ARQUITECTURA CORPORATIVA

DE WEB SERVICES

Memòria del projecte
d'Enginyeria en Informàtica
realitzat per
Javier Vacas Gallego
i dirigit per
Joan Borrell Viader
J. Carlos Muiño Gallego
Bellaterra, 10 de Juny de 2008

CERTIFICACIÓ DE DIRECCIÓ

El sotasignat, **Joan Borrell Viader**

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en **Javier Vacas Gallego**.

I per tal que consti firma la present.

Signat: **Joan Borrell Viader**

Bellaterra, 10 de Juny de 2008

CERTIFICACIÓ DE DIRECCIÓ EN EMPRESA

El sotasignat, **J. Carlos Muiño Gallego**

de l'empresa, **Prexon Consulting, S.L.**

CERTIFICA

Que el treball a què correspon aquesta memòria ha estat realitzat en l'empresa sota la seva supervisió mitjançant conveni amb **Javier Vacas Gallego**

firmat amb la Universitat Autònoma de Barcelona.

Així mateix, l'empresa en té coneixement i dóna el vist-i-plau al contingut que es detalla en aquesta memòria.

Signat: **J. Carlos Muiño Gallego**

Santa Coloma de Gramenet, 10 de Juny de 2008

Índice de contenidos

Tabla de figuras

Capítulo 1: Introducción

1.1 Motivación del proyecto

Prexon Consulting S.L. es una consultoría informática especializada en el sector asegurador y financiero, nacida en 2005 y creada por un equipo de titulados en Ingeniería Informática. Desde su creación se ha centrado en dar respuesta a las diversas necesidades que presentan todo tipo de compañías aseguradoras y financieras, tanto software de gestión como soluciones orientadas a la expansión comercial de este tipo de empresas.

Uno de los productos estrella de Prexon es la llamada *Plataforma de Multitarificación de Seguros*. Esta aplicación RIA (Rich Internet Application) es una aplicación web con aspecto similar al de una aplicación de escritorio que permite a un usuario poder consultar el precio de una póliza de seguros para varios productos (autos, hogar, salud, accidentes, etc.) de manera remota y en diversas aseguradoras simultáneamente. Lo que por una parte aumenta la comodidad del cliente final y por otra incrementa la competencia entre compañías.

Mi trayectoria personal en Prexon se remonta a Marzo de 2006. En ese momento me integro en el equipo de trabajo encargado de desarrollar la *Plataforma de Multitarificación*, concretamente trabajando sobre la capa intermedia de la aplicación. Esta capa está destinada, básicamente, a recibir peticiones de los usuarios y hacérselas llegar a cada una de las compañías de las que el usuario desea obtener precios. Esto nos ha permitido conseguir una gran experiencia en este tipo de comunicaciones corporativas, consistentes en

envíos telemáticos de pequeño tamaño pero que contienen datos personales y críticos.

Por otro lado Aura Seguros S.A. es una pequeña aseguradora dedicada especialmente a los productos de decesos y accidentes. Este proyecto responde a una petición de Aura hacia Prexon para desarrollar una interfaz de conexión e integración con la *Plataforma de Multitarificación*. El objetivo comercial de Aura es ampliar su red de servicios convencionales añadiéndole una red externa que amplíe de manera significativa su actual expansión.

En la actualidad las aplicaciones de multitarificación de seguros se están convirtiendo en grandes herramientas de expansión para todas las compañías aseguradoras y en especial para las más pequeñas. No sólo porque el proceso de contratación de pólizas se hace de manera automatizada, ahorrando así recursos, si no porque, además, estas aplicaciones tienen la ventaja de que llegan fácilmente a una gran cantidad de posibles clientes a través de Internet. Gracias a esto las diferencias de presupuesto entre compañías y las inversiones en marketing o en la red comercial de sucursales no son tan definitivas como en los canales de ventas convencionales. De este modo, en el momento de visualizar la lista de precios podemos hacer una comparación de precios y garantías entre cada aseguradora de manera absoluta. Por tanto, una pequeña empresa como Aura tendrá la oportunidad de competir directamente con los grandes gigantes del mundo asegurador.

1.2 Objetivos

Nuestro objetivo principal en este proyecto es llevar a cabo la conexión entre la compañía de seguros Aura y la *Plataforma de Multitarificación* de modo que podamos hacer presupuestos de pólizas para todos los productos que Aura ofrezca a sus clientes, así como tener la posibilidad de llevar a cabo la contratación de una póliza en el caso de que alguno de los precios presupuestados nos interese. Además, si este es el caso, devolveremos al cliente la documentación acreditativa de que efectivamente ya ha realizado la contratación de la póliza.

La disposición de Prexon en la realización de este proyecto se sustenta en el interés de crecimiento del producto de multitarificación. El ofrecer una garantía de competencia habilitando la posibilidad a los clientes de obtener presupuestos en la mayor cantidad de aseguradoras posibles es uno de los puntos clave necesarios para asegurar el éxito comercial de este producto. El objetivo final es que pueda llegar a convertirse en una herramienta de referencia para correduríias de seguros y por qué no, para los usuarios finales. Además, el hecho de poder reutilizar este desarrollo para futuros clientes de Prexon que quieran formar parte de la *Plataforma de Multitarificación,* es un importante activo de cara a su integración en la aplicación.

Como objetivos personales hay que destacar la notable experiencia que obtendré a lo largo del desarrollo, en particular en el mundo de las tecnologías de la información correspondientes a los sectores empresariales, corporativos y comerciales. De la misma, manera también conseguiré importantes conocimientos técnicos durante la realización de este proyecto, que servirán de complemento práctico y real a los estudios adquiridos en la carrera de Ingeniería en Informática. Por último, el hecho de formar parte de un equipo de trabajo dinámico y enérgico, formado por grandes profesionales del sector con abundante experiencia, es una razón significativa de la elección de este proyecto por mi parte.

El hecho de que en Prexon hayamos desarrollado la *Plataforma de Multitarificación* nos permite tener una gran experiencia respecto a este tipo de conexiones entre compañías, de modo que conocemos previamente cual son las técnicas más usadas, así como las que permiten una mayor flexibilidad, seguridad y facilidad de uso. Después de realizar un trabajo previo de selección de la tecnología idónea para las necesidades expuestas por Aura (ver sección 2.2) se ha tomado la decisión de que los web services son la solución adecuada. Sus ventajas respecto al resto de posibilidades radican en su interoperabilidad entre distintas plataformas y aplicaciones, así como en la capacidad que poseen de implementar distintos sistemas de seguridad, debido a que esta tecnología se apoya en HTTP (*HyperText Transfer Protocol*).

Es importante destacar el aspecto de la seguridad, el cual toma especial relevancia dado que se trabajará con datos críticos tanto para el cliente como para la compañía. En el desarrollo de este proyecto se tendrá especial cuidado con esta característica del proyecto, por lo que se seguirán de manera categórica las leyes que regulan el tratamiento de dichos datos, como la Ley Orgánica de Protección de Datos entre otras.

Este proyecto, por tanto, consistirá en desarrollar un web service con la misión principal de dar respuesta a las peticiones de presupuesto y contratación de pólizas en diversos productos, siendo fácilmente escalable, con el objetivo de que en un futuro se pueda ampliar dando cabido a otros servicios como la consulta de pólizas por parte de los agentes de seguros o la gestión de partes de accidentes. Es importante destacar, también, que a pesar de que el desarrollo de este proyecto es particular para la aseguradora Aura, es un requisito básico hacerlo de manera que se pueda reutilizar fácilmente en futuros proyectos corporativos que presenten unas necesidades similares a las comentadas.

1.3 Punto de partida

Partimos desde una situación en la que encontramos un sistema informático de gestión de pólizas para agentes en dos vertientes: interno y externo. El sistema de gestión interno consiste en una aplicación de escritorio que permite llevar a cabo una amplia gestión comercial y económica sobre la compañía. Por otra parte el externo radica en una aplicación web que permite a los agentes disponer de una gestión exhaustiva sobre su cartera de pólizas. Estos dos entornos trabajan sobre una base de datos, la cual será el punto de conexión entre los sistemas ya existentes en la compañía y la solución que desarrollaremos dentro del marco de este proyecto. Como resultado obtendremos un sistema totalmente integrado en el que los web services adoptarán un papel principal como uno de los métodos de expansión comercial de la compañía.

1.4 Planificación

Este proyecto comienza a realizarse en octubre de 2007 en Prexon Consulting S.L. y se hace un cálculo aproximado inicial de seis a ocho meses de duración. La figura 1-1 muestra el diagrama de Gantt de la planificación prevista al comenzar el proyecto incluyendo los principales pasos que se tomarán durante su desarrollo. Aunque este será el proyecto principal al que estaré asignado en Prexon, no trabajaré a tiempo completo en él, pueden existir días o semanas que por carga de trabajo o por otro tipo de aspectos inesperados me tenga que dedicar momentáneamente a otros proyectos.

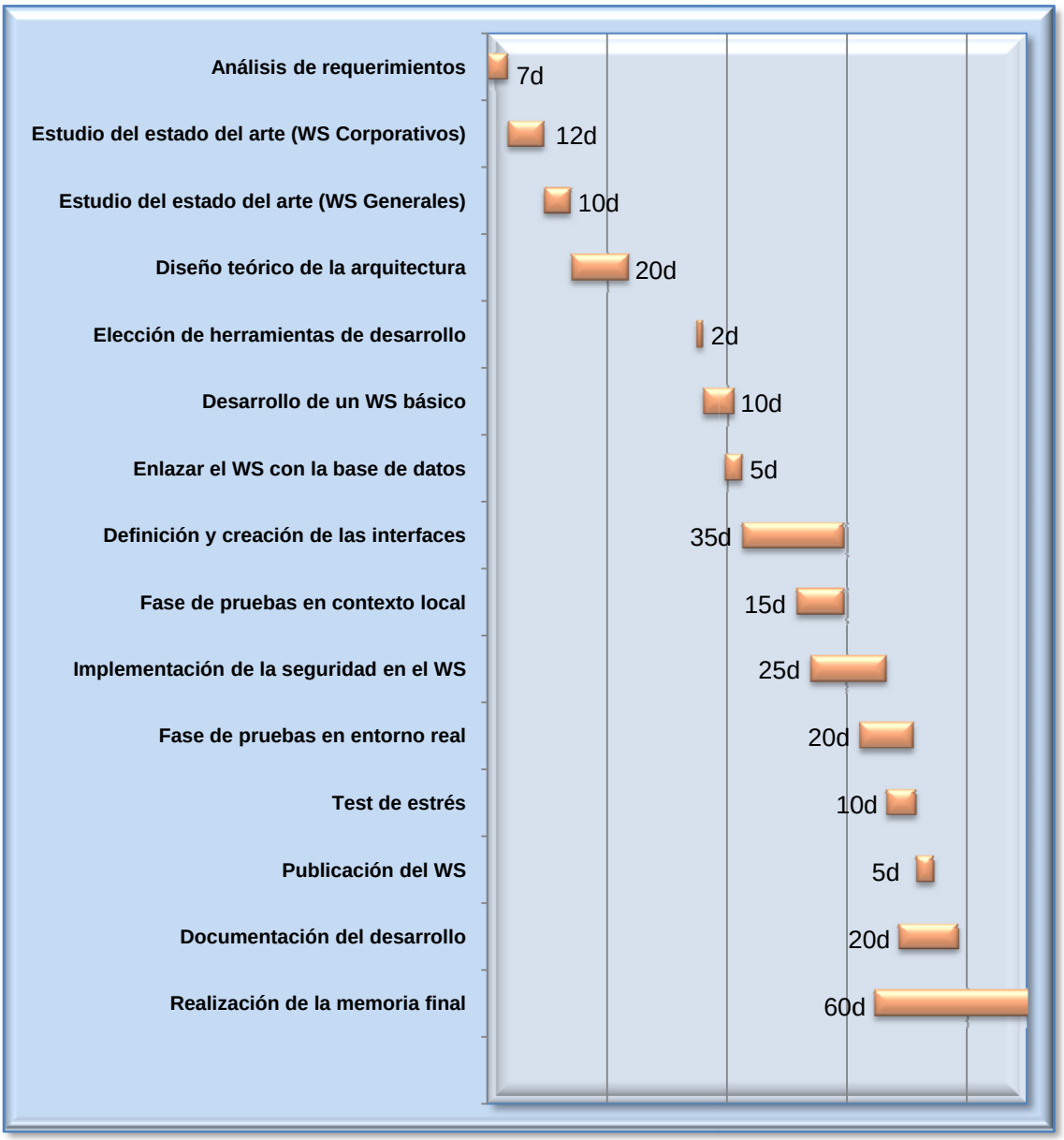

Figura 2-1 - Diagrama de Gantt del proyecto

1.4 Organización de la memoria

La memoria de este proyecto está dividida en cinco capítulos organizados de la siguiente manera:

- Capítulo 2: Entorno, herramientas y alternativas. Se explica el motivo por el cual se han elegido WS para solucionar los requisitos planteados, así como otras alternativas que podíamos haber escogido. Este capítulo también incluye un repaso sobre el estado del arte de esta tecnología, así como un estudio de viabilidad incluyendo las herramientas que usaremos en el desarrollo.

- Capítulo 3: Análisis. En este capítulo se explica qué es un web service y por qué se han elegido web services para desarrollar este proyecto. También de describe la arquitectura del sistema haciendo hincapié en los detalles en los cuales se han tenido que tomar decisiones complicadas. También se hace un repaso de en qué consiste lógica de negocio de la aplicación y en qué lugar del sistema estará situada.

- Capítulo 4: Descripción técnica. Contiene la descripción de la parte técnica del proyecto. También se hace un recorrido por todos los protocolos y estándares que han sido necesarios utilizar a lo largo del desarrollo de todo el proyecto.

- Capítulo 5: Desarrollo. En este capítulo se expone de qué manera hemos usado los protocolos y estándares descritos en el capítulo anterior para realizar este proyecto. También se hace énfasis en las características básicas del proyecto, así como en los puntos clave que le hace diferenciarse de otros proyectos similares.

- Capítulo 6: Pruebas. Se explica las pruebas que se han llevado a cabo, y los resultados finales obtenidos. También daremos un repaso general de cómo debería ser una aplicación cliente viendo una visión general del funcionamiento de la *Plataforma de Multitarificación.*

- Capítulo 7: Conclusiones. En este capítulo se revisan los objetivos y la planificación que nos habíamos propuesto al principio del proyecto, y se valora si se han llegado a conseguir. Además, también se comentan las posibles futuras vías de desarrollo para el sistema desarrollado en este proyecto.

Capítulo 2: Entorno, herramientas y alternativas

2.1 Definición de web service

El término web service describe un método de integración de aplicaciones de tipo web usando los estándares XML (eXtensible Markup Language), SOAP (Simple Object Acces Protocol) y WSDL (Web Services Description Languge) sobre Internet. XML se usa para etiquetar los datos, SOAP para transferirlos y WSDL sirve para describir los servicios de que dispone.

