Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Proyecto Fin de Carrera

Mantenimiento de aplicaciones web

Autor: Daniel Cabrera
Tutor: Loïc Martínez

Índice de contenido

1 Introducción

Una empresa de selección de personal tiene como núcleo de su funcionamiento una página web en la que se listan las ofertas de empleo de clientes al público. Esta empresa tiene varias sedes repartidas por todo el territorio español (Barcelona, Bilbao, Madrid, Sevilla, etc.), y dispone de diferentes divisiones según los sectores laborales que abarca (banca, comercial, ingenieros, recursos humanos, etc.). Las divisiones están compuestas por directores (o managers), consultores y becarios. En adelante y salvo que se quiera recalcar su puesto, estos tres grupos serán referidos simplemente como "consultores".

Para la publicación de ofertas de trabajo, los consultores disponen de una plataforma (a partir de ahora PLF) en la que se gestionan y almacenan los datos de los clientes, los candidatos, las ofertas de empleo, listas (de candidatos/clientes), entre otras funciones.

La página web se aloja en un servidor web que a partir de ahora será llamado SWB. Las ofertas de trabajo son exportadas mediante un procedimiento cada 30 minutos de la plataforma PLF a la base de datos de SWB para un mejor acceso a ellas desde la página web.

Por otro lado, la empresa dispone de un servidor de aplicaciones web internas (a partir de ahora AWI) para facilitar las tareas de los consultores.

Dentro del servidor AWI **destaca la aplicación Búsqueda de candidatos**. A través de ella, los consultores pueden mandar ofertas de empleo por correo electrónico a posibles candidatos que puedan dar el perfil. Esta aplicación conecta con la base de datos de SWB para recuperar las ofertas de trabajo que lleva el consultor, y con la plataforma PLF para importar las listas de candidatos que tenga almacenadas. La mecánica es simple: el consultor selecciona uno de los puestos de trabajo disponibles, una lista de candidatos, y la aplicación genera una plantilla con la oferta de empleo, que se manda por correo electrónico a la lista de candidatos seleccionada.

1.1 Sistema

El sistema está organizado tal y como muestra la Figura 1 con los componentes que se describen a continuación:

- **Plataforma PLF:** es el eje de todo el sistema. Cualquier aplicación, servidor, u otra plataforma que se instale, va a depender directa o indirectamente de PLF. Es un software adquirido de un proveedor, por lo que no es posible modificar su código fuente, con las limitaciones que ello conlleva. Incorpora un servidor de base de datos Sybase (PLFDB).
- **Servidor web (SWB):** es el encargado de alojar las páginas web de búsqueda de empleo.
- **Servidor SQL Server para las páginas web (SWBDB):** importa de PLFDB las ofertas de trabajo y las almacena para ser utilizadas por la web de búsqueda de empleo de la empresa. Estos procedimientos son lanzados periódicamente cada 30minutos, para tener la base de datos y las páginas de internet lo más actualizados posible.
- **Servidor de aplicaciones web internas AWI:** en él se almacenan y se ejecutan los servicios web que utilizan los consultores dentro de la empresa. La aplicación de Búsqueda de candidatos necesita hacer llamadas a la base de datos SWBDB para gestionar las ofertas de trabajo.
- **Servidor SQL Server para las aplicaciones web interas (AWIDB):** las aplicaciones web se apoyan en este servidor de base de datos para obtener la información. En él se almacena el directorio de personal de la empresa, se gestionan las estadísticas y se

registra la actividad de las aplicaciones. También importa diariamente de PLFDB las tablas correspondientes a candidatos, clientes, listas de candidatos y listas de clientes.

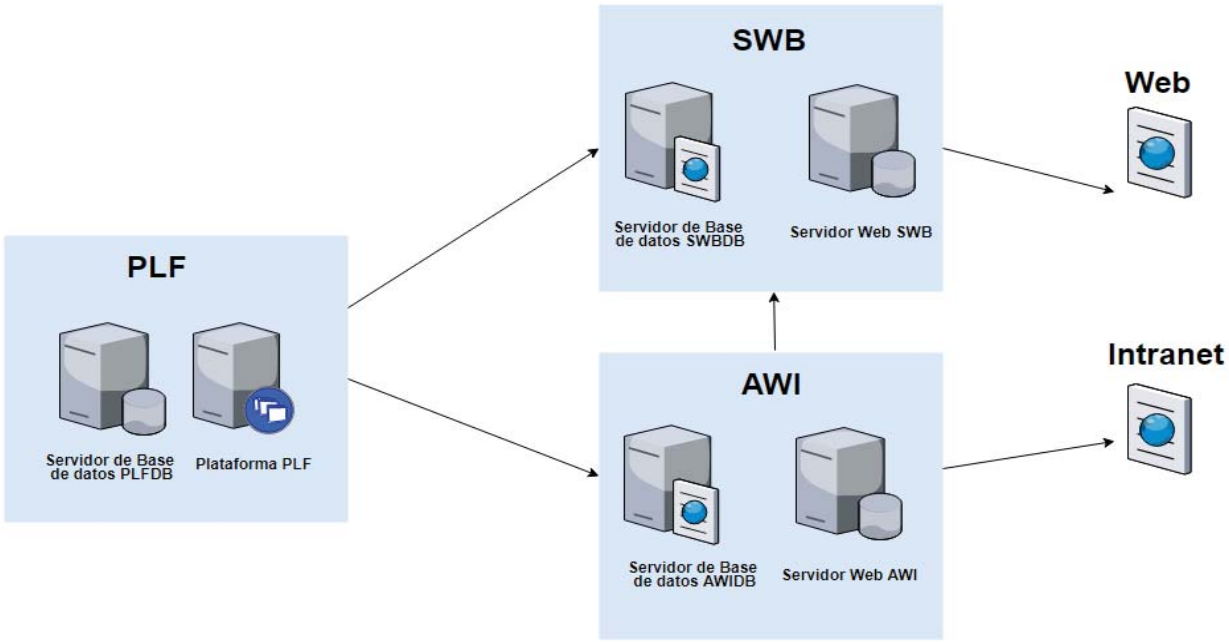

Figura 1. Organización del sistema de selección de personal de la empresa

1.2 Desarrollo

Tanto la aplicación que da vida a la página web, como las aplicaciones internas para consultores, están desarrolladas con Struts (Apache Software Foundation, 2018), un entorno de trabajo bajo la plataforma Java Enterprise Edition basado en el patrón modelo-vista-controlador (Reenskaug, 2004).

La aplicación web se activa en un servidor de aplicaciones WebLogic 8.1 (BEA Systems, 2018). Las aplicaciones internas arrancan su funcionamiento sobre un servidor de aplicaciones JBoss 4.2 (JBoss Inc., 2010).

1.3 Objetivo

Una vez expuesto el funcionamiento del sistema, el cometido en el proyecto presente abarca los siguientes puntos:

- Planteamiento de soluciones para adaptarse a cambios externos de la arquitectura de la herramienta "Búsqueda de candidatos".
- Implementación y codificación de dichas soluciones.
- Pruebas y mantenimiento.

2 Descripción del sistema existente

En este apartado se describe el comportamiento de la aplicación Búsqueda de Candidatos antes de su actualización/mantenimiento.

Como se mencionó anteriormente, esta aplicación se desarrolló con el entorno de trabajo Struts. Las características más representativas de esta herramienta son:

- Asegurar la persistencia de los objetos.
- Separar la lógica de negocio de la interfaz de usuario.
- Ofrecer un mantenimiento sencillo.

El ciclo que sigue para procesar las peticiones HTTP es el mostrado en la Figura 2.

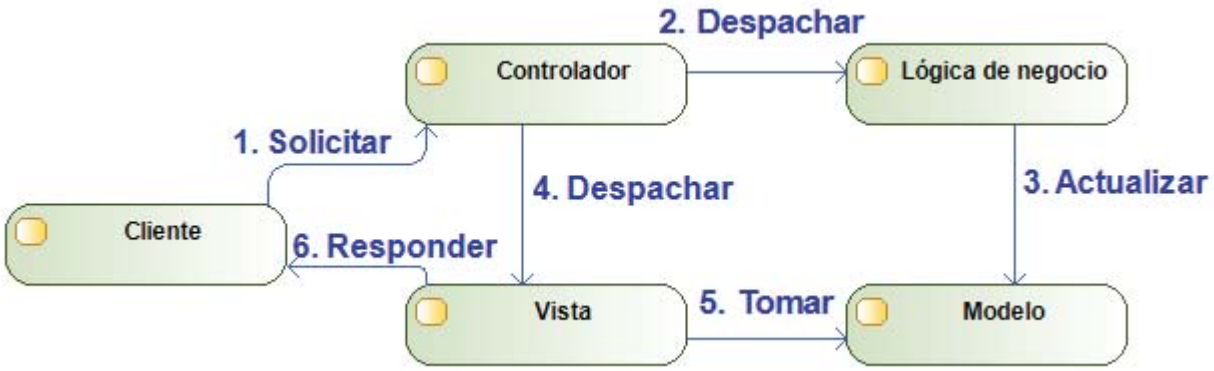

Figura 2. Diagrama de funcionamiento de Struts

1. Solicitar: el cliente realiza una petición, que pasa siempre por el controlador.
2. Despachar: el controlador procesa la petición y dispara la lógica de negocio correspondiente.
3. Actualizar: se actualiza el modelo. Se obtienen los datos, que se almacenan en "beans" (componentes de información reutilizables para facilitar la programación).
4. Despachar: dependiendo del valor devuelto, el controlador redirige a la vista correspondiente.
5. Tomar: la vista obtiene los datos procesados por la lógica de negocio.
6. La vista muestra los datos.

Las aplicaciones también utilizan desde STRUTS unas librerías que permiten la conexión a servidores de bases de datos SQL Server (Microsoft, Microsoft SQL Server 2008, 2009) SWBDB, AWIDB, y al servidor de base de datos de Sybase PLFDB. Esta parte correspondería al paso 3 del diagrama de funcionamiento de Struts.

2.1 Búsqueda de Candidatos - Flujo

Tal y como se ha comentado, antes, un proceso dentro del servidor SQL SWBDB se encarga de importar periódicamente (cada 30 minutos) las ofertas de trabajo de PLF a la base de datos de SWBDB. Esta decisión se tomó para dividir la carga de trabajo en la plataforma PLF, y evitar bloqueos en la misma. De este modo, se pretende utilizar PLF únicamente para gestionar los clientes, los candidatos, y las ofertas de empleo.

Un ejemplo del flujo de importación de ofertas de empleo sería el siguiente:

1. Un consultor da de alta una oferta de trabajo.
2. El proceso SQL de importación, lanzado cada 30 minutos, almacena la oferta de trabajo activa desde PLFDB a SWBDB.
3. Dentro del servidor SWB, la aplicación web de búsqueda de empleo lee la oferta de trabajo activa, y la muestra dentro de la página de la empresa.

Las listas de candidatos, por el contrario, se importan directamente de PLFDW a una base de datos en AWIDB. Esto se hace directamente y bajo demanda cada vez que se hace un envío dentro de la aplicación, por un sencillo motivo: se ha de tener la certeza que las listas de candidatos están en todo momento actualizadas. Esto debe ser así por los siguientes motivos:

- Desde el mismo momento que un candidato se da de baja, no le pueden llegar más envíos.
- Si el candidato está trabajando para un cliente que colabora con la empresa, no le pueden llegar envíos.

Es por esto que es importante tener las listas lo más actualizadas posible. El flujo de importación de una lista sería el siguiente:

1. El consultor hace un envío.
2. El sistema llama a un procedimiento dentro de la base de datos AWIDB, que hace una consulta a la base de datos PLFDW: importar la lista de candidatos a AWIDB.
3. El sistema se hace carga de hacer el envío a todas las direcciones de correo electrónico dentro de la lista de candidatos importada en AWIDB.

Antes se comentó que para no interferir y evitar bloqueos, es prudente no hacer consultas directas a PLF. Sin embargo, estas consultas son inevitables en algunas ocasiones y son pequeñas e imperceptibles dentro de la carga de trabajo de PLF.

Para importar las tablas que contienen las ofertas de trabajo desde un servidor de base de datos a otro, se utiliza la aplicación Microsoft SSIS (Microsoft, SQL Server Integration Services, 2017). Se eligió esta herramienta por su potencia la hora de mover datos entre bases de datos, lo cual facilita enormemente el proceso de importación entre servidores.

Se han de diferenciar tres partes. Por un lado está el servidor de base de datos de PLF, cuyo motor está implementado sobre T-SQL Sybase Central (Sybase Software, 2013). De él es de donde se va a extraer toda la información. Por otro, dentro del servidor de base de datos SWB, hay dos clústeres (agrupación de servidores) implementados sobre SQL:

- Clúster 1: es utilizado por la aplicación interna "Validación de ofertas de trabajo" del servidor AWI. En él se almacenan las ofertas de empleo en la base de datos antes de ser exportadas al clúster 2 y publicadas en la página web.
- Clúster 2: es el utilizado por la web para leer las ofertas de trabajo activas.

2.2 Búsqueda de Candidatos -Diagramas

A continuación, se explica el flujo del diagrama de secuencia ilustrado en la Figura 3 utilizado para la importación de ofertas de trabajo en la herramienta SSIS:

1. Se eliminan todas las filas de la tabla jvi_import_jobs, que contienen los datos de cada

oferta de trabajo.

2. La base de datos de PLF contiene todas las ofertas de trabajo que han existido. Tanto las abiertas como las que ya fueron ocupadas o canceladas. Mediante una consulta a la base de datos, se obtienen todas las ofertas que estén activas, y se copian a la tabla jvi_import_jobs.
3. En este punto, los directores validan (o revocan) las ofertas de trabajo mediante la aplicación de validación de ofertas de trabajo, y el estado se actualiza en la tabla jvi_import_jobs (revocado o validado)
4. Se utiliza una tabla temporal, jvi_import_jobs_temp, que desde el inicio se trunca para ser actualizada.
5. Se copian de la tabla plf_import_jobs (tabla utilizada por la web de ofertas de empleo) del clúster 2 todas las filas de ofertas de trabajo a la tabla temporal jvi_import_jobs_temp.
6. Ahora, si hay alguna oferta que haya sido revocada, validada, o que ya no exista en jvi_import_jobs, se debe eliminar de jvi_import_jobs_temp. Este paso se hace principalmente porque si ya existe una oferta en la tabla plf_import_jobs (es decir, una oferta en línea), y el consultor necesita modificar algo de ella, tiene que pasar de nuevo todo el proceso de validación de la oferta. Es por eso que si en este paso, se detecta alguna oferta validada o revocada que ya esté en línea, se debe eliminar de la tabla jvi_import_jobs_temp, pues significa que su estado ha sido modificado.
7. Ahora se vuelven a leer todas las ofertas de empleo, y las que tienen estado "validado" se copian a la tabla jvi_import_jobs_temp.
8. El siguiente paso es truncar la tabla plf_import_jobs, para actualizarla con la última versión de jvi_import_jobs_temp.
9. Se copia el contenido de jvi_import_jobs_temp a plf_import_jobs
10. Se trunca la tabla v2_jobs, de la que se leen las ofertas de empleo de la web.
11. Se rellena la tabla v2_jobs con el contenido de plf_import_jobs

