Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Creación de una solución de comunicación para el software médico Cosmed Omnia con el objetivo de ser integrado con aplicaciones en la nube.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Virgilio Tello García

Tutores: Jorge Hortelano Otero
Alicia Villanueva García

2018/2019

Resumen

Ante la necesidad de importación de datos del software médico Cosmed OMNIA desde el portal de gestión y mantenimiento SportMediScore de Biit Solutions; se plantea y desarrolla una serie de soluciones para gestionar la comunicación.

Debido a que el software Cosmed OMNIA no posee servicios REST, sino intercambio de información a través de archivos estructurados en formato XML; y éste a su vez se encuentra en una red privada y protegida de un hospital, sin disponibilidad de facilitar la apertura de nuevos puertos o gestionar el intercambio de archivos mediante FTP o VPN; se plantea como primera opción desarrollar e instalar un software en el centro médico para ofrecer mediante servicios REST el intercambio de archivos, pero debido a la gestión de puertos y seguridad del hospital, es desechada.

Por lo que finalmente se desarrolla un software para el intercambio de archivos a un servidor externo, el cual ofrecerá la comunicación de tales con el software SportMediScore mediante servicios REST, bajo nuestro mantenimiento.

Para lograr el intercambio de archivos sin necesidad de abrir puertos en el centro hospitalario, se gestiona la conexión mediante Sockets de Java, manteniendo bajo nuestros servidores en la nube una aplicación servidor a la espera de un cliente, e instalando en el centro médico un simple cliente que conectará a nuestro servidor.

La solución para el intercambio de archivos está compuesta por un total de 3 sockets, un socket para la comunicación y mantenimiento de la conexión, y 2 más para el intercambio de los archivos mediante buffers de datos, siempre manteniendo el socket servidor bajo nuestros dominios.

Y finalmente una vez el intercambio de archivos ha sido resuelto, la solución de servicios REST ya desarrollada, permite un fácil intercambio y la obtención de la información deseada para el software SportMediScore.

Palabras clave: solución, comunicación, hospital, sockets.

Abstract

Given the need to import data from Cosmed OMNIA medical software from the management and maintenance portal SportMediScore of Biit Solutions; a series of solutions is proposed and developed to manage communication.

Because the Cosmed OMNIA software does not have REST services, instead have exchange information through structured files in XML format; and at the same time is located in a private network and protected from a hospital, without availability to facilitate the opening of new ports or manage the file exchange through FTP or VPN; the first option is to develop and install a software in the medical center to offer REST services through file exchange, but due to port management and hospital security, it is discarded.

So finally software is developed for the exchange of files to an external server, which will offer the communication of such with the SportMediScore software through REST services, under our maintenance.

To achieve the exchange of files without opening ports in the hospital, the connection is managed through Java Sockets, keeping a server application under our servers in the cloud waiting for a client, and installing in the medical center a simple client that will connect to our server.

The solution for the exchange of files is composed of a total of 3 sockets, one socket for communication and maintenance of the connection, and 2 more for the exchange of files through data buffers, always keeping the server socket under our domains.

And finally once the file exchange has been solved, the REST services solution already developed, allows an easy information exchange and obtaining the desired information for the SportMediScore software.

Key words: solution, communication, hospital, sockets.

Tabla de contenidos

Tabla de ilustraciones

1. Introducción

En éste documento hablaremos continuamente sobre centros hospitalarios, debido a que la solución desarrollada es aplicada para tales. En éste caso, nos centraremos en centros hospitalarios especializados en la ciencia del deporte y la salud.

Estos centros médicos requieren de completos y complejos sistemas para la obtención y medición de los resultados físicos del paciente. Para ello se requiere de una amplia gama de dispositivos hardware como bicicletas estáticas, cintas de correr, máscaras de altitud, etc.

Además de complejos softwares para recopilar y mostrar la información obtenida, ofreciéndola al médico de una manera sencilla, para que éste pueda analizarla y obtener su diagnóstico. Estos softwares están enfocados a realizar la obtención y muestra de los datos obtenidos, pero ¿qué sucede a la hora de registrar el historial médico del paciente?

Todo centro médico requiere de un software ERP [1] para almacenar y gestionar el historial médico de los pacientes, como las citas, ejercicios a realizar por el paciente, seguimiento y comparativa con resultados anteriores, y un largo etcétera. Toda esa información se introduce manualmente por el médico generalmente, a excepción de los datos que podemos importar gracias a la comunicación entre diferentes softwares.

Y en este punto nos encontramos para afrontar este proyecto, el desarrollo de una solución para la comunicación entre el software para la obtención de datos de las examinaciones del paciente, con el software para almacenar y gestionar su expediente.

1.1 Motivación

Abordar este proyecto me ofrecía dos grandes aspectos que me ayudarían a crecer como profesional de la informática.

Desarrollar una solución real para un cliente extranjero, desarrollándome en el mundo laboral en un idioma no natal, enfrentándome a desarrollar la solución en un tiempo concreto, realizar pruebas en real en el cliente y demostraciones; así como el formar parte de un equipo de desarrollo bajo metodologías ágiles, SCRUM [5] en éste caso, desarrollando mis habilidades tanto sociales como de trabajo en equipo.

Además del aprendizaje e integración de una solución completa y profesional, bajo metodologías de ingeniería del software; desarrollando la solución bajo el sistema de control de versiones Git, desplegando el proyecto a través de Maven y ejecutando los test de integración en la herramienta Jenkins.

Sin olvidar el reto principal que supone desarrollar esta solución, la transmisión de archivos desde un sistema hospitalario a un servidor externo, y viceversa [6] cuando

las soluciones convencionales como una red VPN o un servidor FTP han sido desechadas por limitaciones del centro. Era un reto cuanto menos interesante a lograr.

