WEBSERVER 2.0
Motor de integración

Memoria del proyecto

de Ingeniería Técnica en Informática de Sistemas

realizado por

Alberto Vázquez López

y dirigido por

Jordi Pons Aróztegui

Escola d'Enginyeria

Sabadell, *julio* de *2011*

El abajo firmante, **Jordi Pons Aróztegui**,

profesor de la Escola d'Enginyeria de la UAB,

CERTIFICA:

que el trabajo al que corresponde la presente memoria

se ha realizado bajo su dirección por

Alberto Vázquez López

y para que conste firma el presente

Sabadell, julio de 2011

Firmado: ***Jordi Pons Aróztegui***

El abajo firmante, **Alberto Benito**,

de ***UNIT4 Ibérica***,

CERTIFICA:

Que el trabajo al que corresponde la presente memoria se ha realizado bajo su dirección por

Alberto Vázquez López

y para que conste firma el presente

Barberà del Vallès, julio de 2011

Firmado: ***Alberto Benito***

HOJA RESUMEN – PROYECTO FINAL DE CARRERA DE LA ESCUELA DE INGINIERIA

Titulo del proyecto: WEB Server 2.0: "Motor de Integración"	
Autor: Alberto Vázquez López	**Fecha: *Julio 2011***
Tutores: Jordi Pons Aróztegui (UAB) // Alberto Benito (UNIT4)	
Titulación: Ingeniería Técnica en Informática de Sistemas	

Palabras clave:

- **Catalán:** Web Server, Java, Migració.
- **Castellano:** Web Server, Java, Migración.
- **Inglés:** Web Server, Java, Migration.

Resumen del proyecto

- **Catalán:**

A l'empresa Unit4 es disposa d'un Web Server codificat en Visual Basic que es considera està desfassat i obsolet de manera que el que és desitja és fer una migració a un llenguatge de programació més actual i potent i a més eliminar restriccions de software necessari que té ara aquest servidor a més de millorar el seu rendiment. Aquest projecte consisteix en el al desenvolupament d'aquest nou servidor.

- **Castellano:**

En la empresa Unit4 se dispone de un Web Server codificado en Visual Basic que ha quedado desfasado y obsoleto de forma que lo que se desea es migrarlo a un lenguaje de programación actual y potente y eliminar restricciones de software que tiene ahora además de mejorar el rendimiento. Este proyecto es el referente al desarrollo de este nuevo servidor.

- **Inglés:**

In UNIT4 company provides a Web Server coded in Visual Basic that has become outdated and obsolete so that what is wanted is to migrate to a current programming language and powerful, and delete software restrictions that have in this moment in addition to improving performance. This project does the reference to the development of this new server.

Índice

Índice de tablas

Índice de figuras

CAPÍTULO 1: Introducción

1.1– Presentación del proyecto, estado actual

Este proyecto se realiza en la empresa UNIT4 Ibérica en un convenio para realizar el proyecto de fin de carrera conjuntamente con la Universitat Autònoma de Barcelona con una duración aproximada de 560 horas.

Para empezar se debería hacer una pequeña definición de qué es un Web Server, ya que el proyecto tratará del desarrollo de uno.

Un Web Server (servidor web) es un programa que está implementado en la máquina servidor y procesa aplicaciones. Es decir, un Web Server es una aplicación que mediante conexiones bidireccionales con un cliente debe dar una respuesta, dependiendo cual sea la petición que haya realizado el cliente. Esta aplicación debe esperar hasta que una aplicación cliente haga una petición y el Servidor Web debe decidir con la información de la petición qué debe hacer. Es aquí donde entran en juego los servicios web.

Los servicios web están englobados dentro del Web Server y son los encargados de realizar las operaciones requeridas por el cliente. Puede ser que se haga una llamada para ver una página web, entonces el servidor retornará un código HTML. También puede ser que se quieran hacer operaciones con una base de datos por lo que el servicio gestionará la conexión a la base de datos y realizará consultas o lo que se necesite en cada momento. Esto es solo un ejemplo de los posibles servicios que puede dar un Web Server.

Este proyecto, llamado ***"Web Server 2.0 Motor de integración",*** hace referencia al proyecto que se lleva a cabo en UNIT4 para renovar el servidor actual que tienen en funcionamiento. El servidor actual está codificado en el lenguaje de programación Visual Basic al que se considera en muchos casos un lenguaje de programación desfasado y obsoleto. Este servidor se ha utilizado durante los últimos 10 años, aunque ha sido modificado en los años posteriores de su puesta en funcionamiento para intentar mejorarlo y adaptarlo a las peticiones de los clientes. A pesar de ello la empresa ha decidido que la mejor opción es la de desarrollar un Web Server nuevo en Java ya que consideran que es un lenguaje de programación actual.

El actual Web Server tiene una serie de limitaciones que a la hora de poder venderlo a los clientes hace que sea más difícil implantarlo. Las principales limitaciones están asociadas al lenguaje de programación Visual Basic en el que está codificado. Los principales problemas son que este servidor ha de estar en funcionamiento exclusivamente en máquinas con un sistema operativo Windows. Además no se permite que las máquinas tengan una arquitectura de más de 32 bits y actualmente cada vez salen más computadores con arquitecturas de 64 bits por lo que no se puede aprovechar el potencial de estas máquinas. Además es necesario que funcione bajo un IIS de Microsoft.

Por lo tanto el servidor actual depende mucho de la tecnología de Microsoft. Lo que se quiere hacer es un cambio de tecnología en el nuevo servidor. Se ha decidido que el servidor se ha de migrar a Java, que es un lenguaje de programación más potente y con perspectivas de que cada día mejore mas debido a las actualizaciones periódicas que hace la plataforma Java. Java además, permite que ya no haga falta ejecutar la aplicación en máquinas con sistemas operativos Windows ya que es multiplataforma y podrá ejecutarse sobre cualquier sistema operativo, esto hace que al cliente se le elimine una restricción de forma que podrá elegir el sistema que quiera o que se adapte mejor a su entidad ya sea

por motivos de rendimiento o por motivos económicos ya que por ejemplo un sistema Linux es mucho más económico que uno Windows porque Linux es de libre distribución.

Además con Java podremos usar arquitecturas de 64 bits y aprovechar las mejoras que se hacen en los computadores.

Por otro lado ya no será obligatorio un IIS de Microsoft, podremos utilizar otro tipo de servidores como Apache Tomcat que será el que se utilice en este proyecto y que es de libre distribución por lo que no tiene ningún coste, aunque el cliente podrá decidir qué servidor utilizar.

Por lo tanto mediante la migración del Web Server, conseguimos eliminar la clara dependencia hacia la tecnología de Microsoft.

1.2 - La empresa "UNIT4"

UNIT4 Ibérica es la filial española del grupo empresarial UNIT4 que cuenta con más de 45 años de experiencia en el desarrollo e implementación de software empresarial. Anteriormente esta empresa se llamaba Centro de Cálculo de Sabadell (CCS) fundada el 1963. En 2006 la compañía holandesa UNIT4 Agresso compro CCS y se convirtió en una filial española de Agresso Spain haciendo que cambiara el nombre por CCS Agresso. Finalmente en 2010 se decidió que por motivos de márqueting lo mejor era cambiar el nombre y unificar todas las filiales de la empresa por lo que ahora el nombre es UNIT4.

Tiene oficinas en 17 países europeos, así como en otros siete países repartidos por Norteamérica, Asia, Pacifico y África. La sede central de la empresa está en Sliedrecht (Países Bajos).

UNIT4 fabrica y comercializa software de gestión empresarial al que da también soporte mundialmente de forma ágil y flexible, con el objetivo de ayudar a las organizaciones dinámicas a gestionar sus necesidades de negocio de manera efectiva. UNIT4 destaca por ser el 5º mayor proveedor de software ERP en el mundo y el segundo a escala europea.

La empresa se divide en diferentes departamentos como son I+D o como lo llaman internamente “Fábrica”, ERP, CRM, Comercial y mas departamentos necesarios en una empresa. Este proyecto se desarrolla en el departamento de ERP dentro de un pequeño grupo que se dedica al desarrollo de Web Services para clientes del departamento que necesitan de Web Services. Han desarrollado varios Web Services que utilizan como base y un conjunto de módulos para ir añadiéndoles funcionalidades teniendo en cuenta lo que demanda el cliente. Aun así a veces los módulos hechos no se corresponden a lo que el cliente desea y ahí es donde entra este grupo de desarrollo, realizando web servers a medida de lo que el cliente desea. Este grupo está compuesto por 10 programadores y no trabajan todos juntos, normalmente cada uno tiene su trabajo aunque dependiendo del tamaño del proyecto se asignan mas programadores a un mismo proyecto.

1.3 - ¿Por qué el PFC en una empresa?

Me pareció interesante la idea de realizar el proyecto de final de carrera en una empresa para comprobar cómo funcionaba una empresa dedicada al desarrollo de software. De esta forma antes de integrarme en el mundo laboral, después de haber conseguido el título tendré una idea clara del funcionamiento de una empresa de este tipo.

Mi experiencia aumentará y ya no partiré de cero. Y al estar desarrollando mi proyecto de final de carrera en una empresa y estar en una etapa de formación podré aprender mucho más sin una presión excesiva.

1.4 – Estructura de la memoria

La memoria de este proyecto está dividida en capítulos que a continuación se detallan:

- **Capítulo 1 Introducción:** En este capítulo podremos ver una presentación del proyecto y de la empresa donde se desarrolla.

- **Capítulo 2 Estudio de viabilidad:** En este capítulo se verá el estudio sobre la viabilidad del proyecto con la descripción del proyecto, el estudio de la situación actual, análisis de los requisitos y las alternativas, la planificación que se ha hecho del proyecto, el presupuesto y coste del proyecto y unas conclusiones comentando si el proyecto es o no viable y si es o no rentable.

- **Capítulo 3 Diseño del sistema:** En este capítulo podremos ver el diseño de la aplicación con una visión global de cuál será la arquitectura de la aplicación. Se

verán diagramas de flujo y diagramas de clases con la forma en que se comunicarán las clases entre sí.

- **Capítulo 4 Implementación:** En el capítulo de implementación pasamos a ver todo lo referente a la parte del código de la aplicación. Primero se verá cual es el entorno de desarrollo y el porqué de algunas elecciones hechas. Dividiremos el capítulo también en diferentes partes, para separar los diferentes lenguajes de programación que se utilizan. Se verá el código de las clases diseñadas en el capítulo anterior y cómo se interrelacionan entre ellas. De esta forma podremos ver de una forma más directa cómo se desarrolla el flujo de datos especificado en el diagrama de flujos del capítulo 3.

- **Capítulo 5 Pruebas:** En este capítulo se verá cómo se han hecho las pruebas para garantizar que la aplicación tenga un correcto funcionamiento y que se han corregido los errores encontrados.

- **Capítulo 6 Conclusiones:** Este es el capítulo final donde se verá un análisis de los objetivos conseguidos y no conseguidos, cómo se ha desarrollado el proyecto en referencia a la planificación inicial y se proponen mejoras a la aplicación hecha.

- Finalmente podremos ver un glosario con los términos clave vistos en la memoria explicando cual es su significado y la bibliografía con los documentos y enlaces que han sido consultados.

CAPÍTULO 2: Estudio de viabilidad

2.1 – Introducción

En este capítulo veremos la viabilidad del proyecto haciendo un análisis de los objetivos, de la situación actual con un diagnóstico del sistema y un análisis de los requisitos funcionales y no funcionales del proyecto. Este proyecto es de tipo desarrollo.

2.2 – Objetivos del proyecto

A continuación se verá un listado de objetivos con una breve descripción:

O1. Migrar Web Server a Java. El objetivo fundamental de todo el proyecto es el de migrar el servidor a java para renovarlo.

O2. Hacer Web Server multiplataforma. Al conseguir realizar el objetivo O1 este objetivo se cumple de forma inmediata ya que Java es multiplataforma. Este objetivo va ligado a la necesidad de eliminar la dependencia a las tecnologías de Microsoft.

O3. Hacer módulo de consultas. Otro objetivo principal es el de realizar un módulo en java que sea capaz de hacer consultas a la BBDD y retornar el resultado.

O4. **Hacer módulo de inserciones.** El módulo de inserciones no se considera objetivo principal. En el caso de que haya tiempo se hará un módulo capaz de hacer inserciones de datos en la BBDD.

O5. **Módulos secundarios.** Este objetivo hace referencia al desarrollo de otros módulos para el servidor, un ejemplo sería el de modificaciones de la BBDD o una API.

O6. **Mejorar rendimiento del Web Server**. Otro objetivo es el de mejorar el rendimiento del servidor en comparación al actual mediante un pool de conexiones.

O7. Reducir costes. Un objetivo secundario es el de reducir costes.

Ahora veremos una tabla donde podremos ver de una forma sencilla la prioridad que se le da a cada objetivo.

Objetivo	Crítico	Prioritario	Secundario
O1	X		
O2	X		
O3	X		
O4		X	
O5			X
O6		X	
O7			X

Tabla 1 Prioridades de los objetivos

2.3 – Partes interesadas

En la siguiente tabla podremos ver cuáles son los usuarios que están destinados a utilizar la aplicación y cuáles serán sus responsabilidades.

USUARIOS

Perfil	Responsabilidad
Administrador del sistema	Control del sistema. Gestión de usuarios. Mantenimiento.
Usuario experto	Gestión de información. (Introducción, modificación de la BBDD)
Usuario no experto	Consulta de la información

Tabla 2 Usuarios del sistema

La siguiente tabla muestra los recursos humanos que tiene el proyecto con una breve descripción de cuál será su responsabilidad a la hora de desarrollar la aplicación.

PROJECT TEAM

Descripción	Responsabilidad
Jefe de proyecto	Define, planifica i controla el proyecto.
Director de proyecto	Supervisa el trabajo del alumno.
Analista	Colabora con el jefe de proyecto en el estudio de viabilidad i la planificación. Analiza la aplicación.
Programador	Diseña i desarrolla la aplicación de acuerdo con el análisis y la planificación.
Becario	Participa en todas las fases del proyecto y ayuda a los responsables.

Tabla 3 Recursos del proyecto

2.4 – Estudio situación actual

- ***Contexto***

Actualmente la empresa UNIT4 dispone de un Web Server codificado con el lenguaje de programación Visual Basic que se considera que cada vez se está quedando más desfasado. Debido al lenguaje de programación existen una serie de limitaciones en el momento de instalar el servidor: la necesidad de estar en una máquina con Windows de cómo máximo 32 bits y tener en funcionamiento un IIS de Microsoft. A parte se considera que el rendimiento del servidor actual no es todo lo bueno que debería ser y tiene un coste elevado.