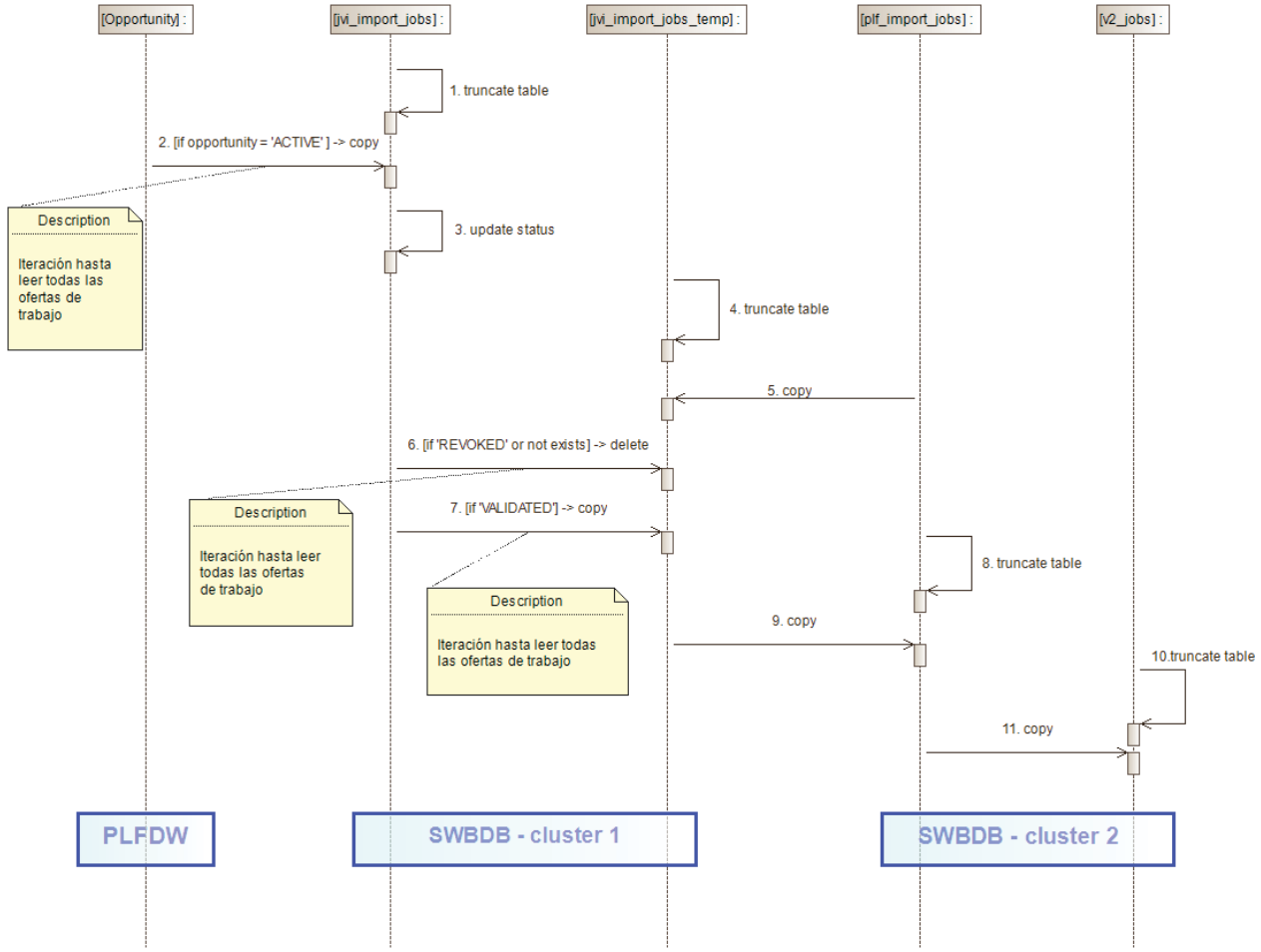

Figura 3. Diagrama de secuencia de importación de ofertas actual

Observaciones:

- Este procedimiento cuenta con un gran número de tablas intermedias. De la que se exporta (opportunity de PLF) a la que se importan los datos (v2_jobs de SWBDB), se usan un total de seis tablas intermedias. Las razones de esta decisión no son claras, pues es un procedimiento muy antiguo que no ha sido apenas actualizado en los últimos años. Es posible que fuera para guardar copias de seguridad, o porque el acceso a las tablas fallaba, o la base de datos PLF se bloqueaba si se hacían demasiadas consultas sobre ella. Aun así, hay tablas temporales innecesarias, por lo que queda pendiente una revisión de todo el procedimiento como mantenimiento, como se verá en el siguiente apartado.
- Los puntos 6-7 dan un comportamiento indeseado evitable. Al darse el caso en que un consultor aplique unos cambios a una oferta ya en línea, si el director descarta sus cambios, la oferta desaparece de la web. Ni siquiera aparece con los cambios antiguos.

A modo de ejemplo, siguiendo el diagrama de casos de uso de la aplicación mostrado en la Figura 4, el funcionamiento de la aplicación Búsqueda de Candidatos con el patrón modelo-vista-controlador de Struts para el caso de uso "Enviar a validar oferta de trabajo" sería el siguiente:

1. El consultor, al entrar a la aplicación, solicita al controlador la petición de acceso.
2. El controlador procesa la petición y dispara la lógica de negocio que muestre la página

de inicio de la aplicación.

3. Se actualiza el modelo obteniéndose los datos para trabajar: las ofertas de trabajo y las listas de candidatos del consultor.
4. Se llama a la "vista" para que muestre al consultor la página de inicio con los datos del modelo.
5. El consultor rellena los datos y hace clic al botón que activa el caso de uso "Hacer envío de oferta", lo que llama al controlador.
6. El controlador dispara la lógica de negocio que procese el envío de datos.
7. El modelo almacena y procesa los datos, enviando los correos electrónicos a los candidatos de la lista del consulto.
8. El controlador llama a la "vista", que muestra al consultor el resultado de su acción (si salió bien o no).

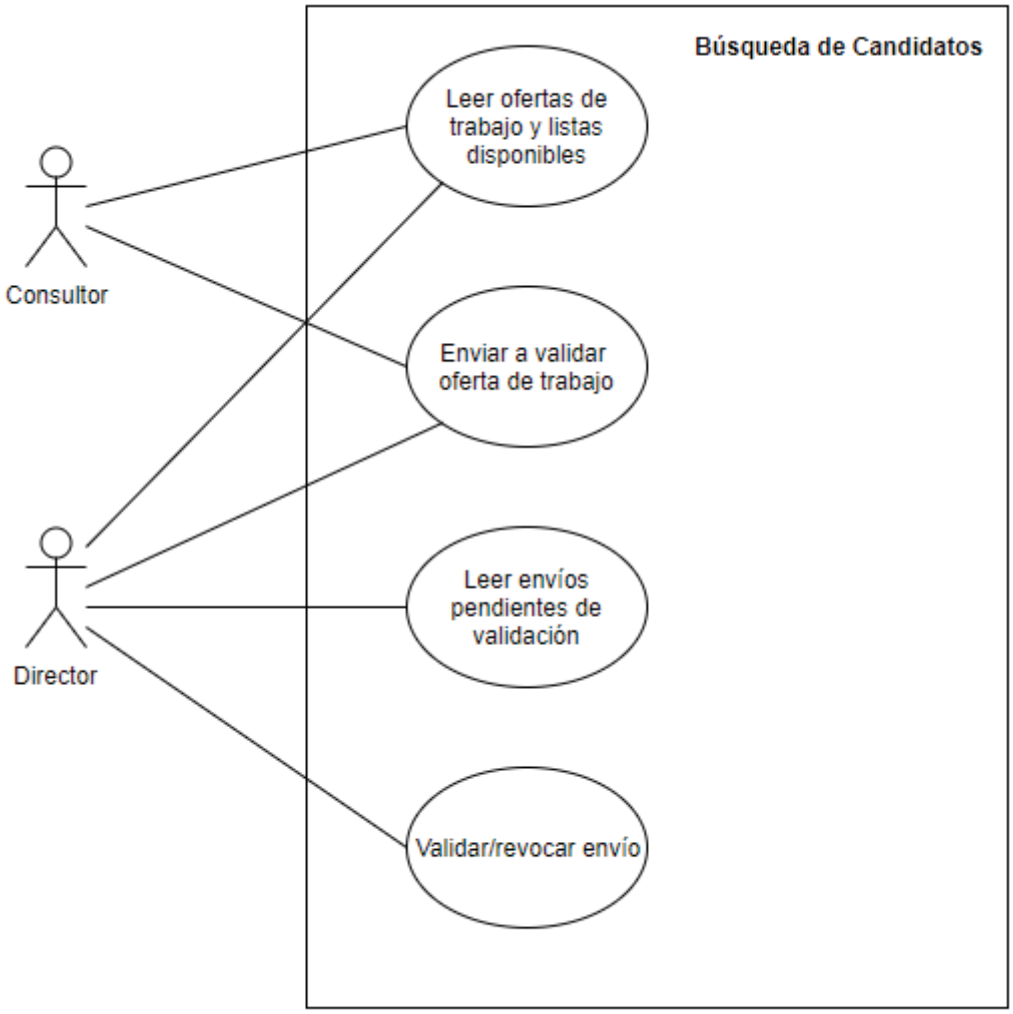

Figura 4. Diagrama de casos de uso de "Búsqueda de Candidatos"

3 Planteamiento del problema

En este apartado se procede a mostrar los cambios por los que va a pasar el sistema, y a estudiar y detectar los efectos colaterales que se darán en la aplicación de Búsqueda de Candidatos, con el fin de encontrar soluciones factibles y acordes a las demandas planteadas.

3.1 Cierre del servidor web

La empresa alojaba la página web en su servidor SWB. El negocio, con el fin de abaratar costes y retirar servidores antiguos y costosos, opta por un proceso de externalización para que la gestión se lleve desde un proveedor externo. Una vez ejecutado el proceso, el servidor SWB sería retirado de servicio y sustituido por un servidor web externo, gestionado por un proveedor de servicios (Figura 5).

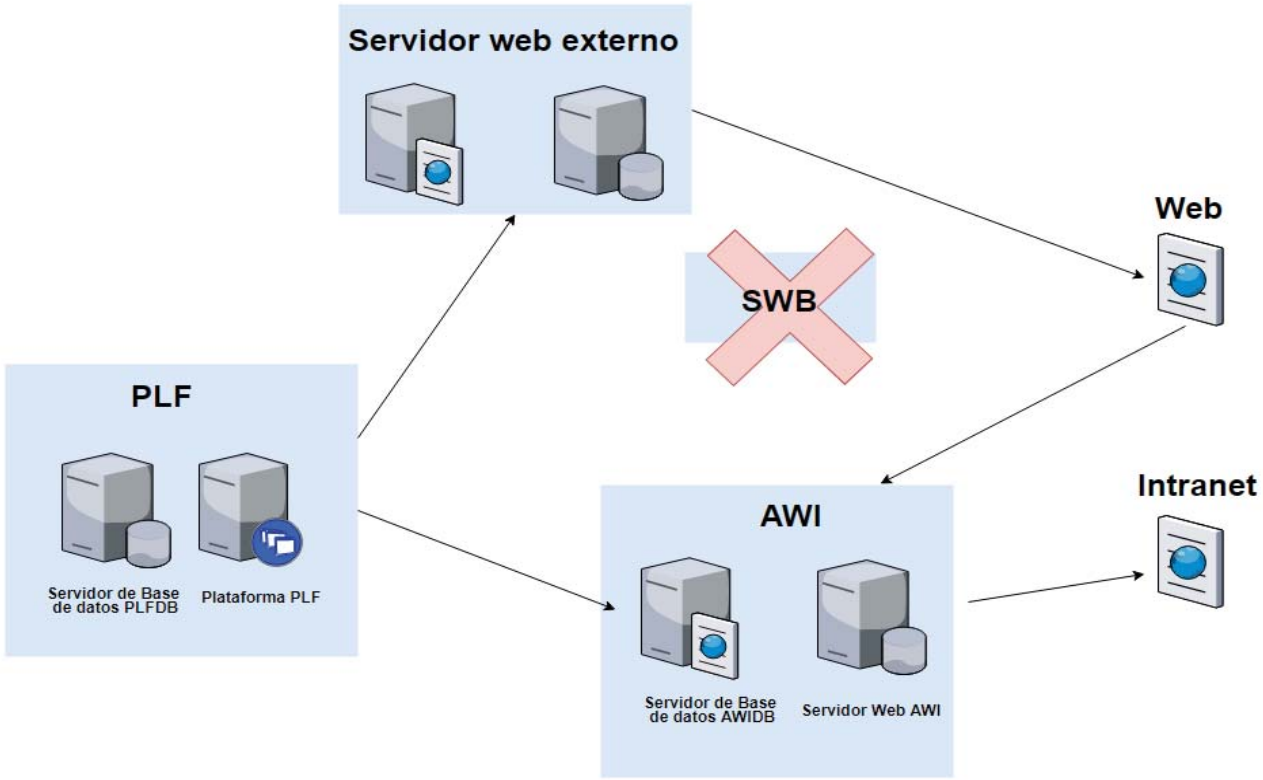

Figura 5. Organización del sistema de la empresa tras la externalización

El proceso de externalización implica asimismo cesar el servicio del servidor de bases de datos SWBDB. Por lo tanto, el procedimiento mostrado en el apartado 2.2 desaparecería.

3.2 Servicios afectados

A la hora de desarrollar aplicaciones, se teme que este nuevo enfoque por razones obvias no permita el acceso a la nueva base de datos externa. Todos los procedimientos de importación desde PLF, como el descrito en la Figura 3, se pierden.

La aplicación Búsqueda de Candidatos depende para funcionar de diversos factores que es necesario analizar si se ven afectados.

- **Listas de candidatos:** las listas son procesadas a diario cada mañana por un procedimiento SSIS que las importa desde la base de datos de la plataforma PLFDB al servidor AWIDB. Por lo tanto, este factor no se ve afectado por el cierre del servidor web.
- **Ofertas de trabajo:** son leídas desde el servidor AWIDB contra la base de datos SWBDB, de donde eran importadas por un procedimiento SSIS cada 30 minutos desde la plataforma PLF. Este objeto se ve afectado dentro de la aplicación.
- **Envíos:** los envíos son guardados mediante un procedimiento SQL en una tabla dentro del servidor AWIDB. Puesto que contienen información relativa a las ofertas de trabajo, estos objetos también se ven afectados.
- **Consultores/Directores:** los datos de los consultores y los directores encargados de validar los envíos se encuentran dentro de una tabla de la base de datos del servidor AWIDB. No se ven afectados.

Por tanto, la aplicación Búsqueda de Candidatos se va a ver afectada por dos factores: la ausencia de un origen para leer las ofertas de trabajo, y un procedimiento de creación de envíos incapaz de ejecutarse sin disponer de datos de ofertas de trabajo como parámetro de entrada.

3.3 Modificaciones previstas

Los requisitos para la aplicación Búsqueda de Candidatos son los mismos: "enviar correos electrónicos de ofertas de empleo a candidatos". La interfaz será igualmente la misma: "El sistema almacenará la plantilla de correo y la enviará a la lista de correo indicada". Sin embargo, el conocimiento del dominio ha cambiado.

Una vez detectados los servicios afectados, se plantean unos interrogantes para su estudio en profundidad.

- De dónde importar las nuevas ofertas de trabajo: sólo queda o bien extraerlas desde la plataforma PLF, o bien el proveedor debe ofrecer algún tipo de medio o tecnología que permita a la aplicación conectar con sus servidores para extraer las ofertas de trabajo.
- Dónde almacenar las ofertas de trabajo: tal vez será necesario crear un procedimiento como el de la Figura 3, pero adaptado al nuevo problema. Los datos se deberán importar a una base de datos interna, esta vez necesariamente dentro del servidor AWI.
- Cómo modificar el procedimiento SQL encargado de generar los envíos programados.

La solución deberá en la medida de lo posible respetar y adaptar el diseño y la lógica del software para poder ofrecer un servicio viable, persistente, y sencillo de mantener.

4 Solución

El ciclo de vida de software para este proyecto sería una variante entre el ciclo de vida iterativo y el evolutivo (Cantone, 2006), pues el proyecto en realidad tiene dos tipos de expertos. En un primer momento, el cliente (consultores/directores) especificó el tipo de demanda que quería a los expertos (la empresa). Por ello, por cada demanda del cliente, se iteraba un nuevo ciclo de vida en cascada. Ahora, puesto que parte del software se ha externalizado, los mismos expertos se convierten en clientes que deben especificar sus demandas al nuevo experto (el proveedor) para que el software creado originalmente pueda seguir ofreciendo sus servicios de una manera práctica y fiable. Esto es, se comienza con un alto grado de incertidumbre, pues no se trata de mejorar el sistema, sino de que evolucione para que persista su funcionalidad.

4.1 Comunicación: alternativas

Se negocia con el proveedor una manera de poder acceder a las ofertas de empleo desde la aplicación. Se presentan las siguientes alternativas.

4.1.1 Acceso al servidor externo de base de datos

La posibilidad de acceso al servidor externo del proveedor desde la base de datos de la empresa implicaría poder importar directamente las ofertas desde las propias tablas del proveedor. Se ubicarían permisos de acceso y de lectura únicamente para las tablas específicas que contienen los datos de las ofertas de empleo. El resto podría directamente restringirse su acceso.

Para que fuese posible el acceso a la base de datos externa, SQL Server ofrece una forma de conexión denominada servidores vinculados (Figura 6). Su configuración permite la ejecución de una instrucción SQL que incluya las tablas de otra instancia de bases de datos (SQL Server, Oracle, Sybase, etc.). El proveedor crearía el servidor vinculado en su servidor de base de datos mediante un origen de datos OLE DB, el cual especifica la base de datos a la que se puede tener acceso mediante OLE DB. La empresa establecería la conexión una vez obtenida la fuente de datos del servidor externo.

Figura 6. Esquema de servidores vinculados

Para acceder a las tablas del servidor externo, se utilizaría la instrucción OpenQuery (Microsoft, 2017):

```
OPENQUERY (servidor_vinculado, 'consulta')
```

4.1.2 Importar las ofertas de trabajo directamente de la plataforma PLF

El proveedor externo importa primeramente las ofertas de trabajo de la plataforma PLF de la empresa. Esta tenía antes su propio algoritmo de importación de ofertas de trabajo a la base de datos (Figura 3), en vías de desaparecer, SWB.

Esa metodología, tal y como se explicó en el apartado 2.1, podría intentar reproducirse directamente desde la base de datos PLFDB a la base de datos de aplicaciones AWIDB. Es decir, se propone la implementación de un procedimiento bajo demanda basado en el algoritmo de importación hasta ahora utilizado, que importe bajo demanda las ofertas de cada consultor cuando estos accedan a la aplicación.

