples y anidadas

IF, IF ELSE, IF ELSE IF ELSE

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

SWITCH CASE

```
switch (condition) {
```

```
        case A:
        ...
        break;
        case B: {
        ...
        } break;
        default:
        ...;
}
```

16. Sentencias de Lazo

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).

WHILE

Una sentencia while debe tener la siguiente forma:

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

DO WHILE

Una sentencia do-while debe tener la siguiente forma:

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

FOREACH

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

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

}
```

- **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;
}
```

17. Espacios en blanco

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);
```

- **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.

18. Practicas de Programación

Proporcionando acceso a variables de instancia y de clase

No hacer ninguna variable de instancia o clase pública sin una buena razón. A menudo las variables de instancia no necesitan ser asignadas/consultadas explícitamente, a menudo esto sucede como efecto lateral de llamadas a métodos.

Referencias a variables y métodos de clase

Evitar usar un objeto para acceder a una variable o método de clase (static). Usar el nombre de la clase en su lugar.

Constantes

Las constantes numéricas (literales) no se deben codificar directamente, excepto -1, 0, y 1, que pueden aparecer en un bucle for como contadores.

Asignaciones de variables

Evitar asignar el mismo valor a varias variables en la misma sentencia. Es difícil de leer.

Paréntesis

En general es una buena idea usar paréntesis en expresiones que implican distintos operadores para evitar problemas con el orden de precedencia de los operadores. Incluso si parece claro el orden de precedencia de los operadores, podría no ser así para otros, no se debe asumir que otros programadores conozcan el orden de precedencia.

Expresiones antes de '?' en el operador condicional

Si una expresión contiene un operador binario antes de '?' en el operador ternario ?: , se debe colocar entre paréntesis. Ejemplo:

(x >= 0) ? x: -x;