- ***Lógica del sistema***

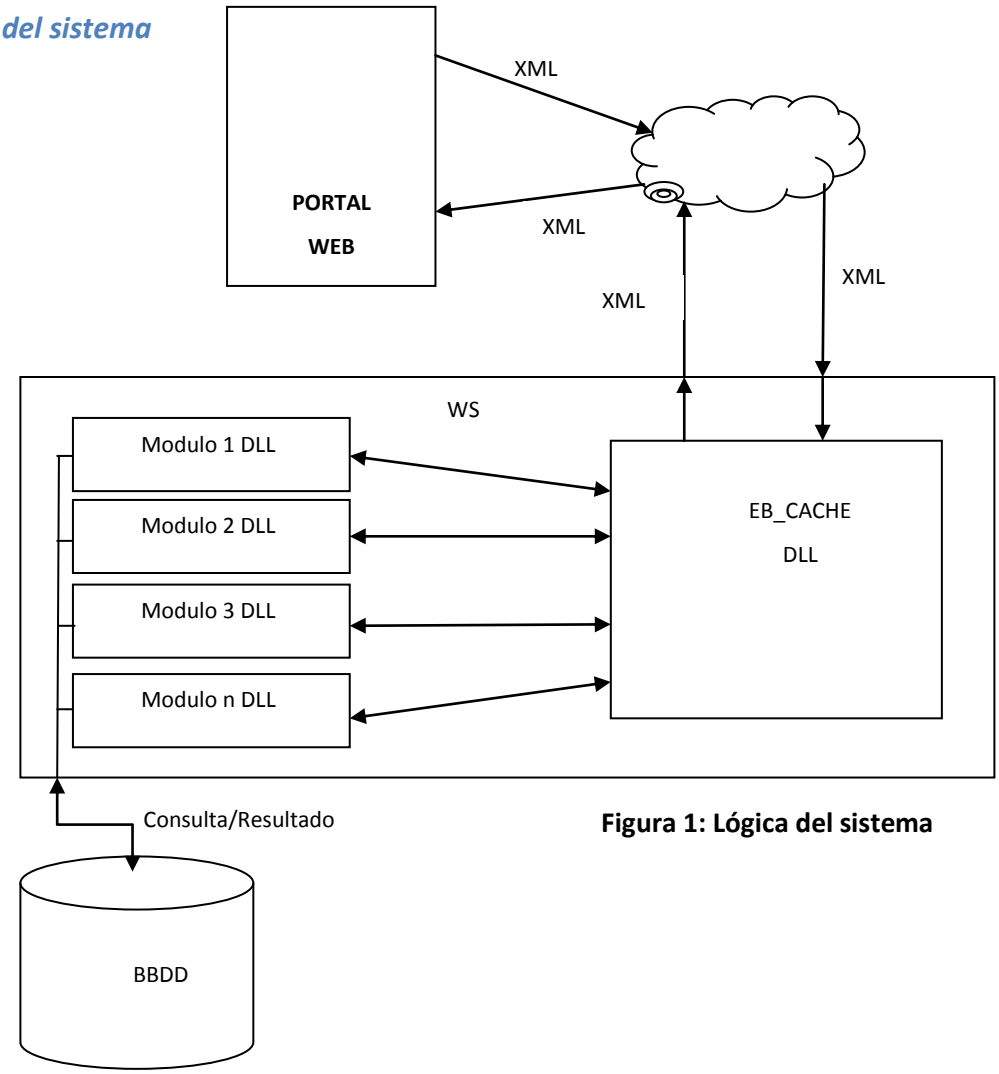


Figura 1: Lógica del sistema

En el esquema de la figura 1 podemos ver cuál es la lógica del sistema actual la cual se explicará más detalladamente a continuación.

Cuando un cliente realiza una consulta en el portal web, se codifica y se envía en formato XML a través de la red hasta llegar al servidor.

Dentro del servidor todo se divide en diferentes módulos que tienen sus funciones determinadas. El módulo ***"EB_CACHE"*** es un dispatcher, un distribuidor, encargado de recibir el archivo XML con la consulta que se desea hacer con los parámetros de la consulta. El módulo leerá los parámetros y los extraerá. Una vez obtenidos todos los datos, decidirá a qué módulo se debe hacer la llamada, consultas, inserciones u otros módulos, y enviará los datos a ese módulo para que realice el trabajo que se haya demandado.

Los otros módulos son los módulos que tienen lógica de negocio. Serán los encargados de realizar el trabajo con la BBDD o de transformar resultados. Cada módulo tiene su función de forma que queda todo separado. Una vez el módulo tenga el resultado, ya sea el resultado correcto o reporte de un error, éste se envía al módulo ***"EB_CACHE"*** que realizará las transformaciones necesarias para enviar a través de la red el resultado en formato XML y hacérselo llegar al cliente del portal web.

- ***Descripción física***

La descripción física de este sistema es bastante simple. Solo consta de un PC cliente que envía datos a través de la red. El servidor recibe los datos, trabaja con ellos, consulta o modifica la BBDD y después retorna la información al cliente.

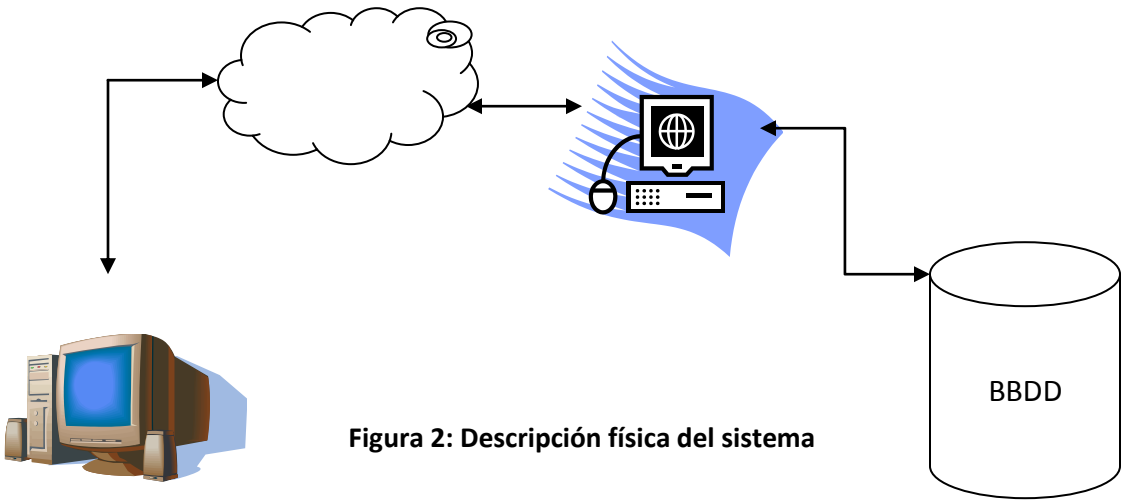


Figura 2: Descripción física del sistema

- ***Usuarios y/o personal del sistema***

En la siguiente tabla podemos ver de forma general el personal del sistema que se va a desarrollar con una breve descripción de lo que podrán realizar.

Descripción	Responsabilidad
Desarrolladores	Personas que hacen mantenimiento, mejoran o programan el servidor.
Clientes	Personas que trabajan con la aplicación y necesitan información de la BBDD.

Tabla 4 Personal del sistema

- ***Diagnóstico del sistema actual***

En la siguiente tabla podemos ver un diagnóstico del sistema que podemos encontrar actualmente en la empresa con las deficiencias que se han diagnosticado y las posibles mejoras que se esperan obtener al realizar el proyecto.

Deficiencias
- Lenguaje VB obsoleto y desfasado. - Muy dependiente de Microsoft. - No es multiplataforma. - Sólo funciona con IIS (Internet information server, Microsoft). - No funciona en arquitecturas de 64 bits. - Tiene un coste elevado.
Mejoras
- Lenguaje Java nuevo y más potente que además tiene actualizaciones periódicas. - Mejoras del rendimiento mediante lenguaje nuevo y pool de conexiones. - Multiplataforma (que funcione independientemente del sistema operativo). - Más económico, ya que todo funcionará mediante programas de libre distribución.

Tabla 5 Diagnóstico del sistema actual

2.5 – Requisitos funcionales

A continuación veremos un listado con una breve descripción de los requisitos funcionales.

- **RF1 Integración en Java.** Es fundamental que la aplicación este totalmente integrada en el lenguaje de programación Java, para ayudarnos a mejorar la aplicación, y por que la política de la empresa es llevarlo todo hacia Java. Ya se han hecho aplicaciones para clientes en Java y por lo tanto es necesario hacerlo así para una correcta sincronización de todos los sistemas.
- **RF2 Control de acceso de usuarios.** Se debe tener en cuenta un control de los usuarios ya que en la BBDD hay información de carácter confidencial, de esta forma hace falta que para acceder a esta información sea necesario identificarse.
- **RF3 Comunicación con el portal web**. Es indispensable tener una buena comunicación con el portal web de donde se reciben las peticiones para evitar pérdidas de información.
- **RF4 Consultar información en BBDD**. El módulo de consultas a BBDD se considera indispensable para este proyecto.
- **RF5 Inserciones de información en la BBDD.** El módulo de hacer inserciones se considera por motivos de tiempo secundario.
- **RF6 Modificaciones/Eliminaciones en la BBDD.** Son los módulos encargados de modificar registros en la BBDD y se consideran secundarios.
- **RF7 Creación de pool de conexiones**. Un pool de conexiones es una forma de evitar que un usuario deba abrir una conexión con la BBDD cada vez que desea algo de ella. La creación de un pool de conexiones es necesaria para poder mejorar el rendimiento del servidor.

- **RF8 Control de todas las acciones hechas en el sistema (log).** Las acciones hechas en el sistema como conexión al servidor, conexión a la BBDD, registro de errores, desconexiones e información de diferentes procesos quedarán guardadas en un archivo de texto “log”.

2.6 – Requisitos no funcionales

- **RNF1 Tolerancia a errores.** Tener un sistema capaz de registrar y visualizar los errores de forma que nos ayude mas tarde a encontrar donde está el fallo y corregirlo.
- **RNF2 Cumplimiento de la normativa.** Es necesario el cumplimiento de las normativas de proyectos tanto de UNIT4 y de la UAB.
- **RNF3 Mejorar rendimiento.** Se debe intentar mejorar el rendimiento del servidor respecto al actual.

2.7 – Restricciones del sistema

Las principales restricciones a la hora de desarrollar el proyecto son:

- La aplicación se ha de desarrollar en Java íntegramente. Es la política de la empresa ya que otros productos ya están desarrollados en java y así se podría integrar perfectamente junto a los otros productos.
- Para el desarrollo de la aplicación se ha de usar Eclipse Helios, Apache Tomcat, Axis2 y como base de datos SQL server 2005. Estas son las herramientas que se utilizan en la empresa para desarrollar aplicaciones.
- La aplicación se ha de adaptar al sistema físico de la empresa de forma que el cliente no tenga la impresión de que se haya hecho ningún cambio en el sistema. El cambio ha de ser totalmente transparente.
- El proyecto ha de estar finalizado el 28 de junio de 2011.

2.8 – Catalogación y priorización de los requisitos

- ***Requisitos funcionales***

En la tabla siguiente podemos ver una catalogación de los requisitos funcionales de forma que podemos ver cuáles son esenciales, condicionales u opcionales.

	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RF8
Esencial	X			X				
Condicional			X		X		X	
Opcional		X				X		X

Tabla 6 Catalogación de los requisitos funcionales

- ***Requisitos no funcionales***

En la tabla siguiente podemos ver una catalogación de los requisitos no funcionales de forma que podemos ver cuáles son esenciales, condicionales u opcionales.

	RNF1	RNF2	RNF3
Esencial		X	X
Condicional			
Opcional	X		

Tabla 7 Catalogación de los requisitos no funcionales

- ***Relaciones entre requisitos y objetivos***

En la siguiente tabla se puede ver cómo se relacionan los objetivos de este proyecto con los requisitos funcionales y no funcionales que hemos comentado en el apartado anterior.

	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RF8	RNF1	RNF2	RNF3
O1	X			X	X	X				X	X
O2	X										X
O3				X			X			X	X
O4					X		X			X	X
O5						X	X			X	X
O6	X	X	X	X	X	X	X	X	X		X
O7	X			X	X	X					X

Tabla 8 Relaciones entre requisitos y objetivos

2.9 – Alternativas y solución propuesta

En este proyecto no se han barajado diferentes propuestas ya que es un proyecto cerrado y ya estaba definido que la única solución era la migración del servidor a Java, debido a que las aplicaciones que maneja la empresa están programadas en Java y se considera que para una mejor integración de todos los sistemas todo debería estar desarrollado con un mismo lenguaje de programación.

2.10 – Planificación del proyecto

En este apartado podremos ver un calendario general del proyecto, un listado de los recursos del proyecto, las fases que se llevan a cabo en el desarrollo del proyecto con un calendario mas especifico y las horas que se han planificado para cada fase, un diagrama para ver la planificación de forma gráfica, un listado de los posibles riesgos que hay y sus posibles soluciones con un plan de contingencia.

- ***Calendario general***

En la siguiente tabla se puede ver una breve descripción de cuál ha sido el calendario del proyecto.

Calendario del proyecto	El proyecto se llevará a cabo a partir del mes de noviembre de 2010 hasta el mes de junio de 2011 con una dedicación de 20 horas semanales
Fecha de inicio	8 de noviembre de 2010
Fecha de finalización	17 de junio de 2011
Previsión de horas	558 horas

Tabla 9 Calendario general

- ***Recursos del proyecto***

Estos son los recursos del proyecto con sus correspondientes costes.

Recursos Humanos	Coste
Director de Proyecto	**150 €**
Jefe de proyecto	**55€**
Analista	**45€**
Programador	**35€**
Becario	**5€**

Tabla 10 Recursos del proyecto

- ***Calendario con fases.***

Calendario de fases con las horas y fechas de inicio y fin.

	Fase	Inicio	Finalización	Duración
1	**Inicio del proyecto**			**2h**
2	- Asignación y matriculación	8/11/2010	8/11/2010	2h
3	**Formación**			**120h**
4	- Comprensión de entorno y aprendizaje	8/11/2010	23/12/2010	120h
5	**Planificación**			**30h**
6	- Estudio de viabilidad	23/12/2010	12/01/2011	16h
7	- Aprobación EV	12/01/2011	12/01/2011	1h
8	- Plan de proyecto	12/01/2011	17/01/2011	12h
9	- Aprobación PP	17/01/2011	17/01/2011	1h
10	**Modulo base**			**107h**
11	- Análisis de la aplicación base	18/01/2011	21/01/2011	16h
12	- Diseño de la aplicación base	24/01/2011	27/01/2011	16h
13	- Desarrollo de la aplicación base	28/01/2011	21/02/2011	66h
14	- Test y pruebas jmeter	21/02/2011	23/02/2011	8h
15	- Aprobación	23/02/2011	23/02/2011	1h
16	**Modulo consultas**			**105h**
17	- Análisis del módulo consultas	23/02/2011	1/03/2011	16h

18	- Diseño del módulo consultas	1/03/2011	7/03/2011	16h
19	- Desarrollo del módulo consultas	7/03/2011	29/03/2011	64h
20	- Test y pruebas jmeter	29/03/2011	31/03/2011	8h
22	- Aprobación	31/03/2011	31/03/2011	1h
23	**Módulo de inserciones**			**89h**
24	- Análisis del módulo inserciones	1/04/2011	5/04/2011	12h
25	- Diseño del módulo inserciones	6/04/2011	8/04/2011	12h
26	- Desarrollo del módulo inserciones	11/04/2011	6/05/2011	56h
27	- Test y pruebas jmeter	9/05/2011	10/05/2011	8h
28	- Aprobación	11/05/2011	11/05/2011	1h
29	**Otros módulos**			**61h**
30	- Análisis módulos secundarios	11/05/2011	13/05/2011	8h
31	- Diseño módulos secundarios	13/05/2011	17/05/2011	8h
32	- Desarrollo módulos secundarios	17/05/2011	31/05/2011	40h
33	- Test y pruebas jmeter	31/05/2011	1/06/2011	4h
34	- Aprobación	1/06/2011	1/06/2011	1h
35	**Implantación**			**16h**
36	- Instalación	1/06/2011	2/06/2011	4h
37	- Pruebas reales	2/06/2011	6/06/2011	8h
38	- Formación usuarios	6/06/2011	7/06/2011	4h
39	**Generación de documentos**	**7/06/2011**	**15/06/2011**	**20h**
40	**Cierre de proyecto**	**15/06/2011**	**16/06/2011**	**4h**
41	**Defensa del proyecto**	**16/06/2011**	**17/06/2011**	**4h**

Tabla 11 Calendario de fases

A continuación se puede ver un gráfico de la metodología a la hora de planificar el proyecto, y en la figura 4 el diagrama de Gantt para poder ver la planificación que se ha hecho. Se sigue la metodología en espiral. Esto es a causa de que esta aplicación es modular y no se construyen todos los módulos a la vez, si no que se van construyendo por separado. Esto quiere decir que cada vez que vayamos a desarrollar uno de los diferentes módulos se tendrá que realizar cada una de las fases de análisis, diseño, desarrollo, pruebas y aprobación. También quiere decir que lo que hayamos desarrollado en un momento nos servirá para el desarrollo del posterior módulo.