Sin embargo, como ya se señaló, el algoritmo utilizado era algo ineficaz y redundante. Entonces, se propone como posibilidad acceder directamente a la base de datos PLFDB, bajo demanda desde la aplicación, sin procedimientos de importación a tablas de bases de datos secundarias, como era el caso anteriormente.

Bien, esta opción trae algunas objeciones. El servidor de base de datos PLFDB debe ser utilizado exclusivamente para lo que la plataforma PLF ofrece, que es la administración de ofertas de trabajo, clientes/candidatos y organizaciones por parte de los consultores. Es el motor de la plataforma. Que independientemente se lancen peticiones continuadas por otro lado podría traer consigo el riesgo de que la plataforma se bloquee.

Se pasa a hacer un cálculo aproximativo sobre el coste que supondría acceder directamente a la base de datos de PLFDB. Para empezar, se puede conocer el acceso medio por consultor a la aplicación haciendo una consulta a la base de datos, en una tabla que registra la creación de cada envío masivo de emails a candidatos. Como ejemplo, se toma la media de consultores por día que accedieron en diciembre de 2017:

```
SELECT count(*), year(daterequested), month(daterequested), day(daterequested)
FROM [dbo].[tb_bc_envios]
where year(daterequested)=2017 and month(daterequested)=12
group by year(daterequested), month(daterequested), day(daterequested)
order by year(daterequested), month(daterequested), day(daterequested) desc
```

Se obtendría el resultado reflejado en la Tabla 1.

Número de envíos	Año	Mes	Día
35	2017	11	29
37	2017	11	28
34	2017	11	27
59	2017	11	26
26	2017	11	23
66	2017	11	22
67	2017	11	21
59	2017	11	20
47	2017	11	19
4	2017	11	18
48	2017	11	16
114	2017	11	15
90	2017	11	14
97	2017	11	13
50	2017	11	12
66	2017	11	9
85	2017	11	8
69	2017	11	7
158	2017	11	6
48	2017	11	5
112	2017	11	2
129	2017	11	1

Tabla 1

En el peor de los casos, se coge el valor más alto de envíos diarios: 158.

Cuando un consultor crea un envío, el número total de veces que se accede a la tabla de ofertas de trabajo es 3:

- Cuando el envío es creado por el consultor.
- Cuando el manager visualiza el envío en la pestaña de validación antes de validarlo.
- Cuando el manager valida el envío: la aplicación accede a la tabla para crear la plantilla de correo electrónico.

No se tiene en cuenta que un consultor entre sin crear un envío, o que un manager revoque un envío, pues aparte de no ser medible, es raro que estos casos ocurran.

Entonces

158 envíos x 3 accesos a PLFDB = 474 accesos

Lo siguiente que se va a hacer es mirar el número de páginas diarias leídas en la plataforma. Para ello se debe hacer un cálculo entre los valores totales de dos días consecutivos en la tabla de estadísticas de la base de datos, tal y como muestra la Figura 7.

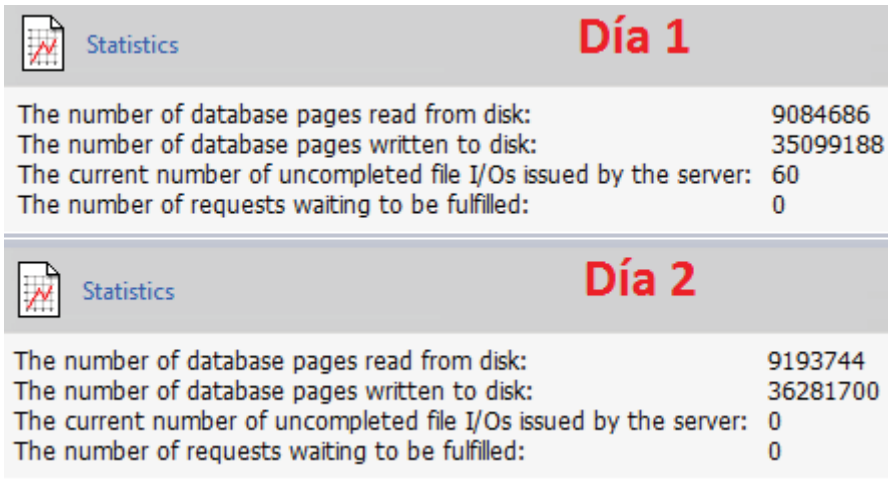

Figura 7. Estadísticas de paginación de PLFDB

La media diaria de páginas leídas sería:

$$9193744 - 9084686 = 109058$$

Ahora, se comprueba en PLFDB el número total de páginas que abarca la tabla en la que se guardan las ofertas de trabajo.

Name	Owner	Type	# T Pgs.	% T Used	# I Pgs.	% I Used	% File
opportunity	profile	Table	5171	95	2858	74	0

Figura 8. Tablas de Página de la tabla de ofertas de empleo

Tal y como se puede ver en la Figura 8, el número en el momento en que se hizo la consulta es 5171 tablas de página. El siguiente paso es conocer el número de filas que contiene dicha tabla actualmente:

```
select count(opportunity_ref) from opportunity
```

El resultado de la consulta es 148176.

Para conocer aproximadamente lo que ocupa una única fila, se divide el total de páginas actual por el número de filas:

$$\frac{5171}{148176} = 0{,}0348976892344239$$

Es decir, menos del valor de una tabla de página. Tomando que cada acceso costara una página entera, serían 474 lecturas frente a un total de 109058 antes calculado (más o menos 1 lectura de 230). Es una cifra muy pequeña y fácilmente asumible.

4.1.3 Exportación de las ofertas de trabajo

El proveedor podría compartir las ofertas de empleo de alguna de las siguientes maneras propuestas:

4.1.3.1 Fichero compartido en línea

Generar un fichero XML en línea con las ofertas de trabajo, por ejemplo en una URL, mediante DOM (World Wide Web Consortium, 2015); una interfaz de plataforma independiente del lenguaje que permite acceder y actualizar el contenido, la estructura y el estilo de documentos (en este caso ficheros XML) de forma dinámica.

El funcionamiento consistiría en exportar todas las ofertas de trabajo a un fichero XML en línea cada 30 minutos (para que se acerque al modelo anterior). Posteriormente, estas podrían ser importadas directamente por un procedimiento programado a una tabla de la base de datos AWIDB. Otra opción podría ser leer bajo demanda el fichero XML, según accedan a la aplicación los consultores. En ese caso, las ofertas correspondientes a cada consultor podrían ser almacenadas en estructuras de datos temporales, que serían posteriormente utilizadas y almacenadas por el procedimiento de creación de envíos.

4.1.3.2 Servicios web

Enviar las ofertas de trabajo mediante servicios web (Guru99.com, 2016). El proveedor crearía un servicio web, mediante el cual atendería las peticiones provenientes de la aplicación. Por ejemplo, al acceder un consultor a la aplicación, se activaría un servicio web que pidiera las ofertas de trabajo pertenecientes a ese consultor. En este caso se establece el mismo dilema sobre su almacenamiento. Una opción podría ser crear una tabla específica para las ofertas de trabajo dentro de la base de datos. Al llamar al servicio web, se comprobaría si la oferta ya existe en la tabla. De ser así, se actualizaría, y de lo contrario, se insertaría en la tabla.

El desarrollar esta opción implicaría grandes cambios internos dentro de la estructura software, pero mejoraría la interoperabilidad entre máquinas con respecto a la opción anterior.

4.1.3.3 Fichero compartido mediante SFTP

Volcar las ofertas de trabajo en un fichero CSV, y almacenar este en un servidor SFTP (Ellingwood, 2013). Los pasos que se deben seguir serían los siguientes:

- El procedimiento del proveedor exporta las ofertas desde la base de datos.

- Las ofertas son almacenadas en un fichero CSV local.
- Se ejecuta un script que transfiere el fichero CSV a una carpeta dentro del servidor SFTP.
- La empresa implementa un script similar que conecta con el servidor SFTP y descarga el fichero CSV a su servidor local.
- Se vuelca la información a una nueva tabla en la base de datos o a una estructura de datos.

Como se puede comprobar, esta opción es similar a la de la generación de un fichero XML.

4.2 Estudio de viabilidad

La empresa externa analiza las opciones propuestas y da su valoración.

4.2.1 Acceso al servidor externo de base de datos

Se desecha la idea de la compartición/acceso a una base de datos común pues, aunque es posible, la principal tarea que tiene el proveedor es hacer funcionar la página web de la empresa con sus servidores, y el hecho de compartir las bases de datos requeriría un estudio aparte bastante costoso. Habría que analizar la tecnología que podría utilizarse y los test se alargarían, pues no es la tecnología que utiliza esta empresa normalmente. Por otro lado, habría que considerar el impacto de los test sobre el rendimiento del servidor que ofrece más servicios a otros sitios web.

4.2.2 Importar las ofertas de trabajo directamente de la plataforma PLF

Aunque no habría inconveniente, como se mostró anteriormente, en acceder a la base de datos PLFDB, el proveedor comunica que su plataforma externa, al importar los trabajos, no los publicará directamente. Habrá una interfaz dentro de la web en la que los managers de los consultores validarán/revocarán las ofertas de trabajo para que estas aparezcan en línea o no. Entonces, si la empresa importa las ofertas de trabajo directamente de la plataforma, sin conocer si están en línea o no, esto puede llevar a confusiones entre los candidatos y los consultores. Por ejemplo, una oferta que haya sido descartada en el último paso, pero se haya enviado el email mediante la herramienta, tal y como ilustra la Figura 9.

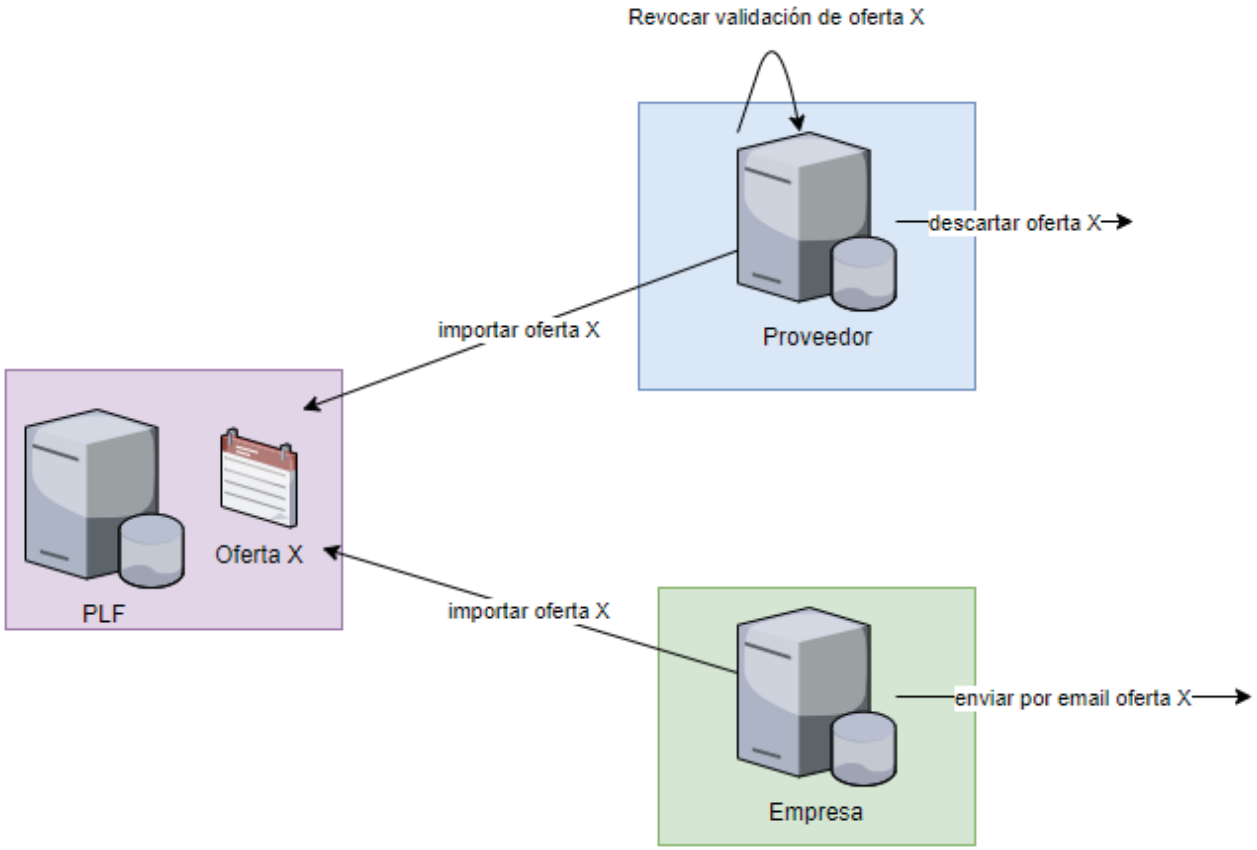

Figura 9. Importación directa de ofertas de trabajo desde la plataforma PLF

4.2.3 Exportación de las ofertas de trabajo

El proveedor está de acuerdo con esta propuesta por ser la más sencilla de implementar. Ofrece generar unos enlaces web públicos XML que contengan las ofertas de trabajo que estén en línea, con las siguientes puntualizaciones:

- Generar un fichero completo XML "jobs.xml" dos veces diarias: una a primera hora de la mañana, y otra a medio día. Se encargaría de contener todas las ofertas de trabajo que estén publicadas en la web.
- Generar un fichero "parcial" XML, "jobs_incremental.xml" a cada hora (salvo las dos horas reservadas para la generación del fichero anterior). Este archivo contendría únicamente las ofertas de trabajo que fuesen publicadas a lo largo del día.

La decisión de utilizar dos ficheros diferentes es debido a que, según señala el proveedor, la cantidad de trabajos a exportar es bastante considerable, lo que conlleva un consumo de recursos excesivo en sus servidores. Por ello, el generar un fichero completo a primera hora, y otro fichero completo a medio día durante la hora del almuerzo, es aceptable puesto que son horas del día en las que la carga de trabajo es muy baja. El resto de horas dentro de la jornada laboral, este fichero permanecería sin modificar, y sería el fichero parcial el que sería actualizado a cada hora, únicamente con las ofertas de trabajo que fueran creadas, editadas, o eliminadas a lo largo del día.

Sin embargo, no es una solución eficaz del lado del "cliente" (es decir, la empresa de cara al

proveedor) en comparación con el resultado que se conseguía anteriormente. El procedimiento de importación anterior corría cada 30 minutos. Ahora, con la solución planteada, desde que un consultor publica una oferta, hasta que esta aparece en la aplicación, podría pasar hasta una hora.

Se le plantea el siguiente razonamiento al proveedor:

Al acceder a la base de datos de la plataforma PLFDB, se estima que hay cerca de 3000 ofertas de trabajo activas:

```
select count(distinct opp.opportunity_ref)
from opportunity opp
inner join opport_role oppr on oppr.opportunity_ref = opp.opportunity_ref
inner join person p on p.person_ref = oppr.person_ref and (oppr.role_type='U1' or
oppr.role_type='U2')
inner join person_type pt on pt.person_ref = p.person_ref
inner join staff s on s.person_type_ref=pt.person_type_ref
where record_status='L'
and responsible_team like 'P%'
and date_closed is null
and s.email_address like '%@%.%'
and opp.responsible_team = s.team
```

Y las ofertas que han sido modificadas en un día promedio han sido alrededor de 150:

```
select *
from opportunity
where update_timestamp > CONVERT(date, getdate())
```

Es decir, frente a la exportación general de ofertas al fichero XML, la carga de exportación del fichero parcial es un 95% menos costosa. Por tanto, se propone lo siguiente:

- Mantener la ejecución del fichero completo “jobs.xml” dos veces al día
- El fichero parcial “jobs_incremental.xml” debe actualizarse bajo demanda por cada oferta que sea creada, editada o eliminada a lo largo del día en la página web.

La solución de este último punto sería la más conveniente para que la herramienta de búsqueda de candidatos continuara siendo igual de productiva o más. Tal y como ha quedado demostrado, con una media tan baja de ofertas modificadas al día, no debería existir riesgo de que el sistema se sobrecargue.

Sin embargo, el proveedor y parte del negocio alegan que la prioridad es la migración de las páginas web desde el servidor de la empresa al del proveedor, y estos pequeños detalles pasan a un plano secundario dada la magnitud del proyecto principal. Por tanto, el equipo de desarrollo deberá trabajar en una solución que se encargue de importar los datos registrados en los ficheros XML.

4.3 Diseño software

Una vez especificada la fuente de la que se extraerán los datos, se pasa a concebir la mejor manera de proveer a la aplicación del nuevo origen de las ofertas de trabajo. Se barajan las siguientes posibilidades para su estudio:

4.3.1 Modificar aplicación sustituyendo lectura de SWBDB por lectura de XML

La base de datos SWBDB va a desaparecer, y con ella los registros de las ofertas de trabajo. Los datos de cada una de ellas estarán disponibles en un fichero XML online. Por ello, se propone sustituir el código fuente de acceso a esta base de datos, por un código fuente de lectura del fichero XML.