Los web services se usan básicamente como método de comunicación entre empresas o entre empresas y clientes. Permiten intercambiar datos entre diferentes organizaciones sin la necesidad de tener un conocimiento recíproco profundo de cómo están implementados los sistemas informáticos de cada una de ellas.

A diferencia de los modelos tradicionales de tipo cliente/servidor, los web services no proporcionan a los usuarios una GUI (Graphic User Interface). En lugar de esto los web services permiten compartir lógica de negocio, datos y otros procesos mediante una interfaz que actúa a través de la red. Los desarrolladores pueden, si lo desean, añadir el web service a una interfaz gráfica como una página web o un programa ejecutable para ofrecer su funcionalidad a los usuarios finales.

Los web services permiten que diferentes aplicaciones que provienen de diferentes fuentes se comuniquen entre ellas sin necesidad de desarrollar código hecho a medida debido a que toda la comunicación se lleva a cabo con

XML. La concepción básica no está vinculada a ningún sistema operativo o lenguaje de programación. Por ejemplo, un web service desarrollado en Java podría ser consumido desde C++ (en la sección 2.3 veremos cómo esto no siempre es así), así como una aplicación Windows podría comunicarse con sistemas UNIX. Los web services tampoco requieren el uso de navegadores o código HTML (HyperText Markup Language).

2.2 Razón de la elección de web services

Para desarrollar nuestro proyecto se ha procedido a realizar de manera previa un estudio de las tecnologías y técnicas disponibles en la actualidad que nos permitan solucionar los requisitos expuestos para su realización. Después de descartar muchas de estas tecnologías por su falta de seguridad o por su elevada complejidad se ha llegado a la conclusión de que los web services son la mejor alternativa posible ya que posee diversas posibilidades respecto a la integración de la seguridad, así como una complejidad mínima en el tratamiento de los datos. Esto es ideal para una comunicación ágil y eficaz, necesaria en el entorno corporativo en que nos moveremos, en el que tendremos picos de tráfico de docenas de transacciones por minuto.

Pese a que se fueron descartando distintas tecnologías, en la recta final todavía teníamos en consideración un par de ellas, como son por una parte web services y por otra CORBA (Common Object Request Broker Architecture). CORBA es un estándar usado para el procesamiento de objetos distribuidos. Básicamente su objetivo es crear un paquete con código de un lenguaje de programación en concreto, en el que además se añade información referente a sus capacidades y a la forma de llamar a los métodos que contiene. De esta manera ese código puede ser ejecutado por otra máquina desde un programa no necesariamente escrito en el mismo lenguaje. Además, CORBA también está diseñado para ser independiente del sistema operativo. El descarte de CORBA como tecnología para desarrollar nuestro proyecto fue su elevada complejidad, necesaria para otro tipo de aplicaciones, pero no para la nuestra, ya que CORBA está considerado como middleware, y esto nos podría producir

algún problema. El uso de CORBA está mayoritariamente orientado al comercio on-line, banca y finanzas.

Por tanto, la elección de web services se basa en dos aspectos fundamentales para cumplir con éxito los objetivos del proyecto como son la facilidad y eficacia de uso, y la seguridad necesaria para no comprometer datos críticos, asegurando que éstos lleguen al otro extremo de manera totalmente privada. Cabe destacar que la experiencia obtenida con *Plataforma de Multitarificación* también ha hecho decantar la balanza hacia los WS ya que un elevado número de aseguradoras con las que trabajamos también usan esta tecnología, algunas de manera altamente satisfactoria y otra no tanto. Finalmente la fuerte auge que los web services están experimentando en la actualidad y la inmensa documentación sobre esta materia ha hecho decantarnos por ellos como la tecnología en que sustentará el desarrollo de nuestro proyecto.

2.3 Estado del arte

Como ya se ha comentado anteriormente este proyecto se ha realizado en una consultoría informática muy especializada, esto ha permitido adquirir una gran experiencia respecto a los web services y otras tecnologías similares en el mundo corporativo y sobretodo asegurador. Se ha podido observar una gran disparidad de opciones en el uso de este sistema de comunicación y algunos otros, cosa que ha dificultado significativamente la integración de diversas de estas compañías en un sistema global como la *Plataforma de Multitarificación*.

Actualmente podemos encontrar varios escenarios cuando hablamos de WS corporativos. Existen ciertas entidades que hacen uso del término web services erróneamente ya que la tecnología de la que realmente están usando es un simple "*http post*". Aunque en principio los web services no están atados a ningún lenguaje de programación en particular, existen algunas ampliaciones que sí lo son. De esta manera nos encontramos compañías que dificultan el desarrollo de la aplicación cliente destinada a consumir su web service haciéndolo dependiente de software. Este sería el caso, por ejemplo, de los

web services que usan ciertos tipos de datos adjuntos en el intercambio de información, ya que según su tipo es necesario usar obligatoriamente ciertas herramientas de desarrollo concretas ya que sólo es soportado por estas.

La seguridad también es un asunto en el que de nuevo podemos observar gran disparidad de opciones. Existen compañías cuyos web services se basan únicamente en una autenticación de tipo usuario y contraseña (incrustados en los datos a intercambiar) para dar acceso a su funcionalidad, con el riesgo que esto conlleva de una posible intercepción del envío por parte de un tercero si no usamos cifrado, pudiendo obtener información crítica como números de cuentas bancarias o tarjetas de crédito. Muchas entidades, sin embargo, usan un certificado para autentificar al consumidor del web service además de hacer uso del protocolo SSL (*Secure Sockets Layer*) para garantizar que la comunicación se produce con total garantía de privacidad. Otro nivel de seguridad vendría dado por la necesidad de firmar electrónicamente algunos (o todos) los datos a enviar.

El intercambio de datos en un WS va claramente asociado a XML (*eXtensible Markup Language)*. XML es un lenguaje de marcas que nos permite intercambiar con gran facilidad información estructurada entre diferentes plataformas, de manera que cualquier dato necesario para realizar una operación determinada viajará dentro de un documento XML. También existe una alternativa para transmitir los datos en un web service como son los *attachments* o datos adjuntos, esta tecnología se suele utilizar normalmente en casos en los que hay que enviar grandes cantidades de datos, como pueden ser documentos de gran tamaño o imágenes.

2.4 Estudio de viabilidad

La realización de este proyecto supone un paso adelante respecto al uso de los web services en el ámbito corporativo, ya que la inexistencia de un formato estándar de comunicación e intercambio de información provoca que cada compañía realice su propia arquitectura, dificultando así una integración global

de varios de estos servicios en una misma aplicación como la *Plataforma de Multitarificación*.

En el sector asegurador estamos viendo como el mercado on-line crece día a día. Podemos observar, por ejemplo, que gran parte de las compañías de seguros ofrecen en su web un servicio de tarificación de seguros para diversos ramos o productos. Este sistema obliga al cliente a introducir sus datos en cada una de las compañías en las que quiera consultar precios, por este motivo están surgiendo motores de multitarificación que facilitan en gran medida la comparación entre varias aseguradoras sin necesidad de introducir los mismos datos una y otra vez. Por tanto, si las compañías de seguros desean ser competitivas en el mercado on-line deben disponer de un sistema ágil y eficaz de comunicación con estos motores. Es aquí donde los web services pueden suponer una gran ayuda para abordar esta situación, de modo que se postulan como la solución de futuro ideal para llevar a cabo la comunicación entre aseguradores y motores de multitarificación.

Uno de los aspectos importantes en el desarrollo de este proyecto es que el web service resultante pueda ser consumido desde cualquier plataforma y por tanto no sea dependiente de *software*. Aunque en principio este es un aspecto general de los web services existen ciertas restricciones con algunos entornos de desarrollo, sería el caso, por ejemplo, de los tipos de datos adjuntos (o attachments). Por este motivo se ha ido con especial cuidado respecto a la elección del lenguaje y entorno de desarrollo del proyecto, que ha sido estudiado detenidamente (ver sección 2.5).

Como ya se ha mencionado anteriormente, en este proyecto se trabajará con datos críticos, eso obliga a tener especial cuidado en su manipulación. Como marco regulador de esta situación encontramos la Ley Orgánica de Protección de Datos 15/1999, de 13 de diciembre, de obligado cumplimiento para las empresas que trabajen con este tipo de datos. Por tanto, en la realización de este proyecto se tendrá en cuenta este aspecto legal, de manera que se seguirán las pautas indicadas en esta Ley, siendo, además, una norma general

de uso en la empresa en la que se realizará el proyecto, ya que es común el uso de datos de alta seguridad.

Para concluir subrayaremos que este estudio de viabilidad nos permite afirmar que el proyecto se podrá realizar con éxito dentro de los plazos predispuestos en un principio, cumpliendo con los objetivos y requisitos impuestos por parte del cliente, así como con los impuestos internamente por la política de Prexon.

2.5 Herramientas de desarrollo

Después de un periodo de estudio se ha llegado a la conclusión de que las herramientas ideales para el desarrollo de nuestro proyecto son las siguientes:

- Java. Utilizaremos el lenguaje de programación Java ya que se está convirtiendo en la actualidad en el lenguaje de referencia para aplicaciones web. Además nos facilita la publicación en un servidor web libre como Apache Tomcat.
- Eclipse. Es el entorno de desarrollo de referencia en Java gracias a su sencillez y flexibilidad. Además es libre.
- WTP (Web Tools Platform). Es un proyecto que extiende la plataforma Eclipse con nuevas herramientas que nos facilitan el desarrollo de todo tipo de aplicaciones web, especialmente web services.
- Apache Tomcat. Es un servidor web libre necesario para publicar el web service. Sólo será usado para realizar pruebas.
- IBM WebSphere Application Server. Servidor web y de aplicaciones que ya utilizaba la compañía Aura previamente a la concepción de este proyecto. Por tanto, la versión final del sistema se publicará aquí.
- Microsoft SQL Server 2005. Es el sistema de gestión de base de datos que se ha venido usando en la compañía Aura y, por tanto, se usará también para este proyecto. Además permite el uso de XML de forma nativa.

La involucración de Aura en el desarrollo de nuestro proyecto es máxima, así que se dispone de servidores de última generación para albergar la base de datos, así como el web service. Esto es importante ya que en un corto espacio de tiempo se esperan recibir picos de docenas de cotizaciones por minuto. De la misma manera pueden existir instantes en los que haya dos o tres peticiones simultáneamente.

Para la implementación del código también se dispone de una estación de trabajo totalmente preparada para hacerse cargo de todo el desarrollo.

2.6 Sumario del capítulo

En este capítulo hemos visto un punto básico de este proyecto, como es el de la definición de web service, así como una argumentación de por qué se ha elegido esta tecnología para solucionar los problemas que se nos habían planteado y no otra. También se incluye los motivos por los que otras tecnologías han sido descartadas a favor de web services.

De la misma manera, se ha hecho una detallada descripción de la situación del estado del arte respecto a web services en la actualidad, del cual tenemos gran experiencia por haber estado trabajando en la *Plataforma de Multitarificación* de Prexon.

También se ha hecho referencia al estudio de viabilidad del proyecto, el cual nos indica que el proyecto se podrá realizar con éxito. Por último se hace un repaso de las herramientas de desarrollo que se usarán durante la implementación del proyecto.

Capítulo 3: Análisis

3.1 Requisitos

Los requisitos finales que se acuerdan entre Prexon y Aura en el momento de empezar nuestro proyecto son los siguientes (después se modificarían algunos y se añadirían otros):

- Disponer de un sistema escalable, fiable y seguro que permita a terceros, ya sean los agentes de la compañía, corredurías o los propios clientes finales llevar a cabo una comunicación telemática con Aura con un doble objetivo: en primer lugar, ofrecer un servicio de presupuestación de pólizas en cada uno de los productos ofrecidos regularmente por las vías tradicionales de venta. En segundo lugar, dar la posibilidad de contratar una póliza de manera real, devolviendo por parte de la compañía la documentación acreditativa de que la emisión se ha realizado exitosamente. Por supuesto para llevar a cabo esta interacción será necesaria una capa de visualización que interactúe con nuestro sistema y muestre toda la información que se obtenga por pantalla, es aquí donde la *Plataforma de Multitarificación* desarrollada por Prexon entra en funcionamiento. Pese a esto, de ninguna manera el WS estará limitado a funcionar únicamente con esta plataforma, contrariamente los datos de conexión y comunicación con nuestro WS serán informados a la mayor parte posible de estos motores de multitarificación existentes en la actualidad para así permitir una expansión comercial lo más extensa posible.

- Desarrollar este sistema de manera que su funcionalidad pueda ser incrementada en un futuro para permitir la incorporación de herramientas como el hecho de ofrecer la posibilidad de introducir partes de siniestros en el sistema, ahorrando así gran cantidad de papeleo. Otro aspecto que interesa añadir en un futuro será la consulta de pólizas por parte de agentes o corredores de la compañía.

Por otra parte, se incluye como requisito interno de Prexon el hecho de que este sistema no sea un desarrollo específico para Aura Seguros sino que interesa reutilizarlo en otros proyectos, ya no sobre aspectos únicamente de seguros, sino más bien sobre encargos corporativos en general.

Es importante destacar de nuevo que el producto que resulte de este proyecto, necesitará una aplicación cliente para poder aprovechar su funcionalidad y visualizar sus resultados. Está aplicación no queda dentro de los objetivos del proyecto aunque sí que veremos algún aspecto básico de cómo debería funcionar, así como el ejemplo de la *Plataforma de Multitarificación*.

3.2 Descripción de la arquitectura

Después de un cierto periodo de tiempo como desarrollador de la *Plataforma de Multitarificación* hemos podido observar diversos tipos de aplicaciones con diferentes arquitecturas, de las cuales todas tenían el mismo objetivo. También hemos podido evaluar cuales de ellas son más estables, fiables, seguras y sencillas de usar. Esto nos ha servido para coger lo mejor de cada una de estas arquitecturas con el objetivo de desarrollar un sistema que cumpla con los requisitos de cliente.

La primera cuestión que hay que tener en cuenta es el hecho de desarrollar un solo web service que dé cabida a todas las operaciones para todos los productos que ofrece la compañía, o bien, hacer un web service diferente para cada una de estas operaciones. Aplicar la lógica directa nos sugeriría que ¿para qué vamos a hacer varios web services si en uno podemos dar cabida a todas las operaciones que necesitamos simplemente creando diferentes

interfaces? Pues bien, el dilema surge cuando pensamos en el hecho de que existirán diversos usuarios que puedan consumir nuestro web service, es decir, puede haber un corredor al que sólo le permitamos hacer presupuestos de pólizas pero no contrataciones, o un agente que sólo trabaje con el producto de decesos, por ejemplo. Por este motivo necesitamos algún sistema de autorización que nos permita diferenciar quien está intentando hacer una petición, de modo que según nuestra política interna decidamos darle respuesta a su solicitud o no.

De este modo, disponemos de dos posibilidades, la primera, como ya se ha mencionado, sería incluir todas las operaciones en el mismo web service de manera que hagamos la comprobación de las credenciales del usuario que haga la solicitud dentro del código del web service. La segunda, contrariamente consistiría en ofrecer diferentes web services para cada tipo de operación y/o producto que ofrezcamos dando a los clientes diferentes URLs para cada uno de ellos.