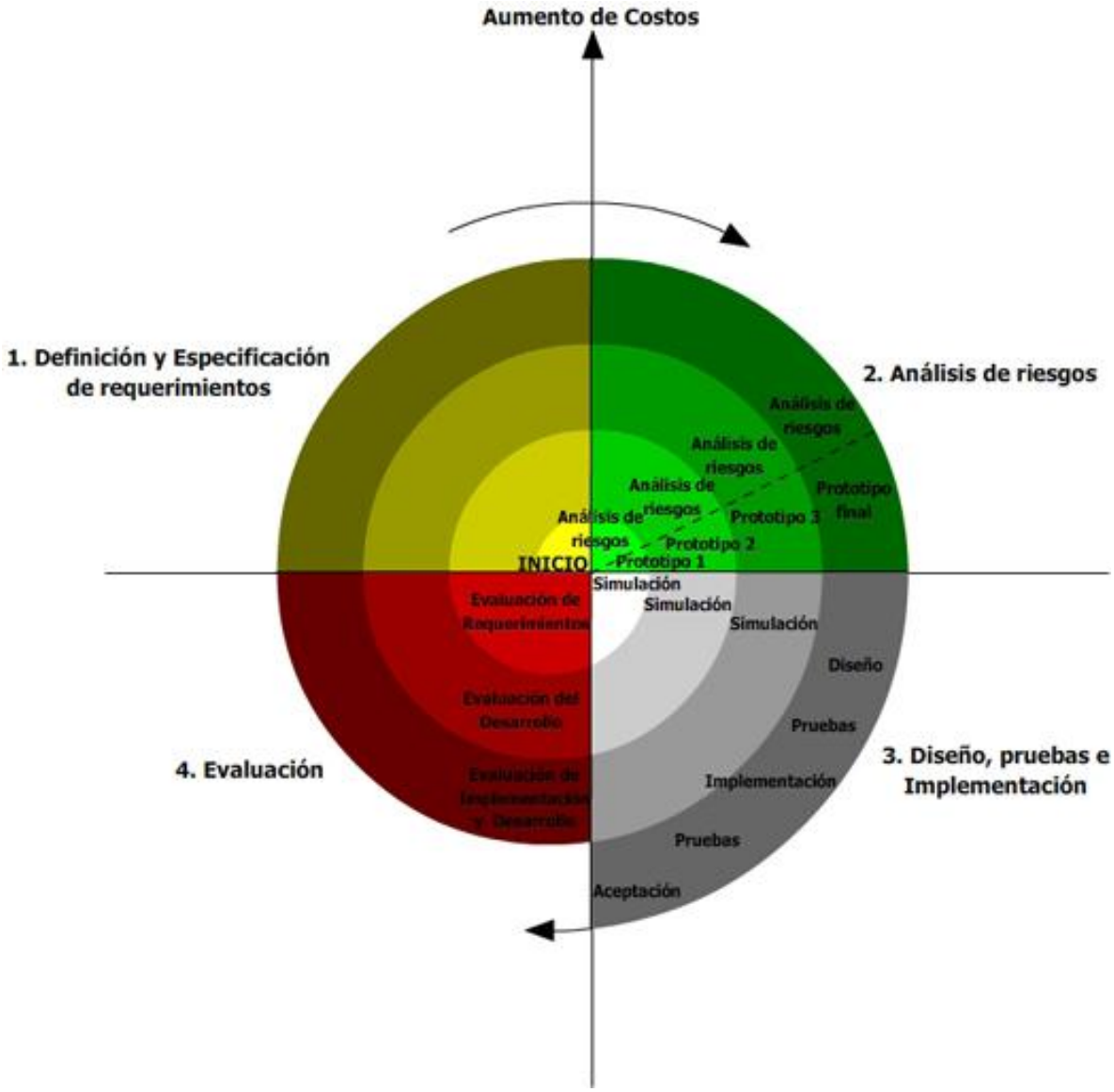


Figura 3 Gráfico metodología temporal en espiral

Figura 4 Diagrama de Gantt

- ***Lista de riesgos y catalogación***

En la siguiente tabla podemos ver los principales riesgos que se pueden encontrar a la hora de desarrollar el proyecto con las probabilidades de que ocurran y el impacto que tienen.

Nombre	Descripción	Probabilidad	Impacto
R1	Planificación temporal optimista. El proyecto no finaliza en la fecha prevista.	Alta	Critico
R2	Falta alguna tarea necesaria en la planificación. No se cumplen los objetivos.	Alta	Critico
R3	Presupuesto pequeño, disminuye la calidad.	Alta	Critico
R4	Pocos recursos, se retrasa la finalización del proyecto.	Alta	Critico
R5	No se hacen correctamente las pruebas, la calidad disminuye.	Alta	Critico
R6	Incumplimiento de alguna normativa, repercusiones legales.	Baja	Critico
R7	Abandono del proyecto antes de la finalización, pérdidas económicas.	Baja	Catastrófico

Tabla 12 Riesgos del proyecto

- ***Plan de contingencia***

En la tabla vemos las posibles soluciones a los riesgos mencionados en la tabla 12.

	Solución
R1	Retrasar alguna funcionalidad. Hacer un seguro. Asumir pérdidas económicas.
R2	Revisar bien el plan del proyecto y modificarlo.
R3	Renegociar con el cliente y asumir posibles pérdidas económicas.
R4	Pedir un posible retraso del proyecto, negociar con el cliente.
R5	Diseñar pruebas con anterioridad, negociar un contrato de mantenimiento.
R6	Consultar a un experto, asumir repercusiones penales.
R7	No tiene solución.

Tabla 13 Plan de contingencia

2.11 – Presupuesto

En esta sección del estudio de viabilidad se encuentra la valoración de costes referentes al proyecto. Podremos encontrar una valoración del coste del personal y una del coste de los recursos. También podremos ver un desglose del proyecto viendo cual es el coste de cada una de las fases del proyecto. Finalmente encontraremos la valoración del coste total que se atribuye a este proyecto.

- ***Estimación de coste del personal.***

Recurso	Horas	Coste
Director de proyecto	0,75 horas	112,14 €
Jefe de proyecto	75,48 horas	4151,19 €
Analista	62,79 horas	2825,50€
Programador	303,056 horas	10606,94 €
Becario	303,33 horas	1516,67 €

Tabla 14 Coste personal

- ***Valoración del coste de los recursos.***

  Dos PC de 1200 € con un coste de amortización de 100 €.

  Microsoft Office para la documentación con un precio de 250 € y un coste de amortización de 22 €.

  No es necesario nada más porque el software que se utilizará en la planificación, diseño y desarrollo son de libre distribución y no conllevan un coste asociado.

  Por tanto el coste total asociado a los recursos es de 222 €.

- ***Desglose coste fases del proyecto.***

Fase	Horas	Coste
Inicio del proyecto	2 horas	85 €
Formación	120 horas	3220 €
Planificación	30 horas	1900 €
Módulo Base	107 horas	3300,22 €
Módulo consultas	105 horas	3100 €
Módulo Inserciones	89 horas	2375,12 €
Otros Módulos	61 horas	2055 €
Implantación	16 horas	1500,09 €
Generación de documentos	20 horas	1200 €
Cierre de proyecto	4 horas	300,01 €
Defensa de proyecto	4 horas	190 €

Tabla 15 Desglose de costes de fases

- Resumen del coste total.

  El coste asociado al proyecto es de 19.225,44 € y el coste de los recursos es de 222 € por lo que el coste total asociado a este proyecto es de:

  19.447,44 €

2.12 – Conclusiones viabilidad

Puede parecer que el coste de este proyecto es elevado pero los beneficios que podemos obtener son importantes, ya que al terminar el proyecto el servidor será multiplataforma, es decir no será necesaria la tecnología Microsoft, esto hace que los pagos por licencias de Visual Basic, Windows, IIS dejen de ser necesarios, por lo que este servidor será más fácil de vender a los clientes nuevos sin ningún coste añadido y poder recuperar más fácilmente la inversión hecha en su desarrollo.

En cuanto a la viabilidad del proyecto los únicos inconvenientes que podemos atribuir a este proyecto es el tiempo que se pierde en realizarlo y que los desarrolladores no podrán realizar otros proyectos de la empresa y el periodo de formación para los usuarios. Por otro lado los beneficios son muchos tales como la mejora del rendimiento, la introducción de un lenguaje de programación nuevo y con perspectivas de mejora, una relación perfecta entre el servidor nuevo y las aplicaciones desarrolladas por la empresa que deben conectarse al servidor y como se ha dicho anteriormente más barato por la liberación del pago de licencias por herramientas de pago.

Por todo esto podemos decir que este proyecto es viable.

CAPÍTULO 3: Diseño del sistema

3.1 – Introducción.

En este capítulo vamos a poder ver el diseño de la aplicación que se va a desarrollar, veremos la arquitectura de la aplicación, un diagrama donde podrá verse cuál es el flujo de datos que tiene el sistema, diagramas con todas las clases que deben implementarse en la etapa de desarrollo y cómo se relacionan entre ellas. Finalmente podremos ver una visión global de todo el sistema.

Esta etapa de diseño es muy importante ya que si la realizamos de manera correcta y obtenemos un buen diseño podremos obtener el esqueleto principal de la aplicación que deberemos rellenar con el código necesario y además puede ahorrarnos mucho tiempo en la etapa de desarrollo ya que la base de la aplicación estará bien definida.

3.2 – Tecnología de desarrollo.

A continuación podremos ver una breve explicación de las tecnologías que se utilizarán a la hora de desarrollar esta aplicación. Al ser un proyecto cerrado en la empresa, no se pudo hacer una valoración de alternativas a las tecnologías elegidas.

Aun así procederemos a comentarlas de forma que se tenga una idea más clara de lo que se llegará a utilizar en la implementación del proyecto.

- **Java**

  Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems en los años 90. Este lenguaje de programación fue liberado en su mayor parte, de forma que ahora es software libre.

El lenguaje Java se creó bajo cinco objetivos principales:

- Debería usar la metodología de la programación orientada a objetos.
- Debería ser multiplataforma.
- Debería incluir por defecto soporte para trabajo en red.
- Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
- Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

Ventajas:

- Java es un lenguaje multiplataforma.
- Es un lenguaje diseñado para que los programas se puedan transferir y ejecutar con total seguridad.
- Java es un lenguaje dinámico. La máquina virtual de java enlaza los programas java en tiempo de ejecución, eliminando la necesidad de enlazarlos con las librerías en tiempo de compilación.
- Eliminación del acceso directo a memoria y de la aritmética de punteros.
- La orientación a objetos: Una de las ventajas que supone empezar desde el principio se traduce en el hecho de que Java ha sido creado inicialmente como lenguaje orientado a objetos. Teniendo en cuenta que la orientación a objetos es tal vez la tendencia más popular en el desarrollo de software moderno.
- Es un lenguaje preparado para la red. La máquina virtual de Java evita que los programas transferidos al ordenador lo destruyan, facilita la rápida transferencia de dichos programas y los ejecuta de modo que no dependan del sistema operativo subyacente.
- Java tiene la característica de que libera la memoria automáticamente, por lo cual uno no se tiene que preocupar por la pérdida de ésta.

Desventajas:

- Al ser un lenguaje interpretado y tener la necesidad de la máquina virtual para interpretarlo resulta ser más lento que otro tipo de lenguajes.
- Es un lenguaje de programación y tiene una curva de aprendizaje algo complicada.

- **XML**

XML significa **eXtensible Markup Language**, o lenguaje de anotación extensible. XML no es un lenguaje para hacer mejores páginas web, sino un lenguaje para información auto-descrita, o al menos, auto-descrita si las etiquetas están bien puestas. XML no es un lenguaje para crear páginas web, nos permite estructurar la información que deseamos mostrar y después aplicar fácilmente transformaciones para su posterior presentación. Lo más común es que la información esté almacenada en una base de datos, se obtenga esta información y se convierta a XML y luego se transforme para servirlo al cliente.

El lenguaje XML funciona básicamente a base de etiquetas, cada elemento tiene dos etiquetas una que indica el comienzo del elemento y otra para indicar el final. Dentro de cada elemento se pueden poner varios elementos, de forma que podemos tener un elemento llamado “Libros” y después en el interior tener varios elementos llamados “Libro” con sus respectivos elementos como nombre o descripción. De esta forma y si están bien colocadas todas las etiquetas en el documento tendremos un documento XML bien formado y de esta forma tendremos la información bien organizada.

- **Tomcat**

  Tomcat es el servidor Web más utilizado a la hora de trabajar en entornos web con Java. Tomcat es una implementación completamente funcional de los estándares de JSP y Servlets. Tomcat también puede especificarse como el controlador de las peticiones de JSP y servlets recibidos por servidores Web comunes, como el servidor Apache HTTP de la Fundación de software de Apache o el servidor Microsoft Internet Information Server (IIS). Tomcat está integrado en la implementación de referencia Java 2 Enterprise Edition (J2EE) de Sun Microsystems.

- **SQL Server 2005**

  Microsoft SQL Server es un sistema para la gestión de bases de datos producido por Microsoft basado en el modelo relacional. Sus lenguajes para consultas son T-SQL y ANSI SQL. Microsoft SQL Server constituye la alternativa de Microsoft a otros potentes sistemas gestores de bases de datos como son *Oracle*, *PostgreSQL* o *MySQL*.

3.3 – Arquitectura de la aplicación.

A continuación se explicará cuál es la arquitectura que seguirá la aplicación que procederemos a crear. En los apartados siguientes se verá de forma gráfica mediante diagramas de flujo de datos y diagramas de clases para una mayor comprensión.

La aplicación se ha diseñado de forma modular. De esta forma se ha dividido en varios módulos pequeños para una implementación más sencilla.

El Web Server se divide en un módulo principal llamado ***"EB_CACHE"*** y varios módulos donde serán implementados los diferentes Web Services que queramos desarrollar para nuestro servidor.

"EB_CACHE" es el módulo que recibirá la petición del cliente para realizar el servicio deseado. Este módulo no tiene ninguna lógica de negocio, simplemente es un distribuidor. En cuanto se reciba la petición, que ha de tener una estructura determinada, el módulo analizará la petición y realizará la llamada al servicio que corresponda a la operación que haya pedido el cliente.

Los servicios estarán implementados en pequeños módulos de forma que cada servicio estará en un módulo individual distinto. Por ejemplo un módulo para el servicio de consultas a BBDD o un módulo de inserciones BBDD. En este módulo, dependiendo de los parámetros que se habrán recibido de la petición del cliente, se hará una configuración del servicio a partir de unos ficheros XML que se encontrarán en el servidor. En estos ficheros XML se encuentran las configuraciones de los servicios a falta de introducir parámetros. Por lo tanto en estos módulos, en los que si hay lógica de negocio, se configurará el servicio mediante el fichero XML y los parámetros, se realizará la operación correspondiente como podría ser una consulta o una inserción a BBDD y se retornará el resultado al módulo EB_CACHE. Este módulo finalmente retornará el resultado final al cliente.

3.4 – Diagramas de flujo de datos.

A continuación podremos ver de forma gráfica la arquitectura de la aplicación y cuál es el flujo de datos del sistema.