En la Figura 10 se muestra esquemáticamente el funcionamiento de la aplicación, centrado en el acceso a las ofertas de trabajo, empleando los casos de uso ilustrados en la Figura 4. Se han obviado procedimientos como acceso a las listas o revocación de envíos, los cuales no es necesario modificar.

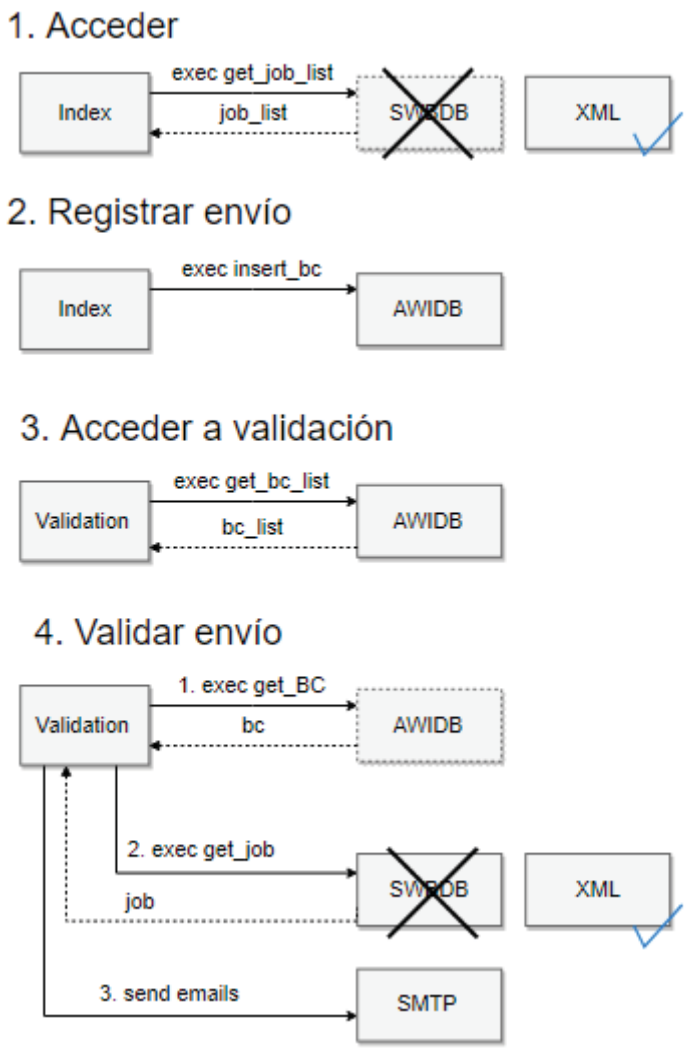

Figura 10. Posibilidad de sustituir el acceso a BD por XML

1. **Acceder:** al entrar el consultor en la aplicación, se lee de la base de datos SWBDB la lista de trabajos activos que tiene el consultor, y estos son almacenados en un modelo de

componentes dentro del programa. Se propone sustituir la lectura de SWBDB por la lectura del fichero XML.

2. **Registrar envío:** cuando el consultor selecciona la oferta de trabajo a enviar a candidatos, al pulsar el botón de envío se crea un registro en la base de datos AWIDB, en la tabla bc_deliveries que como campo relevante tiene la referencia a la oferta de trabajo, y ningún otro dato más relacionado con esta.
3. **Acceder a la validación:** este paso lo realiza el manager, cuando accede al apartado de validación de la aplicación. Se lee de la base de datos AWIDB todos los envíos pendientes de validar.
4. **Validar envío:** al pulsar el manager el botón de validar envío, se lee de la base de datos AWIDB la oferta registrada en la tabla bc_deliveries, y con la referencia de la oferta se lee de la base de datos SWBDB todos los datos de dicha oferta, para generar una plantilla que posteriormente es enviada a todos los candidatos por correo electrónico. Se propone sustituir la lectura de SWBDB por la lectura del fichero XML.

Aunque la propuesta es aceptable y viable, esta solución tiene una serie de desventajas importante:

- Leer de un XML no es igual que leer de una base de datos. Independientemente del número de ofertas de empleo que tenga el consultor, el fichero XML siempre se tendrá que leer entero. El tiempo de espera de por ejemplo un consultor con una oferta en línea, y el de un consultor con treinta ofertas en línea, será el mismo. Es muy posible que el acceso a la aplicación se ralentice mientras se lea del fichero XML online.
- Sustituir el código fuente de consulta a la base de datos, supone cambiar modelos de componentes, modificar funciones y clases, prescindir de procedimientos de la base de datos que son perfectamente reutilizables, y en general cambiar gran parte de la lógica de la aplicación.
- Son dos ficheros XML a los que se deberá acceder indistintamente, con las modificaciones extra que ello conlleva.
- Si por algún problema del proveedor, los ficheros no fuesen accesibles, tuviesen algún error o estuviesen vacíos, la aplicación permanecería inservible e impactaría a toda la empresa el tiempo que el fichero tuviese una anomalía.

4.3.2 Implementar un procedimiento que importe los datos XML

Tal y como se ha mostrado anteriormente, la aplicación sólo accedía a una tabla de la base de datos situada en el servidor SWBDB: la que contenía las ofertas de trabajo. Ya que ahora todas las ofertas de trabajo están en un XML, se propone el implementar una tarea que importe todas las ofertas XML, y las vuelque a una tabla nueva creada en el servidor AWIDB (Figura 11), que sea básicamente una réplica estructural de la existente en SWBDB.

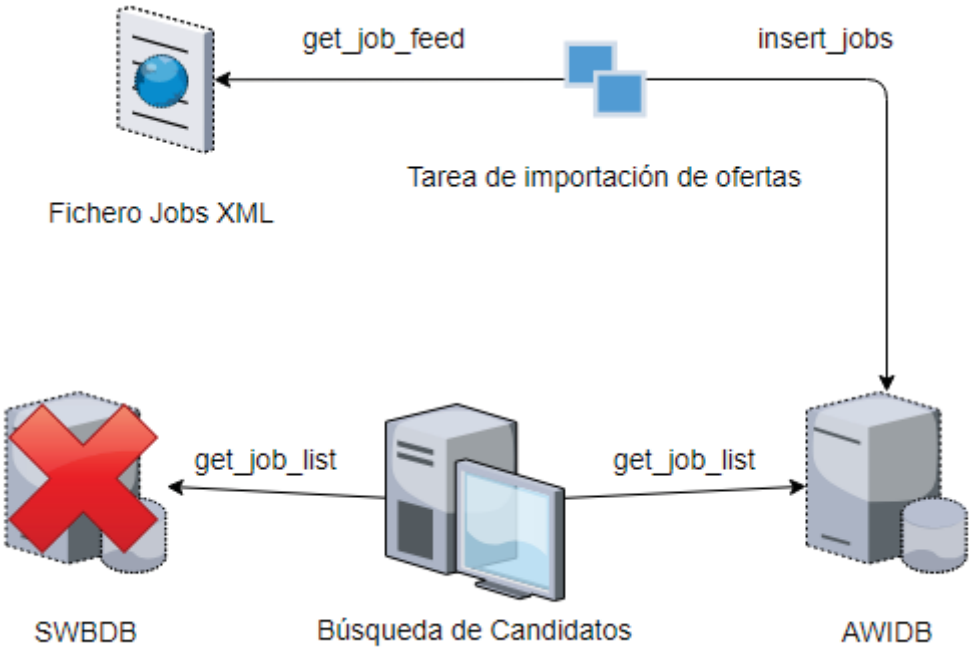

Figura 11. Esquema de procedimiento de importación XML

Esta solución parece a simple vista más eficaz que la anterior, pues no sería necesario modificar código fuente. El proyecto con el que está implementada la aplicación de Búsqueda de Candidatos posee un fichero de configuración de propiedades no compilable en el que se establecen los nombres de los servidores y bases de datos a los que conectar.

Ahora, como principal inconveniente se tiene lo siguiente: el proveedor informó que de los dos ficheros XML, el completo se actualiza una hora por la mañana, una hora por la tarde; y el parcial se actualiza el resto de horas. Esto quiere decir que desde que el consultor publica una oferta, hasta que esta es importada a la base de datos AWIDB y por tanto visible en la aplicación, puede llegar a pasar hasta dos horas, como se puede ver en la Figura 12. El tiempo que pasa desde que un consultor envía una oferta para validar, y esta es validada por su director, no queda reflejado pues se tiene en cuenta desde que la oferta ya ha sido publicada.

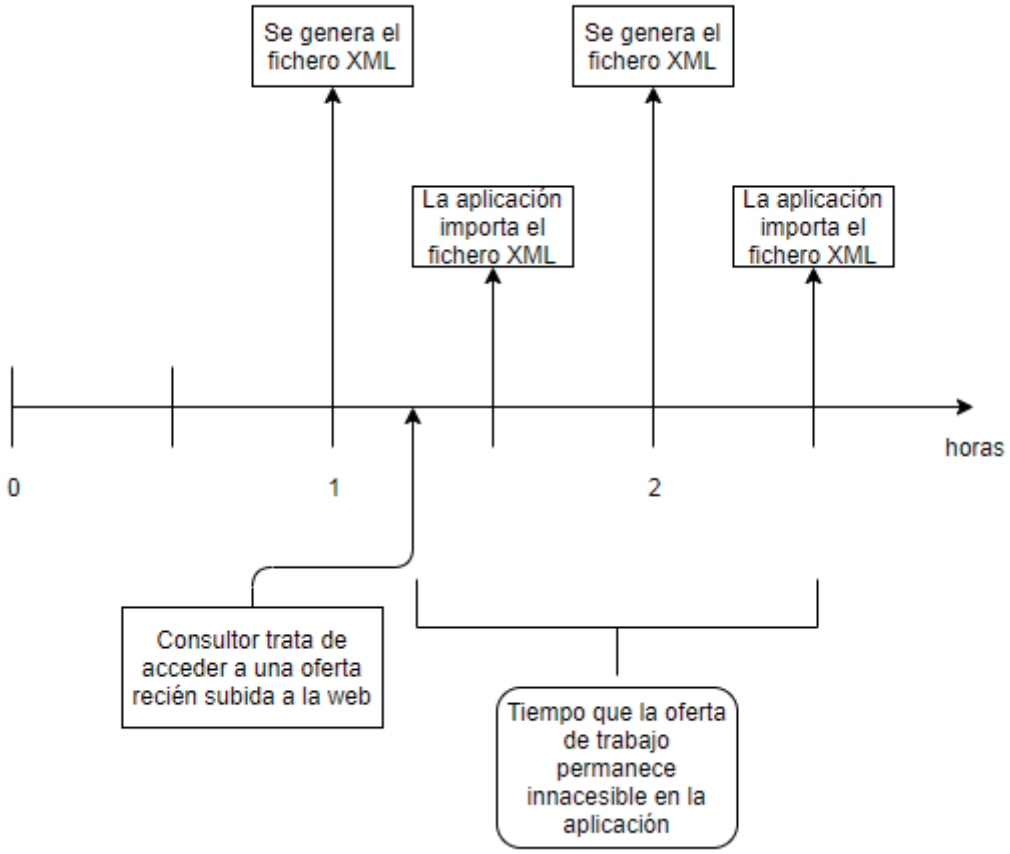

Figura 12. Tiempo de espera para que una oferta de empleo sea visible

Por tanto, es muy importante dejar establecidas con el proveedor las horas en las que se actualizarán los ficheros XML, para minimizar en la medida de lo posible el tiempo de espera del consultor desde que publica una oferta hasta que esta aparece disponible en la aplicación.

4.3.3 Diseño de la solución elegida

Una vez elegida la solución software a diseñar, ahora es necesario concebir la lógica de dicha solución.

El primer paso que se da es definir una nueva tabla en el servidor AWIDB que replique los campos de la tabla de ofertas de empleo del servidor SWBDB (Figura 13).

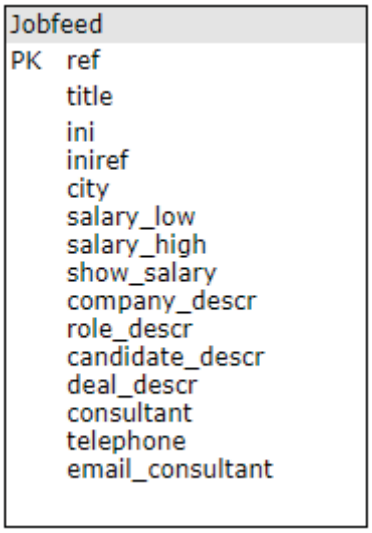

Figura 13. Esquema de tabla de BD Jobfeed

El proveedor facilita la estructura que tendrán los ficheros XML:

```
<jobs>
	<job>
		<uniqueJobID></uniqueJobID>
		<language></language>
		<location>
			<code></code>
			<text></text>
			<term></term>
		</location>
		<id></id>
		<ref></ref>
		<country></country>
		<description>
			<role></role>
			<bulletPoints>
				<bulletPoints_0></bulletPoints_0>
				<bulletPoints_1></bulletPoints_1>
			</bulletPoints>
			<candidate></candidate>
			<company></company>
			<deal></deal>
		</description>
		<status></status>
		<active></active>
		<client_authorised></client_authorised>
		<searchable></searchable>
		<title></title>
		<updated></updated>
		<published></published>
		<summary>
			<content></content>
			<title></title>
		</summary>
		<created></created>
		<brand></brand>
		<consultant>
```

```
                        <name></name>
                        <email></email>
                        <cvxemail></cvxemail>
                        <office/>
                        <telephone></telephone>
                    </consultant>
                    <contractType></contractType>
                    <employer>
                        <name></name>
                        <id></id>
                        <hide></hide>
                    </employer>
                    <industry>
                        <code></code>
                        <term></term>
                    </industry>
                    <international></international>
                    <logoImage>
                        <type></type>
                        <tmp_name></tmp_name>
                        <name></name>
                        <h></h>
                        <w></w>
                    </logoImage>
                    <minisite>
                        <type></type>
                        <ref></ref>
                    </minisite>
                    <productType></productType>
                    <salary>
                        <max></max>
                        <currency></currency>
                        <min></min>
                        <period></period>
                        <show></show>
                    </salary>
                    <sector>
                        <code></code>
                        <term></term>
                    </sector>
                    <subSector>
                        <code></code>
                        <term></term>
                    </subSector>
                    <job_level></job_level>
                    <Contract_Duration></Contract_Duration>
                    <Executive_NonExecutive></Executive_NonExecutive>
                    <company_type></company_type>
                    <Job_Detail_URL></Job_Detail_URL>
                    <Job_Start_Date></Job_Start_Date>
                    <Opportunity_Type></Opportunity_Type>
                </job>
            </jobs>
```

De todos los campos, y tal y como se ha mostrado en la Figura 13, la tabla de la base de datos para la aplicación de búsqueda de candidatos necesita los siguientes:

```
<jobs>
```

```
        <job>
                <title></title>
                <ref></ref>
                <location>
                        <code></code>
                        <text></text>
                        <term></term>
                </location>
                <salary>
                        <max></max>
                        <min></min>
                        <show></show>
                </salary>
                <description>
                        <role></role>
                        <candidate></candidate>
                        <company></company>
                        <deal></deal>
                </description>
                <consultant>
                        <name></name>
                        <email></email>
                        <telephone></telephone>
                </consultant>
        </job>
</jobs>
```

Faltan dos campos que no son proporcionados en el fichero XML:

- “ini”: el nombre de cuenta de cada consultor
- “iniref”: el nombre de cuenta de cada consultor unido al número de referencia de una oferta de trabajo

Cuando un consultor accede desde su PC a la aplicación web de Búsqueda de Candidatos, esta contiene un procedimiento de autenticación SSO (Single Sign-On) que toma su nombre de usuario de Windows. Con él, lanza un procedimiento en SQL dentro del servidor AWIDB que recopila toda la información relacionada con ese usuario (tal y como se mencionó brevemente en el apartado 1.1, hay una tabla específica para almacenar los datos de todos los consultores de la empresa) en un modelo de componentes. Entre otros datos, uno de los que almacena es “ini”. Este parámetro era utilizado, entre otras cosas, para enlazar cada oferta de trabajo con su consultor responsable. Ahora el proveedor ha decidido prescindir de él, pero en la aplicación no se puede descartar, puesto que es utilizado para diferentes procedimientos.

No obstante, el XML proporciona igualmente el correo electrónico del consultor. Dentro de la tabla de consultores, se puede conseguir el parámetro “ini” mediante el correo electrónico.

La Figura 14 presenta un primer borrador básico del funcionamiento de la tarea de importación de ofertas de empleo, que temporalmente se llama JobFeedImport.