Está claro que este último método puede funcionar para dos o tres operaciones diferentes, pero en el caso de tener más se podría convertir en un caos, con lo que conllevaría mucha complejidad a las aplicaciones clientes que consuman nuestro web service. En nuestro caso concreto trataremos con diversos tipos de operaciones, ya no sólo hacer una tarificación o una emisión de una póliza, sino también debemos ofrecer otros métodos necesarios para ciertos aspectos concretos del producto con el que estemos tratando. Éste sería el caso, por ejemplo, de los seguros de automóviles, en que debemos ofrecer códigos internos que permitan a las aplicaciones clientes hablar el mismo idioma que nosotros. Es decir, si quieren decirnos que el coche que se está intentando asegurar tiene un dispositivo GPS que queremos declarar en la póliza, es necesario que nosotros entendamos lo que es. Si nos pasaran directamente que tiene un accesorio llamado GPS, podríamos malinterpretarlo porque igual nosotros en vez de llamarlo GPS lo llamamos dispositivo de localización. Por tanto, una de las interfaces que ofreceremos en nuestro web service será una lista de accesorios con un código, el cual será el que deberán especificar en su solicitud.

La solución a la que llegamos es finalmente presentar de manera única un solo web service que ofrezca una interfaz para cada una de las operaciones que haya disponibles. Para llevar a cabo el aspecto de la autorización de usuarios usaremos un sistema de credenciales incrustadas en el documento XML de la solicitud, que al viajar cifrado por SSL no supondrá ningún riesgo. De todas maneras, también será necesario disponer de un certificado en el lado del cliente para comprobar su identidad, de modo que estas credenciales sólo supondrán un método de autorización pero de ninguna manera de autenticación.

De este modo guardaremos en la base de datos la lista de agentes, corredores o cualquier usuario que este autorizado a consumir nuestro web service, junto a las operaciones que les está permitido hacer. Así se consultará si las credenciales que nos han hecho llegar le permiten acceder a la petición que han hecho, y si no es así se les devolverá una respuesta indicándoles que no están autorizados a ejecutar la operación en concreto.

En el caso en que la operación que nos ha sido solicitada sea una contratación de una póliza estaremos obligados a retornar unos documentos acreditativos de que efectivamente la póliza ha sido emitida y está en vigor, así como los documentos informativos de las condiciones generales y particulares del seguro. Para hacer llegar a los clientes estos documentos utilizaremos los propios XML's de que enviamos en las respuestas, de manera que incluiremos los documentos en formato PDF y codificados en base64 para que puedan viajar incrustados en el XML de manera simple y efectiva. Además, será necesario firmar de manera digital algunos de estos documentos que contengan la información esencial, como es el documento acreditativo de la emisión, el cual demuestra que se ha contratado la póliza.

En los capítulos posteriores se entrará más profundamente en este aspecto, pero es interesante comentar que existirán cinco tipos de esquemas del documento XML que usaremos para transmitir la información. Por una banda tenemos el esquema de solicitud de cotización, que será el mismo para todos los productos. Esto facilita mucho tanto al consumidor del web service como a nosotros el procesamiento de los datos. Por el mismo motivo el esquema de

respuesta también será el mismo para todos los productos. Del mismo modo, usaremos dos esquemas más para la emisión de la póliza (solicitud y respuesta). Por último, existirá un último tipo de esquema que usaremos para devolver las respuestas a las peticiones de listas de valores, como la del ejemplo de los accesorios de automóviles.

Finalmente, la arquitectura de nuestro sistema quedará de la siguiente manera: tendremos un web service que ofrecerá diversas interfaces, cada una de las cuales ejecutará una operación diferente. Cuando llegue una petición, el documento XML de la solicitud se insertará directamente en la base de datos que mediante un trigger comezará a hacer los cálculos necesarios para devolver el resultado de la solicitud, o bien, generará un error con la descripción del problema que se haya podido producir. Esta respuesta será recogida de nuevo por la capa Java que la enviará a través del web service al solicitante. Para implementar la interacción con la base de datos se usará la novedosa tecnología JPA (Java Persistance API) (ver sección 4.6). El uso de esta tecnología no estaba pensado en el momento de empezar la realización del proyecto, pero se integró a la mitad del desarrollo ya que ofrece muchas posibilidades para facilitar la independencia entre la capa Java y la capa de base de datos. Gracias a esto, uno de los objetivos principales de este proyecto, como es el de hacer un sistema reutilitzable, se puede asegurar todavía en más medida, ya que incluso, gracias a JPA, es posible ignorar qué tipo de sistema de gestión de base de datos se está usando, debido a que su funcionamiento permite trabajar con distintas bases de datos sin necesidad de retocar el código con las peculiaridades de cada uno de estos sistemas.

3.3 Lógica de negocio

En el momento inicial del planteamiento de la arquitectura que utilizaremos en el desarrollo del proyecto se abren varios caminos acerca de dónde situar la lógica de negocio del sistema. Esta lógica contiene, entre otras cosas, la normativa de contratación de la compañía. Por ejemplo, el hecho de que a un menor de 25 no le esté permitida la contratación seguro de automóviles o que

no se le haga un seguro de decesos a una persona mayor de 65 años. También será necesario hacer las comprobaciones rutinarias de consistencia en los datos, por ejemplo, la comprobación de que una cuenta bancaria es correcta o, que si se trata de un seguro de automóviles, la matrícula no corresponda a una fecha anterior a la de fabricación.

De modo que se nos abren dos caminos, por una parte tenemos la posibilidad de situar la lógica de negocio directamente en la base de datos. En este supuesto el desarrollo sería más orientado al nivel de datos y la capa Java se utilizaría únicamente como simple interlocutor de comunicaciones, es decir, la capa Java consistiría en un WS simple que recoja los datos que le hagan llegar y llame directamente a un procedimiento almacenado de la base de datos. Desde ahí se llevarían a cabo todas las comprobaciones correspondientes a la normativa de contratación y se validaría la consistencia de los datos de la solicitud.

Por otra parte, toda esta lógica de negocio se podría situar en la capa Java, relegando a la base de datos a un simple almacén de la información recibida y transmitida. De modo que ésta trabaje con mucha menos carga de trabajo y éste recaiga sobre la capa Java que se dividiría en la parte de comunicación en la que estaría implementado el WS y en la parte de la lógica de negocio.

Finalmente se decide hacer una mezcla entre las dos posibles vías de implementar el sistema, pese a que la mayoría de la lógica de negocio recaerá sobre la base de datos. Aunque es cierto que está decisión podría comprometer el rendimiento global del sistema, es necesaria para poder cumplir exitosamente el requisito interno de poder reutilizar el desarrollo en el futuro para futuros proyectos de ámbito corporativo que le puedan surgir a Prexon. Incluir esta lógica en el núcleo de la aplicación Java no nos permitiría reaprovechar el sistema ya que tendríamos que hacer un cambio radical de su código cada vez que queramos usarlo para desarrollar un sistema diferente.

Por tanto, en la capa Java situaremos toda la lógica posible que sea transversal para cualquier tipo de aplicación corporativa, es decir, comprobaciones de la consistencia de los datos como NIF’s, cuentas bancarias, códigos postales, etc, y dejaremos la parte de normativa de contratación en la base de datos. Como

uno de los objetivos principales es que el funcionamiento sea ágil se usará un servidor de última generación para albergar todo el sistema, además, se usará XML (eXtensible Markup Language) como el método de representación de datos nativo. Esto nos permitirá disminuir en gran medida la carga de trabajo de la base de datos ya que se utilizará Microsoft SQL Server 2005, el cual se caracteriza por el soporte que ofrece para este formato, de manera que podremos trabajar directamente con sentencias que actúen contra documentos XML en lugar de tablas.

3.4 Esquema de la arquitectura

La figura 3-1 muestra el esquema de la arquitectura del sistema:

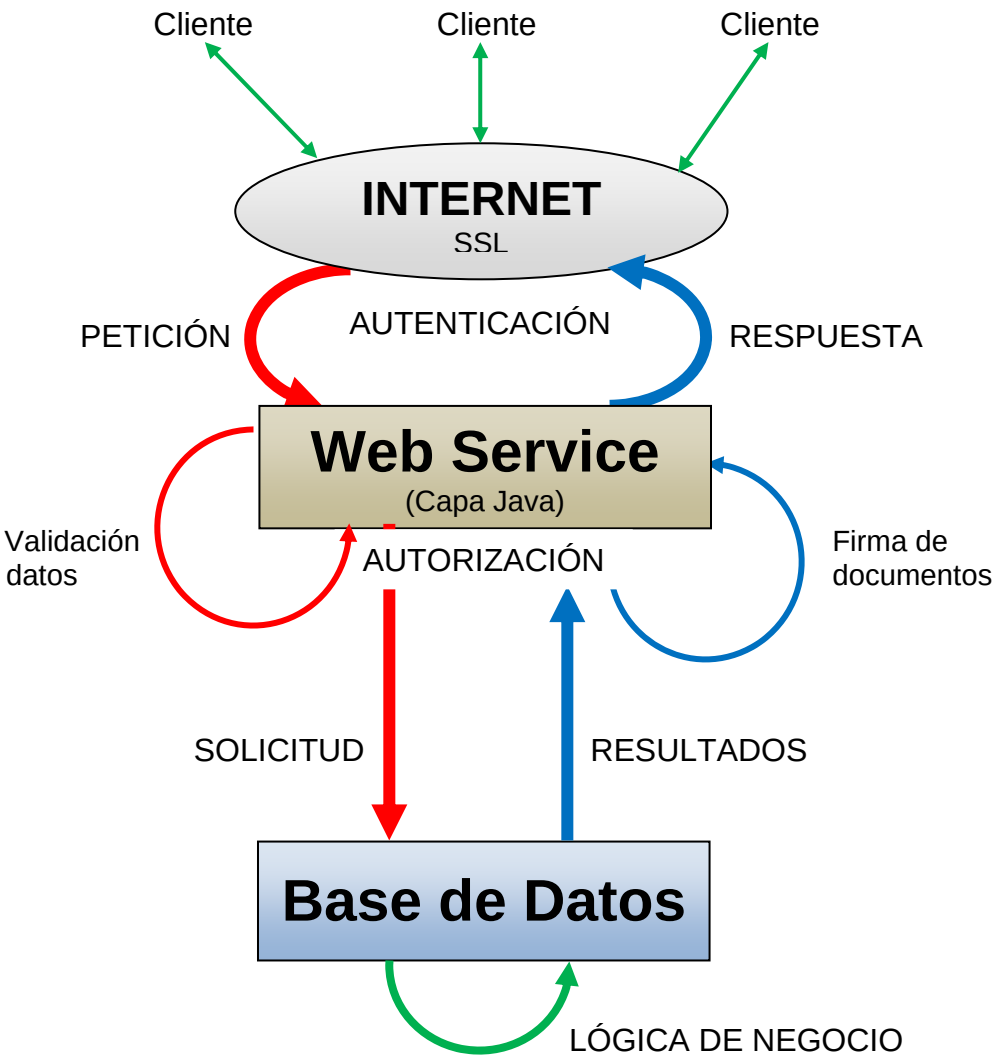

Figura 3-1 – Esquema de la arquitectura del sistema

3.5 Seguridad

La seguridad también es un aspecto básico de este proyecto. Como ya se ha mencionado, los datos de alta seguridad serán de uso habitual dentro del flujo de comunicación entre compañía y clientes. Datos tales como cuentas bancarias, números de tarjetas de crédito, etc. serán usados habitualmente en el momento de la contratación de las pólizas.

Para asegurar que estos datos críticos viajan a través de Internet de forma segura nuestro web service sólo será consumible mediante SSL. Esto nos asegurará privacidad en el envío y recepción de los datos. Además, será necesario, por parte de los clientes que se quieran conectar a nuestro web service, usar un certificado generado por una entidad de certificación acreditada que les autentique, de manera que nosotros nos aseguremos de con quién estamos intercambiando información. Ellos nos pasarán la clave pública de este certificado que nosotros introduciremos en una lista blanca. De este modo, nosotros podremos identificar la entidad o persona que se está intentando comunicar con nosotros en cada momento y permitirle o negarle el acceso según sea el caso.

Independientemente de esto, con el objetivo de afianzar la seguridad global del sistema, se habilitarán reglas especiales en el firewall hardware del que dispone Aura. De esta manera nuestros posibles clientes deberán indicarnos previamente a la comunicación telemática desde que IP se van a conectar de modo que abriremos una regla en el firewall que permita la entrada de datos a nuestro sistema únicamente para esa IP. Por supuesto esto sólo lo deberán hacer cada vez que cambien la IP desde la que se conectar a nuestro web service.

3.6 Sumario del capítulo

En este capítulo hemos hecho un análisis pormenorizado de cada uno de los aspectos clave del proyecto. En primer lugar hemos mencionado cual son los requisitos que tiene nuestra aplicación. En segundo lugar, hemos hecho una

descripción detallada de cómo será la arquitectura del sistema, así como una explicación de en qué consiste la lógica de negocio y qué papel ocupará en el desarrollo global. Por último hemos hablado de un aspecto tan importante como la seguridad y de los métodos que usaremos para garantizarla, como el uso de la tecnología SSL y certificados.

Capítulo 4: Descripción técnica

4.1 Introducción

Para desarrollar este proyecto se han usado diversos protocolos y estándares que definen el funcionamiento general de los web services y de nuestro sistema en particular. En este capítulo veremos una descripción superficial de estos protocolos y estándares, lo suficiente para que los no iniciados puedan tener una visión general del proyecto. Seguidamente, en el siguiente capítulo, veremos la manera específica en que los hemos usado e interrelacionado para obtener el resultado final de este proyecto.

4.2 XML

4.2.1 Introducción

XML (eXtensible Markup Language) es uno de los aspectos básicos en el proyecto ya que se usa en todos los niveles. Es nuestro modo de representar la información, tanto cuando trabajamos contra la base de datos como cuando intercambiamos datos con el cliente.

Se trata de un metalenguaje extensible de etiquetas que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Usa un documento DTD (*Document Type Definition*) o un XSD (XML *Schema Definition)* (ver sección 4.3) para definir la estructura de los datos que contiene. La flexibilidad que proporciona XML es uno de sus puntos

fuertes ya que a diferencia de otros lenguajes parecidos como HTML (*Hypertext Markup Language)*, las etiquetas usadas no están prefijadas sino que es el autor el que define sus propias etiquetas y su propia estructura. Es importante destacar que XML por sí sólo no tiene ninguna funcionalidad ya que ha sido diseñado para almacenar, transportar e intercambiar datos.

4.2.2 Usos

En la actualidad muchos sistemas informáticos y bases de datos contienen información en diversos formatos los cuales son incompatibles entre ellos. El intercambio de estos datos entre aplicaciones supone un gran desafío para los programadores. Gracias a XML esta complejidad se reduce y nos permite crear documentos legibles por cualquier tipo de aplicación, ofreciendo un medio de intercambio independiente tanto de software como de hardware. Además, cada vez más aplicaciones hacen uso de XML para almacenar información en ficheros o directamente en bases de datos.