1.2 Objetivos

El crecimiento tanto personal como profesional. Desarrollando mis habilidades y capacidades para formar parte de un grupo profesional; compartiendo responsabilidades, tiempo y espacio junto a ellos. Así como enfrentarme a una situación de venta de un producto para un cliente real de lengua y cultura extranjera. Por otra parte, el afianzamiento y mejora de conocimientos ya adquiridos previamente, el aprendizaje de nuevas tecnologías para el desarrollo de software y puesta en práctica de metodologías de la ingeniería del desarrollo del software, tanto estudiadas en la carrera cómo por aprender.

El objetivo de la solución propuesta, es desarrollar un software para el intercambio de archivos, a un servidor externo, desde un equipo privado situado dentro de una red hospitalaria. Éste ofrecerá la comunicación de tales con el software en la nube SportMediScore, mediante servicios REST bajo nuestro mantenimiento.

1.3 Que es BiiT. Prácticas de empresa

BiiT Sourcing Solutions, S.L. [15] es una empresa privada dedicada al desarrollo de soluciones software a la medida del cliente. Focalizada en el publico holandés, en centros hospitalarios de rendimiento físico, pero abierta al desarrollo para cualquier campo de clientes.

Estableciendo su estabilidad bajo su producto principal, SportMediScore, un ERP de software médico del que hablaré en el capítulo *2.2* SportMediScore.

La empresa continúa expandiendo su campo de acción, trabajando actualmente en múltiples proyectos de diferentes caracteres; desde una novedosa idea para la gestión empresarial, formando parte de un gran equipo de desarrollo compuesto por informáticos desde India hasta Canadá, pasando por soluciones de mediana envergadura, como la que envuelve este TFG, hasta pequeñas soluciones modulares para su software principal SportMediScore.

1.4 Descripción del problema

Nos encontramos ante la necesidad de establecer una comunicación entre el software médico Cosmed OMNIA [13] y el ERP de gestión hospitalaria en la nube, SportMediScore, para la facilitación del trabajo a los médicos de los diferentes centros con lo que BiiT trabaja.

Para la obtención de datos del software médico Cosmed OMNIA, tan solo disponemos de un protocolo basado en el intercambio de archivos de estructura XML. Lo cual plantea un primer problema para la comunicación, ya que requerimos de comunicarlos fuera de la red del hospital; algo aparentemente sencillo pensando en soluciones como, habilitar una conexión VPN o un servidor FTP, pero al tratarse de un centro hospitalario, la seguridad es primordial y no se da la posibilidad de la apertura de nuevos y/o vulnerables puertos en el proxy, para facilitar estas soluciones.

En segundo lugar, encontramos el problema de comunicar una aplicación en la nube a través de documentos de texto, lo cual sería extremadamente lento, así como la implementación de un sistema de comunicación anticuado y vulnerable a la extracción de los datos incluidos en los ficheros en texto plano.

1.5 Estructura del trabajo

A continuación, expongo la solución desarrollada en dos fases. La primera fase consta de la implementación de una solución para comunicación entre el software médico Cosmed OMNIA y la aplicación en la nube SportMediScore. Esta solución es un software servidor encargado del intercambio y traducción de archivos con el software Cosmed OMNIA a servicios REST accesibles por la aplicación en la nube SportMediScore. Esta solución fue insuficiente hasta el desarrollo de la solución explicada en segundo lugar, en el momento que volvió a formar parte del conjunto proyecto final.

En segundo lugar, expondré la solución que he desarrollado para el envío de los ficheros generados para la comunicación por el software Cosmed OMNIA fuera de la red del centro hospitalario sin la necesidad de apertura de nuevos puertos, ni la exposición de datos privados de los pacientes en el proceso.

2. Contexto

2.1 Introducción a la situación previa al proyecto

Cuando me incorporé a la empresa, se hallaba en el punto de necesidad de solventar la problemática de comunicación entre el software médico Cosmed OMNIA y el servidor REST externo para la comunicación con la aplicación en la nube SportMediScore.

Se había desarrollado una primera versión del servidor REST y su traducción entre archivos de estructura XML para la comunicación con OMNIA y servicios REST para la comunicación con SportMediScore. Pero se requería de un software que pudiese

transferir los archivos fuera de la red del centro hospitalario desde la carpeta en la cual OMNIA los genera, así como obtenerlos desde una red externa hasta la susodicha carpeta en la cual OMNIA los procesa; bajo las regulaciones de seguridad establecidas, aplicadas en parte bajo ciertas reglas por el proxy.

2.2 Aplicación en la nube SportMediScore

SportMediScore es una solución software en la nube desarrollada por BiiT Sourcing Solutions S.L. para la gestión de centros hospitalarios centrados en el rendimiento físico y el bienestar. Ofreciendo funcionalidades desde gestionar las citas con los pacientes a generar informes médicos en base al estado y resultado de las pruebas de salud y rendimiento del paciente.

Desplegada por Maven, SportMediScore es una solución de estructura software compleja, trabajando en unión con distintos servicios. Bajo el flujo de motor de trabajo Activiti BPM, desarrollada en Java con una capa cliente implementada con el framework Vaadin, utilizando como almacén de datos MySQL y servida en la web gracias al servidor Apache, obtendríamos el conjunto base de tecnologías que envuelven el desarrollo y mantenimiento de SportMediScore.

Pero este software no se reduce tan solo a servir una página web para la gestión hospitalaria, sino también a la generación de informes de diagnóstico para diferentes tipos de examinaciones realizadas a los pacientes. Para ello, requiere de un servidor Drools de gestión de reglas de negocio al que se envían los datos, y éste, a través de las reglas definidas, obtiene unos resultados/conclusiones que están almacenados en el portal de gestión de contenidos Liferay; obteniendo así toda la información necesaria para generar y componer un reporte que el paciente pueda comprender.

2.3 Software médico OMNIA

OMNIA es una solución software desarrollada por Cosmed S.R.L. diseñado para la gestión de datos, la interpretación y generación de importes de test. Realizados por toda una gama de productos para la examinación de capacidades físicas de los pacientes, desde equipo para la espirometría hasta medir la capacidad pulmonar y la evaluación de la composición metabólica, hasta la composición corporal.