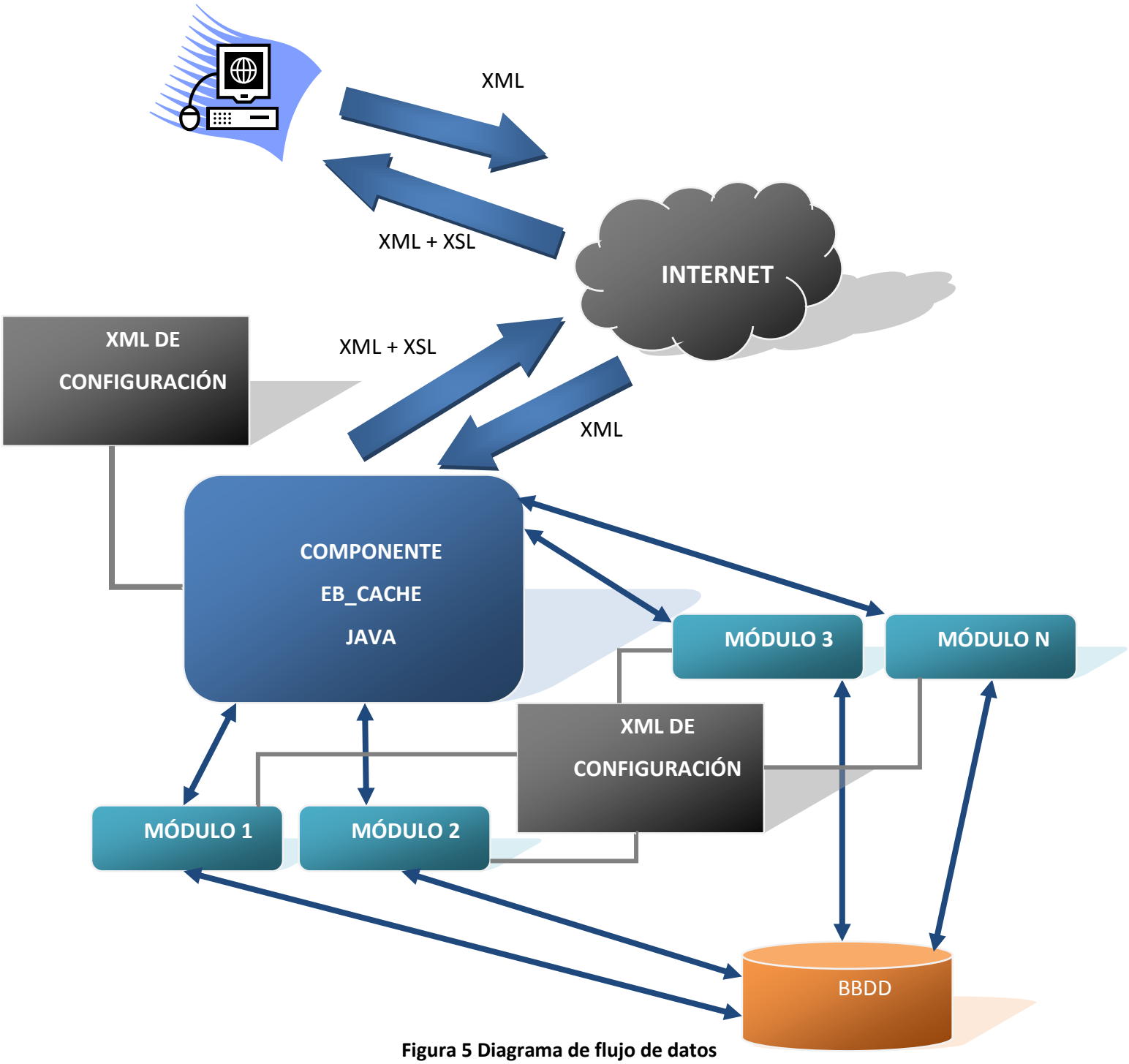


Figura 5 Diagrama de flujo de datos

3.5 – Diagrama de clases.

A continuación podremos ver el diagrama de clases de la aplicación. Se podrá ver el diagrama de clases del módulo "EB_CACHE", el módulo de consultas y el módulo de inserciones. Después de cada diagrama tendremos una breve explicación para poder entender las funciones y parámetros de cada clase.

MÓDULO EB_CACHE

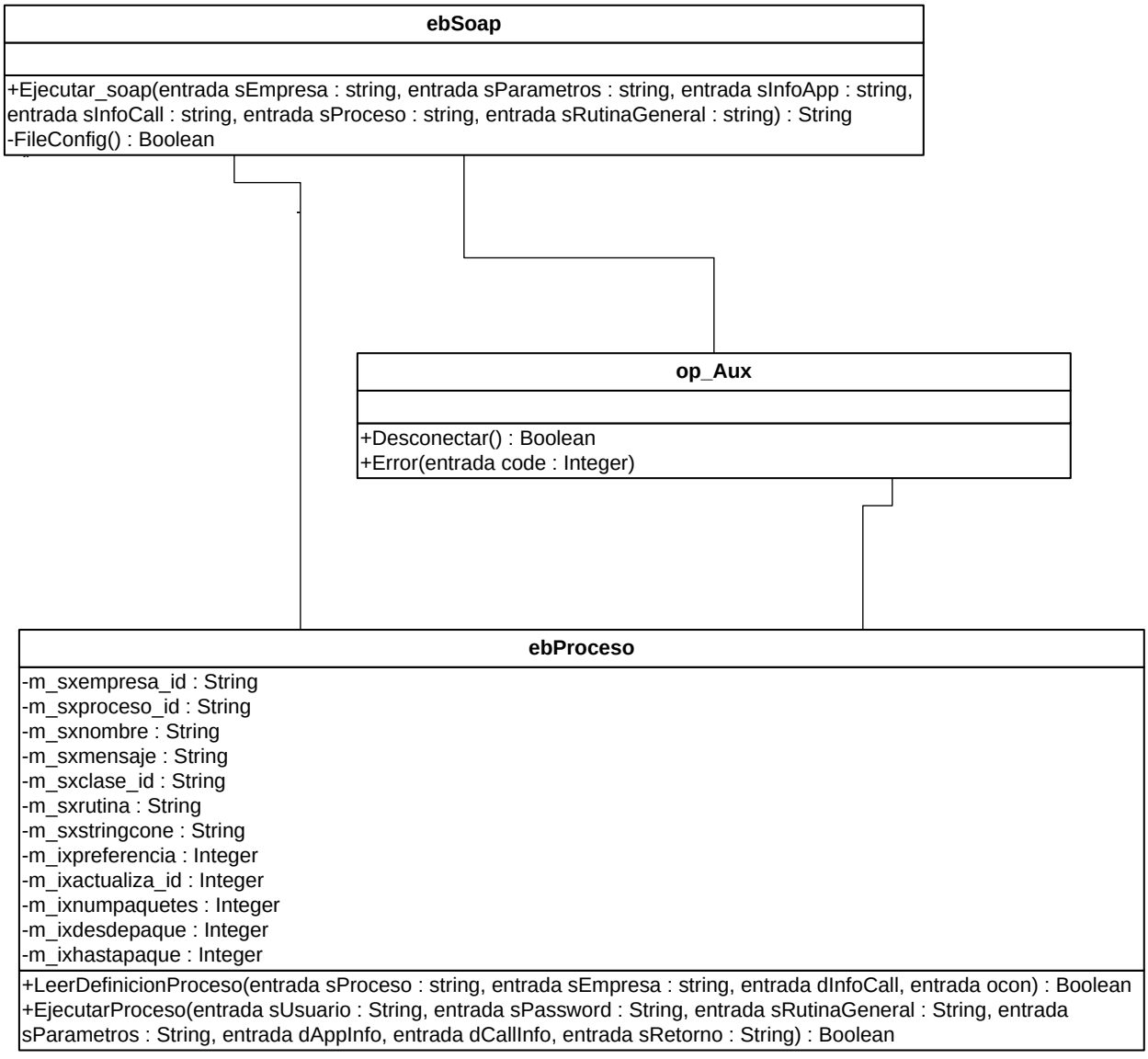


Figura 6 Clases EB_CACHE

Clases de EB_CACHE:

Clase ebSoap: Esta clase es la encargada de recibir la petición del cliente. Espera a recibir la llamada a la función Ejecutar_soap y a continuación analiza la petición, que si es correcta realiza la llamada a las funciones de la clase ebProceso.

- **Función Ejecutar_soap:**

  Esta función se encarga de recibir los parámetros de la consulta y de hacer las correspondientes llamadas para la configuración del servicio y la ejecución del proceso.

  Parámetros:

  Parámetro sEmpresa (String): Nombre de la empresa.

  Parámetro sParametros (String): Parámetros del proceso al que llamará el web server.

  Parámetro sInfoApp (String): Información asociada a la aplicación que ejecuta la llamada. Usado internamente en la fase de implantación para su depuración.

  Parámetro sInfocall (String): Información asociada a la llamada. Usado internamente en la fase de implantación para su depuración.

  Parámetro sProceso (String): Nombre del proceso al que se quiere hacer la llamada.

  Parámetro sRutinaGeneral (String): Rutina a la que se debe llamar.

- **Función FileConfig:**

  FileConfig busca el fichero motorintegracion.xml de configuración y lo deja cargado en la variable doc en caso de no encontrarlo da un mensaje de error.

  Retorno existe (boolean): Retorna True si existe el fichero y false en caso contrario.

Clase ebProceso: Esta clase se encarga de configurar el servicio correspondiente y ejecutarlo haciendo la llamada de la clase del servicio correspondiente de una forma dinámica.

- **Función LeerDefinicionProceso:**

  Esta función mira cual es el Proceso demandado y configura la aplicación extrayendo los datos de motorintegración.xml donde podemos encontrar la configuración de los servicios disponibles.

  Nota: Los archivos XML han de estar codificados en UTF-8, sino java no lo reconocerá.

  Parámetro sProceso (string): Nombre del proceso al que se quiere hacer la llamada.

  Parámetro sEmpresa (string): Nombre de la empresa.

  Parámetro dInfoCall (Document): Documento donde se guarda información de la llamada.

  Retorno LeerDefinicionProceso (boolean): Retorna True si a leído bien la definición del proceso a realizar, en caso contrario retorna False.

- **Función EjecutarProceso:**

  Función que ejecuta el servicio que se ha configurado. Crea una objeto dinámicamente, el objeto dependerá de m_sxclaseid que será la variable que nos dirá a que clase debemos ir.

  Parámetro sUsuario (string): Nombre Usuario.

  Parámetro sPassword (string): Contraseña del usuario.

  Parámetro sRutinaGeneral (string): Rutina a la que se debe llamar.

  Parámetro sParametros (string): String con los parámetros del cliente.

  Parámetro dAppInfo (Document): Documento utilizado para depuración.

Parámetro dCallInfo (Document): Documento utilizado para depuración.

Parámetro sRetorno (string): Resultado de ejecutar el servicio.

Retorno EjecutarProceso (boolean): Si se ejecuta el proceso correctamente retorna True en caso contrario False.

La clase op_Aux se comentará más adelante ya que es una clase común para todos los módulos.

MÓDULO newConsultaSQL

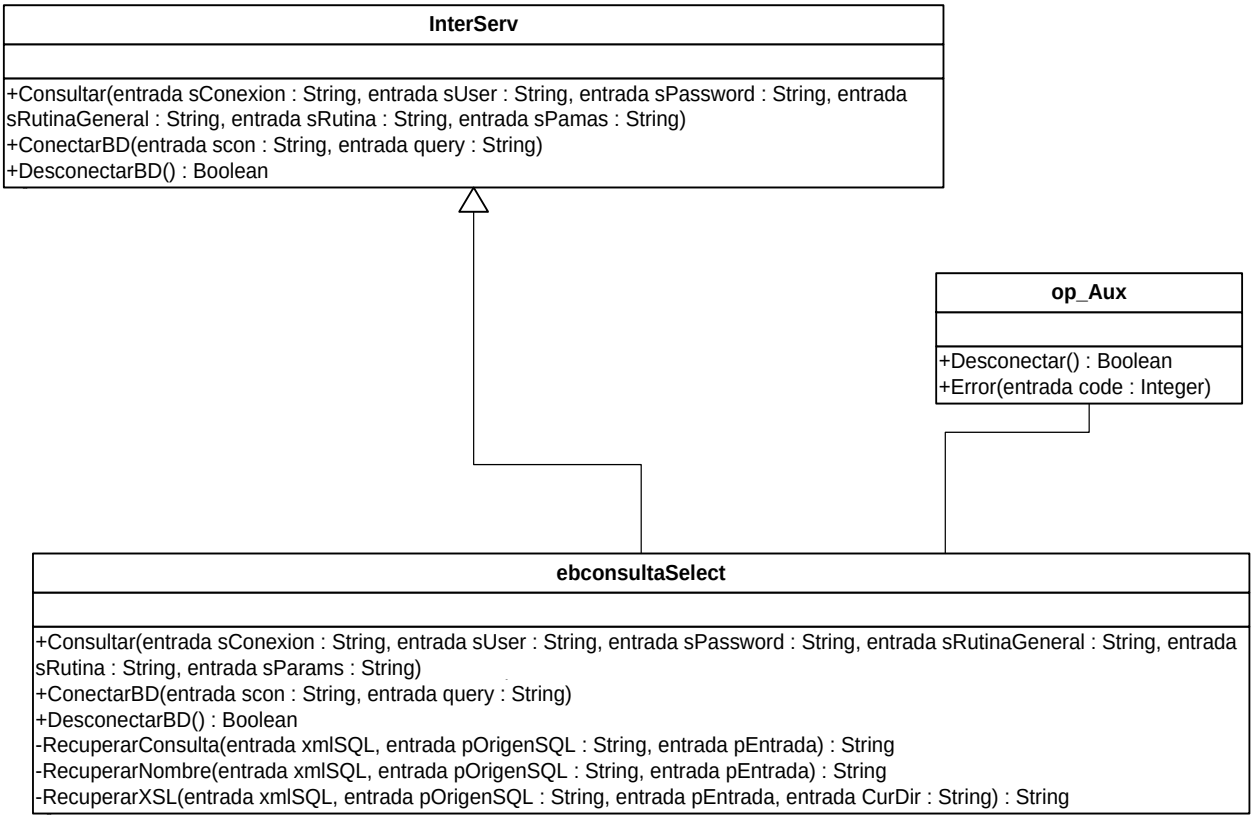


Figura 7 NewConsultaSQL

InterServ es una Interface de Java dónde se describen las cabeceras de las funciones principales que deberán tener todos los servicios del Web Server. Esto sirve para que en posibles desarrollos posteriores de servicios todos sigan un patrón similar y puedan funcionar correctamente en el servidor ya que las funciones que se ponen en la Interface son las estrictamente necesarias para ello. Una vez el servicio nuevo haya seguido el patrón de la interface el desarrollador podrá implementarlas como quiera y añadir todas las funciones que desee.

Clase ebConsultaSelect: Esta clase es la encargada de configurar la consulta que se hará a BBDD. Obtendrá los parámetros de la consulta, buscará la consulta en el archivo XML de configuración y rellenará los campos de la consulta con los parámetros que el cliente haya introducido en su petición. Finalmente conectará con la BBDD, realizará la consulta y construirá un archivo XML con el resultado. Una vez obtenido el resultado se tendrá que mirar el archivo de configuración por si hay que aplicar algún archivo XSL para la presentación del resultado.

- **Función Consultar**:

  Esta función será la encargada de ejecutar la consulta en la BBDD.

  Parámetro sConexion (String): String de conexión a la BBDD.

  Parámetro sUser (String): Usuario que ejecuta la consulta.

  Parámetro sPassword (String): Contraseña del usuario que ejecuta la consulta.

  Parámetro sRutinaGeneral (String): Rutina a la que se debe llamar.

  Parámetro sRutina (String): Ruta del archivo de configuración del servicio.

  Parámetro sParams (String): Parámetros de la consulta.

  Retorno Salida (Document): Documento XML con la consulta realizado.

- **Función ConectarBD:**

  Esta función crea la conexión con la BBDD.

  Parámetros scon (String): Cadena con la URL de conexión.

- **Función DesconectarBD:**

  Esta función cierra la conexión con la BBDD.

  Parámetro con (Connection): Conexión a la BBDD.

- **Función recuperarConsulta**:

  Esta función obtendrá la sentencia SQL para la consulta que se debe hacer.

  Parámetro xmlSQL (Document): Documento donde están los datos de la configuración del servicio.

  Parámetro pOrigenSQL (String): ruta dónde se encuentra el XML de configuración del servicio con la sentencia.

  Parámetro pEntrada (Document): XML con los parámetros de la consulta.

  Retorno sSentencia (String): Cadena con la sentencia SQL.

- **Función recuperarNombre:**

  Esta función recupera el nombre en el idioma deseado de la consulta que se mostrará en el resultado.

  Parámetro xmlSQL (Document): Documento donde están los datos de la configuración del servicio.