4.2.3 Sintaxis

Las reglas de sintaxis de XML son simples y fáciles de usar pero muy estrictas, así que el desarrollo de software que pueda leer y manipular XML es significativamente asequible.

En el ejemplo siguiente podemos ver un documento sencillo en el cual observamos que XML es un lenguaje que se auto-describe:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<automovil>
        <marca>Audi</marca>
        <modelo>A4</modelo>
        <versión>1.8T (150 CV)</versión>
        <matricula>3213CRD</matricula>
</automovil>
```

En la primera línea de este ejemplo definimos la versión y la codificación usada en el documento. En la siguiente línea encontramos el elemento raíz

<*automovil*> el cual nos indica que la información que encontraremos en el documento estará relacionada con un automóvil. Esta información se encuentra en las cuatro líneas siguientes las cuales son los elementos hijos de la raíz. Finalmente encontramos el fin del elemento raíz. Vemos por tanto, que este lenguaje es auto-descriptivo ya que a primera vista quedan bastante claras las características del automóvil en cuestión. También observamos que los documentos XML no son más que texto plano, esto hace que cualquier editor de textos sea capaz de leerlos y/o editarlos.

A continuación se describen los requisitos básicos de XML:

- Todos los elementos en un documento deben tener una etiqueta de clausura que indique su cierre. La declaración XML no tiene esta etiqueta de clausura ya que no forma parte del documento XML en sí y por tanto no es un elemento.
- Todas las etiquetas son sensibles a diferencias entre mayúsculas y minúsculas. Por tanto, las etiquetas de apertura y clausura deben estar escritas de la misma manera.
- Los elementos deben estar correctamente anidados, el último elemento en abrirse debe ser el primero en cerrarse. Ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<raiz>
  <hijo>
    <subhijo>.....</subhijo>
  </hijo>
</raiz>
```

- Todos los documentos XML deben tener un elemento raíz único en el cual estarán incluidos el resto de elementos.
- Los elementos XML pueden tener atributos en pares de nombre-valor. Estos atributos deben estar entrecomillados.
- Algunos caracteres tienen un significado especial dentro de un documento XML. Para usarlos utilizaremos referencias igual que hacemos en HTML (por ejemplo los caracteres “<”, “>” y “&” pasarán a ser “<”, “%gt” y “&” respectivamente.

- La sintaxis para escribir comentarios es la siguiente:
  <!-- Esto es un comentario -->

4.2.4 Elementos

Un elemento XML es cualquier cosa que esté entre la etiqueta de principio de elemento y la etiqueta de final (ambas incluidas). Un elemento puede contener otros elementos, texto simple o una mezcla de ambos, y aunque se puede usar cualquier nombre para un elemento, existen ciertas restricciones que se deben seguir. Son las siguientes:

- Los nombres de los elementos no pueden comenzar por números o signos de puntuación.
- Los nombres de los elementos no pueden empezar por la secuencia de letras “xml” (ya sea en mayúscula o minúscula).
- Los nombres de los elementos no pueden contener acentos.

Una importante característica de los elementos XML es que son extensibles. Añadir nuevos elementos en un documento no impide que una aplicación que no los espere funcione correctamente. Este, por tanto, es uno de los aspectos positivos de XML, ya que se puede extender sin perjudicar el funcionamiento de ninguna aplicación que esté usando ese tipo de documento en concreto.

4.2.5 Atributos

Los atributos pueden proporcionar información adicional sobre los elementos que los contienen. Esta información no suele formar parte de los datos en sí mismos, sino que se trata más bien de una indicación para las aplicaciones que quieran tratar ese documento XML.

En cambio, hay ocasiones en que un atributo se puede usar para representar la misma información con la que normalmente usaríamos un elemento. Aunque no hay reglas sobre cuando usar atributos y cuando elementos, ya que se permite completamente el uso de atributos para representar cualquier tipo de

información, la recomendación es usar siempre que nos sea posible elementos en lugar de atributos. En el siguiente ejemplo podemos observar esta situación:

```
<vehículo tipo=”motocicleta”>
        <marca>Kawasaki</marca>
        <modelo>Ninja</modelo>
</vehículo>
```

```
<vehículo>
        <tipo>motocicleta</tipo>
        <marca>Kawasaki</marca>
        <modelo>Ninja</modelo>
</vehículo>
```

Por tanto, es recomendable evitar cuando nos sea posible los atributos, ya que éstos no pueden contener múltiples valores ni estructuras en forma de árbol como los elementos. Además, los atributos no son tan fácilmente expandibles como los son los elementos. A continuación podemos observar un ejemplo de lo que no deberíamos hacer con el uso de atributos:

```
<mensaje id=”250” dia=”10” mes=”05” año=”2008”
de=”Alberto” para=”Juan”
titulo=”Hola” cuerpo=”Qué tal?”>
</mensaje>
```

Y a continuación vemos esa misma información representada de manera adecuada.

```
<mensaje id=”250”>
        <fecha>
                <dia>10</dia>
                <mes>05</mes>
                <año>2008</año>
        </fecha>
        <de>Alberto</de>
        <para>Juan</para>
        <titulo>Hola</titulo>
        <cuerpo>Qué tal?</cuerpo>
</mensaje>
```

Como vemos, sólo usamos un atributo para representar el número de identificador del mensaje de ejemplo que usamos en este caso. Este identificador no forma parte de la información que queremos representar sino de una información adicional y anexa. Por tanto, lo recomendado será usar los atributos para almacenar los datos sobre los datos (metadatos) y los elementos para los datos en sí mismos.

4.2.6 Sección CDATA

Normalmente en un documento XML el texto es procesado por el *parser*, pero este será ignorado si se encuentra dentro de una sección CDATA. Cuando un elemento se procesa, el texto entre las etiquetas también es procesado. Esto es debido a que un elemento puede contener otros elementos en su interior y por tanto el *parser* necesita tratar el texto que hay dentro del elemento para comprobar la existencia de sub-elementos. Este tipo de datos son los llamados PCDATA (Parsed Character Data).

El término CDATA ((Unparsed) Character Data), por tanto, se usa para indicar un fragmento de texto que no se procesará por el *parser*. Esto es especialmente útil si queremos almacenar en el documento XML algún tipo de código, por ejemplo, ya que este puede contener caracteres como “<” o “&” que el *parser* interpretaría de forma errónea. En el siguiente ejemplo podemos ver como usaríamos la sección CDATA:

```
<script>
      <![CDATA[
        function factorial(a)
        {
            if (a < 2) then
            {
                return 1;
            }
            else
            {
                return a * factorial(a-1);
            }
        }
      ]]>
</script>
```

4.2.7 Namespaces

En XML los nombres de los elementos están definidos manualmente por el diseñador. Esto suele acabar en conflicto en el momento que nos interesa mezclar documentos XML de diferentes aplicaciones. Los *namespaces* o espacios de nombres nos proporcionan un método para evitar conflictos de nombre entre elementos.

Este problema se puede solucionar de manera sencilla usando prefijos para diferenciar las diferentes interpretaciones que queramos darle a los elementos con el mismo nombre. Cuando usamos estos prefijos en XML se debe definir un espacio de nombres para cada uno del tipo: *xmlns:prefix="URI"*. Además, en el momento que definamos uno de estos prefijos en un elemento, todos sus hijos con el mismo prefijo también quedarán asociados con el mismo espacio de nombres. Los espacios de nombres se pueden crear directamente en los elementos que van a hacer uso de ellos o directamente en la raíz del documento XML. A continuación podemos ver un ejemplo del uso de los espacios de nombres:

```
<raiz xmlns:c="http://www.prexon.com/Coches"
      xmlns:l="http://www.prexon.com/Libros">

      <c:producto>
         <c:marca>Audi</c:marca>
         <c:modelo>A4</c:modelo>
      </c:producto>

      <l:producto>
         <l:nombre>Un mundo sin fin</l:nombre>
         <l:autor>Ken Follet</l:autor>
      </l:producto>

</raiz>
```

Como podemos ver el objetivo es dar a los nombres de espacios un nombre único, aunque en ocasiones algunas compañías lo usan para indicar una página web que contenga información sobre este espacio de nombres. En el caso de que definamos el espacio de nombres directamente en cada uno de los elementos no será necesario repetir el prefijo en cada hijo de ese elemento.

4.3 XSD

4.3.1 Introducción

Para poder enviar los datos desde el cliente hacia nosotros o a la inversa, es esencial que ambos esperamos la misma estructura en el contenido de los datos que nos van a llegar. Esto lo llevamos a cabo mediante los esquemas XSD (XML *Schema Definition*) que nos permiten describir la estructura de un documento XML, de modo que el que envía los datos lo hará de una manera que el receptor pueda entender. Podemos decir que los esquemas XSD son los esqueletos de los documentos XML.

Un esquema XSD nos define los bloques con los que construiremos un documento XML de manera que nos indica:

- Qué elementos pueden aparecer en el documento XML.
- Qué atributos pueden aparecer en el documento XML.
- Qué elementos son elementos hijo.
- El orden de los elementos hijo.
- El número de los elemento hijo.
- Si un elemento está vacío o puede contener texto.
- Los tipos de datos para los elementos y atributos.
- Valores fijos y/o por defecto para los elementos y atributos.

Otras características importantes de los esquemas XSD es que son fácilmente extensibles, están escritos en XML y son tipados. Esto nos permite utilizar esquemas dentro de otros esquemas, crear nuestros propios tipos de datos derivados de los tipos estándar y referenciar múltiples esquemas en el mismo documento.

4.3.2 Sintaxis

Para que un esquema sea correcto y no contenga errores el primer paso es asegurarnos de que está bien formado. Las reglas básicas para comprobar esto son:

- Debe empezar con una declaración XML.
- Sólo puede tener un único elemento raíz.
- Las etiquetas de comienzo deben tener etiquetas de finalización.
- Hay distinción entre mayúsculas y minúsculas en el nombre de los elementos.
- Los elementos deben estar anidados correctamente.
- Los valores de los atributos deben estar entrecomillados.

A continuación podemos ver un esquema XSD que nos define los elementos de un documento XML:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.prexon.es"
           xmlns="http://www.prexon.es"
           elementFormDefault="qualified">

      <xs:element name="tomador">
          <xs:complexType>
            <xs:sequence>
                <xs:element name="nombre" type="xs:string"/>
                <xs:element name="apellidos" type="xs:string"/>
                <xs:element name="direccion" type="xs:string"/>
                <xs:element name="poblacion" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
      </xs:element>

</xs:schema>
```

En este ejemplo podemos ver el uso de *complexType,* usado para indicar que un elemento contiene hijos. El resto de elementos son de tipo simple ya que no contienen otros elementos. El elemento <*schema*> es siempre la raíz de cualquier esquema XSD y puede contener diversos atributos con información sobre espacios de nombres, etc. A continuación vamos a ver una breve descripción de los atributos que usamos en la raíz del esquema:

- **`xmlns:xs=”http://www.w3.org/2001/XMLSchema”`**
  Indicamos que los tipos de datos usados en el esquema provienen de *http://www.w3.org/2001/XMLSchema*.

- **targetNamespace="http://www.prexon.es"**
  Indicamos que los elementos definidos en este esquema provienen de http://www.prexon.es.
- **xmlns="http://www.prexon.es"**
  Indicamos que el espacio de nombres por defecto es http://www.prexon.es.
- **elementFormDefault="qualified"**
  Indicamos que cualquier elemento que se use por un documento XML y que esté declarado en este esquema debe ser asignado a un espacio de nombres.

En el siguiente ejemplo observamos cómo el documento XML correspondiente hace referencia a su esquema XSD:

```
<?xml version="1.0"?>
<tomador xmlns="http://www.prexon.es"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.prexon.es mensaje.xsd">

        <nombre>Alberto</nombre>
        <apellidos>Gómez Pérez</apellidos>
        <direccion>Avda. Mediterraneo 3, 4 A</direccion>
        <poblacion>Badalona</poblacion>

</tomador>
```

Vemos como también tenemos atributos en el elemento raíz del documento, su significado es el siguiente:

- **xmlns="http://www.prexon.es"**
  Indicamos que el espacio de nombres por defecto, en este caso http://www.prexon.es. Esta declaración señala que todos los elementos usados en este documento XML están declarados en el espacio de nombres http://www.prexon.es.
- **xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**
  Definimos la instancia del espacio de nombres del esquema XSD.

- **`xsi:schemaLocation="http://www.prexon.es mensaje.xsd"`**
  Este atributo tiene dos parámetros. El primero es el espacio de nombres que utilitzaremos. En el segundo indicamos la localización del esquema XSD que vamos a usar para ese espacio de nombres.

4.3.3 Tipos simples

Un elemento de tipo simple es un elemento XML que únicamente contiene texto. No puede contener otros elementos o atributos. Para definir uno de estos elementos se debe indicar su nombre y su tipo. También podemos especificar los valores por defecto o fijos de los elementos. Vamos a ver algunos ejemplos:

```
<marca>Audi</marca>
<antigüedad>4</antigüedad>
<fecha_matriculacion>2004-04-27</fecha_matriculacion>
<color>blanco</color>
<tipo>coche</tipo>
```

```
<xs:element name="marca" type="xs:string"/>
<xs:element name="antigüedad" type="xs:integer"/>
<xs:element name="fecha_matriculacion" type="xs:date"/>
<xs:element name="color" type="xs:string" default=”blanco”/>
<xs:element name="tipo" type="xs:string" fixed=”coche”/>
```

Todos los atributos son definidos como tipos simples, así que su declaración es similar a la de los elementos:

```
<marca id=”23”>Audi</marca>
```

```
<xs:attribute name="id" type="xs:integer" use=”required”/>
```

Con los atributos también podemos definir valores por defecto o fijos, pero como vemos en el anterior ejemplo también podemos indicar que son obligatorios (por defecto son opcionales).

También podemos hacer uso de las llamadas restricciones, usadas para indicar los valores que son aceptados para elementos XML o atributos. Existen más de diez tipos de restricciones distintos. A continuación podemos ver un ejemplo de un par de ellas. En el primero estamos definiendo el conjunto de valores que serán aceptados en el elemento *marca*, y en el segundo decimos el rango de valores correcto para el elemento *edad*.

```
<xs:element name="marca">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Seat"/>
    <xs:enumeration value="Renault"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

```
<xs:element name="edad">
<xs:simpleType>
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="120"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

4.3.4 Tipos complejos

Un el elemento de tipo complejo contiene otros elementos y/o atributos. Disponemos de cuatro tipos de elementos complejos: los vacíos, los que contienen únicamente otros elementos, los que contienen sólo texto y los que contienen ambos elementos y texto. En el siguiente ejemplo vemos la manera en que definiremos elementos complejos:

```
<vehiculo>
        <marca>Audi</marca>
        <modelo>A4</modelo>
</vehiculo>
```

```
<xs:element name="vehiculo">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="marca" type="xs:string"/>
      <xs:element name="modelo" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

En el anterior ejemplo vemos como el elemento <*vehículo*> hace uso del tipo complejo. También podemos observar el uso del indicador <*sequence*> que nos señala que los elementos hijo deben situarse en el mismo orden en que han estado declarados.