Este software ofrece al médico la recopilación de los valores obtenidos a través de examinaciones al paciente, para éste obtener un diagnóstico sobre el cual actuar y mejorar el estado del paciente.

Cosmed ofrece además tanto la versión Standalone como la versión en red del producto OMNIA, así como una larga extensión de módulos para acercarse más a las necesidades del centro hospitalario.

Sin olvidar OMNIA Control Protocol que nos permite la comunicación del software OMNIA con terceros a través de archivos de composición XML; permitiendo así recopilar información, ejecutar pruebas e integrar datos desde software ajeno a las soluciones Cosmed.

2.4 Integración de SportMediScore con OMNIA

Se propone unificar dos softwares de uso diario facilitando el trabajo a los médicos en el centro hospitalario, evitándoles la necesidad de introducir a mano los datos obtenidos de un software de análisis a un software de gestión reduciendo su carga laboral, así como tiempo invertido en la misma operación, además de evitar así también el posible error humano.

Para ello, es necesario de convertir los ficheros generados por OMNIA a servicios REST y viceversa, es decir, convertir las peticiones REST de SportMediScore a comandos encapsulados en archivos XML.

2.4.1 Protocolo de OMNIA

El OMNIA Control Protocol (OCP) habilita un puente entre aplicaciones de terceros y OMNIA utilizando una arquitectura a tres capas:

- Una capa de transporte que crea un punto final donde OMNIA puede ser conectado.
- Una capa de codificación que maneja la sintaxis y la gramática de la comunicación.
- Una capa de traducción que actúa como intérprete de lenguajes.

Cada capa esta también desacoplada de su vecina:

- La capa de transporte proporciona un punto final y transporta datos en ambas direcciones; no sabe nada sobre la información transmitida, solo opera con flujos de bytes.
- La capa de codificación formatea flujos de datos en formularios bien formados y viceversa. Conoce todo acerca de la sintaxis, rechazando cualquier mensaje que no la respete, pero no comprende nada acerca del contenido del mensaje.
- La capa de traducción es un intérprete que traduce mensajes de lenguaje desconocido para OMNIA, a un lenguaje que pueda interpretar y viceversa, desconociendo que acción conlleva el mensaje en cuestión.

OMNIA soporta múltiples sintaxis de lenguajes (XML, cadenas de texto) y múltiples codificaciones (ASCII, UTF-8, UTF32, Unicode, Big Endian Unicode).

Además de la clásica arquitectura cliente/servidor, OMNIA implementa un tipo de arquitectura publicación/suscripción para mensajes no solicitados por sus clientes, si el usuario inicia un nuevo test OMNIA avisa a sus clientes. Esto se debe a que mientras un test está en progreso, ciertos comandos no son aceptados y debe ser considerado por los clientes, por lo que, si el cliente desease realizar otra operación antes de que terminase la actual, recibiría una notificación de espera hasta poder realizarla.

Por razones de seguridad, todos los comandos que alterarán de alguna manera los datos almacenados en OMNIA quedan registrados en la base de datos. Para realizar cualquier comando OCP, primero debes identificarte y obtener un token de identidad que expira cuando la sesión es finalizada.

Ilustración 1. OMNIA OCP

1.1.1 Importar datos del XML a SportMediScore

Para ello se requiere de una solución software comenzada a desarrollar previo a mi incorporación a la empresa. Se trata de un servidor que escucha una carpeta compartida, en la cual se tratarían los archivos XML de OMNIA, consumiendo el archivo en cuestión y traduciendo su contenido a una petición REST que contiene la información requerida y enviándola a SportMediScore. O, por el contrario, traduciendo las peticiones REST por parte de SportMediScore a archivos XML que OMNIA pueda procesar.

2. Solución propuesta

2.1 Metodologías y tecnologías implementadas

A continuación, introduciré y presentaré a grandes rasgos las tecnologías y metodologías de la ingeniería del software que han sido implementadas para el desarrollo de ésta solución.

2.1.1 SCRUM, marco de desarrollo ágil

Scrum es el nombre que recibe el marco de metodologías ágiles, siendo un proceso de buenas prácticas que se aplican para trabajar en equipo intentando lograr el mejor resultado posible para el desarrollo de un proyecto.

Basado en la comunicación, en las entregas parciales y entregas regulares del proyecto, desarrolladas en base a la preferencia y necesidad del proyecto. Enfocado a proyectos de cierta complejidad y necesidad de obtener resultados lo antes posible, donde los requisitos son poco definidos o con facilidad a cambiar.

El equipo que forma Scrum está compuesto por tres roles:

- Product owner: En este caso representado por Henny van Doorn, fundador de BiiT Sourcing Solutions S.L.
- Scrum master: Personificado por mi tutor de prácticas, Jorge Hortelano.
- Development team: Desenvuelto por mi persona.

Taiga

Para dar soporte al marco de desarrollo ágil Scrum, se ha utilizado el software de código abierto Affero GPL, Taiga [12]; mediante esta herramienta se facilita la gestión y colaboración de proyectos ágiles, así como gestión de problemas y Wiki enfocados al proyecto.

Elegido este software entre tantas opciones por sus ventajas tales como:

- Potencia: fácil personalización y mayor control sobre tu proyecto.
- Simple e intuitivo: la línea de aprendizaje es mínima y su nivel de usabilidad es muy alto.
- Modulable: permite la configuración que uno desee para sus proyectos pudiendo extender sus funcionalidades gracias a la cantidad de módulos disponibles.
- Licencia: software bajo licencia de código abierto Affero GPL.

2.1.2 Git, gestor de control de versiones

Los sistemas de control de versiones son una categoría de herramientas de software que ayudan a un equipo de software a administrar los cambios en el código fuente a lo largo del tiempo. El software de control de versiones realiza un seguimiento de cada modificación del código en un tipo especial de base de datos.