  Parámetro pOrigenSQL (String): Ruta donde se encuentra el fichero XML.

  Parámetro pEntrada (Document): Parámetros dados por el cliente.

  Retorno nombre (String): Retorna el nombre de la consulta a realizar.

- **Función recuperarXSL:**

  Esta función recupera la ruta del fichero XSL.

  Parámetro xmlSQL (Document): Documento donde están los datos de la configuración del servicio.

  Parámetro pOrigenSQL (String): Ruta donde se encuentra el fichero XML.

  Parámetro pEntrada (Document): Parámetros dados por el cliente.

  Parámetro curDir: Directorio donde se encuentran los ficheros de configuración del servicio.

  Retorno XSL: Se recupera el fichero XSL para la transformación del resultado de la consulta.

MÓDULO newInsertSQL

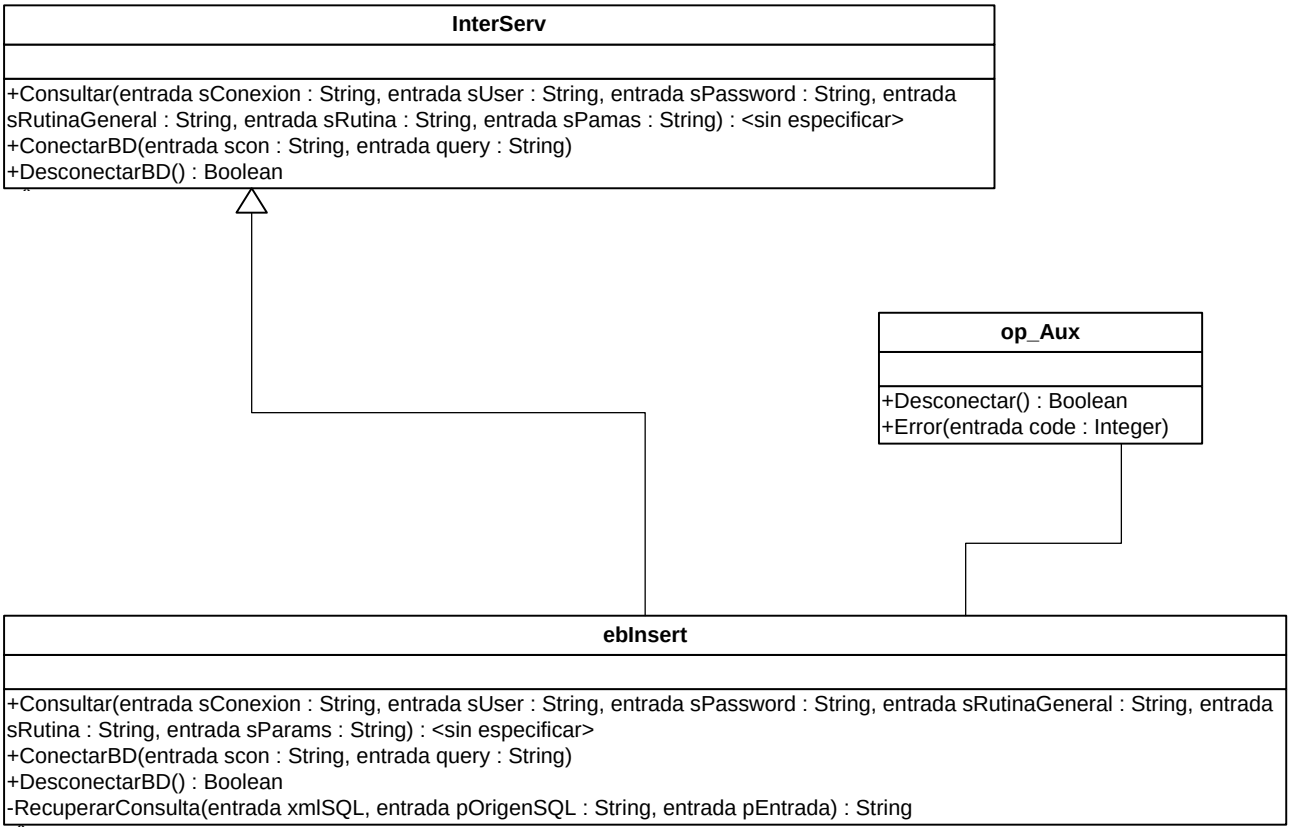


Figura 8 newInsertSQL

En este módulo podemos ver también la interface InterServ que tiene el mismo cometido que en el módulo de consultas y por eso podemos ver que las clases de este servicio tienen la misma estructura que el servicio de consultas.

Clase ebInsertSelect: Esta clase es la encargada de configurar la sentencia SQL para poder realizar una inserción en la BBDD. Primero obtendrá la sentencia, la configurará con los datos que se deban insertar, una vez obtenida la sentencia SQL definitiva conectará y realizará la inserción, y si el proceso se ha completado correctamente se recibirá un fichero XML avisando que la inserción de datos se ha hecho de forma correcta y en caso de no se correcto un XML con el error.

- **Función Consultar**:

  Esta función será la encargada de configurar y ejecutar la inserción de datos a la BBDD.

  Parámetro sConexion (String): String de conexión a la BBDD.

  Parámetro sUser (String): Usuario que realiza la inserción.

  Parámetro sPassword (String): Contraseña del usuario que realiza la inserción.

  Parámetro sRutinaGeneral (String): Rutina a la que se debe llamar.

  Parámetro sRutina (String): Ruta del archivo de configuración del servicio.

  Parámetro sParams (String): Parámetros de la inserción.

  Retorno Salida (Document): Documento XML con información del proceso realizado.

- **Función ConectarBD:**

  Esta función crea la conexión con la BBDD.

  Parámetros scon (String): Cadena con la URL de conexión.

- **Función DesconectarBD:**

  Esta función cierra la conexión con la BBDD.

  Parámetro con (Connection): Conexión a la BBDD.

- **Función recuperarConsulta**:

  Esta función obtendrá la sentencia SQL necesaria para realizar la inserción en la BBDD.

  Parámetro xmlSQL (Document) Documento donde están los datos de la configuración del servicio.

  Parámetro pOrigenSQL (String): ruta donde se encuentra el XML de configuración del servicio con la sentencia.

  Parámetro pEntrada (Document): XML con los parámetros de la consulta.

  Retorno sSentencia (String): Cadena con la sentencia SQL.

Como hemos podido observar, los dos servicios que se han diseñado, consultas e inserciones, tienen la misma estructura gracias a la interface InterServ que nos sirve de plantilla para implementar servicios, de esta forma si hacemos que los módulos de servicios sigan una estructura prefijada anteriormente, cada vez que queramos desarrollar un módulo nuevo no tendremos que adaptar el Web Server ya que serán el resto de módulos nuevos los que se tendrán que adaptar a la aplicación principal.

La clase **Op_aux** es una clase para poner operaciones y funciones auxiliares que puedan ser comunes para toda la aplicación.

Tiene una función llamada **Error** que recibe un código dado dependiendo de la parte de la aplicación que llame a esta función. De esta forma obtenemos un documento XML con información sobre cuál ha sido el error, con un breve descripción y su código para posteriormente poder realizar las correcciones necesarias.

También hay una función llamada Desconectar que únicamente sirve para terminar los procesos una vez el servicio se ha realizado de forma correcta.

CAPÍTULO 4: Implementación

4.1 – Introducción.

En esta sección vamos a detallar el proceso y los métodos que se han llevado a cabo durante el desarrollo de la aplicación. Detallaremos cual es el entorno de desarrollo que hemos decidido utilizar y también dividiremos la implementación en desarrollo Java y desarrollo XML que son los dos lenguajes de programación utilizados.

En este apartado de Implementación es donde damos forma a todo lo que se ha diseñado en el apartado anterior (Diseño) y donde se tendrán más problemas, ya que en el momento del desarrollo vemos los errores que hemos podido realizar en las fases anteriores y hay que corregirlos y además podemos encontrar que alguna cosa planificada o diseñada llega a ser imposible de realizar.

4.2 – Entorno de desarrollo.

Antes de empezar a escribir código debemos preparar el entorno en el que realizaremos el desarrollo de nuestra aplicación.

La mayor parte de la aplicación se desarrolla en el lenguaje de programación java y se ha decidido utilizar **Eclipse Helios**, ya que facilita mucho la tarea del programador avisando de los posibles errores que realizamos durante la implementación del código y además facilita la compilación y la ejecución de las aplicaciones que se desarrollan mediante un simple clic. Además se ha configurado de forma que se pueda trabajar mediante Eclipse con el servidor Tomcat 5.5 y con Axis2, de esta forma podemos hacer las pruebas de los webservices desde el propio Eclipse.

También se ha instalado el servidor **Tomcat 5.5** y **Axis2** que es el motor de servicios web de Apache. Con esto instalado conseguimos tener preparada la máquina para poder hacer funcionar los servicios web desde el servidor para su utilización.

Además para desarrollar los ficheros XML de la configuración de servicios web se utilizará el programa Notepad++, un programa libre desde el cual se puede escribir cualquier tipo de código.

Finalmente se instalará Microsoft SQL Server 2005 para crear una pequeña base de datos con una tabla para poder hacer las pruebas de consultas de los servicios que se van a desarrollar.

Una vez instalado y configurado todo el software se podrá empezar a desarrollar la aplicación.

4.3 – Desarrollo Java.

En este apartado se comentará la parte del desarrollo de la aplicación que se realizará en Java. Como se ha comentado anteriormente el grueso de la aplicación es en Java. Los diferentes módulos que se comentan en anteriores secciones como ***EB_CACHE*** o los módulos de consultas o inserciones, serán clases de Java.

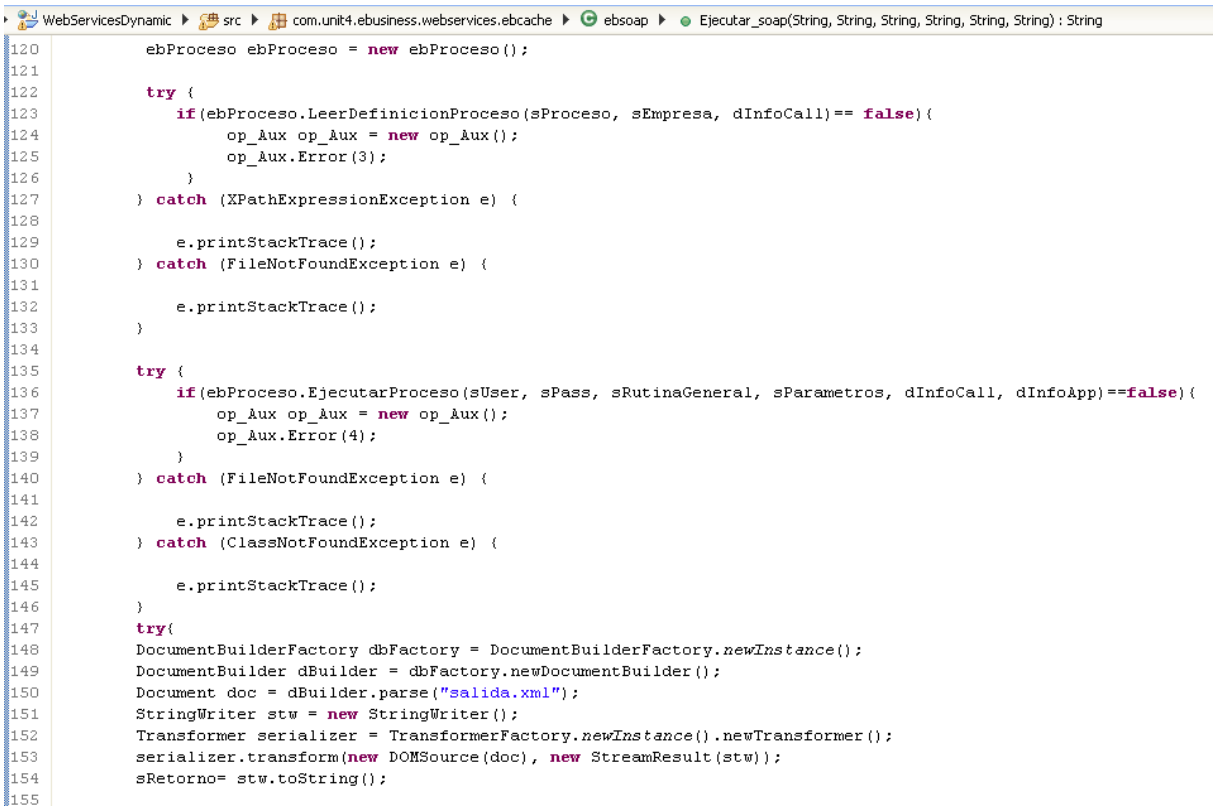
En la sección comentaremos cómo se ha ido implementando el código, problemas surgidos durante el desarrollo y además se comentaran librerías y operaciones que a priori son poco utilizadas y poco conocidas y que pueden resultar curiosas y muy útiles.

El módulo ***EB_CACHE*** es el módulo que recibe la petición del cliente. Este módulo se divide en dos clases, una es ***ebsoap*** que es propiamente la que recibe los parámetros y la otra es ***ebProceso*** que será la encargada de decidir a qué clase se deberá llamar para realizar la operación deseada.

Figura 9 Cabecera de ebsoap

En la figura 8 podemos ver un ejemplo del código de la clase ***ebsoap*** donde se encuentra la cabecera y podemos ver los parámetros necesarios para el método implementado en la clase, que se llama ***Ejecutar_soap***. Los parámetros necesarios en realidad son ***sEmpresa***, ***sParametros*** y ***sProceso***. Estos tres parámetros son indispensables para poder realizar una operación, los otros tres parámetros (***sInfoApp***, ***sInfocall*** y ***sRutinaGeneral***) son para los programadores y se utilizan para depuraciones de la aplicación. A lo largo de toda la aplicación estos parámetros aparecen en varias ocasiones, pero la aplicación no los necesita para su correcto funcionamiento.

Cuando se hace la llamada al método ***Ejecutar_soap*** podemos ver en la figura 8 cómo se hace una llamada a una función llamada ***FileConfig,*** con la que se comprueba si existe o no el fichero XML llamado motorintegracion.xml donde se encuentra toda la configuración. En el caso de no existir se lanzaría un error.




```
120          ebProceso ebProceso = new ebProceso();
121
122          try {
123             if(ebProceso.LeerDefinicionProceso(sProceso, sEmpresa, dInfoCall)== false){
124                  op_Aux op_Aux = new op_Aux();
125                  op_Aux.Error(3);
126              }
127         } catch (XPathExpressionException e) {
128
129             e.printStackTrace();
130         } catch (FileNotFoundException e) {
131
132             e.printStackTrace();
133         }
134
135         try {
136             if(ebProceso.EjecutarProceso(sUser, sPass, sRutinaGeneral, sParametros, dInfoCall, dInfoApp)==false){
137                 op_Aux op_Aux = new op_Aux();
138                 op_Aux.Error(4);
139             }
140         } catch (FileNotFoundException e) {
141
142             e.printStackTrace();
143         } catch (ClassNotFoundException e) {
144
145             e.printStackTrace();
146         }
147         try{
148         DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
149         DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
150         Document doc = dBuilder.parse("salida.xml");
151         StringWriter stw = new StringWriter();
152         Transformer serializer = TransformerFactory.newInstance().newTransformer();
153         serializer.transform(new DOMSource(doc), new StreamResult(stw));
154         sRetorno= stw.toString();
155
```

Figura 10 Ejemplo código ebsoap

Una vez hechas las comprobaciones de que se puede continuar con la ejecución de la aplicación, procedemos a crear un objeto de la clase ***ebProceso*** y realizamos la operación ***LeerDefinicionProceso*** y ***EjecutarProceso*** que están explicadas en el apartado de la clase ***ebProceso***. Una vez ejecutada la función ***EjecutarProceso*** obtenemos el resultado de la operación deseada y procedemos a convertir el resultado que se ha guardado en XML a una cadena para poder mostrar el resultado por pantalla.