Un elemento complejo vacío no podrá tener ningún contenido excepto por lo que se refiere a atributos. En el siguiente ejemplo vemos como habría que declarar uno de estos elementos vacíos:

```
<poliza id="2344" />
```

```
<xs:element name="poliza">
  <xs:complexType>
    <xs:attribute name="id" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

Los indicadores se usan para controlar de qué manera vamos a controlar los elementos en los documentos XML. En total hay siete indicadores divididos en tres grupos:

- Orden: estos indicadores se usan para fijar el orden en que debe aparecer cada elemento.
- Ocurrencia: indica el número mínimo y/o máximo de veces que se puede repetir un elemento.
- De grupo: se usan para definir conjuntos de atributos.

4.4 SOAP

4.4.1 Introducción

SOAP (Simple Object Acces Protocol) es el pilar en que se basa este proyecto así como cualquier web service. Como ya hemos explicado, cada vez se están invirtiendo más esfuerzos en el desarrollo de aplicaciones para optimizar la comunicación entre ellas. Hasta ahora cuando una aplicación quería comunicarse con otra de manera remota se solía usar *Remote Procedure Calls* (RPC), pese a que este sistema presenta problemas de compatibilidad y seguridad debido a que el tráfico que genera es normalmente bloqueado por *firewalls* y *proxy*s. Para mejorar esta comunicación se pensó en usar *HyperText Transfer Protocol* (HTTP) que, aunque no está diseñado para esta situación, está soportado por la gran mayoría de sistemas. De este modo nació SOAP, el cual ofrece el medio perfecto para que las aplicaciones se comuniquen entre ellas independientemente de que estén ejecutándose en diferentes sistemas operativos o estén desarrolladas con distintos lenguajes de programación.

4.4.2 Sintaxis

Un mensaje SOAP no es más que XML ordinario que contiene los siguientes elementos:

- *Envelope:* contiene la identificación del documento XML como un mensaje SOAP.

- *Header*: es un elemento opcional que contiene la información de cabecera.
- *Body*: contiene toda la información ya sea de llamada o de respuesta.
- *Fault*: contiene información de los posibles errores que hayan ocurrido en el procesamiento del mensaje. Es un elemento opcional.

Existen ciertas reglas de sintaxis que se deben cumplir cuando hablamos de mensajes SOAP:

- Debe estar en formato XML.
- Debe usar los *espacios de nombres* de *Envelope* y *Encoding*.
- No debe contener ninguna referencia a documentos DTD (*Document Type Definition*)
- No debe contener ninguna instrucción de procesamiento de XML.

En la siguiente figura se puede observar el esqueleto de un mensaje SOAP:

```
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
  ...
  ...
</soap:Header>
<soap:Body>
  ...
  ...
  <soap:Fault>
    ...
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

4.4.3 Envelope

El elemento *Envelope* debe ser la raíz del documento XML en el que éste se define como un mensaje SOAP. Dentro de este elemento encontramos la definición del espacio de nombres *xmlns:soap*, el cual debe estar siempre asociado con `"http://www.w3.org/2001/12/soap-envelope"`, de manera que si

se usa un espacio de nombres diferente la aplicación debe generar un error y descartar el mensaje. Por otra parte el atributo *encodingStyle* se usa para definir los tipos de datos usados en el documento. Este atributo puede aparecer en cualquier elemento del mensaje, de manera que se aplicará sobre los contenidos de ese elemento y sus elementos hijos. Un aspecto importante a destacar es que un mensaje SOAP no tiene ningún tipo de dato definido por defecto.

4.4.4 Header

El elemento H*eader* contiene información específica del mensaje como podría ser la autenticación. Este elemento es opcional y en el caso que exista debe ser el primer elemento hijo de *envelope*. SOAP define tres atributos en el espacio de nombres por defecto, los cuales son *actor, mustUnderstand* y *encodingStyle*. A continuación veremos una breve descripción de cada uno de ellos:

- *actor*: cuando un mensaje SOAP es enviado puede pasar por varios puntos intermedios antes de llegar al destino final. En ocasiones nos puede interesar pasar parte del mensaje a uno de estos puntos intermedios. El atributo *actor*, por tanto, se usa para dirigir el elemento *Header* a algún punto intermedio particular.
- *mustUnderstand*: este atributo se usa para indicar cuando el receptor debe procesar el elemento *Header* obligatoriamente o bien hacerlo de manera opcional.
- *encodingStyle*: como hemos explicado antes este atributo indica los tipos de datos de los que se va a hacer uso en el elemento que lo contenga (en este caso *Header*).

4.4.5 Body

El elemento *Body* (de obligada presencia) es el que contiene el mensaje que se desea hacer llegar al punto de destino final. El ejemplo siguiente consiste en un mensaje SOAP el cual solicita el precio de un libro concreto y recibe como resultado otro mensaje con la información requerida. Nótese que el hijo del elemento *Body* debe siempre contener el espacio de nombres al cual pertenece.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
   <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
      <m:Item>World without end</m:Item>
   </m:GetPrice>
</soap:Body>
</soap:Envelope>
```

Respuesta SOAP:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
   <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
      <m:Price>19.95</m:Price>
   </m:GetPriceResponse>
</soap:Body>
</soap:Envelope>
```

4.4.6 Fault

El elemento opcional *Fault* contiene los errores que se hayan podido producir y el estado del mensaje SOAP. Si este elemento está presente debe aparecer como hijo de *Body* y sólo en una ocasión. Dentro de *Fault* podemos encontrar los siguientes subelementos:

- *faultcode*: código identificatorio del error. Existen los siguientes valores para este subelemento:

    - *VersionMismatch*: el espacio de nombres del elemento *Envelope* es incorrecto.
    - *MustUnderstand*: el atributo *mustUnderstand* tiene valor 1 (procesamiento obligatorio) pero, a su vez, el elemento *Header* no se ha entendido.
    - *Client*: el mensaje no está formado correctamente o contiene información inválida.
    - *Server*: hubo un problema con el servidor y el mensaje no pudo ser procesado.
- *faultstring*: explicación del error en formato legible.
- *faultactor:* contiene información acerca de quién ha causado el error.
- *detail*: contiene información específica del error.

4.4.7 SOAP y HTTP

La transferencia de un mensaje SOAP no es más que un par petición-respuesta HTTP que cumple con las reglas de codificación de SOAP. El envío de la petición SOAP se puede realizar ya sea mediante un HTTP POST o bien con un HTTP GET. En el caso de llevar a cabo el envío a través de un HTTP POST se deben incluir de manera obligatoria las siguientes cabeceras HTTP: *Content-Type, e*n la que indicaremos que se trata un mensaje SOAP; y *Content-Length*, la cual contendrá el número de bytes que contiene el elemento *Body*.

4.4.8 Nota crítica

Como cualquier tecnología existente, SOAP tienes ciertas ventajas y ciertos inconvenientes. En la siguiente lista se pueden ver algunas de las más significativas:

- Ventajas
    - Usar SOAP sobre HTTP facilita la comunicación a través de proxys y firewalls.

- SOAP es muy versátil, de manera que puede ser usado con diferentes protocolos de comunicación. El protocolo estándar es HTTP aunque también se pueden usar otros como SMTP.

- Desventajas
  - XML usa texto raso para representar la información, esto provoca que SOAP sea considerablemente más lento que otras tecnologías como CORBA (Common Object Request Broker Architecture). Esto no es muy grave si los mensajes enviados son de reducido tamaño, como es el caso de este proyecto.

4.5 WSDL

4.5.1 Introducción

WSDL es un lenguaje escrito en XML que nos permite describir web services. Un documento WSDL especifica la localización del servicio así como las operaciones o métodos de que dispone. Por tanto, lo usaremos en este proyecto para indicar a nuestros clientes la funcionalidad del web service así como el lugar desde donde puede ser consumido.

4.5.2 Estructura del documento

Para describir un web service los documentos WSDL disponen de los siguientes elementos:

- <*portType*>: contiene las operaciones que el web service puede llevar a cabo.
- <*message*>: contiene los mensajes usados por el web service.
- <*types*>: contiene los tipos de datos que el web service utiliza.
- <*binding*>: especifica los protocolos de comunicación del web service.

Un documento WSDL puede contener, también, otros elementos de extensión o bien un elemento del servicio que se encarga de hacer posible la agrupación de las definiciones de diversos WS en un único documento WSDL.

4.5.3 Elemento <*portType*>

Este elemento es el más importante de un documento WSDL. En él se incluye la descripción del web services, las operaciones que pueden ser ejecutadas así como los mensajes que están involucrados.

Si los analizamos desde el punto de vista de un lenguaje de programación tradicional el elemento <*portType*> se puede equiparar con una librería de funciones (o con una clase).

4.5.4 Elemento <*message*>

Este elemento permite definir los datos de cualquiera de las operaciones que realice el web service. Cada uno de estos mensajes puede consistir en una o varias partes, las cuales, siguiendo el símil de los lenguajes tradicionales son comparables a los parámetros pasados a una función al llamarla.

4.5.5 Elemento <*types*>

Como su nombre indica este elemento define qué tipos de datos van a ser usados por el web service. Para evitar problemas de dependencias de plataformas, WSDL usa el esquema XML para definir estos tipos de datos.

4.5.6 Elemento <*binding*>

Este elemento simplemente nos indica el formato del mensaje y los detalles del protocolo usado por cada una de las operaciones definidas en el elemento <*portType*>.

4.5.7 Ejemplo

A continuación se puede observar lo que sería una versión bastante simplificada de un documento WSDL:

```
<message name="getPriceRequest">
   <part name="book" type="xs:string"/>
</message>

<message name="getPriceResponse">
   <part name="price" type="xs:string"/>
</message>

<portType name="bookPrices">
  <operation name="getPrice">
      <input message="getPriceRequest"/>
      <output message="getPriceResponse"/>
  </operation>
</portType>
```

En este ejemplo podemos ver como el elemento <*portType*> nos indica que hay un puerto con nombre *bookPrices* el cual contiene la operación *getPrice*. Esta operación tiene el mensaje de consulta *getPriceRequest* y el mensaje de respuesta *getPriceResponse*.

Por otra banda en los elementos <*message*> observamos las partes de cada mensaje (en este caso sólo una por mensaje) así como los tipos de datos asociados.

Si lo comparamos con un lenguaje de programación tradicional *bookPrices* sería la librería de funciones, *getPrice* sería una función concreta que tendría como parámetro de entrada *getPriceRequest* y como parámetro de salida *getPriceResponse*.

4.5.8 Tipos de operaciones

Básicamente los tipos de operaciones más utilizados en WSDL son los pares petición-respuesta, pero también disponemos de los siguientes tipos:

- Unidireccional: la operación puede recibir un mensaje pero no retornará ninguna respuesta.
- Petición-respuesta: como se ha mencionado anteriormente la operación puede recibir una petición y enviará una respuesta.
- Solicitud-respuesta: la operación puede enviar una petición y esperará una respuesta.
- Notificación: la operación puede enviar un mensaje pero no esperará ningún tipo de respuesta.

4.5.9 Operación unidireccional

Un ejemplo de operación de tipo unidireccional sería el siguiente:

```
<message name="newBookPrice">
   <part name="book" type="xs:string"/>
   <part name="price" type="xs:string"/>
</message>

<portType name="bookPrices">
   <operation name="setPrice">
      <input name="newPrice" message="newBookPrices"/>
   </operation>
</portType >
```

En este ejemplo vemos que el puerto *bookPrices* define un tipo de operación unidireccional llamado *setPrice*, el cual permite la entrada de nuevos mensajes de tipo *newBookPrice* con los parámetros de entrada *book* y *price*, sin embargo observamos como no hay ningún parámetro de salido definido.

4.5.10 Operación petición-respuesta

El siguiente ejemplo muestra como sería una operación petición-respuesta:

```
<message name="getPriceRequest">
   <part name="book" type="xs:string"/>
</message>

<message name="getPriceResponse">
   <part name="price" type="xs:string"/>
</message>

<portType name="bookPrices">
  <operation name="getPrice">
      <input message="getPriceRequest"/>
      <output message="getPriceResponse"/>
  </operation>
</portType>
```

Aquí observamos como existe una operación de petición-respuesta con nombre *getPrice* que requiere un mensaje de entrada, en este caso *getPriceRequest* con parámetro *book*, y devuelve un mensaje llamado *getPriceResponse* con parámtro *price.*