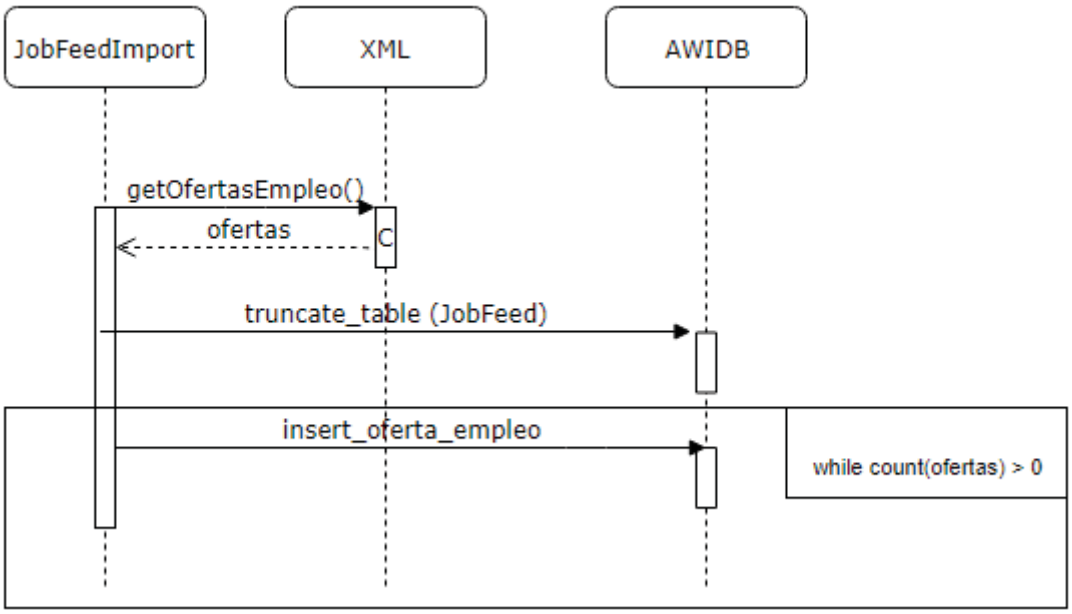

Figura 14. Borrador del diagrama de secuencia del proceso de importación de ofertas de empleo

El procedimiento insert_oferta_empleo será el encargado de almacenar cada oferta en la tabla JobFeed, accediendo a la tabla Consultores para rescatar el campo "ini" necesario. La Figura 15 muestra una primera aproximación de lo que sería necesario.

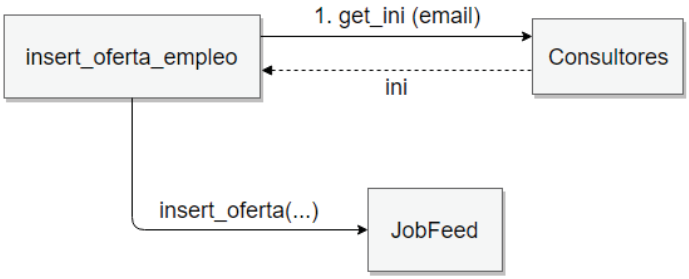

Figura 15. Funcionamiento de la función que inserta la oferta en la BD

Una vez establecido el proceso a implementar, se pasa a precisar las reglas. Es necesario tener en cuenta los siguientes puntos:

- Si el fichero XML está vacío, o es inaccesible, o hubiera algún tipo de excepción al acceder, el procedimiento debe parar ahí. No debe seguir porque se eliminarían los valores de la tabla.
- No conviene descartar que los datos del XML fueran erróneos. Por ejemplo, referencias de empleo no numéricas, o correos electrónicos mal construidos, o incluso campos vacíos. Es necesario establecer control de excepciones para todos los campos que vayan a ser introducidos en la base de datos. Si alguna oferta estuviera mal construida al ser importada, o tuviese algún valor erróneo, se debe descartar y continuar con el proceso. El error debe ser registrado en un fichero ".log" histórico.

- No se debería descartar, que en el peor de los casos, todos los datos del fichero XML fueran erróneos (pese a que en la página web se muestren correctamente las ofertas de empleo). En ese caso, tras obtener los datos erróneos mediante la función getOfertasEmpleo(), la tabla JobFeed sería borrada, y al tratar de introducir los valores en la tabla, estos serías descartados por ser incorrectos y la tabla acabaría vacía. Sería conveniente crear una tabla temporal en la que introducir todas las ofertas, y hacer la comprobación de si está o no vacía. De estarlo, la tabla JobFeed permanecería intacta y la aplicación podría seguir utilizándose hasta que se arreglara el fichero XML.
- Cada oferta de trabajo será almacenada en un modelo de componentes, y cada modelo en una lista de objetos. De esta manera, un procedimiento procesará la lista y por cada elemento, hará la inserción en la base de datos.

Teniendo en cuenta los puntos anteriores, la Figura 16 muestra cómo evoluciona el diagrama de secuencia de importación de ofertas.

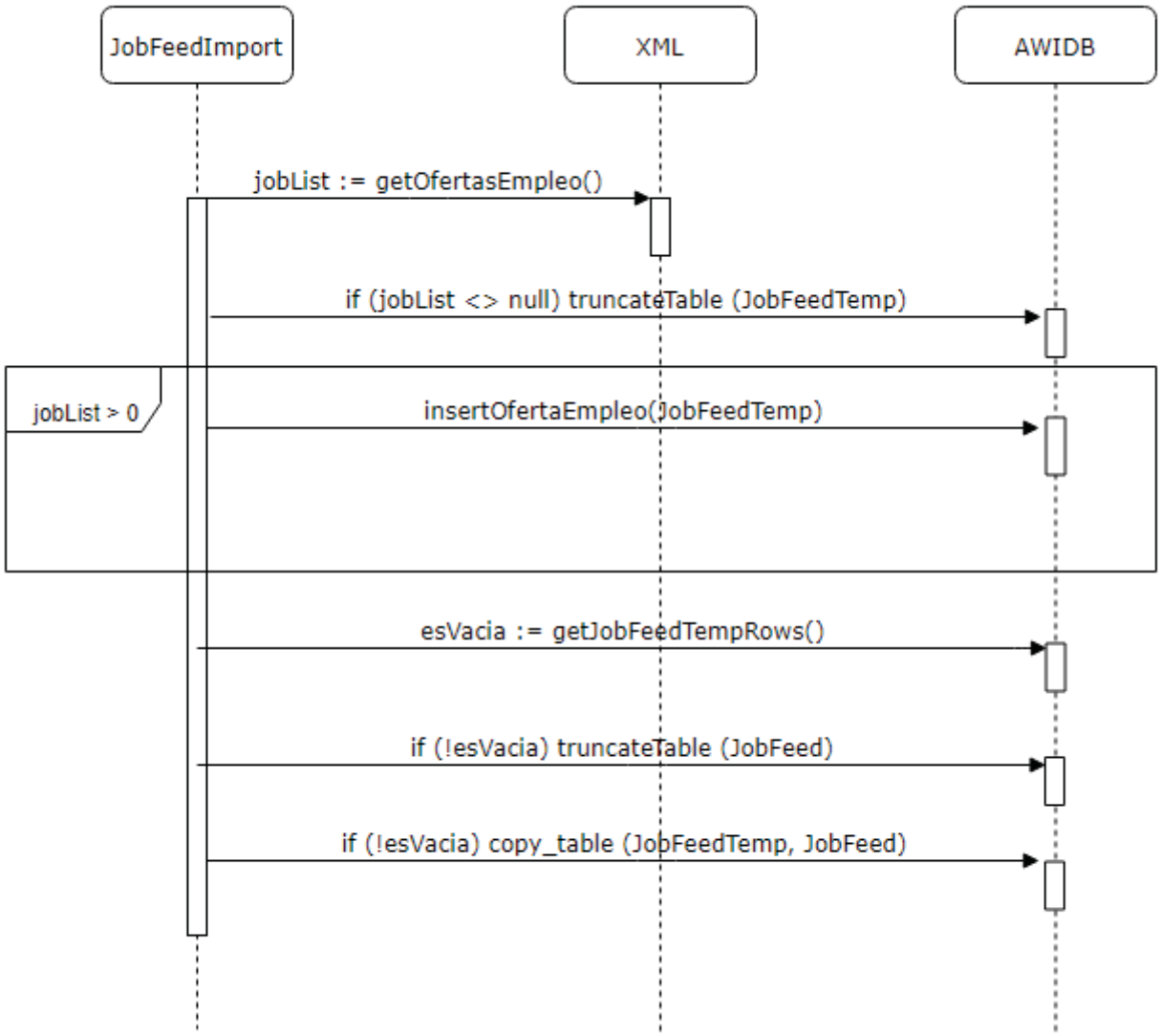

Figura 16. Nuevo diagrama de secuencia de importación de ofertas

A continuación, se explica el flujo del diagrama de secuencia ilustrado en la Figura 16 utilizado para la importación de ofertas de trabajo desde el fichero completo XML que genera el proveedor:

1. Se leen las ofertas de empleo del fichero completo XML y se almacenan una por una en un modelo de componentes, y el conjunto de modelos en una lista JobList.
2. Si la lista no es vacía, se eliminan todos los registros de la tabla temporal JobFeedTemp.
3. Por cada elemento que tenga la lista, se introducen los campos de cada elemento en la tabla de base de datos temporal JobFeedTemp.
4. Se cuentan las filas de la tabla temporal.
5. Si la tabla temporal no está vacía, se procede a borrar la tabla final JobFeed.
6. Se copian los datos de la tabla temporal a la tabla final.

Caminos alternativos:

- Lista vacía

2. Si la lista es vacía, o ha habido alguna excepción no controlada, se registra el fallo en un fichero histórico.
3. Se sale del programa.

- Tabla temporal vacía
  5. Si la tabla temporal es vacía, se registra el fallo en un fichero histórico.
  6. Se sale del programa.

El diagrama muestra dos procedimientos que es necesario desarrollar para su posterior codificación:

getOfertasEmpleo()

Esta función lee una por una las ofertas contenidas en el fichero completo XML y almacena cada una en un modelo de componentes. Lo que debe procesar por cada oferta es lo siguiente:

5. Leer campo title.
6. Leer campo ref. Se comprueba que el número de caracteres es igual a 6, y que todos son numéricos.
7. Leer campo text.
8. Leer campo salary_low.
9. Leer campo salary_high.
10. Leer campo show_salary.
11. Leer campo company.
12. Leer campo role.
13. Leer campo candidate.
14. Leer campo deal.
15. Leer campo name
16. Leer campo email. Comprobar que no es vacío, que contiene "@" y contiene ".".
17. Leer campo telephone.

Antes de pasar a la siguiente oferta, el modelo de componentes que acaba de guardar toda la información de una oferta precisa es almacenado en una lista.

Caminos alternativos:

- Campo ref no numérico
  2. Si el campo ref no es numérico, o el número de caracteres es distinto de 6, registrar el error en un fichero histórico.
  3. Ir a la siguiente oferta.
- Campo email incorrecto
  12. Si el campo email es vacío, o no contiene "@" o ".", registrar el error en un fichero histórico.
  13. Ir a la siguiente oferta.

InsertOfertaEmpleo(jobList)

Se coge de entrada la lista de objetos que contiene todas las ofertas, y por cada una de ellas se llama al procedimiento de base de datos insert_oferta_empleo.

1. Obtener de la tabla Consultores el campo ini mediante el campo de correo eléctronico,

2. Si se ha devuelto en la consulta el ini, insertar en la tabla temporal JobFeedTemp todos los campos de entrada, más el ini y el iniref (resultado de unir el campo ini y el ref).

Caminos alternativos:

2. Si no se devuelve campo ini, no hacer inserción en la tabla. Devolver un error.

En el caso que la llamada al procedimiento devolviese un error, este debe registrarse en un fichero histórico.

getJobFeedTempRows()

Desde el procedimiento, se hace una consulta a la base de datos que devuelva el número de filas que contiene la tabla JobFeedTemp. Si es mayor que cero, devuelve un valor buleano a true. De lo contrario, a false.

truncateTable()

Se hace una llamada a la base de datos para truncar la tabla que se pase por argumento.

copyTable()

Se invoca un procedimiento en la base de datos que copie los datos de la tabla temporal a la tabla final.

La tarea JobFeedImport es la encargada de leer el fichero XML final. Aparte, para leer el fichero XML parcial será necesario implementar otra tarea similar, a la que temporalmente se llamará JobFeedIncremental (Figura 17). Compartirá gran parte de los procedimientos, pero estos variarán ligeramente en sus funciones. Como punto importante, se da por hecho que la tabla JobFeed contiene filas y no está vacía. De no ser así, esta tarea pierde su funcionalidad.

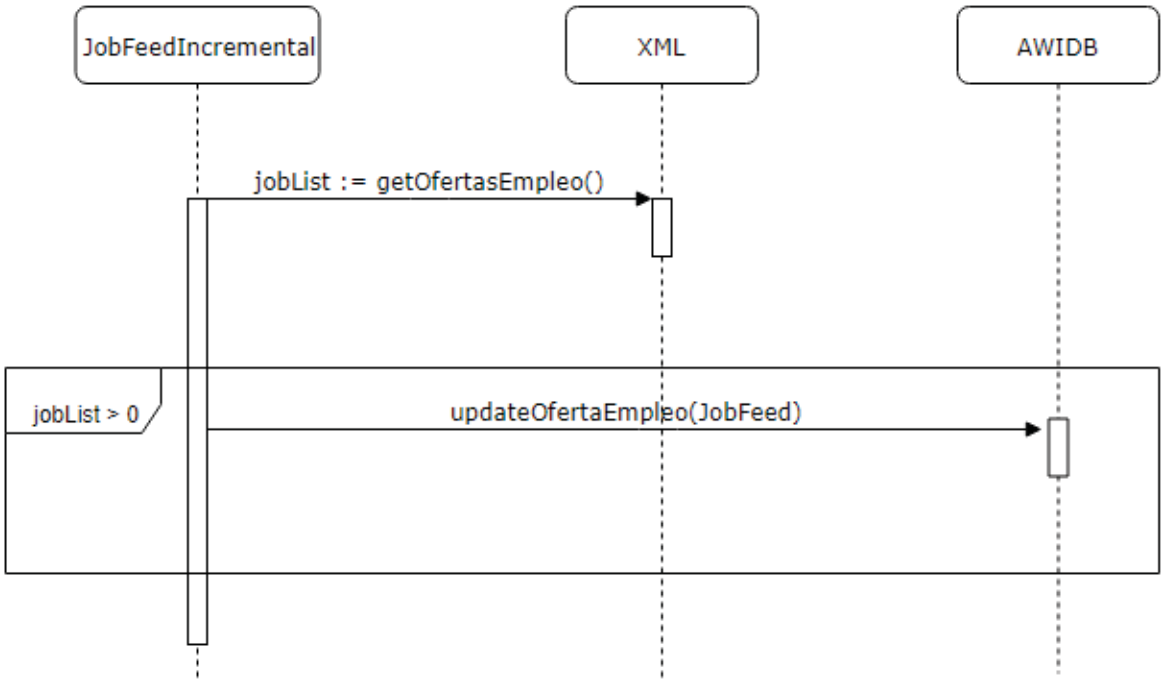

Figura 17. Diagrama de secuencia para la importación parcial de ofertas de empleo

El procedimiento getOfertasEmpleo() atiende a la misma lógica que en la tarea JobFeedImport, salvo por el fichero XML al que accede.

getOfertasEmpleo()

Esta función lee una por una las ofertas contenidas en el fichero parcial XML y almacena cada una en un modelo de componentes. Lo que debe procesar por cada oferta es lo siguiente:

1. Leer campo title.
2. Leer campo ref. Se comprueba que el número de caracteres es igual a 6, y que todos son numéricos.
3. Leer campo text.
4. Leer campo salary_low.
5. Leer campo salary_high.
6. Leer campo show_salary.
7. Leer campo company.
8. Leer campo role.
9. Leer campo candidate.
10. Leer campo deal.
11. Leer campo name
12. Leer campo email. Comprobar que no es vacío, que contiene "@" y contiene ".".
13. Leer campo telephone.

Antes de pasar a la siguiente oferta, el modelo de componentes que acaba de guardar toda la información de una oferta precisa es almacenado en una lista.

Caminos alternativos:

- Campo ref incorrecto
  2. Si el campo ref no es numérico, o el número de caracteres es distinto de 6, registrar el error en un fichero histórico.
  3. Ir a la siguiente oferta.
- Campo email incorrecto
  12. Si el campo email es vacío, o no contiene "@" o ".", registrar el error en un fichero histórico.
  13. Ir a la siguiente oferta.

El procedimiento encargado de insertar los datos en la tabla ha variado ligeramente, respecto a la tarea JobFeedImport.

updateOfertaEmpleo(jobList)

Se coge de entrada la lista de objetos que contiene todas las ofertas, y por cada una de ellas se llama al procedimiento de base de datos update_oferta_empleo.

1. Obtener de la tabla Consultores el campo ini mediante el campo de correo eléctronico,
2. Si se ha devuelto en la consulta el ini, hacer una consulta a la tabla JobFeed para comprobar si existe algún registro con el valor del campo ref igual al parámetro ref de entrada.
3. De ser así, actualizar en la tabla JobFeed todos los campos de entrada, más el ini y el iniref (resultado de unir el campo ini y el ref).