Figura 11 Ejemplo código ebProceso

En la clase ebProceso existen dos funciones que son llamadas desde la clase ***ebsoap***, son ***LeerDefinicionProceso y EjecutarProceso.*** En la figura 10 podemos ver un ejemplo de código de la función ***LeerDefinicionProceso*** donde podemos ver como con la librería ***DocumentBuilder*** de java podemos cargar el documento XML de configuración para poder leer y escribir en él. Seguidamente extraemos los datos necesarios para realizar la operación demandada por el cliente mediante la librería ***XPATH.*** Con esta librería y sus funciones podemos extraer los datos de forma rápida ya que en una sola línea podemos poner los parámetros para acotar la búsqueda y obtener el resultado. Una vez obtenidos todos los datos necesarios esta función finaliza.

La siguiente función es ***EjecutarProceso,*** mediante la cual procedemos a hacer la llamada a la clase necesaria para realizar la operación (Consultas, Inserciones) de forma dinámica. Esto significa que no tenemos que realizar un bloque de elección dependiendo de la clase a la que se quiere llamar. En la siguiente figura se podrá ver el código en el que está implementada.

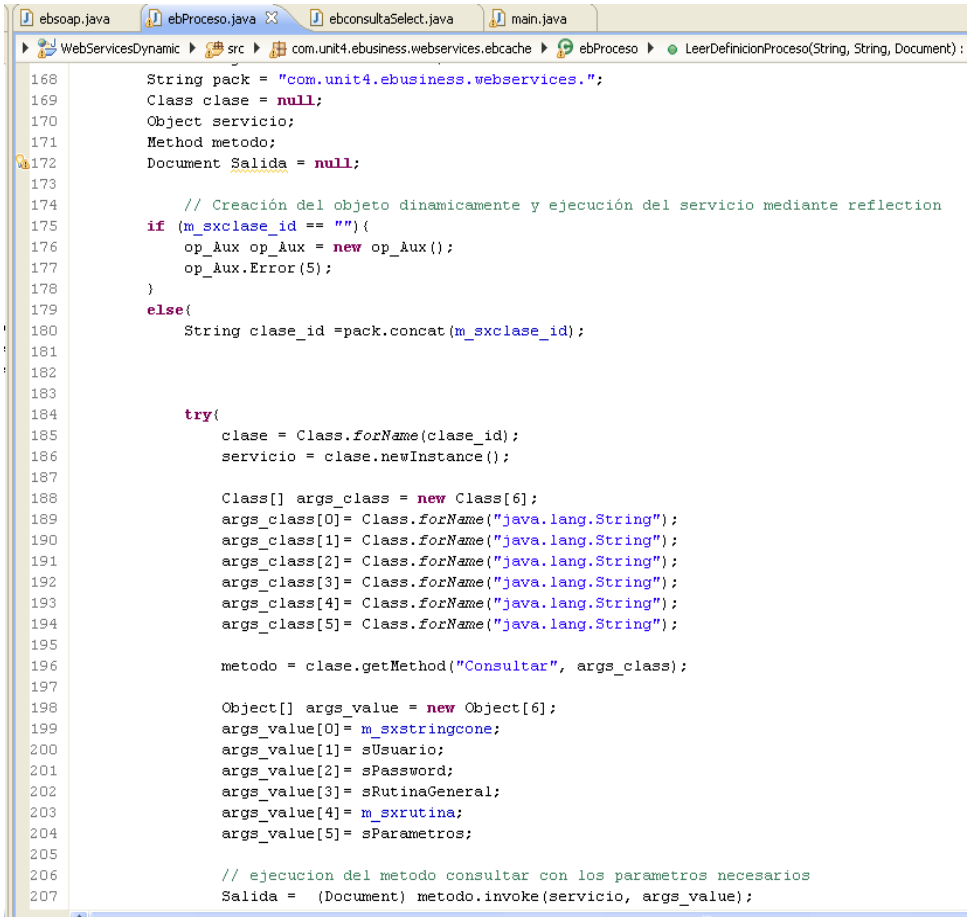


Figura 12 ebProceso llamada dinámica a otra clase

En la figura 11 podemos ver la llamada dinámica a otra clase. Con la librería ***reflect*** de java podemos conseguir que mediante los parámetros extraídos del fichero XML de configuración se llame a la clase necesaria. Vemos cómo con el parámetro ***clase_id*** extraído creamos una nueva instancia del objeto que se quiere crear. Una vez decididos los parámetros que se van a comunicar al nuevo objeto y especificado de qué tipo van a ser procedemos a configurar el método que va a ser llamado. Toda clase que especifica un servicio a realizar tiene un método llamado ***Consultar***, así que llamaremos a ese método. Finalmente asignamos los parámetros e invocamos el método. De esta forma hemos ejecutado una clase dinámicamente.

La librería ***reflect*** es poco utilizada y poco conocida, pero resulta muy útil ya que en este momento, aunque con solo dos posibles servicios implementados puede parecer un poco largo y complicado de implementar, en un web server totalmente construido, con muchos más servicios, nos ahorra el hecho de tener que construir casos para cada uno de ellos.

Con esto finalmente conseguimos el resultado de la operación demandada y la tenemos lista para que ***ebsoap*** pueda mostrarla.

Después de esto pasamos a los módulos donde se encuentran los servicios a realizar. Primero veremos una Interface java llamada ***InterServ*** en la que se estipulan cómo han de ser las cabeceras de tres funciones que deben implementar todos los servicios que se quieran desarrollar en este servidor. De esta forma nos aseguramos que a la hora de hacer llamadas al servicio exista el método predefinido al que se llamará.

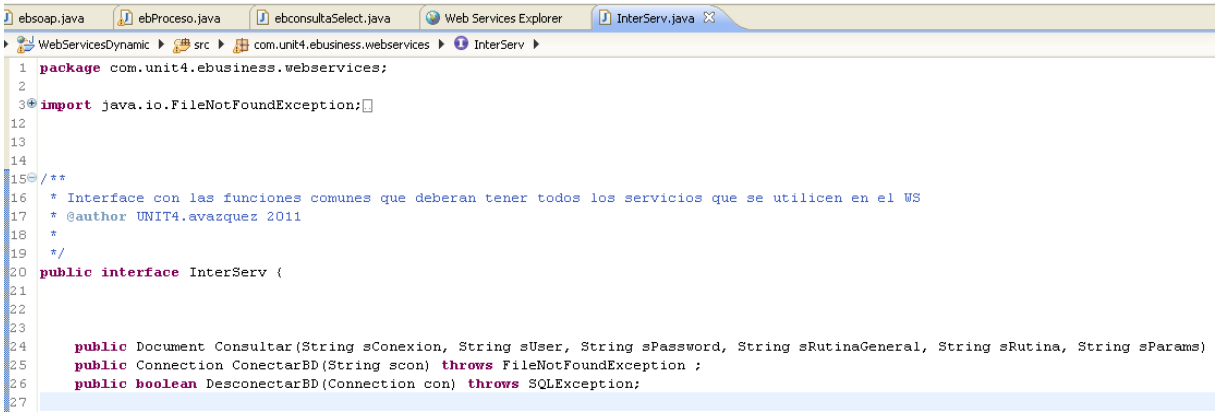


Figura 13 Código InterServ

Podemos ver tres funciones que son ***Consultar***, ***ConectarBD*** y ***DesconectarBD***. Estas tres funciones, como mínimo, deberán estar en todos los servicios que se quieran implementar.

Veremos primero la clase para realizar consultas a base de datos. Las clases de consultas y de inserciones son prácticamente idénticas, sólo se diferencian en pequeñas cosas.

La clase que realiza consultas se llama ***ebconsultaSelect***. La primera función que nos encontramos es Consultar como estaba implementado en la Interface de servicios.

Figura 14 Ejemplo ebconsultaSelect

Esta función, lo primero que hace es cargar los parámetros de la consulta que se quiere realizar y que el cliente nos ha dado. Una vez obtenidos los parámetros debemos obtener la consulta SQL, el nombre de la consulta y el fichero XSL para la posterior conversión del resultado obtenido. Como podemos ver en la figura 13 esto lo conseguimos con tres funciones: ***recuperarConsulta***, ***recuperarNombre*** y ***recuperarXSL***. Cada servicio tiene un fichero XML de configuración que, mediante los parámetros y datos obtenidos de motorintegracion.xml, podemos obtener la ruta donde se encuentra y todo lo necesario. Estas operaciones son similares a la forma de obtener datos de un XML que se llevaron a cabo en la clase ***ebProceso***.

Una vez obtenidos los datos, podemos proceder a realizar la consulta. Primero debemos abrir una conexión a la base de datos, que se realiza con la función ***ConectarBD,*** y una vez obtenida la conexión ya podemos realizar la consulta cuyo resultado guardaremos en una variable. Después de obtener el resultado, se realiza la construcción del fichero XML trabajando con las filas y columnas de cada registro para saber que ***tags*** hay que poner en el fichero son sus respectivos valores.

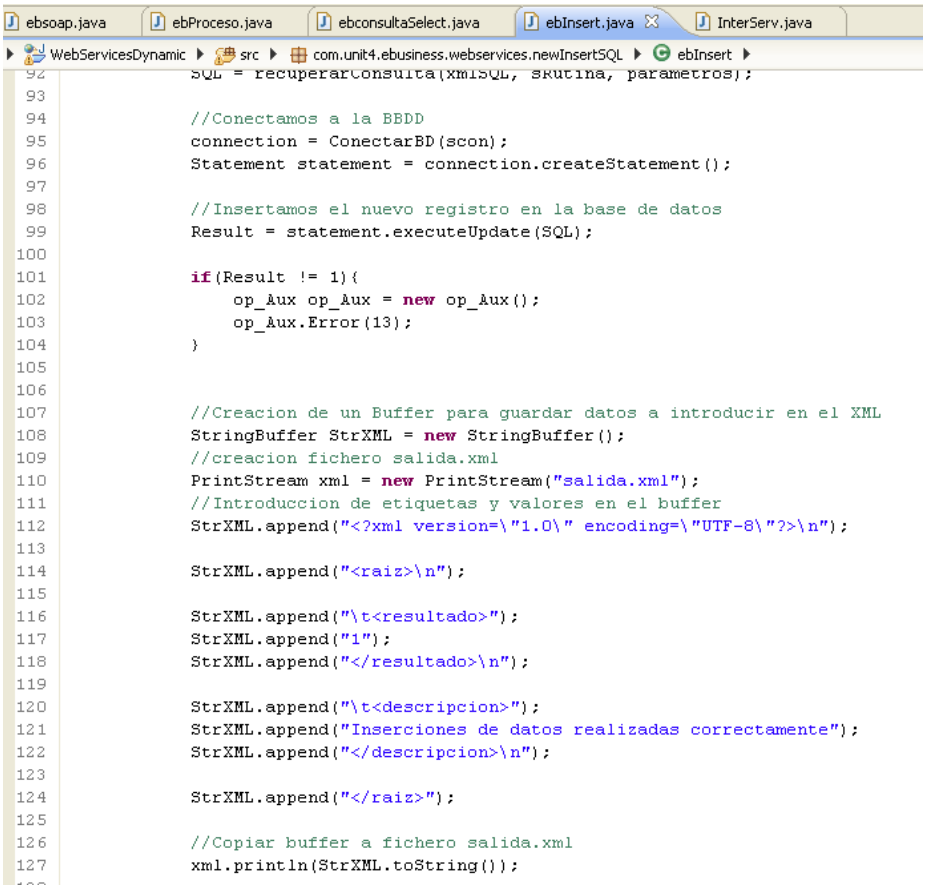
Figura 15 ebConsultaSelect

En la figura 14 podemos observar la construcción del XML de salida. Primero se concatena todo en un ***string*** con la cabecera y los ***tags*** necesarios y, registro a registro, lo vamos añadiendo a la cadena. Una vez finalizada la concatenación lo imprimimos todo en un fichero XML temporal llamado temp.xml. Se utiliza un archivo temporal por qué seguidamente procederemos a transformar el archivo XML mediante el fichero XSL para su presentación. De esta forma podemos evitar conflictos, y una vez obtenido el fichero salida.xml eliminaremos el archivo temporal.

Con esto hemos realizado la consulta. A continuación se retorna primero a la clase ***ebProceso*** y éste lo retorna a la clase ***ebsoap*** que finalmente la mostrará al cliente que hizo la petición.

Finalmente queda la clase ***ebInsert*** dedicada a realizar inserciones en la base de datos. La mecánica de esta clase es prácticamente idéntica y tiene las mismas funciones que la clase de consultas.

Las mayores diferencias que se pueden ver se encuentran a la hora de recuperar la consulta ya que en el fichero de configuración de las inserciones existe una consulta base en la que los parámetros que queremos introducir a la base de datos han de ser añadidos. Para una mayor facilidad en la consulta base, en el lugar donde deben ir estos parámetros se ha señalizado con el símbolo ***"#"*** de esta forma en nuestra función podemos localizar la parte de la consulta donde ha de ir el parámetro y introducirlo. De esta forma podemos completar una consulta que ya está medio hecha en lugar de tener que construirla desde el inicio.




```
92              SQL = recuperarConsulta(xmlSQL, sRutina, parametros);
93
94              //Conectamos a la BBDD
95              connection = ConectarBD(scon);
96              Statement statement = connection.createStatement();
97
98              //Insertamos el nuevo registro en la base de datos
99              Result = statement.executeUpdate(SQL);
100
101             if(Result != 1){
102                 op_Aux op_Aux = new op_Aux();
103                 op_Aux.Error(13);
104             }
105
106
107             //Creacion de un Buffer para guardar datos a introducir en el XML
108             StringBuffer StrXML = new StringBuffer();
109             //creacion fichero salida.xml
110             PrintStream xml = new PrintStream("salida.xml");
111             //Introduccion de etiquetas y valores en el buffer
112             StrXML.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
113
114             StrXML.append("<raiz>\n");
115
116             StrXML.append("\t<resultado>");
117             StrXML.append("1");
118             StrXML.append("</resultado>\n");
119
120             StrXML.append("\t<descripcion>");
121             StrXML.append("Inserciones de datos realizadas correctamente");
122             StrXML.append("</descripcion>\n");
123
124             StrXML.append("</raiz>");
125
126             //Copiar buffer a fichero salida.xml
127             xml.println(StrXML.toString());
```

Figura 16 Ejemplo código ebInsert

En la figura 15 podemos ver la parte en la que se realiza la conexión con la base de datos y se realiza la inserción. Si al realizar la inserción, la operación devuelve un valor de 1 significará que la inserción ha tenido éxito y se procederá a construir un XML para comunicar al cliente que la inserción se ha hecho de forma correcta. En caso contrario se lanzaría un error y también se informaría al cliente de que se ha producido un error.

También se ha creado una clase llamada ***op_Aux*** en la que se codifican funciones auxiliares, de momento solo hay dos, que son ***Desconectar*** y ***Error***.

ebsoap.java | ebProceso.java | ebconsultaSelect.java | ebInsert.java | InterServ.java | op_Aux.java

WebServicesDynamic ▸ src ▸ com.unit4.ebusiness.webservices ▸ op_Aux ▸

```
30    public void Error(int code) throws FileNotFoundException{
31
32        String error = null;
33        switch (code){
34
35        case 1: error = "Fichero de configuracion no encontrado (motorintegracion.xml)"; break;
36        case 2: error = "Error al leer: sInfocall, sInfoApp";break;
37        case 3: error = "Error al leer la definicion del proceso en motorintegracion.xml";break;
38        case 4: error = "Error al ejecutar el proceso";break;
39        case 5: error = "No se puede continuar, falta conocer la clase a la que acceder";break;
40        case 6: error = "Error al ejecutar el sevicio";break;
41        case 7: error = "Error al conectar con la BBDD";break;
42        case 8: error = "Error en el proceso de conexion con la BBDD";break;
43        case 9: error = "Error al leer el fichero XML del servicio";break;
44        case 10: error = "Error, no se a podido recuperar la sentencia SQL";break;
45        case 11: error = "Error, no se a podido recuperar nada de la BBDD";break;
46        case 12: error = "Error, no se ha podido realizar la operacion del servicio requerido";break;
47        case 13: error = "Error, no se ha podido insertar la información en la base de datos";break;
48        }
49        //Creacion de un Buffer para guardar datos a introducir en el XML
50        StringBuffer StrXML = new StringBuffer();
51        //creacion fichero salida.xml
52        PrintStream xml = new PrintStream("retorno.xml");
53        //Introduccion de etiquetas y valores en el buffer
54        StrXML.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
55
56        StrXML.append("<raiz>\n");
57
58        StrXML.append("\t<resultado>");
59        StrXML.append("-1");
60        StrXML.append("</resultado>\n");
61
62        StrXML.append("\t<descripcion>");
63        StrXML.append(error);
64        StrXML.append("</descripcion>\n");
65
66        StrXML.append("\t<salida>");
```

Figura 17 Ejemplo código Op_Aux

En la figura 16 podemos ver un ejemplo de la función Error. Esta función es llamada desde algunos puntos de la ejecución de la aplicación y ayudan a la hora de hacer una depuración más rápida de la aplicación. Se crea un fichero XML con el código que describe cuál ha sido el error que se ha producido. Esto nos ayuda a encontrar donde se ha producido el error de forma más rápida y su posterior corrección.