4.5.11 Binding

Para poder describir mejor la funcionalidad del elemento *binding* veámos el siguiente ejemplo:

```
<message name="getPriceRequest">
   <part name="book" type="xs:string"/>
</message>
<message name="getPriceResponse">
   <part name="price" type="xs:string"/>
</message>

<portType name="bookPrices">
  <operation name="getPrice">
      <input message="getPriceRequest"/>
      <output message="getPriceResponse"/>
  </operation>
</portType>

<binding type="bookPrices " name="b1">

<soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http" />
  <operation>
    <soap:operation
     soapAction="http://ejemplo.com/getPrice"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

En este ejemplo el elemento *binding* tiene dos atributos como son el nombre y el tipo. Este último nos apunta hacia el puerto del *binding* que en este caso es *bookPrices*.

De la misma manera el elemento *soap:binding* también contiene dos atributos. Por una parte el atributo *style* puede tener o bien el valor “rpc”, o bien el valor “document”. Por otra parte el atributo *transport* define la versión concreta del protocolo SOAP que se va a usar, en este caso HTTP.

Finalmente el elemento *operation* define cada una de las operaciones que incluye el puerto. Para cada una de estas operaciones se debe definir la correspondiente acción SOAP, así como la manera en que los parámetros de entrada y salida se codifican, en nuestro caso usamos “literal”.

4.5.12 Sintaxis

A continuación se expone un listado de la sintaxis completa de WSDL 1.2.

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri">
    <import namespace="uri" location="uri"/> *
    <wsdl:documentation .... /> ?
    <wsdl:types> ?
        <wsdl:documentation .... /> ?
        <xsd:schema .... /> *
    </wsdl:types>
    <wsdl:message name="ncname"> *
        <wsdl:documentation .... /> ?
        <part name="ncname" element="qname"? type="qname"?/> *
    </wsdl:message>
    <wsdl:portType name="ncname"> *
        <wsdl:documentation .... /> ?
        <wsdl:operation name="ncname"> *
            <wsdl:documentation .... /> ?
            <wsdl:input message="qname"> ?
                <wsdl:documentation .... /> ?
            </wsdl:input>
            <wsdl:output message="qname"> ?
                <wsdl:documentation .... /> ?
            </wsdl:output>
            <wsdl:fault name="ncname" message="qname"> *
                <wsdl:documentation .... /> ?
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:serviceType name="ncname"> *
        <wsdl:portType name="qname"/> +
    </wsdl:serviceType>
    <wsdl:binding name="ncname" type="qname"> *
        <wsdl:documentation .... /> ?
        <-- binding details --> *
        <wsdl:operation name="ncname"> *
            <wsdl:documentation .... /> ?
            <-- binding details --> *
            <wsdl:input> ?
                <wsdl:documentation .... /> ?
                <-- binding details -->
            </wsdl:input>
            <wsdl:output> ?
                <wsdl:documentation .... /> ?
                <-- binding details --> *
            </wsdl:output>
            <wsdl:fault name="ncname"> *
                <wsdl:documentation .... /> ?
                <-- binding details --> *
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="ncname" serviceType="qname"> *
        <wsdl:documentation .... /> ?
        <wsdl:port name="ncname" binding="qname"> *
            <wsdl:documentation .... /> ?
            <-- address details -->
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

4.6 JPA

4.6.1 Introducción

JPA (Java Persistence API) es una plataforma desarrollada para el lenguaje de programación Java que permite gestionar datos relacionales con el objetivo de usar las ventajas del modelo de orientación a objetos al interactuar con bases de datos relacionales. El funcionamiento de JPA se basa en POJOs (Plain Old Java Object) para usar persistencia en nuestras aplicaciones.

4.6.2 Entidades

Cuando hablamos de persistencia normalmente nos referimos a los datos que son permanentes en nuestra aplicación y que se suelen almacenar en medios tales como bases de datos o ficheros. Sin embargo en JPA, estos datos se llaman entidades. Estas entidades hacen referencia a una colección lógica de datos que pueden ser almacenados y recuperados como un todo.

Las entidades forman el núcleo de JPA y disponen de diferentes características y propiedades. Las más importantes son las siguientes:

- **Persistencia**. Se encarga de almacenar y recuperar las entidades de un medio persistente como una base de datos.
- **Identidad.** Esta propiedad se usa para identificar una entidad única entre diferentes entidades en una base de datos.
- **Transaccionabilidad.** Las operaciones de crear, actualizar y borrar de las entidades se llevan a cabo en un contexto transaccional.

4.6.3 Mapeado objeto-relacional

Sabemos que el modelo de orientación a objetos funciona con clases, de la misma manera que las bases de datos lo hacen con tablas. A partir de ahora, gracias al uso de JPA, no trabajaremos con tablas sino con objetos, y, por tanto, realizaremos todas las operaciones y consultas contra objetos. Esto nos

permitirá evitar la utilización del lenguaje relacional asociado a tablas, columnas, claves, etc. Por tanto, usar este modelo nos proporcionará características propias del paradigma de orientación a objetos como por ejemplo la herencia.

4.6.4 Metadatos

En cada entidad tendremos asociado un conjunto de metadatos que la describirán y permitirán a la capa de persistencia que sea reconocida, interpretada y gestionada de manera correcta, desde el momento en que se carga hasta que es invocada. Existen dos maneras distintas de especificar los metadatos dependiendo, únicamente, de la elección del desarrollador. Son las siguientes:

- **Anotaciones.** Permiten añadir metadatos al código de una manera estructurada. Fueron introducidas por primera vez como parte de la plataforma Java SE 5. Es el tipo de metadatos que hemos usado en este proyecto.
- **XML.** Esta modalidad permite especificar los metadatos de cada entidad de manera externa al código mediante un documento XML.

4.7 Sumario del capítulo

En este capítulo hemos hecho un extenso repaso a las tecnologías usadas en la realización del proyecto. Hemos empezado por la base de representación y almacenamiento de datos, como es XML, y hemos visto como definimos la estructura de estos datos para coordinarnos con los clientes. También hemos hecho un repaso por el protocolo de comunicación que hemos utilizado en el proyecto, como es SOAP, así como la manera en que describimos esta comunicación, WSDL. Por último hemos repasado una innovadora tecnología de persistencia, llamada JPA, que nos permite separar la capa de base de datos totalmente de la capa Java.

Capítulo 5: Desarrollo

5.1 La base: XML

Podemos considerar que en este proyecto el lenguaje XML toma un papel especialmente significativo, ya que no sólo lo usamos para el envío y la recepción de los datos que necesitaremos para llevar a cabo todas las operaciones de nuestro web service, sino que, además, utilizaremos XML como sistema nativo de datos. Es decir, en la base de datos guardaremos y recuperaremos la información directamente en formato XML.

Como ya hemos comentado XML se ha convertido en uno de los métodos más usados para representar datos siendo independiente de plataforma. Gracias a esto, su uso es de mucha utilidad para llevar a cabo transacciones de información entre distintas aplicaciones o sistemas. Las ventajas que ofrece el hecho de almacenar documentos XML en una base de datos son amplias. También lo es el tipo de aplicaciones que se pueden llegar a desarrollar con esta tecnología, por ejemplo:

- Aplicaciones que permiten recuperar ciertos documentos basándose en su contenido.
- Aplicaciones que sólo quieran recuperar un contenido parcial del documento.
- Aplicaciones con la posibilidad de agregar documentos.

La razón por la que hemos usado una base de datos relacional para almacenar datos XML en este proyecto, es porque Microsoft SQL Server 2005 nos

proporciona poderosas herramientas respecto a la consulta y modificación de datos, de la misma manera que lo hace con los datos relacionales tradicionales. Para llevar a cabo esto se ha creado en SQL Server 2005 un nuevo tipo de datos nativo llamado XML, que nos permite almacenar en la misma base de datos tanto datos relacionales como documentos XML simultáneamente.

Para llevar a cabo estas operaciones y consultas de los datos almacenados en forma de XML se utiliza el lenguaje XQuery. Además, XQuery también incluye XPath, que nos permite modificar directamente los datos XML, como por ejemplo, añadir o borrar sub-árboles del documento o actualizar ciertos valores.

En forma de conclusión, destacaremos los grandes beneficios que aporta para el desarrollador el hecho de no tener que diseminar los datos para almacenarlos en la base de datos, y aún más importante, tener que reconstruir el documento XML manualmente que se va a enviar como respuesta a una solicitud.

5.2 Esquemas de comunicación

5.2.1 Introducción

De nada nos valdría realizar un sistema de comunicación como nuestro web service si no sabemos qué datos nos van a llegar ni cómo tratarlos. Así pues, necesitamos definir una estructura fija que sepamos tanto nosotros como el consumidor del web service, de manera que lo que obtengamos de esa transmisión de datos sea una información, para la cual, nuestros sistemas estén preparados para procesar.

Sabemos que en nuestro web service dispondremos de diversos tipos de operaciones que nos definirán toda la funcionalidad del sistema. Algunas de estas operaciones tendrán muchos puntos en común respecto a los datos que necesitan para llevarse a cabo. Por ejemplo, los datos del tomador del seguro serán necesarios tanto para un producto del ramo de autos, como de accidentes o de hogar. Por este motivo definiremos básicamente cinco

esquemas XSD que darán forma a los documentos XML, los cuales contendrán todos los datos con los que trabajaremos.

5.2.2 Esquema de solicitud de cotización

El esquema de solicitud de cotización nos definirá la estructura del documento XML que nuestros clientes nos harán llegar cuando deseen hacer una petición de cualquier tipo de operación de cotización. Es decir, se usará el mismo esquema ya sea una solicitud para una póliza de decesos, salud, autos, etc. y siempre cuando se trate de una cotización, ya que para las emisiones se usará otro tipo de esquema.

El primer elemento de este esquema será el de identificación, en él haremos constar los datos básicos del tomador del seguro, ya sea persona física o jurídica, así como sus datos de contacto. En este elemento como datos opcionales dependiendo del tipo de producto del que se trate, también incorporamos los datos del propietario, este sería el caso, por ejemplo, de los seguros de hogar o automóviles.

El segundo y principal elemento que incluiremos en nuestro esquema es el que contendrá los datos del riesgo. La información que este elemento aporte nos servirá para hacer el cálculo del precio de la póliza para la cual el cliente haya hecho la solicitud. Como nuestro objetivo es hacer un sistema reutilitzable daremos soporte a los ramos más comunes del mundo asegurador, aunque en un principio Aura no los comercialice. Por tanto, para especificar el riesgo dispondremos de los siguientes elementos:

- **Personales.** Utilizaremos este elemento cuando lo que queramos asegurar sea una persona, por ejemplo, estaríamos hablando de seguros de decesos, accidentes, salud o repatriación. Necesitaremos, por tanto, datos como fechas de nacimiento, sexo o profesión, así como país de destino en el caso del seguro de repatriación para inmigrantes.
- **Inmuebles.** Este elemento lo usaremos con los ramos relacionados con los bienes inmuebles, básicamente el seguro de hogar. En el incluiremos

la información de la localización del inmueble, los datos a cerca de su construcción como materiales, años de antigüedad, etc. También necesitaremos hacer constar los datos sobre la seguridad de la que dispone (alarmas, vigilante, etc.). Por último, también será necesaria especificar los capitales, es decir, contenido, continente, objetos de valor, caja fuerte, etc.

- **Embarcaciones.** Este es un elemento simple ya que nos sirve para el ramo de embarcaciones. En él simplemente incluiremos los datos de la embarcación que se desee asegurar.

- **Vehículos.** Este es uno de los ramos más complejos, ya que se incluyen automóviles de diversas categorías como motocicletas o coches. Lo primero que necesitaremos saber es de que modelo de vehículo de trata, esta información la obtendremos a partir del código correspondiente de la base de datos oficial de modelos de automóviles de España llamada Base SIETE. A partir de aquí se harán tres distinciones dependiendo de la categoría del vehículo:

    - **Categoría 1 (Coches).** Si el automóvil del que se quiere hacer el seguro es un coche, necesitaremos saber qué uso se le va a dar (particular, transporte público, etc), los accesorios que se incluyen y su valor, y el código postal de circulación.

    - **Categoría 2 (Vehículos industriales).** Para esta categoría lo primero que necesitamos saber es qué tipo de vehículo es y las características que tiene. También nos interesará saber su peso, el uso que se le va a dar, así como la mercancía que va a transportar.

    - **Categoría 3 (Motos).** En este caso simplemente nos interesará saber el valor de los accesorios que pueda tener y el código postal de circulación.

  En estos tres casos, por supuesto, también nos interesará saber la matrícula. Una vez que tengamos los datos del vehículo, necesitamos obtener los del conductor o conductores. Estos datos serán simplemente

el tipo de conductor (habitual u ocasional), su número de identificación y su sexo. Por último también nos interesará saber la siniestralidad que ha sufrido el conductor habitual en los últimos años para poder hacer un cálculo de una posible bonificación ya sea positiva o negativa.