Caminos alternativos:

- No existe campo ini
  2. Si no se devuelve campo ini, no hacer inserción en la tabla.
  3. Devolver un error.
- No existe el campo ref
  3. Si no se encontrara el valor del parámetro ref, entonces insertar en la tabla JobFeed todos los campos de entrada, más el ini y el iniref (resultado de unir el campo ini y el ref).

En el caso que la llamada al procedimiento devolviese un error (por consecuencia de entrar en caminos alternativos), este debe registrarse en un fichero histórico.

4.4 Codificación

Una vez definido el diseño software, es necesario elegir los lenguajes de programación más convenientes para su desarrollo.

4.4.1 Bases de datos

Como ya se mencionó en el apartado anterior, la creación de nuevas tablas y procedimientos de bases de datos se realizará en el servidor AWIDB. Esto implica que el lenguaje de bases de datos utilizado será T-SQL dentro de un sistema Microsoft SQL Server (Microsoft Corporation). A continuación se muestran las implementaciones más significativas:

- Las dos tablas temporal y final. El campo ref es una clave primaria, pues no deberían aparecer ofertas duplicadas.

```
CREATE TABLE [jobFeedTemp](
	[title] [nvarchar](100) NULL,
	[ini] [varchar](4) NOT NULL,
	[ref] [int] NOT NULL PRIMARY KEY,
	[iniref] [varchar](10) NOT NULL,
	[city] [nvarchar](50) NULL,
	[salary_low] [int] NULL,
	[salary_high] [int] NULL,
	[show_salary] [bit] NULL,
	[company_descr] [text] NULL,
	[role_descr] [text] NULL,
	[candidate_descr] [text] NULL,
	[deal_descr] [varchar](255) NULL,
	[consultant] [nvarchar](50) NULL,
	[telephone] [varchar](30) NULL,
	[email_consultant] [varchar](60) NOT NULL,
)


CREATE TABLE [jobFeed](
	[title] [nvarchar](100) NULL,
	[ini] [varchar](4) NOT NULL,
	[ref] [int] NOT NULL PRIMARY KEY,
	[iniref] [varchar](10) NOT NULL,
	[city] [nvarchar](50) NULL,
	[salary_low] [int] NULL,
	[salary_high] [int] NULL,
	[show_salary] [bit] NULL,
	[company_descr] [text] NULL,
	[role_descr] [text] NULL,
	[candidate_descr] [text] NULL,
	[deal_descr] [varchar](255) NULL,
	[consultant] [nvarchar](50) NULL,
	[telephone] [varchar](30) NULL,
	[email_consultant] [varchar](60) NOT NULL,
)
```

- El procedimiento encargado de insertar las ofertas a la tabla temporal

```
CREATE procedure [insert_oferta_empleo]
	@title nvarchar(100),
	@ref int,
	@city  nvarchar(50),
	@salarylow int,
	@salaryhigh int,
	@showsalary bit,
	@companydescr text,
	@roledescr text,
	@candidatedescr text,
	@dealdescr varchar(255),
	@consultant nvarchar(50),
	@telephone varchar(30),
	@emailconsultant nvarchar(60)
as
```

```
declare @ini varchar(4)
declare @iniref varchar(10)
declare @err_message nvarchar(255)

IF NOT EXISTS (Select ini from [Consultores] where email = @emailconsultant)
BEGIN
SET @err_message = 'Variable ini no encontrada para consultor con email '
      + @emailconsultant + '. Oferta ' + @ref + ' no insertada en la tabla'
RAISERROR (@err_message, 11,1)
END

select
@ini = ini
from [Consultores]
where email = @emailconsultant;

insert into JobFeedTemp values
(@title, @ini, @ref, @ini + CONVERT(VARCHAR(6), @ref), @city, @salarylow,
 @salaryhigh, @showsalary, @companydescr, @roledescr, @candidatedescr,
@dealdescr,
 @consultant, @telephone, @emailconsultant);
```

En la Figura 15 se vio esquemáticamente la lógica de este procedimiento. De los parámetros de entrada se toma el campo "@emailconsultant", dirección de correo electrónico del consultor, y se busca en la tabla Consultores que contiene todo el directorio activo de consultoría si existe alguna fila que tenga el mismo valor que ese campo. En caso afirmativo, se coge de esa fila el campo "ini" necesario para incluir en la tabla temporal JobFeedTemp. También se combina el campo "ini" y el campo "ref" para formar el campo "iniref". Si no existiese ninguna fila con el correo electrónico proporcionado, se interrumpiría el proceso y se lanzaría un error.

- El procedimiento encargado de actualizar las ofertas

```
CREATE procedure [update_oferta_empleo]
      @title nvarchar(100),
      @ref int,
      @city  nvarchar(50),
      @salarylow int,
      @salaryhigh int,
      @showsalary bit,
      @companydescr text,
      @roledescr text,
      @candidatedescr text,
      @dealdescr varchar(255),
      @consultant nvarchar(50),
      @telephone varchar(30),
      @emailconsultant nvarchar(60)
as

declare @ini varchar(4)
declare @ref_old int
declare @iniref varchar(10)
declare @err_message nvarchar(255)

IF NOT EXISTS (Select ini from [Consultores] where email = @emailconsultant)
BEGIN
SET @err_message = 'Variable ini no encontrada para consultor con email '
```

```
        + @emailconsultant + '. Oferta ' + @ref + ' no actualizada en la tabla'
RAISERROR (@err_message, 11,1)
END

select @ini = ini
from [Consultores]
where email = @emailconsultant;

set @iniref = @ini + CONVERT(VARCHAR(6), @ref)

select @ref_old = count(ref) from tb_as_feed_jobfeed where ref=@ref

if (@ref_old > 0)
        begin
                update JobFeed
                set title = @title, ini = @ini, iniref = @iniref, city = @city,
                        salary_low = @salarylow, salary_high = @salaryhigh,
                        show_salary = @showsalary, company_descr = @companydescr,
                        role_descr = @roledescr, candidate_descr = @candidatedescr,
                        deal_descr = @dealdescr, consultant = @consultant,
                        telephone = @telephone, email_consultant = @emailconsultant
                where ref=@ref
        end
else
        begin
                insert into JobFeed values
                (@title, @ini, @ref, @iniref, @city, @salarylow,
                 @salaryhigh, @showsalary, @companydescr, @roledescr,
                 @candidatedescr,
                 @dealdescr, @consultant, @telephone, @emailconsultant);
        end
```

En el procedimiento de actualización de ofertas, se busca igualmente el campo “ini” dentro de la tabla Consultores perteneciente al consultor con correo electrónico el email introducido como parámetro de entrada. Si existe, se almacena en una variable, y en caso contrario se lanza un error. Acto seguido se mira si el parámetro ref no existiese ya dentro de la tabla JobFeed. De existir, se actualizaría la tabla con los parámetros de entrada. De lo contrario, se insertaría un nuevo registro en la tabla con la nueva oferta de empleo.

- Consulta para copiar los datos de la tabla temporal a la final (invocado desde copyTable()):

```
insert into JobFeed
select * from JobFeedTemp;
```

- Consultas para truncar las tablas temporal y final:

```
truncate table JobFeedTemp;

truncate table JobFeed;
```

- Consulta para conseguir el número de filas de la tabla temporal

```
select count(*) from JobFeedTemp;
```

4.4.2 Código fuente de las tareas de importación

Tal y como se señaló en la introducción de este documento, las aplicaciones del sistema están desarrolladas con Struts (Apache Software Foundation, 2018), un entorno de trabajo bajo la plataforma Java Enterprise Edition basado en el patrón modelo-vista-controlador.

Debido a la similitud entre la aplicación Búsqueda de Candidatos y las tareas de importación (en cuanto a conexiones, bases de datos accedidas, etc.), se decide que lo más seguro y adaptable es crear una pequeña aplicación en Java Struts encargada de las tareas de importación.

En la Figura 18 se ven dos clases creadas para la importación. Una "JobFeedImport" correspondiente al proceso encargado de procesar el fichero XML completo, y otra "JobFeedIncremental" para el fichero XML parcial. Ambas se apoyan en una clase pública, un modelo de componentes, encargada de almacenar todos los datos del XML para luego volcarlos a las base de datos.

Figura 18. Diagrama de clases del proceso de importación de ofertas de trabajo

Se implementa el código para importación del fichero XML final y posterior volcado en un archivo JobFeedImport.java. Para la importación del fichero parcial, se implementa el código en un archivo JobFeedIncremental.java.

- La función getOfertasEmpleo() que comparten ambas tareas importa en bucle una por una todas las ofertas del fichero XML, y las almacena en una lista de objetos JobFeedImportBean:

```
SAXBuilder builder = new SAXBuilder();
String ref = "";
String email = "";
URL url = new URL(rb.getString("JOBFEED_XML_URL"));
URLConnection connURL;
connURL = url.openConnection();
BufferedReader in = new BufferedReader(new
InputStreamReader(connURL.getInputStream()));
```

```
Document document = (Document) builder.build(in);
in.close();
Element rootNode = document.getRootElement();
List<Element> list = rootNode.getChildren("job");
List<JobFeedImportBean> joblist = new Vector<JobFeedImportBean>();
JobFeedImportBean jobs = null;
for (int i = 0; i < list.size(); i++)
{
       jobs = new JobFeedImportBean();
    Element tabla = list.get(i);
    jobs.setTitle(tabla.getChildTextTrim(Constants.TITLE));
    ref = tabla.getChildTextTrim(Constants.REF);
    if (ref.length() > 6 || !ref.matches("^-?\\d+$")) {
       Logger.error("Oferta " + ref + " no introducida por formato desconocido");
       continue;
    }
    jobs.setRef(Integer.parseInt(ref));
    List<Element> city = tabla.getChildren(Constants.LOCATION);
    jobs.setCity(city.get(0).getChildTextTrim(Constants.CITY));
    List<Element> salaries = tabla.getChildren(Constants.SALARY);
    Element campo = salaries.get(0);
    if (!campo.getChildTextTrim(Constants.SALARY_LOW).equals(""))
      jobs.setSalaryLow(Integer.parseInt(
                  campo.getChildTextTrim(Constants.SALARY_LOW)));
    if (!campo.getChildTextTrim(Constants.SALARY_HIGH).equals(""))
      jobs.setSalaryHigh(Integer.parseInt(
                  campo.getChildTextTrim(Constants.SALARY_HIGH)));
    jobs.setShowSalary(Integer.parseInt(
            campo.getChildTextTrim(Constants.SHOW_SALARY)));
    List<Element> description = tabla.getChildren(Constants.DESCR);
    campo=description.get(0);
    jobs.setCompanyDescr(campo.getChildTextTrim(Constants.COMPANY_DESCR));
    jobs.setRoleDescr(campo.getChildTextTrim(Constants.ROLE_DESCR));
    jobs.setCandDescr(campo.getChildTextTrim(Constants.CANDIDATE_DESCR));
    jobs.setDealDescr(campo.getChildTextTrim(Constants.DEAL_DESCR));

    List<Element> consultant = tabla.getChildren(Constants.CONSULTANT);
    campo=consultant.get(0);
    jobs.setCons(campo.getChildTextTrim(Constants.CONS_NAME));
    jobs.setTelephone(campo.getChildTextTrim(Constants.CONS_PHONE));
    email = campo.getChildTextTrim(Constants.CONS_EMAIL);
    Pattern patronEmail =
    Pattern.compile("^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)"
            + "@[A-Za-z0-9-]+(\\.[A-Za-z0-9]+)(\\.[A-Za-z]{2,})$");
    Matcher mEmail = patronEmail.matcher(email.toLowerCase());
    if (mEmail.matches()){
      jobs.setEmailCons(email);
    } else {
      Logger.error("Oferta " + ref + " no introducida por email " +
                  email + " incorrecto");
    }
    joblist.add(jobs);
}
```

Puntos a señalar del código:

- “rb” es la instanciación de un objeto ResourceBundle que almacena variables en

un fichero de propiedades ".properties". Para la dirección web XML completa, lee de la variable "JOBFEED_XML_URL"; para la parcial, lee de la variable "JOBFEED_INC_XML".

- "Constants" es la clase de un fichero "Constants.java", en el que se han creado las variables con valores correspondientes a los campos de los ficheros XML.

4.5 Pruebas

Una vez terminado de desarrollar el código, es hora de arrancarlo y someterlo a las pruebas necesarias para asegurar su correcto funcionamiento. Puesto que el proveedor no es capaz de proporcionar un fichero XML provisional sobre el que realizar pruebas, se crea uno manualmente para llevar a cabo las pruebas precisas.

4.5.1 Pruebas para JobFeedImport

4.5.1.1 Básico

1. Lanzar la aplicación.
2. Verificar que se ha ejecutado correctamente y sin fallos.
3. Verificar que se han volcado a la base de datos el mismo número de ofertas que las que incluía el fichero XML.
4. Verificar que todos los campos están rellenos correctamente.

4.5.1.2 Incluir una oferta que contenga un campo ref incorrecto

1. Hacer la prueba con un campo que contenga caracteres alfabéticos, que esté vacío, y que contenga signos de puntuación.
2. Verificar que esa oferta no se ha introducido en la tabla.
3. Revisar que el resto de ofertas sí lo hayan hecho.
4. Verificar que sus campos estén rellenos correctamente.
5. Verificar que se ha ejecutado correctamente y sin fallos.

4.5.1.3 Incluir una oferta que contenga un campo email incorrecto

1. Hacer la prueba con un campo que contenga una dirección que no esté construida correctamente, y con un campo vacío.
2. Verificar que esa oferta no se ha introducido en la tabla.
3. Revisar que el resto de ofertas sí lo hayan hecho.
4. Verificar que sus campos estén rellenos correctamente.
5. Verificar que se ha ejecutado correctamente y sin fallos.

4.5.1.4 Importar un fichero XML correcto, y seguidamente uno vacío

1. Comprobar que la tabla JobFeedTemp está vacía, y la tabla JobFeed llena.

4.5.1.5 Utilizar un fichero XML que tenga varios campos ref duplicados

1. Verificar que sólo se introduce uno de ellos en la tabla
2. Verificar que el resto de ofertas se hayan introducido correctamente
3. Verificar que los campos estén rellenos correctamente.
4. Verificar que todo se ha ejecutado correctamente y sin fallos.

4.5.2 Pruebas para JobFeedIncremental

4.5.2.1 Básico

1. Lanzar la aplicación.
2. Verificar que se ha ejecutado correctamente y sin fallos.
3. Verificar que se han volcado/actualizado a la base de datos el mismo número de ofertas que las que incluía el fichero XML.
4. Verificar que todos los campos nuevos están rellenos correctamente.

4.5.2.2 Incluir una oferta que contenga un campo ref incorrecto

1. Hacer la prueba con un campo que contenga caracteres alfabéticos, que esté vacío, y que contenga signos de puntuación.
2. Verificar que esa oferta no se ha introducido/actualizado en la tabla.
3. Revisar que el resto de ofertas sí lo hayan hecho.
4. Verificar que sus campos estén rellenos correctamente.
5. Verificar que se ha ejecutado correctamente y sin fallos.

4.5.2.3 Incluir una oferta que contenga un campo email incorrecto

1. Hacer la prueba con un campo que contenga una dirección que no esté construida correctamente, y con un campo vacío.
2. Verificar que esa oferta no se ha introducido/actualizado en la tabla.
3. Revisar que el resto de ofertas sí lo hayan hecho.
4. Verificar que sus campos estén rellenos correctamente.
5. Verificar que se ha ejecutado correctamente y sin fallos.

4.5.2.4 Importar un fichero XML correcto teniendo la tabla JobFeed vacía

1. Comprobar que en la tabla todas las ofertas han sido insertadas correctamente.

4.5.3 Pruebas para la aplicación “Búsqueda de Candidatos”

4.5.3.1 Básico

1. Entrar en la aplicación, verificar que hay ofertas disponibles para elegir y ser enviadas.
2. Crear envío y enviar para validar.
3. Entrar en el panel de validación.
4. Seleccionar el envío enviado y validar.
5. Comprobar que se ha enviado correctamente por email y que la oferta se muestra correctamente.

4.5.3.2 Importar un fichero XML lleno, y seguidamente uno vacío

1. Entrar en la aplicación.
2. Comprobar que hay ofertas seleccionables.

4.6 Implementación

Una vez a punto el código fuente del proyecto, se debe proceder a instalar el resultado en la máquina de cliente. Es necesario conocer a qué horas se debe lanzar la tarea completa, y a qué horas la tarea parcial.

4.6.1 Instalación en servidor AWIDB

El servidor AWI funciona con un sistema operativo Windows Server 2012, el cual dispone de una aplicación propia llamada "Programador de tareas" (Microsoft, 2018), en la que se puede programar las horas a las que se debe lanzar una determinada aplicación.