En el futuro esta clase tendrá más funciones ya que irá creciendo a medida que sea necesario y de esta forma no habrá que codificarlas en cada clase y así los códigos de las otras clases no estarán tan cargados.

Estas son todas las clases que han sido desarrolladas en Java. A medida que sea necesario se irán implementando mas clases para cada servicio que se quiera introducir en este web server, siempre siguiendo la pauta de la interface creada para los servicios. Lo que nunca se debe modificar es el módulo ***EB_CACHE*** que es el módulo central que ya está completo.

4.4 – Desarrollo XML.

La parte de desarrollo en XML es menor que la parte de desarrollo en Java. Pero sigue siendo una parte muy importante del proyecto ya que todos los ficheros de configuración que se manejan en el web server están codificados en lenguaje XML y los datos que viajan de cliente-aplicación y aplicación-cliente también son ficheros XML. Es por eso que en la parte de la aplicación que se ha desarrollado en Java se ven tantas operaciones y funciones dedicadas a extracción de datos de un fichero XML y creación de ficheros XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<MotorIntegracion>
    <Empresa id='CCS'>
        <Proceso id='SERVCONS'>
            <Nombre>Consultas Laboro</Nombre>
            <Mensaje>Consultas ebusiness Laboro</Mensaje>
            <Clase>newConsultaSQL.ebconsultaSelect</Clase>
            <Actualiza></Actualiza>
            <Preferencia>1</Preferencia>
            <NumPaquetes>1</NumPaquetes>
            <DesdePaquetes>1</DesdePaquetes>
            <HastaPaquetes>1</HastaPaquetes>
            <Rutina>sqls\servicios_cons.xml</Rutina>
            <StringConexion>jdbc:odbc:jdbcodbcSQLServer,imp,ccsccs</StringConexion>
            <TipoSeguridad></TipoSeguridad>
        </Proceso>
        <Proceso id='SERVINS'>
            <Nombre>Inserción BBDD</Nombre>
            <Mensaje>Inserción Datos </Mensaje>
            <Clase>newInsertSQL.ebInsert</Clase>
            <Actualiza></Actualiza>
            <Preferencia>1</Preferencia>
            <NumPaquetes>1</NumPaquetes>
            <DesdePaquetes>1</DesdePaquetes>
            <HastaPaquetes>1</HastaPaquetes>
            <Rutina>sqls\servicios_ins.xml</Rutina>
            <StringConexion>jdbc:odbc:jdbcodbcSQLServer,imp,ccsccs</StringConexion>
            <TipoSeguridad></TipoSeguridad>
        </Proceso>
    </Empresa>
```

Figura 18 Ejemplo Motorintegracion.xml

En la figura 17 podemos ver el código del fichero XML motorintegracion.xml. Que es el que se usa para la configuración de la aplicación. De el podemos extraer datos que se comentarán a continuación.

Las principales etiquetas de este fichero que son necesarias para que la aplicación se ejecute correctamente, son:

- **Empresa:** Dentro de este fichero puede haber varias empresas y cada una de ellas con sus servicios diferentes. Es una forma de acotar el fichero para cada empresa.
- **Proceso id:** Esta etiqueta denota cual será el servicio a realizar y ha de ser especificado por el cliente para saber cuál será la operación.
- **Nombre:** Es el nombre que se le da al servicio y que será mostrado en el resultado de la operación.
- **Mensaje:** Aquí se puede poner lo que se desee mostrar al usuario cuando se haya realizado la operación demandada.
- **Clase:** Esta etiqueta es fundamental ya que indica a que clase java se ha de llamar para poder ejecutar el servicio. Sin esto la llamada dinámica de clases comentada en la sección anterior no podría funcionar.
- **Rutina:** En esta etiqueta se encuentra la ruta del fichero de configuración de cada uno de los servicios que se deberá utilizar en el momento que se ejecute el servicio.
- **StringConexion:** En esta etiqueta se encuentra cual será la cadena para la conexión a la base de datos, con la URL, el login y el password.

Estas etiquetas que se han comentado son las que se extraen en la clase ***ebProceso*** para posteriormente hacer la llamada a la clase del servicio solicitado.

A continuación se puede ver en la figura 18 un ejemplo de código del fichero de configuración del servicio de consultas.

```
<?xml version="1.0" encoding="UTF-8"?>
<raiz>
    <sql>
        <identificador>pruebas</identificador>
        <filexsl>\xml\xsl\msjconsulta.xsl</filexsl>
        <idiomas>
            <idioma>
                <nombre>es</nombre>
                <valor>PRUEBAS</valor>
            </idioma>
        </idiomas>
        <sentencia>
            <![CDATA[
                SELECT * FROM imp.pc_clientes;

            ]]>
        </sentencia>
    </sql>
    <sql>
        <identificador>pruebas_individual</identificador>
        <filexsl>xml\xsl\msjconsulta.xsl</filexsl>
        <idiomas>
            <idioma>
                <nombre>es</nombre>
                <valor>PRUEBAS INDIVIDUAL</valor>
            </idioma>
        </idiomas>
        <sentencia>
            <![CDATA[
                SELECT * FROM imp.pc_clientes
                WHERE (xnombre= #xnombre#);

            ]]>
        </sentencia>
    </sql>
```

Figura 19 Ejemplo servicios_cons.xml

En la clase ***ebconsultaSelect*** se busca este fichero para su correcta configuración. Las etiquetas nos muestran los datos necesarios para poder hacer una consulta a la base de datos:

- **Identificador:** Es el nombre que le damos a la consulta. Cada consulta tiene su propio identificador de forma que dependiendo de lo que queramos consultar en la base de datos podamos diferenciar una consulta de otra. Este identificador debe

ser especificado en los parámetros que introduce el cliente a la hora de hacer la consulta.

- **Filexsl:** Aquí encontramos la ruta del fichero XSL que se utilizará en la transformación del resultado obtenido de la consulta.
- **Idiomas:** Dentro de idiomas tenemos dos etiquetas que son nombre y valor, que se utilizan para indicar en qué idioma está y el nombre que se le ha dado a la consulta. De esta forma podemos introducir varios idiomas y hacer nuestra aplicación multilenguaje.
- **Sentencia:** Dentro de esta etiqueta podemos encontrar un campo ***CDATA***, que es un tipo de campo en el que se indica que si hay algún carácter especial que pudiese ser interpretado no debe ser interpretado. En este campo es donde se almacenan las consultas. En caso de que haya una consulta con parámetros, el parámetro debe estar contenido entre símbolos ***"#"*** para poder ser sustituido posteriormente en la aplicación.