- **Asistencia Jurídica.** Este producto consiste en ofrecer los servicios de un abogado para dar respuesta cuando el cliente lo estime oportuno a las dudas jurídicas que le puedan surgir. Por tanto, simplemente necesitamos el volumen de consultas que se quieren contratar.

Una vez que disponemos de los datos del riesgo, necesitamos otros datos complementarios, los cuales formarán el tercer de los principales elementos de nuestro esquema de solicitud. Estos datos son la fecha de efecto en la que entrará en vigor la póliza, el fraccionamiento con el que el cliente desea hacer la contratación, así como los datos del seguro en vigor (en caso de que ya disponga de uno del mismo tipo en otra compañía).

Por último también se incluirá un elemento en el que el cliente se identificará con unas credenciales que le habremos proporcionado previamente y que servirán para autorizarle a realizar la operación que solicita.

5.2.3 Esquema de respuesta de cotización

En el esquema de respuesta de las cotizaciones dispondremos básicamente la mínima información que el cliente necesite para evaluar si está interesado en la contratación del producto que le ofrecemos. Para ello le enviaremos la siguiente información:

- **Ramo.** En este elemento informaremos del ramo del que se ha obtenido la solicitud.

- **Precios.** Este es un elemento que se puede repetir varias veces ya que en ocasiones ofreceremos diferentes modalidades (como terceros, todo riesgo, etc. en el seguro de autos). Por tanto, este elemento contendrá la modalidad, el importe del primer recibo, de los siguientes recibos y el

precio anual, así como el tipo de fraccionamiento. Por último, también contendrá un identificador como atributo que servirá para identificar esta modalidad en concreto en caso de que el cliente desee contratar la póliza. Dentro de este elemento se incluirá otro que ofrecerá información sobre las coberturas que están incluidas en cada una de las modalidades que se ofrezcan.

- **Condiciones.** En este elemento haremos constar cualquier condición que sea necesaria para contratar la póliza. Por ejemplo, en una póliza de autos podría ser necesario que un perito revisara el coche que se quiere asegurar a todo riesgo para comprobar que está en buen estado.

- **Documentos.** Aquí haremos constar cualquier documento que pueda necesitar el cliente para decidirse por la contratación de la póliza, como por ejemplo, las condiciones generales del seguro. Cualquier documento que deseemos hacer llegar al cliente será en formato PDF y estará codificado en base 64 para poder hacer el envío a través de XML de una manera satisfactorio.

5.2.4 Esquema de solicitud de emisión

En el esquema de cotización hemos pedido ciertos datos personales básicos del tomador como el nombre o el NIF. Sin embargo, en el momento de realizar la contratación, el cliente nos debe informar de todos sus datos de manera pormenorizada, de manera que completemos los datos de los que ya disponíamos en la cotización. Estos nuevos datos consistirán en aspectos como la dirección, población, número o números de teléfono, dirección de correo electrónico, etc. Además, dependiendo del producto también habrá que informar de datos similares para el conductor o conductores si se trata de un seguro de autos.

También es muy importante la información de cobro. Actualmente en Aura, para pólizas nuevas, sólo existe la posibilidad de llevar a cabo el cobro por

domiciliación bancaria. De todas maneras, dispondremos otros medios de pago por si en algún momento necesitan ser utilizados.

Por último, también necesitaremos hacer constar cual es concretamente la modalidad del seguro que el cliente desea contratar. En la respuesta de la solicitud de cotización se le ofrecerán diversos precios, con distintas modalidades y fraccionamientos, cada uno con un identificador único que nos tendrá que comunicar en la solicitud de emisión para que nosotros podamos localizar la oferta que el cliente desea contratar.

5.2.5 Esquema de respuesta de emisión

Esta será la información final que el cliente recibirá y certificará que la póliza ya ha sido emitida de manera correcta. La información que contendrá este último esquema es la siguiente:

- **Número de póliza.** Este es un dato crucial para poder llevar a cabo la identificación de una póliza.
- **Estado**. En ocasiones la póliza no podrá ser emitida de manera directa sino que es posible que se necesite algún tipo de documentación para verificar los datos que el cliente nos ha comunicado. Por ejemplo, si se trata de un seguro de autos, necesitaríamos el carné de conducir, la documentación del automóvil que se quiere asegurar, etc.
- **Requisitos**. Será en este elemento en el que se le comunicará al cliente qué documentos debe aportar para emitir definitivamente su póliza. Se usará sólo en caso de que sea necesario.
- **Documentos**. En este elemento se transmitirán al cliente los documentos que sean necesarios para demostrar que tiene una póliza en vigor. También se le enviarán otro tipo de documento como las condiciones particulares del seguro que ha contratado.

- **Información complementaria**. Cualquier otra información que pueda ser necesaria transmitir al cliente o al corredor se situará en este elemento.

5.2.6 Esquema de listas de valores

Como ya hemos comentado en ocasiones el consumidor de nuestro web service necesitará usar unos códigos para ciertos campos de las operaciones de solicitud, por ejemplo, para especificar los accesorios que tiene un coche. El esquema para este tipo de información será muy simple, ya que únicamente existirá un tipo de elemento que se repetirá tantas veces como sea necesario (dependiendo del tipo de lista que se haya solicitado), donde se especificará un código numérico y su correspondiente descripción. Este código es que nos tendrán que envían para que nosotros reconozcamos a lo que está haciendo referencia.

5.3 Flujo de comunicación

Una vez que ya está clara la forma en que representaremos los datos y la estructura de estos, es el momento para establecer la comunicación. El primer paso, como es lógico, lo establecerá el consumidor del web service, ya sea un agente o corredor, o bien un cliente final, en el caso de que la consulta se realice desde un motor de multitarificación de acceso público.

Una vez que nosotros recibamos el documento XML con la información correspondiente a la solicitud de cotización, haremos las comprobaciones y cálculos pertinentes para devolver al cliente la información que nos ha requerido. En caso de que las características que el cliente nos indica no entren dentro de nuestra política de negocio, también devolveremos una descripción del motivo del rechazo. De la misma manera, si se produce algún error de tipo técnico, como datos incorrectos, lo informaremos en la respuesta, aunque ya será el motor de multitarificación el que se encargue de trasladar el motivo de la problemática al cliente final.

En el caso de que hayamos conseguido realizar correctamente el proceso hasta aquí y el cliente esté de acuerdo con una de las ofertas que le exponemos, deberá enviarnos de nuevo una solicitud de emisión de la póliza indicando qué oferta es la que escoge (si es que le hemos ofrecido más de una), así como el resto de datos necesario para tramitar la contratación del seguro.

El último paso será devolver al cliente una confirmación de la contratación del seguro, con un número de póliza y un conjunto de documentos que lo acrediten. En el caso de que la póliza quede pendiente de emisión, es decir, que está aceptada pero el cliente nos tenga que hacer llegar algún tipo de documento, se le informará y se esperara a la tramitación de estos documentos para emitir la póliza definitivamente.

5.4 Problemas encontrados

En la realización de este proyecto no se han encontrado grandes problemas que hayan propiciado una ralentización significativa del mismo. Por supuesto, han existido ciertas dificultades que se han ido solucionando sin mayor novedad. Básicamente han existido dos puntos clave que nos han generado pequeños contratiempos.

Por una parte nos encontramos con el tema de la seguridad y los certificados digitales, el desarrollo de la infraestructura necesaria para llevar a cabo las autenticaciones de los clientes ha sido una de las partes más costosas de realizar en el sistema.

Por otro lado, la integración de la tecnología JPA también ha sido laboriosa ya que es muy novedosa y no disponíamos de ninguna clase de experiencia en esta materia. Además, su integración se produjo en la recta final del desarrollo. Esto complicó la situación aunque era importante usar JPA y familiarizarse con esta tecnología ya que se iba a tener que utilizar en futuros proyectos por parte de Prexon.

5.5 Sumario del capítulo

En este capítulo hemos comprobado que el lenguaje XML forma el núcleo de todo nuestro sistema ya que es la manera en que tratamos los datos en todas sus capas, desde su transmisión hasta su almacenamiento. Hemos repasado, también, como se estructura toda la información que vamos a intercambiar con los clientes de modo que haya una coordinación que permita llevar a cabo la transacción con éxito. También hemos visto como debería ser un flujo de comunicación ideal entre el consumidor del web service (parte cliente) y nuestra parte (parte servidor). Por último, hemos comprobado que no han existido grandes problemas durante la realización del proyecto sino pequeñas dificultades que no han retrasado la fecha prevista de término.

Capítulo 6: Pruebas

6.1 Introducción

Para poder comprobar que nuestro sistema funciona de manera correcta se ha usado la *Plataforma de Multitarificación* de Prexon. Esto nos ha permitido hacer una evaluación exhaustiva del web service ya que nos proporciona el medio ideal para la introducción de datos y la visualización de resultados de manera directa e intuitiva. Por supuesto, la plataforma se ha tenido que adaptar a las particularidades de nuestro sistema, como lo ha hecho con el de cada una de las compañías con las que trabaja. Podemos incluir, también, esta integración como parte de este proyecto.

Las pruebas se han ido realizando según se desarrollaba cada una de las partes del sistema. Si bien, es en el momento en que el proyecto estaba básicamente acabado cuando se han realizado todo tipo de pruebas y comprobaciones a fondo para poder asegurarnos de que la funcionalidad del sistema es la correcta. No sólo se han probado situaciones lógicas, sino que también se han introducido todo tipo de combinaciones absurdas de los datos o se han intentado realizar operaciones no autorizadas, de modo que viésemos que nuestro sistema genera el error correspondiente a cada una de estas situaciones.

6.2 La *Plataforma de Multitarificación*

Como ya hemos explicado en el capítulo introductorio, esta aplicación es uno de los productos más importantes de Prexon, ya que es usada habitualmente por diversos agentes y corredores, y trabaja directamente con grandes aseguradoras como Allianz, Mapfre o Zurich entre muchas otras. Esta plataforma no es pública, es decir, sólo se usa de manera interna por las correduría, por lo que un cliente final no tendrá acceso a ella. De cualquier modo el departamento comercial de Aura se encargará de ofrecer a otros motores de tarificación (públicos o privados) ya existentes la posibilidad de integración con los web services de Aura. Hay que destacar que sin una aplicación cliente de este estilo todo nuestro sistema no tendría ninguna utilidad ya que necesitamos una aplicación cliente que interactúe con nuestro web service.

La *Plataforma de Multitarificación* es una aplicación RIA (Rich Internet Application) desarrollada en la parte de visualización mediante Adobe Flex 2.0. Esta tecnología es la heredera de Flash ya que amplía su potencialidad de manera significativa aunque también se basa en Action Script. La aplicación está dividida en tres capas: de datos, intermedia y de visualización. Estas tres capas se comunican entre ellas para llevar a cabo toda la funcionalidad del sistema. En particular, es la capa intermedia la que se encarga de comunicarse con las compañías aseguradoras integradas en la plataforma. Por tanto, será en esta capa donde situaremos toda la lógica que necesitemos para que Aura pueda formar parte de *Plataforma de Multitarificación*. También habrá que hacer ligerísimos retoques en las otras dos capas, como por ejemplo almacenar el logotipo de Aura para que se pueda ver en los resultados.

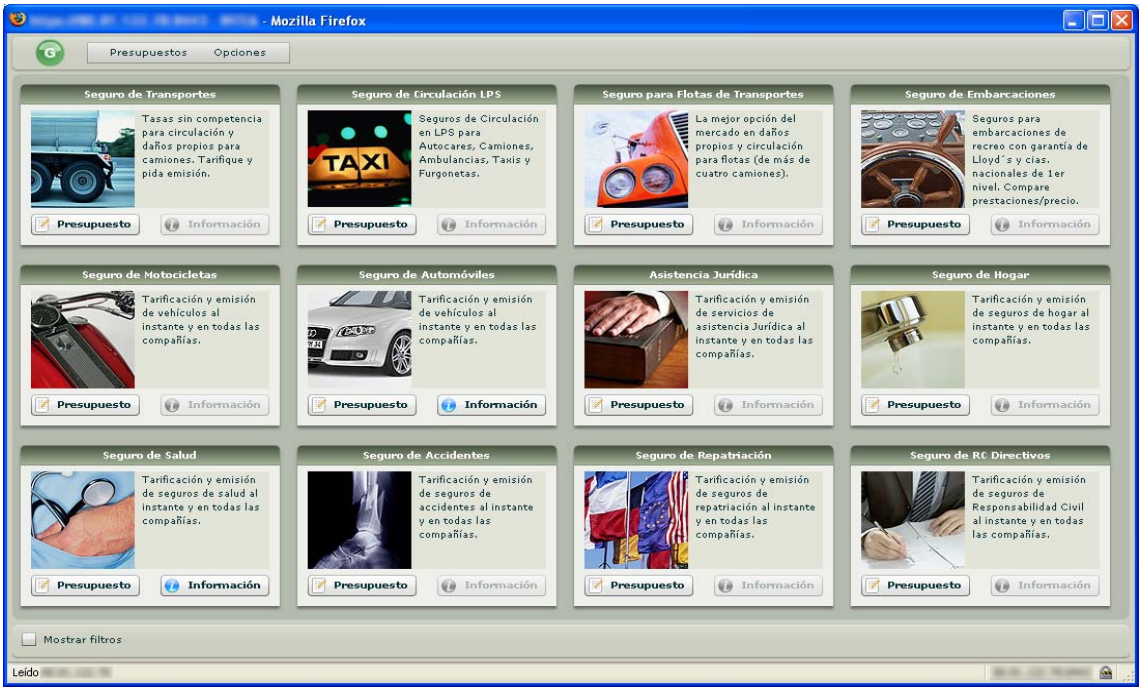

Figura 6-1 – Pantalla principal de la *Plataforma de Multitarificación*

A continuación vamos a ver algunas pinceladas de la *Plataforma de Multitarificación*. En la figura 6-1 podemos a echar un vistazo a la pantalla principal de elección de producto (por supuesto, hay un paso previo que es la introducción de unas credenciales de autorización que no es necesario mostrar).

Como podemos ver en la figura anterior, la pantalla principal de la aplicación nos permite escoger el ramo con el que queremos trabajar. Aura no tiene productos para todos estos ramos por lo que sólo está integrada en algunos de ellos. A continuación vamos a ver una captura de pantalla del formulario del seguro de accidentes, con el que sí trabaja Aura. En este formulario especificaremos la información básica del tomador que necesitemos para calcular el presupuesto de la póliza:

Figura 6-2 – Formulario del tomador de la *Plataforma de Multitarificación*

Para completar la cotización del seguro necesitamos la información específica del riesgo, en este caso, de un seguro de accidentes. En la figura 6-3 podemos observar el formulario de introducción de los datos de riesgo para el producto de accidentes:

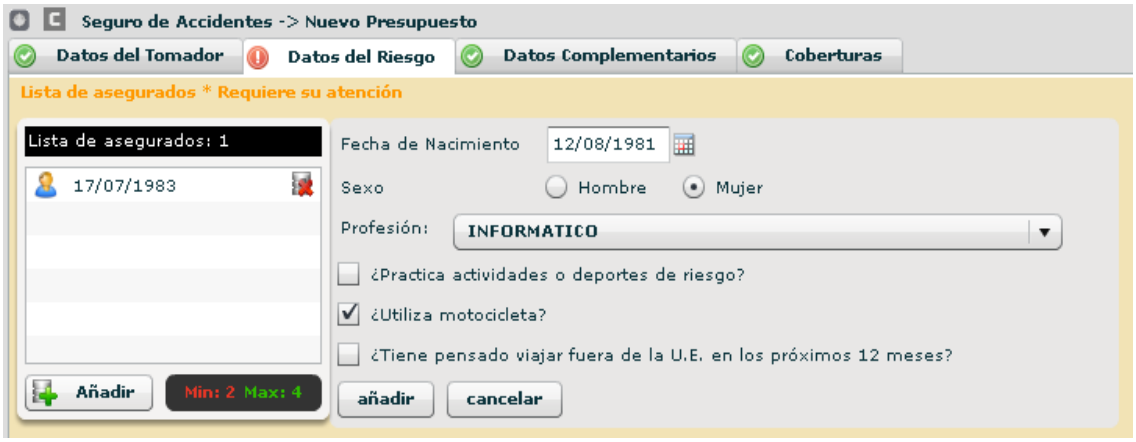

Figura 6-3 – Formulario de riesgo del seguro de accidentes de la *Plataforma de Multitarificación*

La figura 6-3 nos muestra un formulario de riesgo bastante simple. En la figura 6-4 podemos ver uno de los más complejos como es del seguro de automóviles:

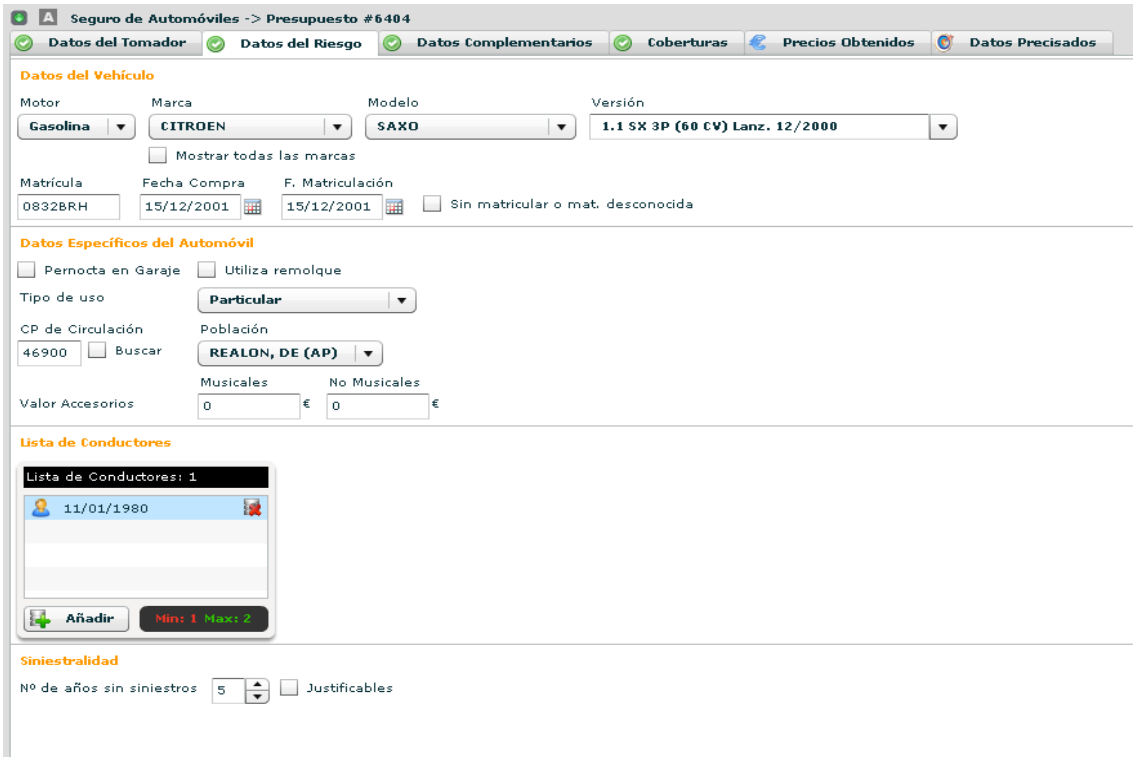

Figura 6-4 – Formulario de riesgo del seguro de automóviles de la *Plataforma de Multitarificación*