Lo que se necesita primeramente es convertir el código fuente del proyecto en un fichero ejecutable. Para ello, utilizando la versión Java instalada en el servidor, se ejecuta una instrucción dentro de la carpeta de ficheros construidos tras compilar el código fuente.

```
C:\Java\jdk1.6.0_13\bin\jar -cfv JobFeedImport.jar .\*
```

Ahora, el fichero "JobFeedImport.jar" contiene las dos clases, JobFeedImport y JobFeedIncremental necesarias para la importación de XML y volcado de información a la base de datos.

La aplicación de Windows, Programador de Tareas, va a lanzar, según proceda, dos ficheros Batch (Microsoft, 2009) diferentes.

- Fichero JobFeedImport.bat

```
@echo off


echo ********************************************
echo *              JOB FEED IMPORT               *
echo ********************************************


set JAR_HOME=C:\Javatasks\JobFeeds

set LIB_HOME=C:\Javatasks\JobFeeds\LIB


set CLASSPATH=%MAIN_PATH%;%JAR_HOME%\JobFeedImport.jar;
%LIB_HOME%\jdom-2.0.6.jar;%LIB_HOME%\log4j-
1.2.8.jar;%LIB_HOME%\sqljdbc4.jar;%LIB_HOME%\commons-
EUDC.jar;%LIB_HOME%\jconn2.jar;%LIB_HOME%\servlet.jar;


java -classpath %CLASSPATH% jobfeedimport.JobFeedImport
```

- Fichero JobFeedIncremental.bat

```
@echo off


echo ********************************************
echo *              JOB FEED INCREMENTAL          *
echo ********************************************


set JAR_HOME=C:\Javatasks\JobFeeds

set LIB_HOME=C:\Javatasks\JobFeeds\LIB
```

```
set CLASSPATH=%MAIN_PATH%;%JAR_HOME%\JobFeedImport.jar;
%LIB_HOME%\jdom-2.0.6.jar;%LIB_HOME%\log4j-
1.2.8.jar;%LIB_HOME%\sqljdbc4.jar;%LIB_HOME%\commons-
EUDC.jar;%LIB_HOME%\jconn2.jar;%LIB_HOME%\servlet.jar;


java -classpath %CLASSPATH% jobfeedimport.JobFeedIncremental
```

Ahora, la aplicación de Programador de Tareas se programa para lanzar estos ficheros ".bat" 10 minutos después de las horas a las que el proveedor haya estipulado. En un primer momento, se decide aplicar este retardo por si acaso llevara tiempo al proveedor el exportar las ofertas, y así asegurar que los ficheros a importar estén actualizados.

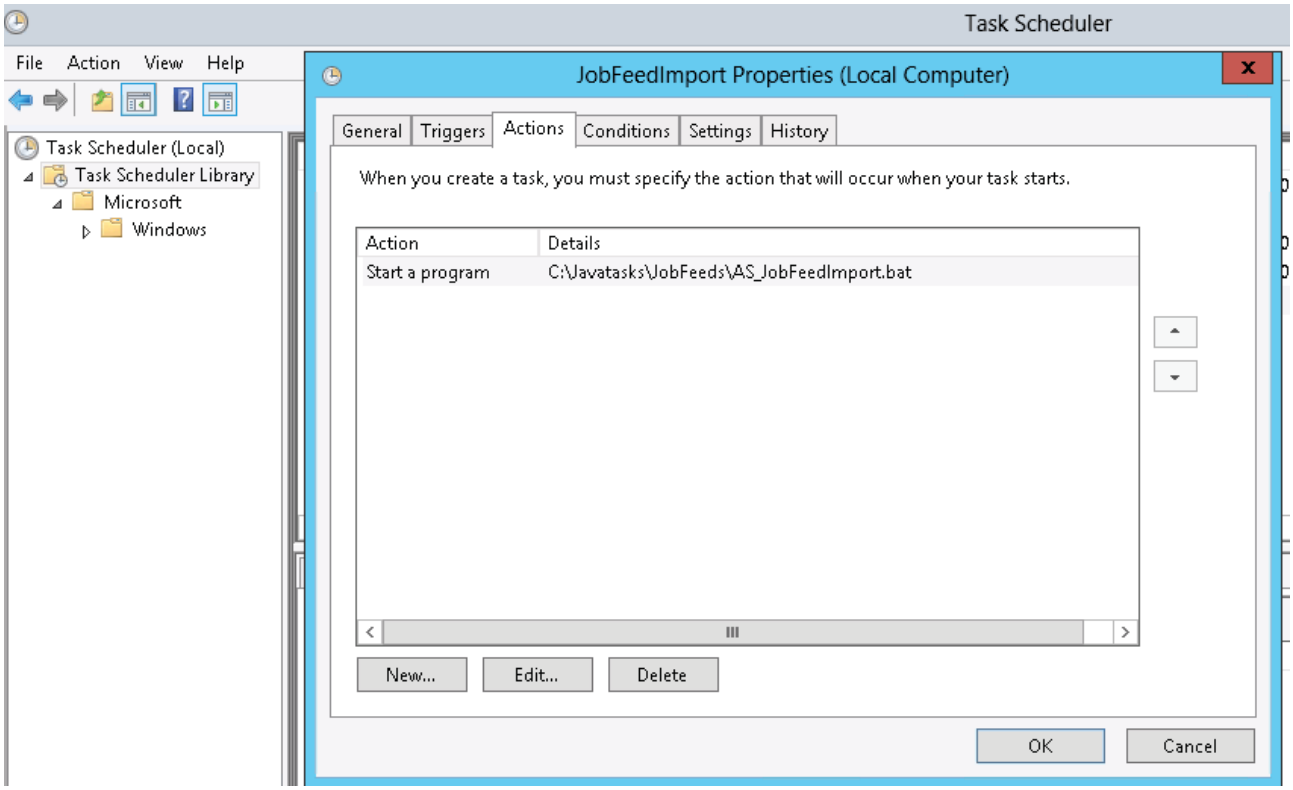

Figura 19. Programación del lanzamiento de las tareas de importación

4.6.2 Aplicación "Búsqueda de Candidatos" actualizada

Aparte de la pequeña aplicación responsable de la importación de ofertas, "Búsqueda de Candidatos" también fue modificada con las credenciales de la nueva base de datos a la que se accede. Puesto que ya está desplegada en el servidor de producción, únicamente es necesario actualizar el fichero de propiedades.

4.6.3 Error una vez aplicados los cambios

El proveedor trabajó por la noche los cambios precisos para que hubiese el menor impacto a la hora de migrar la página web de servidor. Al parecer fue un éxito, y a la mañana siguiente la

nueva página web estaba 100% funcional. Sin embargo, “Búsqueda de Candidatos” estaba inoperativa. Los consultores no eran capaces de visualizar sus ofertas al entrar a la aplicación.

Tras una rápida intervención, se observa que las ofertas no han sido importadas a la nuevas tablas JobFeedTemp y JobFeed, pese a estar las URL de los ficheros XML en línea y completamente legibles.

Se investiga el código y se detecta una anomalía en los campos de entrada: el campo “email” del fichero XML no corresponde con el correo electrónico de ningún consultor. De esta manera no es posible de relacionar las ofertas con los consultores responsables.

Se comunica de inmediato la incidencia al proveedor. La respuesta es un cambio en el modelo de aplicación a las ofertas que no fue comunicado a la empresa. A partir de ahora, las ofertas de trabajo van ligadas a correos electrónicos correspondientes a las diferentes divisiones de la empresa, y no a los consultores particulares. Esto quiebra un aspecto crítico de la solución software nacida del estudio de viabilidad, de su codificación, y es necesario volver atrás y repetir y re-estudiar los pasos dados, con el consiguiente sobrecoste tanto temporal como de eficiencia que supone.

5 Solución (2ª iteración)

Tal y como se comentó en el apartado anterior, con la primera iteración de la solución se partió con un alto grado de incertidumbre. Se dependía mucho de la predisposición del proveedor para facilitar una solución al problema derivado de la externalización de la página web. Como consecuencia, las soluciones derivadas por parte de los expertos en la aplicación tenían muy poco margen de maniobra. Por ello, se emplea un ciclo de vida software evolutivo; para gestionar las anomalías en los requerimientos iniciales.

5.1 Comunicación: alternativas

La empresa se pone nuevamente en contacto con el proveedor, para buscar una alternativa a la ausencia de un campo individual de correo electrónico de consultor. Se sabe que la manera para relacionar una oferta de trabajo con el consultor responsable, es o bien su nombre de usuario, o su campo ini, o bien su correo electrónico corporativo. No hay ninguna otra manera. El proveedor no ha sido capaz de proporcionar ninguno. Es necesario presentar nuevas alternativas.

5.1.1 Exportación de correos electrónicos

Al igual que para las ofertas de trabajo, el proveedor podría generar un fichero XML en línea que relacionara los correos electrónicos de cada consultor con los correos electrónicos de la división a la que pertenecen. No sería necesario más que generarlo una vez diaria, pues no es habitual que los consultores cambien de división frecuentemente.

Dentro del servidor AWIDB se crearía una tabla sencilla con dos campos: correo electrónico de la división, y correo electrónico del consultor. Se podría sopesar el crear tres tablas: una con los correos de cada división, otra con los correos de cada consultor, y una tercera para enlazar los campos de las anteriores, pero sería demasiado aparatoso para lo básico que es.

El procedimiento JobFeedImport accedería a esta tabla para enlazar las ofertas de trabajo con sus consultores responsables.

5.1.2 Importar el correo electrónico desde la plataforma PLF

Se inspecciona la estructura de base de datos de la plataforma PLFDB.

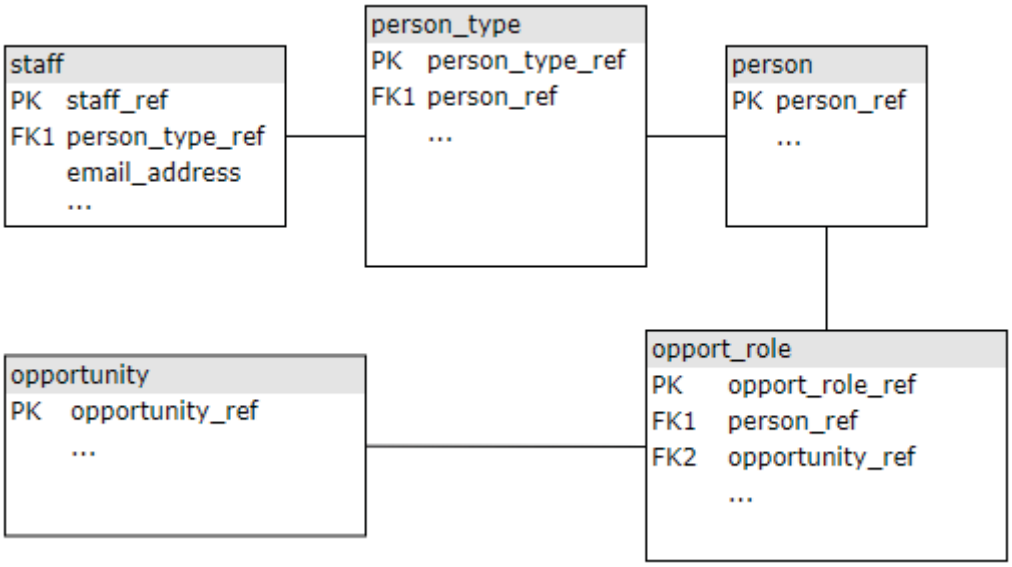

Figura 20. Parte del diagrama de base de datos de la plataforma PLFDB

A continuación se explica brevemente la función de cada tabla que aparece en la figura.

- **Opportunity:** la tabla en la que se almacenan las ofertas de trabajo. Enlaza con la tabla Opport_role mediante el campo "opportunity_ref", que es la clave primaria y el identificador de cada una de las ofertas de trabajo.
- **Opport_role:** tabla de roles de las ofertas de trabajo. Una tabla opportunity tiene varias tablas Opport_role ligadas: una para el consultor primario, otra para un consultor secundario (opcional), y otra para el/los contactos. Enlazan con la tabla person mediante el campo person_ref, que es la clave primaria de dicha tabla e identificador de los individuos que hay en ella.
- **Person:** agrupa consultores, directores, contactos, y candidatos. Enlaza con la tabla Person_type mediante la clave person_ref.
- **Person_Type:** es una precisión de algunos individuos de la tabla Person: más concretamente candidatos temporales, candidatos permanentes y consultores. Estos últimos se encuentran en la tabla Staff y están enlazados mediante la clave person_type_ref.
- **Staff:** tabla con todos los consultores. En esta tabla se encuentra el campo "email" de cada consultor.

Esta asociación parece una vía posible para conectar a un consultor con su oferta de trabajo. Tal y como se estudió en el apartado 4.1, sección "Importar las ofertas de trabajo directamente de la plataforma PLF", sería viable la consulta directa a la base de datos PLFDB, pues el número de accesos es bastante bajo.

5.2 Diseño software y codificación

El proveedor descarta cualquier cambio más, y alega que no puede ofrecer más información que la incluida en el fichero XML ya existente.

Por lo tanto, no es necesario un estudio de viabilidad más que el que ya se ha mostrado en el apartado anterior acerca del acceso al servidor PLFDB. Es necesario cambiar el diseño

secuencial, introduciendo la base de datos de la plataforma PLFDB, que es de donde se extraerá el correo electrónico del consultor.

Como se puede observar en la Figura 21, sería necesario acceder una vez por cada oferta a la base de datos PLFDB, para a través de la referencia de la oferta, obtener el correo electrónico del consultor.

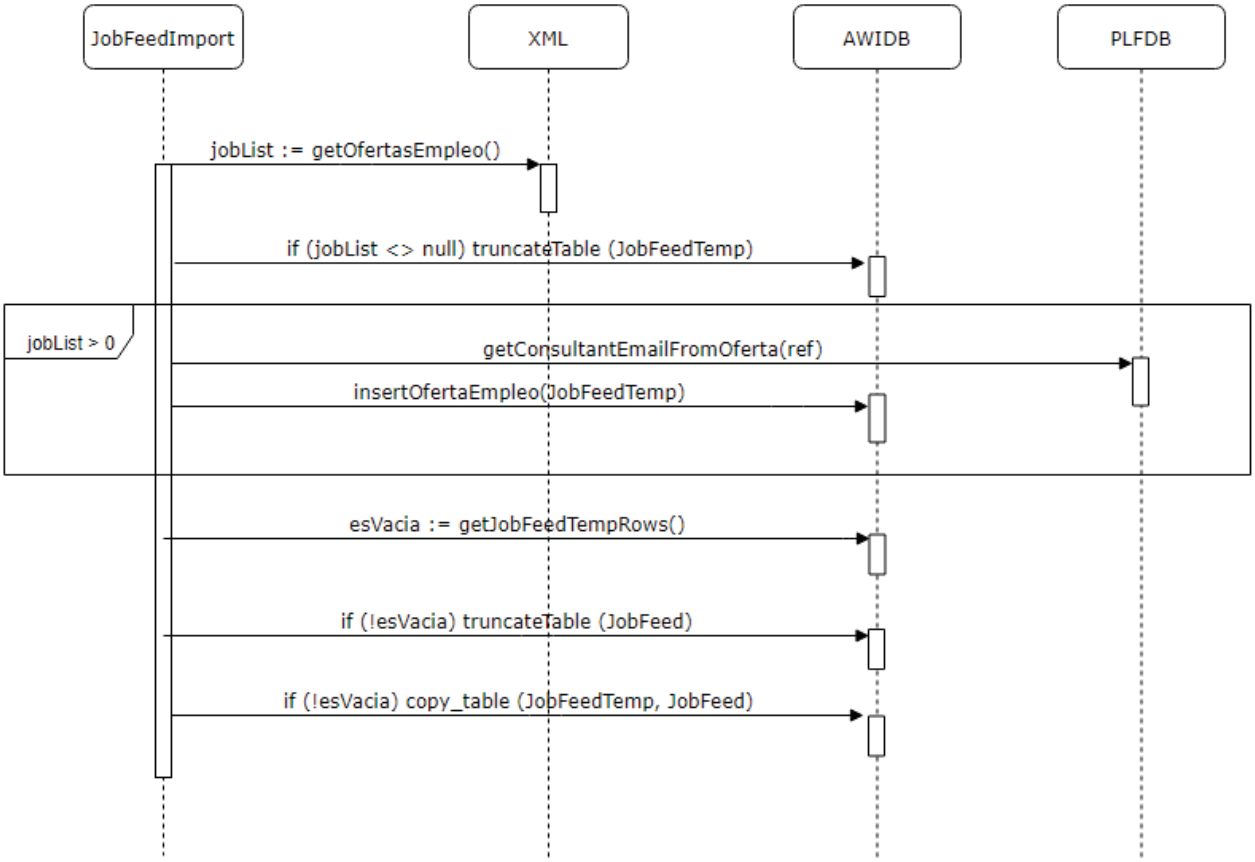

Figura 21. Último diagrama de secuencia de importación de ofertas

Igualmente, la tarea JobFeedIncremental se ve afectada por este cambio (Figura 22).