```
<?xml version="1.0" encoding="UTF-8"?>
<raiz>
    <sql>
        <identificador>pruebas.inserciones</identificador>
        <filexsl>xsl\msjconsulta.xsl</filexsl>
        <idiomas>
            <idioma>
                <nombre>es</nombre>
                <valor>Pruebas de inserciones</valor>
            </idioma>
        </idiomas>
        <sentencia>
            <![CDATA[
                INSERT INTO imp.pc_clientes
                        (xcod_postal, xdomicilio,xnif, xnomabrev, xnombre, xpais, xper_fiscal, xpoblacion, xprovincia_id)
                VALUES
                (#xcod_postal#, #xdomicilio#, #xnif#, #xnomabrev#, #xnombre#, #xpais#, #xper_fiscal#, #xpoblacion#, #xprovincia_id#);
            ]]>
        </sentencia>
    </sql>
```

Figura 20 XML servicio inserciones

En la figura 19 podemos observar el fichero de configuración del servicio de inserciones. Como se puede observar es idéntico al fichero de configuración de consultas, pero podemos ver en el campo sentencia una consulta para realizar una inserción en la base de datos.

Finalmente en la figura 20 podemos ver el resultado de realizar una consulta después de su construcción en XML. Se puede ver cómo los datos se muestran con el correspondiente nombre del campo y su valor para poder identificar cada dato.

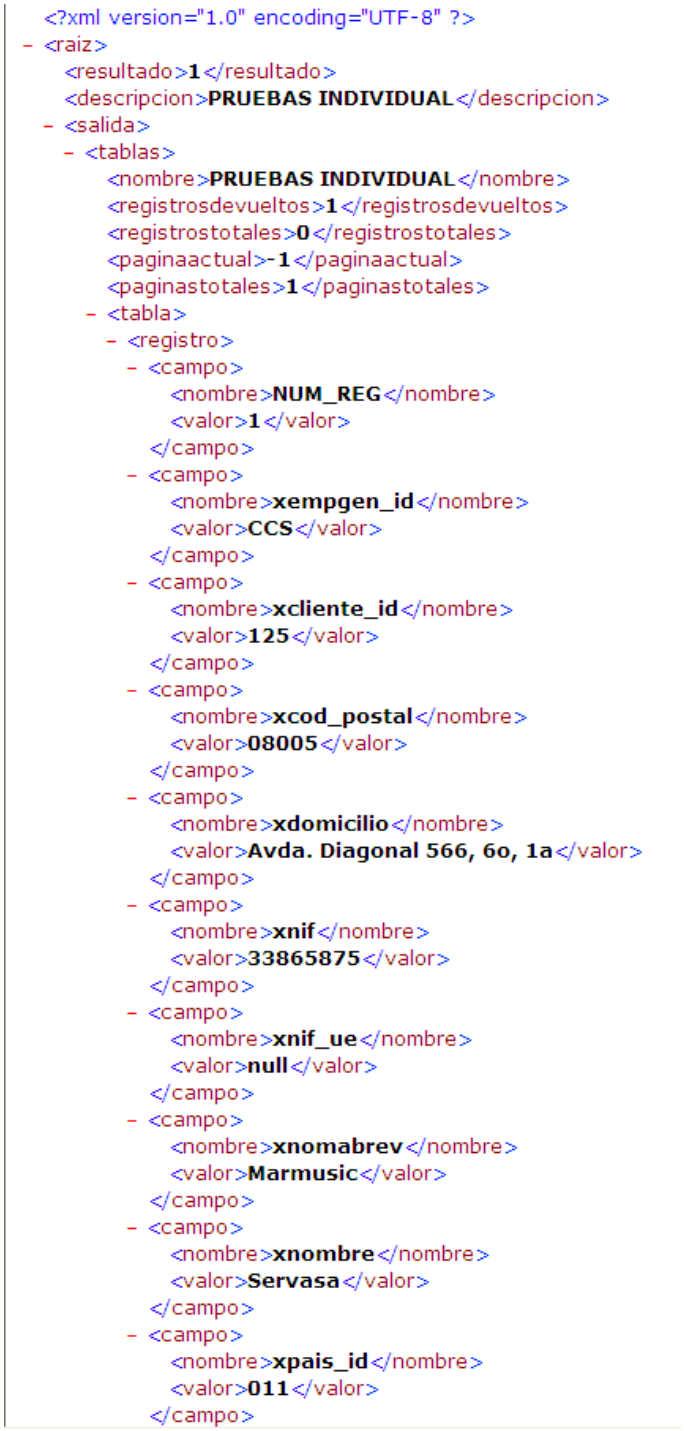


Figura 21 Ejemplo resultado consulta

Una vez está todo implementado tanto en Java como en XML sólo falta hacer las pruebas correspondientes para comprobar que todo funciona de forma correcta y a su vez detectar si se puede mejorar algo. En la siguiente sección se podrán ver cuáles han sido las pruebas realizadas y los resultados que se han obtenido.

CAPÍTULO 5: Pruebas

5.1 – Introducción.

En este capítulo podremos ver las pruebas realizadas a la aplicación. Esta es la última fase que se lleva a cabo en el desarrollo de aplicaciones, es una fase importante ya que se encuentran los posibles fallos que tenga la aplicación que no se hayan detectado durante el desarrollo para de esta forma lograr una aplicación de mejor calidad.

5.2 – Pruebas Unitarias.

Las pruebas unitarias son las que se realizan para comprobar el correcto funcionamiento de los diferentes módulos que tiene nuestra aplicación. Se han hecho diferentes pruebas de ejecución durante el desarrollo de cada uno de los módulos.

Para el módulo EB_CACHE lo importante es que se reciban los datos del cliente. Por lo tanto las pruebas que se realizaron fueron las de transmisión de los datos necesarios para realizar un servicio y una vez enviados comprobar que el módulo los había recibido de una forma correcta y que estaban listos para su posterior manipulación.

También fue necesario realizar una serie de pruebas para la extracción de datos desde la aplicación de ficheros XML ya que en estos ficheros se encuentra la configuración de la aplicación. Las pruebas que se realizaron fueron las de carga de un fichero XML y búsqueda de la entrada que se deseaba mediante acotaciones con los datos obtenidos del cliente.

Finalmente para la creación de objetos dinámicamente, después de una profunda investigación del tema ya que era totalmente desconocido para mí, las pruebas que se realizaron fueron creaciones de objetos de diferentes clases.

Para los módulos de servicios las pruebas realizadas fueron similares para cada módulo. Lo principal fue comprobar que se daba el servicio solicitado.

Así que una de las pruebas fue la extracción de datos necesarios en el XML de configuración del servicio, además de realizar una conexión a la base de datos. Este paso dio algún que otro problema ya que la configuración de la cadena de conexión a la base de datos que reside en el XML de configuración no era aceptada a la hora de realizar la conexión y se tuvo que separar los diferentes datos como URL, usuario y password en diferentes cadenas y de esta forma se solucionó. Después de esto lo que se debía comprobar fue que se realizaba de una manera correcta el fichero XML de salida con todos los datos y la cabecera predeterminada.

5.3 – Pruebas de integración.

Las pruebas de integración son aquellas que se realizan una vez se han comprobado los diferentes módulos mediante las pruebas unitarias. Únicamente son pruebas de todos los elementos unitarios que componen el proceso a realizar de manera conjunta y de una sola vez. Consiste en verificar el funcionamiento del conjunto de partes de la aplicación juntas.

Estas pruebas se realizaron en el momento que se vio que cada módulo funcionaba correctamente de forma separada. El punto clave para estas pruebas fue en la que se crea el objeto dinámicamente, ya que en este punto es en el que saltamos de leer la petición del cliente a ejecutar el servicio que se ha demandado. Una vez que este paso se realiza correctamente y se es capaz de transferir los datos de un módulo a otro la aplicación consigue funcionar de una manera correcta.

5.4 – Tiempos de ejecución.

Para medir el rendimiento de la aplicación a la hora de hacer un serie de consultas a base de datos, se han medido los tiempos en que se realiza la conexión a base de datos y la consulta y el tiempo que tarda la aplicación en realizar el proceso entero.

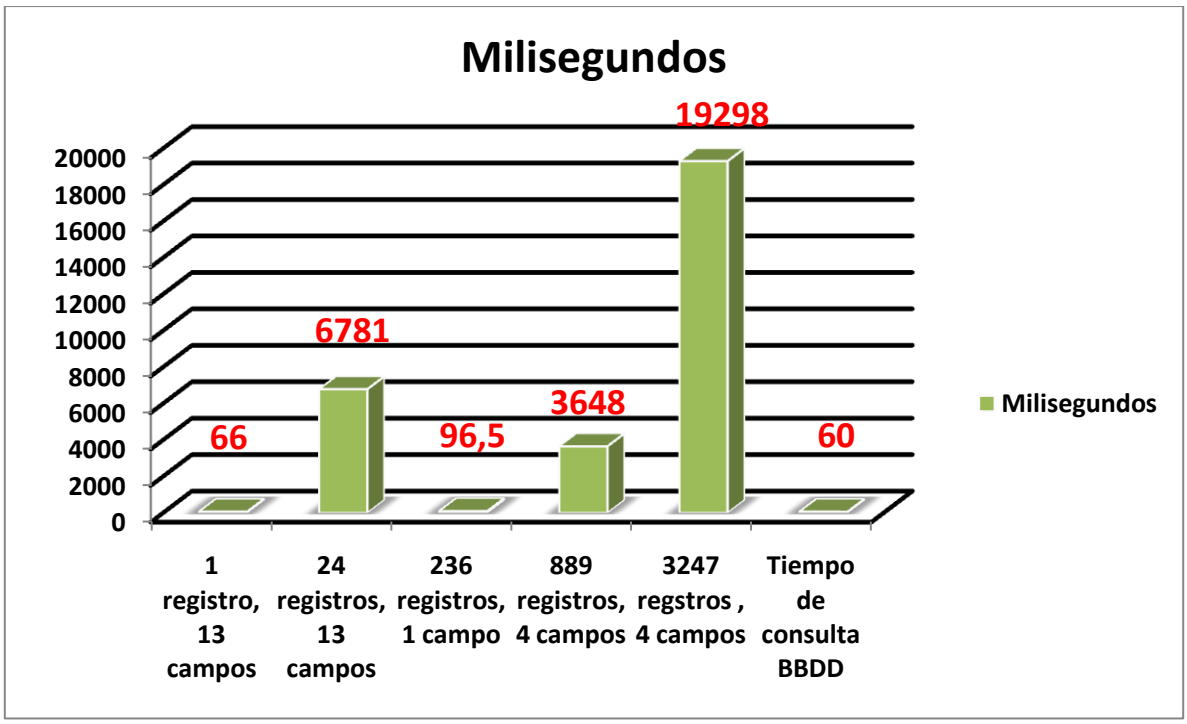


Figura 22 Tiempos de ejecución

En la figura 22 podemos ver un gráfico con los resultados de estas pruebas con diversos valores. A la hora de realizar la conexión con la base de datos y realizar la consulta el tiempo medio es de 60 milisegundos. En cambio a la hora de medir el tiempo de ejecución de la aplicación los tiempos varían en función de los resultados de la consulta. Esto es debido al tiempo que se tarda en la construcción del fichero XML donde se verán todos los resultados. Por lo tanto hay que tener en cuenta el número de registros devueltos y además los campos que tiene cada registro.

Como podemos ver cuantos menos registros y campos se tengan más rápido ira la aplicación, en cambio con 3247 registros y 4 campos la aplicación tarda unos 20 segundos.

5.4 – Problemas.

Durante el desarrollo de la aplicación que se realizó en Eclipse de forma conjunta con el servidor Tomcat y el motor de servicios web Axis2, se logró que todo funcionara de forma correcta y se obtuvieron los resultados correctos. Pero a la hora de intentar instalar este web server dentro del servidor Tomcat sin la ayuda del IDE Eclipse falló la aplicación en el momento que debía realizarse la conexión a la base de datos. Parece ser que Tomcat tiene unos ficheros de configuración en formato XML y deben de modificarse para poder realizar una conexión a la base de datos Microsoft SQL Server 2005. Por esta causa no se pudo probar finalmente en un entorno real.

CAPÍTULO 6: Conclusiones

6.1 – Introducción.

En este capítulo podremos ver las conclusiones que he obtenido del desarrollo de este proyecto. Primero podremos ver cuáles han sido los objetivos que fueron marcados al principio del proyecto y que se han logrado realizar. Seguidamente podremos ver las desviaciones respecto a la planificación original. A continuación señalaremos una serie de mejoras y ampliaciones de este proyecto para un futuro y cuáles son los beneficios que se consiguen después de haberlo desarrollado. Finalmente podremos ver una valoración personal.

6.2 – Objetivos cumplidos.

Los objetivos marcados al inicio del proyecto fueron los siguientes:

O1. Migrar Web Server a Java.

O2. Hacer Web Server multiplataforma.

O3. Hacer módulo de consultas.

O4. **Hacer módulo de inserciones.**

O5. **Módulos secundarios.**

O6. **Mejorar rendimiento del Web Server**.

O7. Reducir costes.

Finalmente a la hora de terminar el proyecto podemos ver que el objetivo 1 se ha logrado, ya que actualmente el web server está en Java en su totalidad. El objetivo 2 se cumple automáticamente ya que Java, por definición, es multiplataforma. Los objetivos 3 y 4 también se han cumplido y los dos módulos funcionan correctamente. El objetivo 5 no se ha cumplido ya que por falta de tiempo no se han podido desarrollar más módulos. El objetivo 6 se ha cumplido aunque se puede mejorar. Y, por último, se han reducido los

costes ya que ahora todo lo que se utiliza en este web server, lenguaje de programación o servidor donde se aloja, se realiza con programas de libre distribución, por lo tanto se aligera el precio gracias al ahorro del pago de licencias.

6.3 – Mejoras y ampliaciones.

Fundamentalmente hay tres mejoras o ampliaciones que deberían llevarse a cabo para mejorar la calidad de la aplicación en un futuro.

El desarrollo de más módulos, implica mas servicios para el web server y si añadimos mas servicios el web server tendrás muchas más funcionalidades por lo tanto podríamos construir un web server mucho más competente.

Si se desarrolla un pool de conexiones a bases de datos aumentaremos de forma considerable el rendimiento de la aplicación ya que no tendremos que abrir una conexión que es un proceso costoso, simplemente utilizaremos una conexión ya abierta.

Finalmente habría que añadir algún sistema de seguridad para cifrar los datos de los ficheros XML que residen en la aplicación o que viajan a través de internet.

6.4 – Beneficios.

Los beneficios que se obtienen a la hora de utilizar la aplicación que se ha desarrollado son los siguientes:

- Mejora del rendimiento del web server.
- Introducimos un lenguaje nuevo y más potente con muchas más posibilidades.
- El web server ahora es multiplataforma y funciona con arquitecturas de 64 bits, por lo tanto podemos aprovechar los recursos de las nuevas máquinas.
- Obtenemos una relación perfecta con el software de la empresa que se ha desarrollado en Java, no hay problemas de compatibilidades.
- Es más económico debido a todos los programas de libre distribución que se utilizan tanto para su desarrollo como para su instalación.

6.5 – Desviaciones respecto a la planificación original.

A continuación en la figura 23 podemos ver un gráfico donde se verán las diferencias en las fases de la planificación.

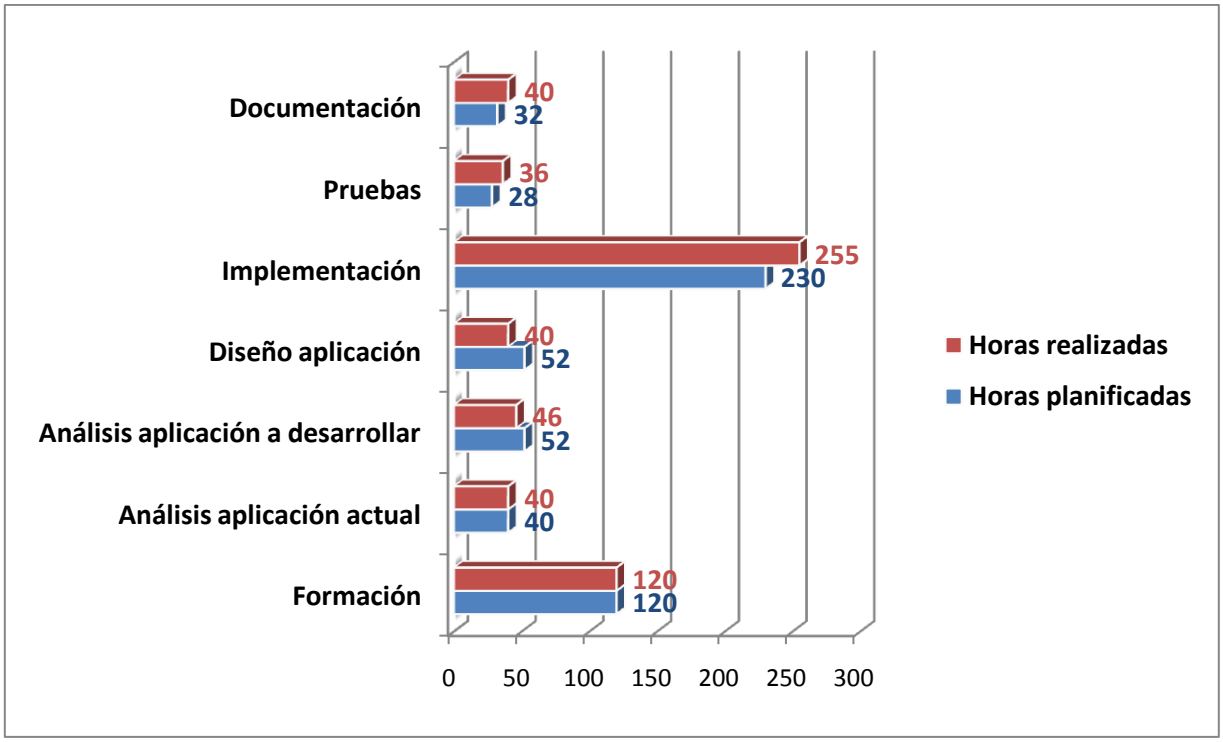


Figura 23 Desviaciones planificación

Como podemos ver las principales diferencias se llevan a cabo en la fase de implementación, pruebas y documentación que necesitaron de más tiempo para poder terminar el proyecto. En cambio la fase del análisis de la nueva aplicación y de su diseño se pudo realizar en un tiempo menor al planificado.

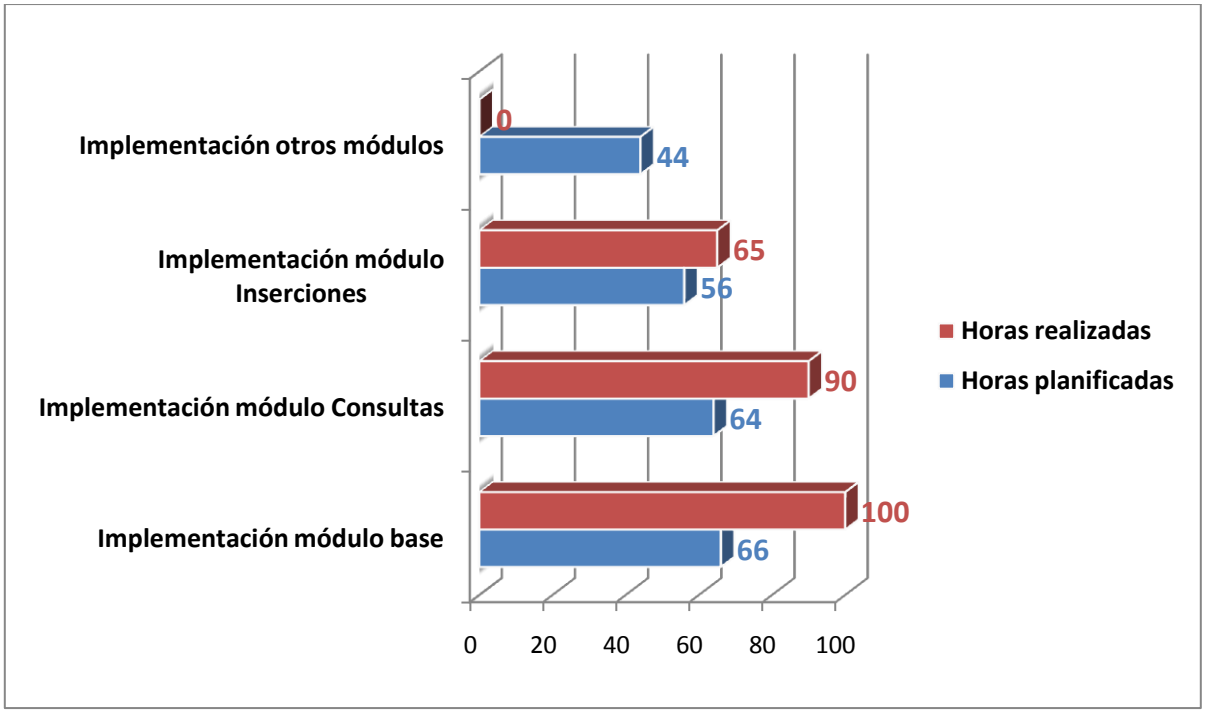


Figura 24 Desviaciones implementación

Me pareció importante destacar la fase de implementación como se ve en la figura 24. Esta fase se dividió en varias partes, la implementación del módulo base, del módulo de consultas, del módulo de inserciones y de otros módulos. Finalmente se desestimó el desarrollo de otros módulos ya que los otros tres módulos necesitaron de más tiempo para su desarrollo.

6.6 – Valoración personal.

Personalmente estoy muy contento con el proyecto. Gracias a este proyecto realizado en una empresa he podido entender de una forma bastante clara como es el funcionamiento de una empresa dedicada al mundo de la informática. Aunque desde un principio tuve una estructura que se me pidió que respetara fue de forma bastante mínima ya que se me dio mucha libertad para la planificación, el diseño y el desarrollo de la aplicación.

Además me ha permitido aprender mucho sobre otros tipos de lenguajes que no conocía y profundizar en el lenguaje Java del que sólo conocía las bases. Mis conocimientos se han visto ampliados en todo el tema de servicios web y estoy seguro que en un futuro me serán de utilidad.

Ha sido una experiencia que ha valido la pena, me siento más preparado para el mundo laboral ahora que si hubiera realizado el proyecto por mi cuenta, pienso que la realización del proyecto en una empresa hace que podamos partir con cierta ventaja.

CAPÍTULO 7: Bibliografía

- Información y tutoriales de XML:
    - Introducción al manejo de XML:
        - http://geneura.ugr.es/~jmerelo/xml/
    - Documentación de Comunicaciones mediante servicios:
        - http://www.programadorphp.org//wp-content/uploads/Cursos_Talleres/JavaWS/manuales_comunicacion.html
- Manuales Java:
    - Java:
        - http://www.proactiva-calidad.com/java/principal.html
    - API Java:
        - http://download.oracle.com/javase/1,5.0/docs/api/
- Información sobre Visual Basic:
    - Tutorial Visual Basic:
        - http://www.elguruprogramador.com.ar/tutoriales/visual-basic
    - Funciones Visual Basic:
        - http://msdn.microsoft.com/es-es/library/32s6akha%28v=VS.80%29.aspx
- Información sobre Tomcat y Pool de conexiones:
    - The Apache Tomcat 5.5:
        - http://tomcat.apache.org/tomcat-5.5-doc/index.html
    - Pool de conexiones: BasicDataSource :
        - http://www.chuidiang.com/java/mysql/BasicDataSource-Pool-Conexiones.php
    - UsingDataSources:
        - http://wiki.apache.org/tomcat/UsingDataSources

Anexo: Contenido del CD

En el CD entregado junto a la memoria se podrá encontrar:

- Memoria en formato PDF.
- Archivo JAR con los ficheros de la aplicación. (Archivos java, clases y ficheros XML)
- Conjunto de presentaciones power point de UNIT4.
- Manual de instalación de Tomcat y Axis2 en Eclipse.
- Archivos .pos de OpenProj con las planificaciones iniciales y finales del proyecto.
- Archivo de Microsoft Visio con los diagramas de clases de la aplicación.

Alberto Vázquez López

Sabadell, 23 de junio 2011