Si se comete un error, los desarrolladores pueden comparar versiones anteriores del código para ayudar a corregir el error y minimizar las interrupciones para todos los

miembros del equipo. El control de versiones protege el código fuente de la catástrofe del error humano y las consecuencias no deseadas.

De entre todos los sistemas de control de versiones disponibles, se decidió usar Git [7] frente a Subversion, por estas razones entre otras:

- Control de versiones distribuido.
- Se trabaja sobre copias locales del repositorio directamente.
- Autorización de acceso para todo el proyecto.
- Seguimiento de cambios basado en contenido.
- Historial de cambios tanto en el repositorio como en las copias locales.
- Solo requiere conectividad de red para la sincronización.

2.1.3 Java, lenguaje orientado a objetos

Java [6] es un lenguaje de programación orientado a objetos, concurrente y de propósito general. Enfocado a facilitar el desarrollo de una solución y poder ejecutarla bajo cualquier tipo de dispositivo que pueda mover la máquina virtual de Java, ya que las aplicaciones Java son compiladas en bytecode, que puede ejecutarse en cualquier máquina virtual Java (JVM), sin importar la arquitectura de la computadora subyacente.

Actualmente uno de los lenguajes de programación más usados en el mundo, así como impartidos por centros educativos de programación; particularmente para aplicaciones de jerarquía cliente/servidor.

Maven

Es una herramienta open source creada con el propósito de simplificar los procesos de compilado. Pero Maven [7] no se centra tan solo en la tarea de compilado, es una herramienta capaz de gestionar un proyecto software completo; desde la etapa de comprobación de la tipificación del código, hasta que se despliega la aplicación, pasando por la ejecución de pruebas y generando informes y documentación.

Por defecto Maven divide el ciclo en las etapas:

- Validación: Validar que el proyecto es correcto.
- Compilación.
- Test: Probar el código fuente usando un framework de pruebas unitarias.
- Empaquetar: Empaquetar el código compilado y transformarlo en algún formato tipo .jar o .war.
- Pruebas de integración: Procesar y desplegar el código en algún entorno donde se puedan ejecutar las pruebas de integración.

- Verificar que el código empaquetado es válido y cumple los criterios de calidad.
- Instalar el código empaquetado en el repositorio local de Maven, para usarlo como dependencia de otros proyectos.
- Desplegar el código en el entorno.

2.1.4 Docker, gestor de contenedores software

Docker [8] es un software diseñado para crear contenedores ligeros y portables para poder ejecutar las aplicaciones software contenidas en cualquier equipo que posea Docker instalado, independientemente del sistema operativo que tenga instalado.

Docker permite integrar en el contenedor las aplicaciones que requiera mi solución software para ser ejecutada (Apache, Java, MySQL, etc.) y la propia solución. Permitiéndome portar mi solución software sin preocuparme más que de tener instalador Docker en la nueva máquina.

2.2 Fase 1. Traducción de archivos OMNIA a servicios REST

En esta primera fase explicaré la solución software desarrollada para la traducción de la comunicación de archivos OMNIA a servicios REST y viceversa. Desarrollando un servidor web intermediario entre ambos softwares desplegado bajo los servidores de BiiT Solutions.

2.2.1 Objetivos

El objetivo principal de esta solución es traducir los archivos XML generados por OMNIA a servicios REST a los que SportMediScore pueda acceder y viceversa, solucionando así el problema de comunicación mediante los diferentes protocolos de ambas plataformas software.

Además, de requerirlo, podría incluir la lógica necesaria para facilitar la comunicación, como algorítmica de datos, solicitudes de permiso para realizar ciertas peticiones, autentificación y desautentificación del usuario, etc. ya que no se descarta la implementación de nuevas funcionalidades en la aplicación web SportMediScore, para la interacción con el software médico Cosmed OMNIA.

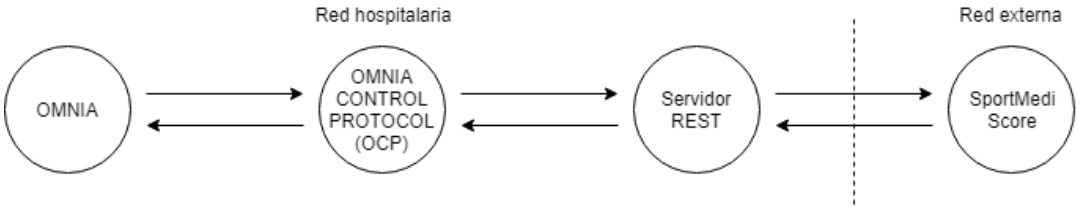

Ilustración 2. OMNIA OCP Fase 1

2.2.2 Seguridad en un Hospital

En un centro médico hospitalario se trabaja y almacena a diario una gran cantidad de datos personales de los pacientes que deben ser protegidos a toda costa por la seguridad de tales.

El robo de la información clínica deriva en diferentes acciones criminales que van, desde el uso de la información robada con fines de fraude administrativo y el consumo ilícito de fármacos, hasta incluso la venta de ficheros de datos a otros cibercriminales. Es importante recordar que la información clínica de un paciente es altamente sensible y el robo y/o mal uso de ésta tiene serias consecuencias para la propia seguridad del paciente o incluso para su elegibilidad en términos de aseguramiento, impacto en sus finanzas (procedimientos costosos realizados a un tercero de forma fraudulenta usando su identidad y cobertura médica), etc.

Por ello, todo centro hospitalario dispone de las más altas medidas de seguridad para prevenir todo tipo de ataques contra su sistema. Por lo que permitir el envío y recepción de los archivos para la comunicación del software OMNIA, que es la solución que ofrece, es totalmente inviable debido a la brecha de seguridad que esto supondría; de hecho, la información transmitida en tales ficheros ni tan siquiera está cifrada, por lo que, de ser robados dispondrían de la información del paciente con total transparencia.