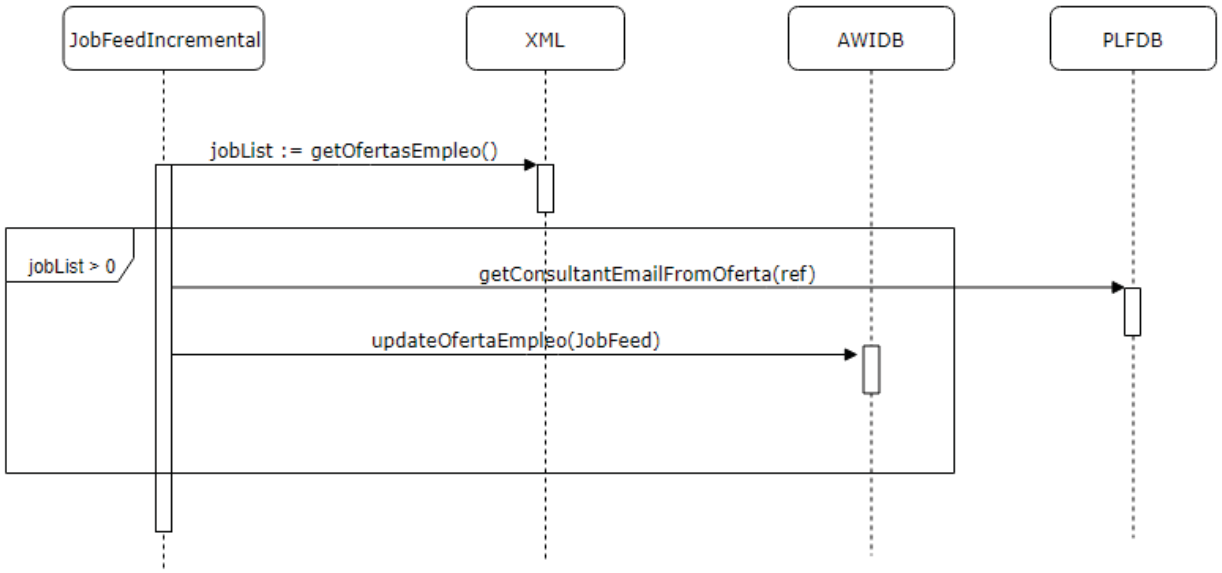

Figura 22. Último diagrama de secuencia del proceso de actualización de ofertas

La consulta a la base de datos que debe realizarse desde la nueva función getConsultantEmailFromOferta() sería la siguiente:

```
select s.email_address from opportunity opp
inner join opport_role oppr on oppr.opportunity_ref=opp.opportunity_ref
inner join person p on p.person_ref=oppr.person_ref and oppr.role_type='U1'
inner join person_type pt on pt.person_ref=p.person_ref
inner join staff s on s.person_type_ref=pt.person_type_ref
where opp.opportunity_ref=?
```

Siendo el símbolo "?" cada una de las referencias de cada oferta de trabajo leídas en el bucle de las listas "jobList".

Tras estos cambios, sería necesario aplicar las pruebas del apartado 4.5 y actualizar la tarea de importación nuevamente, tal y como se expone en el apartado 4.6.

6 Resultados y conclusiones

Si bien es cierto que no se perseguía ninguna mejora, sino la sola adaptación de la aplicación a un cambio externo, gracias a las modificaciones aplicadas, la aplicación Búsqueda de Candidatos consigue estar operativa y funcional.

Las limitaciones del proveedor han sido una dificultad añadida a la hora de elaborar una solución software, pues ha demostrado poca flexibilidad para ayudar a hacer una adaptación óptima de la aplicación a los nuevos cambios.

Es complicado mantener la lógica de una aplicación si los requisitos cambian de una manera tan inesperada. Sin embargo, seguramente el equipo que trabajó por primera vez en la implementación del proyecto tomó como precondición que las ofertas de trabajo serían accesibles permanentemente, y no contempló que en realidad todo engloba un conjunto de sistemas dependientes unos de otros. El proceso de importación de ofertas de trabajo ubicado en el servidor web SWB dependía para su funcionamiento del servidor de base de datos PLFDB, y la aplicación de Búsqueda de Candidatos ubicada en el servidor AWI dependía tanto del servidor AWIDB como de la plataforma PLFDB. A la hora de implementar una solución, se debería haber tenido en cuenta esta dependencia, para introducir algún patrón de diseño que facilitara obtener bajo acoplamiento, baja dependencia entre componentes.

Por ejemplo, un patrón de diseño estructural de tipo fachada habría reducido considerablemente la complejidad (Figura 23).

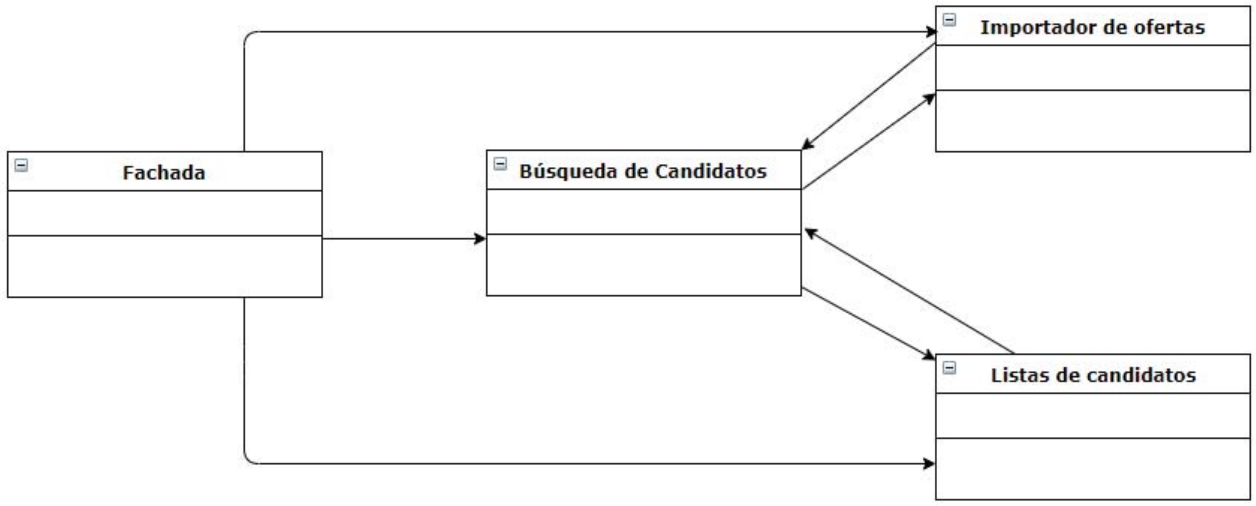

Figura 23. Patrón fachada aplicado a la aplicación Búsqueda de Candidatos

Una solución podría haber sido rediseñar desde cero la aplicación, pero ello habría implicado casi el triple de carga de trabajo para estudiar cada módulo. Por otro lado, en este punto en el que se busca la externalización gradual de los servicios de la empresa, no habría sido una solución viable desconociendo si a corto plazo la aplicación terminara por desaparecer. Por ello, se ha tratado de mantener la lógica interna en la medida de lo posible.

Haciendo una evaluación de las modificaciones, antes de la externalización existía una tarea independiente a la aplicación encargada de importar las ofertas cada 30 minutos a una base de datos de un servidor vinculado al de aplicaciones AWIDB. Después de la externalización, la tarea independiente se dividió en dos. Una tarea encargada de exportar las ofertas desde la web

externa (lado del proveedor) cada 60 minutos, y una tarea encargada de importar esas ofertas (lado de la empresa) cada 60 minutos, con un margen de 10 minutos entre ambas. Es decir, un consultor podría llegar a tardar en ver una oferta hasta 70 minutos (60 minutos de espera máximos + 10 minutos de margen de espera) desde el momento en que es validada, frente a los 30 minutos máximos del procedimiento anterior. Sin embargo, en este caso resultaba imposible mejorar los tiempos si el proveedor no actualiza en menos tiempo el fichero XML en línea.

De todas las alternativas, seguramente la solución más destacada habría sido el emplear servicios web. El hecho de que al acceder el consultor a la aplicación Búsqueda de Candidatos, con tan solo comunicar con su nombre de usuario a la otra máquina, habría simplificado enormemente la complejidad de la aplicación. No habría sido necesario la creación de ninguna tabla adicional dentro de la base de datos. Sin embargo, habría sido necesario modificar la lógica y por tanto el código fuente.

7 Líneas futuras

Tal y como se ha comentado en el apartado anterior, el futuro de la aplicación de Búsqueda de Candidatos es incierto a largo plazo. La empresa atraviesa por un proceso de externalización que teóricamente tendría por objetivo migrar todos los servidores a un proveedor externo, incluido el que contiene a la aplicación.

A corto plazo se contempla estudiar una serie de actualizaciones y mejoras en torno a la aplicación Búsqueda de Candidatos.

7.1 Actualizaciones software

Actualmente se trabaja con tecnología obsoleta que es conveniente renovar, por un lado porque ciertos productos dejan de recibir soporte, y por otro, porque es la mejor manera de evitar errores de software como bugs informáticos.

- Migración a SQL Server 2014: deberían hacerse pruebas de todos los procedimientos SQL invocados desde la aplicación y comprobar si fuera necesario cambiar algunas funciones.
- Actualizar el compilador Java de 1.6 a 1.8: actualmente el código no utiliza ninguna función que esté obsoleta, pero será interesante estudiar nuevas funcionalidades y compatibilidades que la nueva versión pueda ofrecer.
- Actualizar/sustituir el servidor de aplicaciones JBoss 4.2: necesario para garantizar la robustez, escalabilidad y velocidad no sólo de Búsqueda de Candidatos, sino del resto de aplicaciones de la empresa. Dentro de los gratuitos y de código abierto, se pretende analizar JBoss 7, Wildfly, y Glassfish.
- Actualizar Eclipse: es el entorno de desarrollo empleado para implementar la aplicación. La actual versión, Kepler, no soporta Java 8. De hacerse la actualización Java, sería necesario buscar una nueva versión que lo soporte. Por otro lado, se ha de buscar igualmente una versión que soporte el nuevo servidor de aplicaciones elegido.

7.2 Pool de conexiones

Es una técnica utilizada en la que el servidor (y no la aplicación) maneja un número de n conexiones ya creadas y abiertas a la base de datos, y por tanto, reutilizables.

De esta manera, la aplicación se debe modificar para que las conexiones a la base de datos se conviertan en peticiones al pool (cursohibernate.es, 2016).

El pool tiene dos puntos principales de mejora respecto a la situación actual:

- El servidor de aplicaciones tendrá configurado en su pool la conexión a la base de datos que corresponda. La aplicación ya no maneja ni nombres ni instancias a la base de datos. Sólo se debería indicar el nombre del origen de datos que apunta al pool. De haber tenido un pool de conexiones antes de trabajar en este proyecto, no habría habido siquiera que modificar el fichero de configuración de la aplicación de Búsqueda de Candidatos.
- El tiempo que se emplea en crear conexiones y cerrarlas desaparece, pues el servidor tendrá un número n de conexiones abiertas, y la aplicación las irá pidiendo según necesite, y es el pool el que administra cuándo cerrar, o cuándo crear nuevas, en caso de ser necesario y cómo esté configurado. Este punto sería muy útil si se consiguiera configurar un pool contra el servidor de base de datos PLFDB.

Para el caso que nos ocupa, el funcionamiento sería el mostrado en la Figura 24.

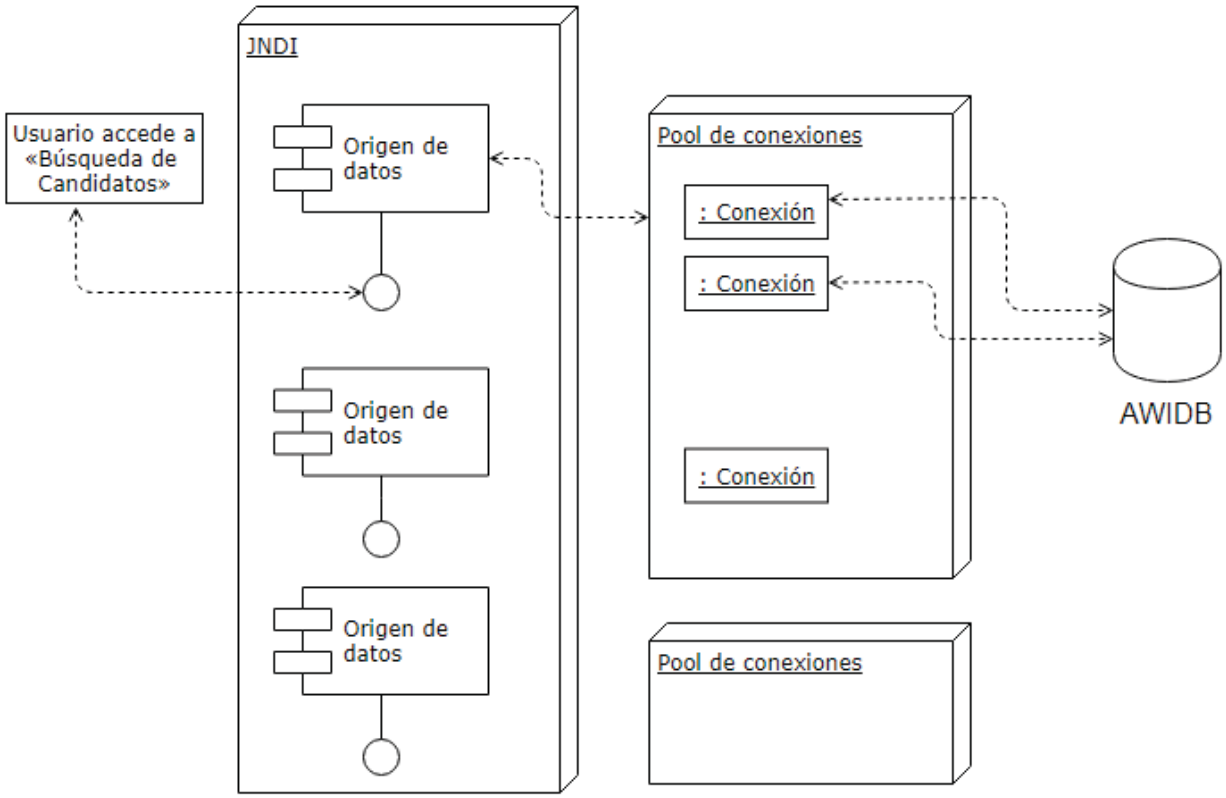

Figura 24. Líneas Futuras: pool de conexiones

Sería también de gran utilidad el estudiar su funcionamiento en aplicaciones independientes como la implementada en este proyecto, JobFeedImport.

8 Bibliografía

Apache Software Foundation. (2018). *Struts*. Obtenido de https://struts.apache.org

BEA Systems. (2018). *BEA WebLogic Platform 8.1 Documentation*. Obtenido de https://docs.oracle.com/cd/E13196_01/platform/docs81/index.html

Cantone, D. (2006). Ciclo de Vida del Software. En D. Cantone, *Implementación y Debugging* (pág. 320). Users.Code.

cursohibernate.es. (2016). *Pool de conexiones.* Obtenido de http://cursohibernate.es/doku.php?id=patrones:pool_conexiones

Ellingwood, J. (2013). *How To Use SFTP to Securely Transfer Files with a Remote Serve.* Obtenido de https://www.digitalocean.com/community/tutorials/how-to-use-sftp-to-securely-transfer-files-with-a-remote-server

Guru99.com. (2016). *What are Web Services? Architecture, Types, Example*. Obtenido de https://www.guru99.com/web-service-architecture.html

JBoss Inc. (2010). *JBoss Application Server 4.2.2 Administration And Development Guide*. Obtenido de https://docs.jboss.org/jbossas/docs/Server_Configuration_Guide/4/html/index.html

Microsoft. (2009). *Microsoft SQL Server 2008*. Obtenido de https://msdn.microsoft.com/en-us/library/hh825826(v=sql.10).aspx

Microsoft. (2009). *Using batch files*. Obtenido de https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb490869(v%3dtechnet.10)

Microsoft. (2017). *OPENQUERY (Transact-SQL)*. Obtenido de https://docs.microsoft.com/es-es/sql/t-sql/functions/openquery-transact-sql?view=sql-server-2017

Microsoft. (2017). *SQL Server Integration Services*. Obtenido de https://msdn.microsoft.com/en-us/library/ms141026(v=sql.120).aspx

Microsoft. (2018). *Task Scheduler*. Obtenido de https://docs.microsoft.com/en-us/windows/desktop/taskschd/task-scheduler-start-page

Reenskaug, T. (2004). *MVC XEROX PARC 1978-79*. Obtenido de http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

Sybase Software. (2013). *Sybase Central*. Obtenido de http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc00170.1540/doc/html/san1283555449750.html

World Wide Web Consortium. (2015). *XML DOM*. Obtenido de https://www.w3schools.com/xml/dom_intro.asp