En la figura 6-5 podemos ver la pantalla de resultados de un seguro de salud. En ella vemos los precios todas las compañías que han respondido a la solicitud (entre ellas Aura). Si nos interesara alguna de estas ofertas, podríamos emitirla pulsando el botón de “Continuar” de las que nos interese.

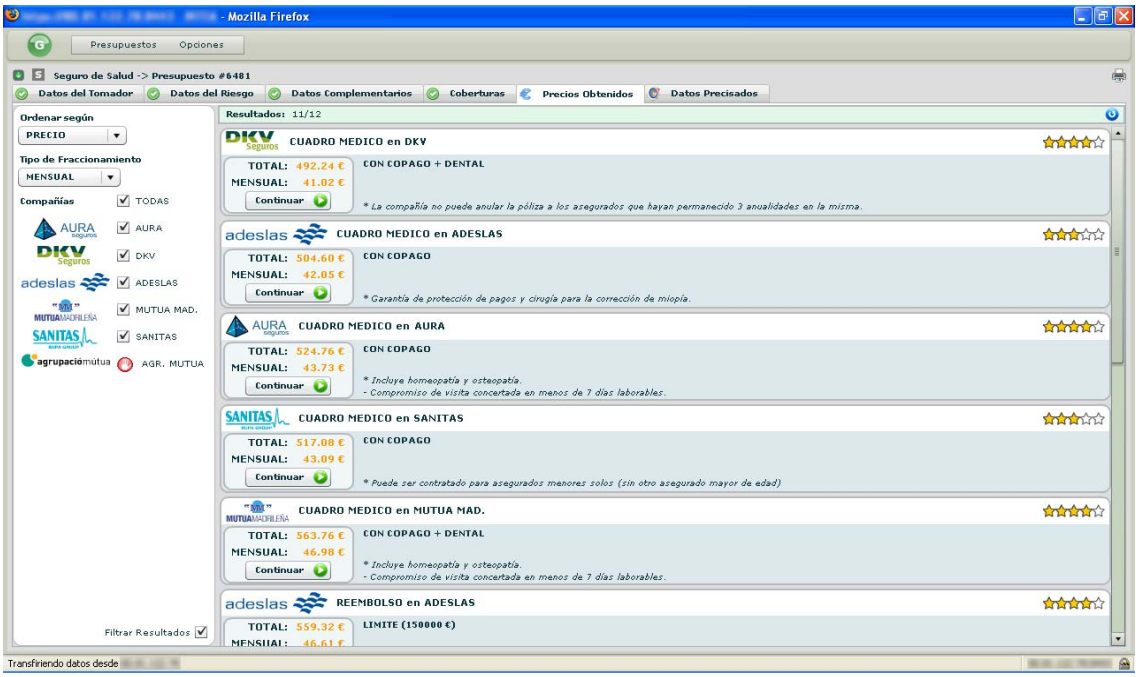

Figura 6-5 – Pantalla de resultados para un seguro de salud de la *Plataforma de Multitarificación*

Finalmente, el último paso sería rellenar el formulario de emisión para contratar la póliza. En este formulario simplemente haremos constar los datos personales que no hemos especificado en la cotización y los datos del pago. En ese momento se hará la solicitud de emisión a la compañía del que el cliente haya seleccionado la oferta. Puede haber diversos resultados:

- **No emitida.** Esto sucederá cuando se produzca algún tipo de error técnico en la comunicación y, o bien la compañía no haya recibido nuestra petición, o la respuesta de la aseguradora viene con errores.
- **Pendiente.** Este será el caso en que el cliente deba llevar a cabo alguna operación física para acabar de emitir la póliza, por ejemplo, enviar algún tipo de documentación.
- **Aceptación comercial.** En este caso la póliza también queda pendiente, pero el que debe resolver la situación será alguno de los comerciales de la compañía, el cual deberá decidir si le interesa a la compañía contratar esa póliza.
- **Emitida.** Todo ha ido bien, la póliza se ha emitido correctamente y el cliente ya puede disfrutar de su seguro.

6.3 Transformaciones XSL

Para que la *Plataforma de Multitarificación* pueda trabajar con distintas compañías es necesario mantener un orden en la representación y almacenamiento de los datos. Cada compañía tiene sus propios esquemas XSD, y mantener la plataforma con decenas de esquemas diferentes simultáneamente es una misión imposible. Por tanto, es obligado tener un sistema que permita pasar del esquema propio de cada una de las compañías al particular que usa la plataforma. Esto se puede llevar a cabo mediante transformaciones XSL (eXtensible Stylesheet Language) y en particular en lenguaje XSLT (XSL Transformations).

Gracias a estas transformaciones podemos pasar de un documento XML con una estructura determinada a otro XML con una estructura distinta, es decir, se lleva a cabo un mapeado de datos. Este mapeado no necesariamente debe ser directo de un elemento a otro, o de un atributo a otro, sino que se pueden añadir operaciones lógicas y aritméticas para transformar los tipos de representación de datos que se usan en un documento al formato del otro. XSLT es un documento XML que usa XPath para navegar por los XMLs con los que va a llevar a cabo las transformaciones.

La plataforma usa esta tecnología para hacer cada una de las transformaciones de su formato propio al de los esquemas de las compañías y a la inversa. De esta manera, se pueden realizar todas las operaciones de manera rápida y eficaz. Por tanto, en la realización de este proyecto se han elaborado cuatro de estas transformaciones (una por cada operación básica de solicitud y respuesta) para poder integrar el sistema de web services de Aura realizado en este proyecto.

6.4 Resultado de las pruebas

Como se ha comentado anteriormente, las pruebas que se han llevado a cabo en este proyecto se han ido haciendo de manera incremental a medida que cada una de sus partes se iba realizando. El proceso de pruebas empezó con

la primera versión del web service, que simplemente nos devolvía un “Hola mundo!”. A medida que se fueron añadiendo nuevas funcionalidades, se iba probando todo el sistema. Por cada nueva interfaz que se realizaba, se hacían pruebas a fondo, incluyendo la comprobación de casos absurdos y datos incoherentes. Con esto no sólo se verificaba que el sistema devolvía las respuestas con los correspondientes precios y modalidades que habían sido solicitados, sino que, además, en caso de que existiese algún tipo de error, nuestro sistema sería capaz de detectarlo y describir el problema al consumidor del web service.

La segunda fase de pruebas que se realizó hacía referencia a la seguridad, se probó todo el sistema usando el protocolo SSL y comprobando que el cliente hacía uso de un certificado emitido por una entidad certificadora acreditada para poder hacer uso de nuestro web service.

Finalmente, después de la integración en la *Plataforma de Multitarificación* de Prexon (parte cliente), se realizó la fase de pruebas más importante. El hecho de haber realizado pruebas duras en la fase de desarrollo evitó que se produjesen errores graves en esta fase de la verificación del sistema. Se han llegado a hacer cientos de cotizaciones y emisiones con todo tipo de datos para comprobar que todo nuestro sistema es fiable al ciento por ciento. El resultado obtenido, es, por tanto, inmejorable.

6.5 Sumario del capítulo

En este capítulo hemos hecho un repaso por las pruebas que hemos realizado para comprobar que nuestro sistema funciona a la perfección. Hemos hecho un pequeño repaso sobre la *Plataforma de Multitarificación* para observar cómo funciona un motor de multitarificación y como deberían comportarse los clientes que se conecten a nuestro web service. Por último, hemos explicado en qué han consistido estas pruebas, así como cada uno de los pasos que hemos dado en su realización, obteniendo unos resultados óptimos.

Capítulo 7: Conclusiones

7.1 Revisión de objetivos

El objetivo principal de este proyecto, que consistía en facilitar la conexión entre la compañía de seguros Aura y la *Plataforma de Multitarificación* de Prexon, así como otros motores de multarificación, se ha cumplido totalmente. A partir de ahora Aura dispondrá de una vía telemática de comunicación con sus agentes y corredores, así como la posibilidad de hacerlo con clientes finales. Esto se sustenta en las interfaces que hemos puesto a disposición de Aura para que pueda ofrecer servicios de cotización y emisión de pólizas de manera remota, asegurando eficacia, rapidez, fiabilidad y seguridad en estas transacciones.

En la parte correspondiente a Prexon también se han cumplido sobradamente los objetivos planteados en un principio. No sólo se ha logrado entregar un proyecto en el plazo acordado, sino que se ha impulsado el crecimiento de la *Plataforma de Multitarificación* integrando una compañía más. Esto ha favorecido a incrementar la competencia entre aseguradoras y, por tanto, reforzar el interés tanto de correduría como de otras compañías para hacerse con el producto.

Un segundo objetivo que impuso Prexon es que la aplicación fuera completamente reutilitzable para futuros proyectos con similares necesidades. Este requisito, que podía parecer difícil de cumplir al ciento por ciento, se ha efectuado con bastante éxito, ya que el uso de JPA o el hecho de ubicar una

parte de la lógica de negocio en la base de datos ha colaborado a su materialización.

Respecto a los objetivos personales planteados también hay que hacer una valoración positiva. En primer lugar, he obtenido valiosa experiencia, así como conocimientos técnicos durante la realización de este proyecto, que complementan así los estudios de Ingeniería en Informática. Además, he tenido la oportunidad de trabajar con un equipo de grandes profesionales que no han hecho más que inculcarme conocimientos técnicos pero también ganas y motivación por el trabajo bien hecho.

En relación a los requisitos técnicos del proyecto cabe destacar que este ha sido uno de los apartados en los que se ha obtenido un éxito más rotundo. Los objetivos de escalabilidad, fiabilidad y seguridad se han cumplido plenamente, haciendo especial hincapié en la seguridad, la cual se ha apuntalado de manera severa debido al uso de datos críticos en las transacciones de información. También se ha verificado que el sistema sea escalable de modo que Aura pueda, en un futuro, incrementar su funcionalidad añadiéndole nuevos ramos o productos, así como otro tipo de operaciones corporativas o de gestión.

7.2 Revisión de la planificación

La planificación global del desarrollo que se había calculado inicialmente es bastante acertada respecto a la duración integral del proyecto. También se aproxima bastante a la duración particular de cada una de las tareas por separado, aunque en este aspecto ha habida pequeñas diferencias en algunas estimaciones de duración. Además, el uso de la tecnología JPA no estaba pensado en un principio, lo que ha hecho variar ligeramente la planificación inicial.

7.3 Futuras vías de desarrollo

Las posibles vías de desarrollo que el sistema desarrollado en este proyecto puede abarcar son numerosas ya que podemos ampliar su funcionalidad de manera que se convierta en un auténtico motor no sólo comercial sino también de gestión de la propia compañía. A continuación podemos ver una lista de estas posibles mejoras:

- **Ampliación de ramos.** Esta mejora ya ha estado prevista durante la realización del proyecto gracias al requisito de escalabilidad. En la actualidad Aura está tramitando la ampliación de ramos con los que trabaja. Es por este motivo que el sistema que hemos realizado ya está completamente preparado para hacerse cargo de estos nuevos productos.

- **Partes de accidentes.** Sería interesante que, igual que un corredor puede emitir una póliza telemáticamente desde Internet, por qué no hacer un parte de siniestro. Esto sería muy útil, por ejemplo, en los seguros de hogar o automóviles, en los que se ahorraría mucho tiempo además de papeleo. De la misma manera que se hace con las cotizaciones o emisiones, existiría un esquema de datos XSD en que hiciéramos constar todos los datos del siniestro, así como documentos escaneados en caso necesario.

- **Gestión de cartera.** De la misma manera que con la mejora anterior, también sería muy útil para los agentes de Aura poder disfrutar de una gestión de su cartera de pólizas, es decir, poder consultar los datos de las pólizas que ha emitido. Incluso, dependiendo del agente, permitirle modificar o dar de baja pólizas.

- **Interfaz gráfica.** Ya hemos comentado en diversas ocasiones que el sistema desarrollado en este proyecto es inútil sin una capa cliente que se encargue de obtener los datos que se necesitan para responder a las

solicitudes que le lleguen y de mostrar los correspondientes resultados que se obtengan. En los objetivos de este proyecto no se hacía constar el desarrollo de una interfaz gráfica ya que, tanto a Aura como a Prexon, les interesaba utilizar la *Plataforma de Multarificación* como la parte cliente desde la cual se consume nuestro web service. A pesar de esto, la realización de una interfaz gráfica sencilla, posiblemente colgada de la página web de Aura, en la que se le ofrezca al cliente la posibilidad de llevar a cabo solicitudes de cotizaciones de seguros e, incluso, llegar a emitirlos, puede ser una posible vía de trabajo fututo que puede dar beneficios comerciales a Aura.

7.4 Valoración y conclusión final

Como conclusiones finales destacaremos que se han cumplido de manera rotunda los objetivos y requisitos impuestos al principio del proyecto, así como la planificación temporal prevista, si bien es cierto que el uso de la tecnología JPA (que no estaba planeado en el arranque del proyecto) ha hecho variar ínfimamente los plazos concretos de término de algunos de los avances planificados.

Mi valoración personal también es favorable. La realización de este proyecto no sólo me ha servido para aprender y asimilar nuevos conceptos en el ámbito técnico, sino que ha colaborado a la integración en un gran equipo de profesionales. He vivido, así, una experiencia en una situación real de un proyecto en el ámbito corporativo con unos requisitos que se deben cumplir y un plazo de entrega que hay que satisfacer.

Bibliografía

[1] Graham, S., Davis, D., Simeonov, S., Daniels, G., Brittenham, P., Nakamura, Y., Fremantle, P., Koenig, D., Zentner, C., “Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI”, Sams, 2nd Edition, 2004.

[2] Keith, M., Schincariol, M., “Pro EJB 3: Java Persistence API”, Apress, 1st Edition, 2006.

[3] Snell, J., Tidwell, D., Kulchenko, P., “Programming Web Services with SOAP”, O’Really Media, 1st Edition, 2001.

[4] W3Schools Online Web Tutorials [en línea], http://www.w3schools.com/

[5] MSDN: Microsoft Developer Network (XML Support in Microsoft SQL Server 2005) [en línea], http://msdn.microsoft.com/en-us/library/ms345117.aspx

Abstract

Este proyecto consiste en la realización de un sistema informático que se encargue de ampliar la red comercial de una compañía de seguros a través de Internet. Para ello se utiliza la tecnología de web services, que nos permite efectuar transacciones de datos de manera rápida, fiable y segura. El web service que se ha diseñado se encarga de resolver y dar respuesta tanto a peticiones de solicitud de precios como de emisión de pólizas en varios ramos. El objetivo es ofrecer al cliente final un método sencillo y próximo de cotización y emisión de seguros.

Aquest projecte consisteix en la realització d'un sistema informàtic que s'encarregui d'ampliar la xarxa comercial d'una companyia d'assegurances mitjançant Internet. Per a això s'utilitza la tecnologia de web services, que ens permet efectuar transaccions de dades de manera ràpida, fiable i segura. El web service que s'ha dissenyat s'encarrega de resoldre i donar resposta tant a peticions de sol·licitud de preus com d'emissió de pòlisses en diversos rams. L'objectiu es oferir al client final un mètode senzill i pròxim de cotització i emissió d'assegurances.

This project consists of a computer based system to be responsible for expanding the business network of an insurance company via the Internet. It uses web services technology, which allows us to make data transactions in a fast, reliable and secure way. The web service that has been designed handles to resolve and respond to requests for prices solicitation and insurance policies emission in several classes. The aim is to give customers an easy and near way of trading and issuance insurances.