2.2.3 Solución con servicios REST

Ante la necesidad de extraer la información fuera del hospital mediante otro método que no fuese el intercambio directo de archivos, se planteó el instalar un servidor que tradujese los archivos OMNIA a servicios REST y viceversa en el centro hospitalario, para poder comunicar con SportMediScore mediante unos servicios más estandarizados, y sin necesidad de comprender el protocolo OCP de Cosmed OMNIA.

Actualmente esta solución software se haya fuera de la red interna del hospital como se indicará más adelante.

2.2.4 Problemas encontrados en producción

Todo este previo contexto a la solución finalmente integrada, fue desechado por las propias medidas de seguridad del hospital, una vez llevadas dichas soluciones a producción. A continuación, se explicarán los puntos clave por los cuales estas soluciones no pudieron ser integradas.

2.2.4.1 Firewall

El hecho de integrar un servidor REST dentro del centro hospitalario evadía la problemática del intercambio de archivos, pero requería la necesidad de la apertura de puertos en el Firewall que gestiona la seguridad del centro hospitalario, y esto no fue permitido por las medidas de seguridad establecidas, por lo que se tuvo que rechazar ésta solución también.

Sin embargo, sería posible evitar este problema mediante el uso de una VPN.

2.2.4.2 Prohibición de VPN

A la hora de plantearse un intercambio de archivos, parece una idea sencilla y segura el plantear como solución una red privada virtual (VPN). Ya que nos ofrece la facilidad de acceder a la carpeta de sincronización fácilmente desde una red externa al centro hospitalario. La idea consiste en hacer una VPN alojada en la nube y conectar el equipo con el software de Omnia a esta VPN como cliente. Dándonos acceso a los puertos necesarios para instalar nuestro servidor REST.

Pero esto no pudo ser debido a que una red VPN comprometería la política de seguridad del hospital, a pesar de estar alojada exteriormente. Más teniendo en cuenta que sería gestionada por una empresa ajena al propio centro hospitalario (BiiT Sourcing en cuestión); por lo que el hospital denegó la propuesta solución de establecer una red VPN.

2.3 Fase 2. Solucionar las limitaciones impuestas

Tras descartar todas las anteriores soluciones debidos a sus respectivos motivos, se debía plantear otra manera de afrontar la problemática.

Se planteó una posible solución, tratándose de desarrollar un servidor en la nube, e instalar un cliente en el centro hospitalario, creando una capa intermedia entre el software médico Cosmed OMNIA y el servidor de traducción y servicios REST. Esta solución es similar al conocido el programa TeamViewer el cual otorga acceso remoto al

equipo en el que esté instalado, se halle dentro de una red privada hospitalaria como si no; por lo que demostrado quedaba que había una solución posible.

Ésta solución no requeriría de abrir puertos en el Firewall del centro hospitalario, permitiendo establecer una comunicación entre ambos puntos mediante el formato de texto JSON [10] y a través de las diferentes tecnologías a evaluar, se podrían transmitir los archivos OMNIA entre cliente y servidor a través de los Sockets.

2.3.1 Objetivos

Sincronizar la carpeta designada para generar/consumir los archivos por Cosmed OMNIA en el centro hospitalario, con la carpeta designada en el servidor para ser generados/consumidos por el traductor y servidor REST; actuando a espaldas del Firewall y ofreciendo la seguridad de no comprometer los datos que se están transmitiendo.

2.3.2 Nueva arquitectura propuesta

La nueva arquitectura se basa en la implantación de una solución intermedia entre el software médico Cosmed OMNIA y el servidor de traducción y servicios REST. Gestionada mediante Sockets TCP/IP Java para la comunicación entre el centro hospitalario y los servicios en la nube de BiiT Sourcing.

El servidor se encuentra desplegado en la nube, a la espera de la petición del cliente para que éste se conecte. El cliente se encontraría desplegado en el centro hospitalario, configurado para conectar a una dirección DNS y un puerto determinado; evitando así la necesidad de cualquier gestión de la seguridad del hospital por parte de su departamento técnico, ya que el centro hospitalario actúa en todo momento como cliente.

La comunicación entre cliente y servidor, se realizará mediante el formato de texto sencillo JSON para el intercambio de datos, debido a su facilidad y extensión en el campo de la comunicación; siendo actualmente uno de los formatos de texto sencillos más comunes de uso.

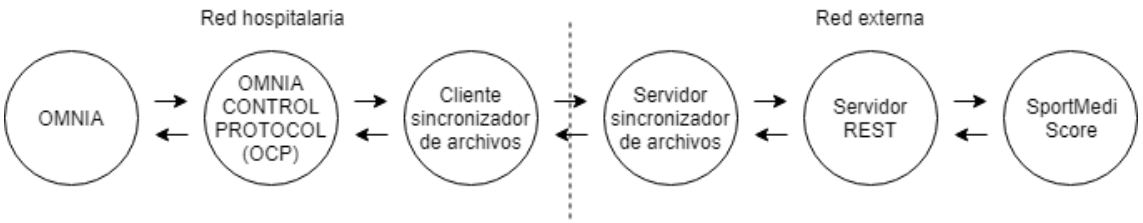

Ilustración 3. OMNIA OCP Fase 2

2.3.2.1 Sockets

Los Sockets Java permiten establecer un enlace entre dos soluciones software que se ejecutan independientemente a través de la red, actuando la clase 'Socket' como cliente y la clase 'ServerSocket' como servidor, requiriendo ambos un puerto al que conectarse/servir respectivamente, y en el caso del cliente, una dirección a la que conectarse.

Implementando todas las funcionalidades que una comunicación cliente/servidor requiere como un tiempo de conexión, establecer tipo de protocolo (TLSv1, TLSv1.1, TLSv1.2, etc.), protocolo de internet IPv4 o IPv6, etc. además de un buffer a través del cual enviar la información pertinente.

Para asegurar la privacidad de los datos de los pacientes, se han utilizado SSLSockets, que implementan Secure Sockets Layer (SSL) que protege contra la modificación de mensajes por un interceptor y encripta los datos transmitidos entre cliente y servidor.

Además, estos SSLSockets requieren de autentificación clave pública/privada, permitiendo gestionar en el servidor, los clientes específicos en los que confiar.

Almacén de claves Java

Tanto para generar las claves públicas de los clientes como para generar el almacén de claves y añadirlas a él, se ha utilizado la herramienta Java Keytool.

El almacén de claves Java permite gestionar las claves públicas de los clientes a atender, almacenándolas encriptadas y requiriendo de una contraseña para acceder a ellas, identificadas por un alias cuando son generadas.

2.3.3 Monitorizar archivos mediante Watch Service de Java

Debido a que la comunicación entre Cosmed OMNIA y el servidor de traducción se da mediante archivos, debemos de conocer cuándo un archivo es creado, eliminado o modificado en las carpetas correspondientes al servidor o cliente, para actuar en consecuencia y comunicar a la otra parte el envío del nuevo archivo o requerir que sea eliminado, manteniendo así la sincronía entre carpetas.

Para ello se ha hecho uso del servicio Java Watch Service, que permite monitorizar un directorio, notificando cuando un archivo es creado, modificado, eliminado, o en caso de que un sistema de ficheros sea más rápido informando de los eventos que la implementación que los procesa, recibiendo una notificación OVERFLOW.

Watcher Service se inicia tras establecer la conexión entre cliente y servidor en ambos extremos bajo un hilo paralelo a la ejecución principal, ya que ambos extremos deben atender a posibles nuevos eventos.

2.3.4 Gestor de configuración de la solución

La solución implementa un gestor para lectura de parámetros necesarios para los diferentes requisitos.

Éste gestor está configurado para atender en primer lugar, a un archivo nombrado 'settings.conf' almacenado en un directorio nombrado 'conf' creado en el mismo directorio donde se encuentra el ejecutable de la solución; en segundo lugar, si no encuentra tal archivo, busca una variable de sistema nombrada 'SYNCHRONIZER_CONF' que contiene la dirección del archivo de configuración. Y, por último, de no encontrar el archivo ni la variable de sistema, atiende a valores hardcodeados en la propia clase Java.

Éstos parámetros a atender son:

 # Parámetros referentes a la conexión
server.name = 127.0.0.1
server.port = 1025
client.data.server.port=1026
server.data.server.port=1027
connection.timeout=60000

 # Parámetros referentes al directorio a monitorizar y almacén temporal de archivos
sinchronization.folder = ../../../FilesLocation
security.synchronization.folder = ./tmp

 # Parámetros de localización del almacén de claves y su contraseña
key.store = ./keystores/keystore.jks
key.store.password = passwd

2.3.5 Gestor de comandos para la comunicación

La comunicación entre cliente y servidor mediante el formato de texto JSON, se ha realizado mediante comandos y subcomandos simples agrupados por funcionalidades.

Éstos comandos son gestionados mediante clases Java de enumerado, diferenciando entre:

- Comandos de acción:

{CREATE, CREATE_OK, CREATE_ERROR, CANCEL, DELETE, DELETE_OK, DELETE_ERROR, DELETE_ALL, DELETE_ALL_OK, DELETE_ALL_ERROR, REQUEST_CLIENT_DATA_SERVER_SOCKET,

CLIENT_DATA_SOCKET_OPPENED, SERVER_DATA_SOCKET_OPPENED, CONNECTED_SERVER_DATA_SOCKET, CONNECTED_CLIENT_DATA_SOCKET, REQUEST_CLOSE_SERVER_DATA_SERVER_SOCKET, REQUEST_CLOSE_CLIENT_DATA_SERVER_SOCKET, REQUEST_CLOSE_CLIENT_DATA_SOCKET, REQUEST_CLOSE_SERVER_DATA_SOCKET, CLIENT_DATA_SOCKET_CLOSED, SERVER_DATA_SOCKET_CLOSED, SERVER_DATA_SERVER_SOCKET_CLOSED, CLIENT_DATA_SERVER_SOCKET_CLOSED, RECONNECT_CLIENT, RECONNECT_SERVER, FILE_INFO, PING, PONG}

- Comandos de contenido:

{MSG, FILE_NAME, FILE_CONTENT, FILE_SIZE, MD5, ERROR}

- Comandos booleanos de respuesta:

{TRUE, FALSE}

2.3.6 Protocolo de comunicación

En un inicio, se comenzó a desarrollar un protocolo de comunicación basado en la implementación de un solo Socket, a través de cual se realizaba tanto la comunicación como el envío de archivos.

Los archivos de Omnia eran convertidos a JSON; traducidos a un Array de bytes, dividido y enviado como texto plano, siendo recompuesto en el receptor. Esto requería de identificar cada paquete con el archivo correspondiente, verificar que todas las partes habían sido recibidas sin perder ningún byte, y en el caso contrario, volver a solicitarla. Ésta solución fue descartada por la complejidad para que los ficheros pudiesen ser transmitidos correctamente (aunque el tamaño de los archivos no llega a un megabyte, la solución se planteó para afrontar la posibilidad de enviar archivos de hasta ochenta megabytes, por lo que este tamaño suponía una gran división de paquetes, lo cual generaba una extrema posibilidad de pérdidas), así como por el tiempo que tomaba este protocolo para el envío completo de un archivo.

Tras descartar ese protocolo, se decidió la necesidad del uso de dos Sockets más, dedicados exclusivamente al envío de los archivos mediante un buffer de datos; un Socket correspondiente al envío de archivos del cliente y otro al envío de archivos del servidor.

Ambos Sockets son abiertos para enviar el archivo, y una vez éste ha sido correctamente enviado son cerrados de nuevo. Son creados en el servidor, ambos, con la diferencia de que si es el Servidor quien requiere enviar un archivo, abre su correspondiente Socket y notifica al cliente para que éste se conecte, una vez conectado notifica al servidor y éste comienza a transmitir los datos; en el caso contrario, si es el cliente quien requiere de enviar un archivo, envía una solicitud al servidor para que abra el Socket, y una vez éste ha ejecutado la orden, notifica al cliente para que éste se conecte y comience a transmitir los datos.

2.3.6.1 Envío de archivo

El proceso de envío de un archivo puede comenzar de dos maneras, tras la detección por el Watcher Service de un nuevo archivo creado o modificado, o tras el envío completo y correcto de un archivo, llamando al siguiente archivo en cola para ser enviado.

A continuación, explicaré el proceso completo del envío de un archivo desde que el Watcher Service detecta el evento. Una vez un evento ya sea de modificación o creado de archivo es detectado, se comprueba que ese archivo no está almacenando en el pool de archivos creados/modificados, si no es así, lo almacena y registra en la cola de archivos creados/modificados.

Al ser añadido al pool, el Listener comunica el evento, notificando a la clase de envío de archivos, si el servidor/cliente no se encuentra enviando otro archivo, procederemos al envío.

Obtenemos el primer archivo en cola para enviar obteniendo el primer archivo en la cola pero sin consumirlo y establecemos el estado a enviando. Procedemos en primera instancia, a comunicar al receptor que vamos a enviar un archivo enviando un comando con la información de tal, que contiene el nombre, peso y MD5; estos datos serán utilizados por el receptor para comprobar que el archivo ha sido enviado correctamente, además de poner el archivo a la cola de archivos creados/modificados, para que el Watcher Service no detecte el archivo que vamos a crear a continuación como un nuevo archivo, generando un bucle infinito de envíos de el mismo archivo de cliente a servidor y viceversa.

A continuación, se procede a abrir y conectar el Socket para el envío de datos; como he descrito anteriormente, si es el Servidor quien requiere enviar un archivo, abre su correspondiente Socket y notifica al cliente para que éste se conecte, una vez conectado notifica al servidor y éste comienza a transmitir los datos; en el caso contrario, si es el cliente quien requiere de enviar un archivo, envía una solicitud al servidor para que abra el Socket, y una vez éste ha ejecutado la orden, notifica al cliente para que éste se conecte.

Una vez el transmisor es notificado de que el Socket está abierto, conectado y listo para enviar, notifica al receptor de que va a comenzar el envío, para que éste comience a recibir los datos a través del Buffer del Socket.

El receptor crea el archivo añadiendo la extensión ‘.tmp’, la cual no será eliminada hasta comprobar que el archivo ha sido escrito correctamente. Una vez comprobado que el peso y MD5 del archivo escrito en el receptor, concuerdan con los datos enviados por el transmisor, la extensión del nombre ‘.tmp’ es eliminada, en caso de que esté configurado de tal manera, se realiza una copia del archivo en la carpeta temporal definida en la configuración y se inicia el proceso de cierre del Socket.

Una vez en Socket está cerrado correctamente, se elimina el archivo del pool en ambos lados, y el transmisor revisa la cola para comprobar si se requiere de enviar otro archivo o esperar a un nuevo evento por parte del Watcher Service.

Diagrama de flujo

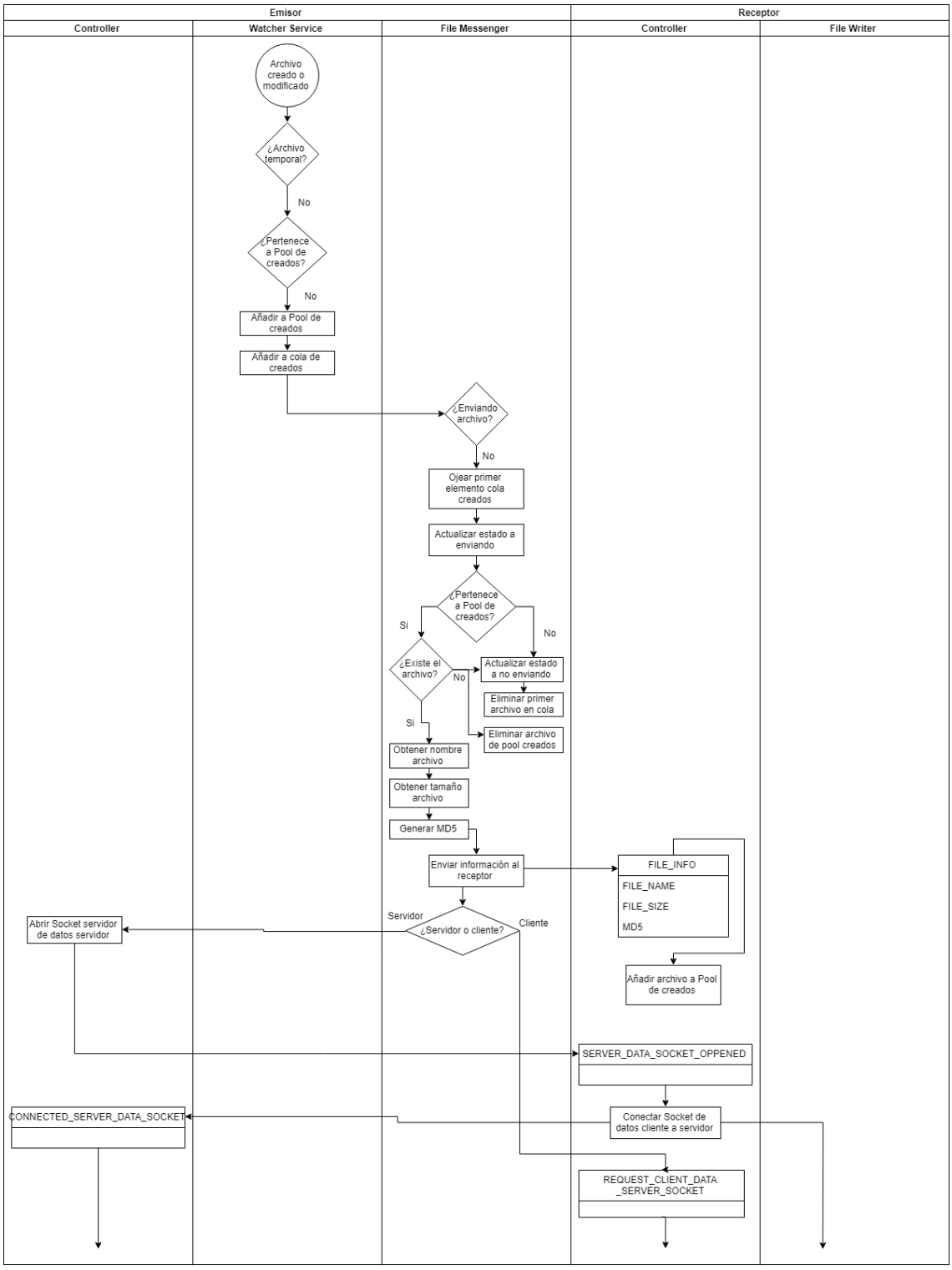

Ilustración 4. Diagrama de flujo: Envío de archivo parte 1

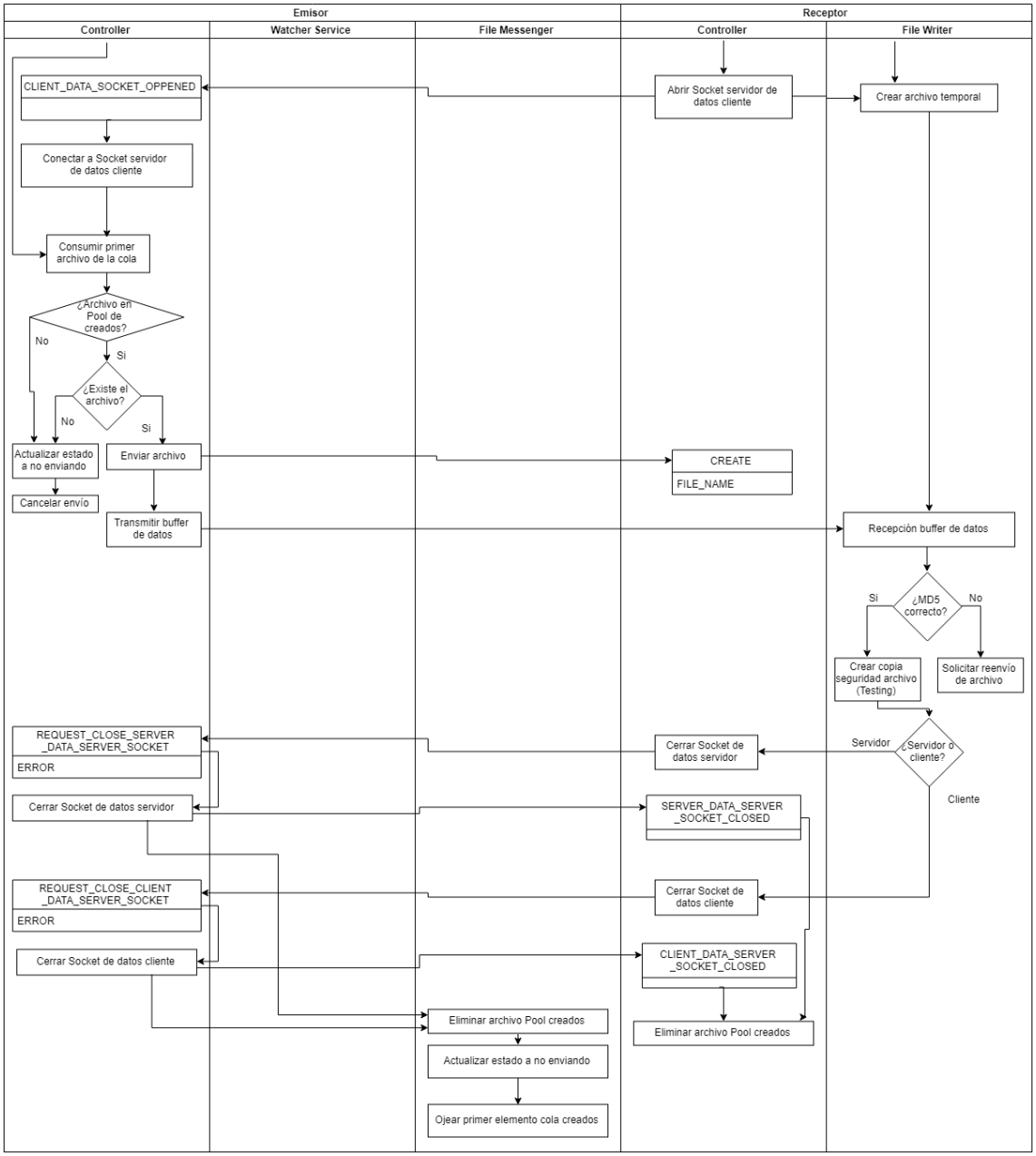

Ilustración 5. Diagrama de flujo: Envío de archivo parte 2

2.3.6.2 Fallo en el envío del archivo

Debido a ciertos errores en la transmisión del archivo por condiciones de la red del centro hospitalario, explicado más adelante en el punto *4.3 Producción*; cabe la posibilidad de que el fallo en la transmisión se vea tan solo reflejado en el transmisor, por lo que, si esto sucede, se inicia el proceso de cierre del Socket, notificando que se ha producido en error en el mismo comando. De ésta manera, el archivo no es eliminado del pool de archivos creados en ninguno de los lados, y el receptor es notificado de ello, eliminando el archivo temporal creado; y el transmisor añade a la cola de nuevo en primer lugar, el archivo en cuestión.

Cuando el proceso de cierre de los Sockets es finalizado, como en el proceso de envío correcto, se comprueba que la cola esté vacía, pero en este caso se hallará el archivo fallido en ella, por lo que comenzará de nuevo el proceso de envío.

Diagrama de flujo

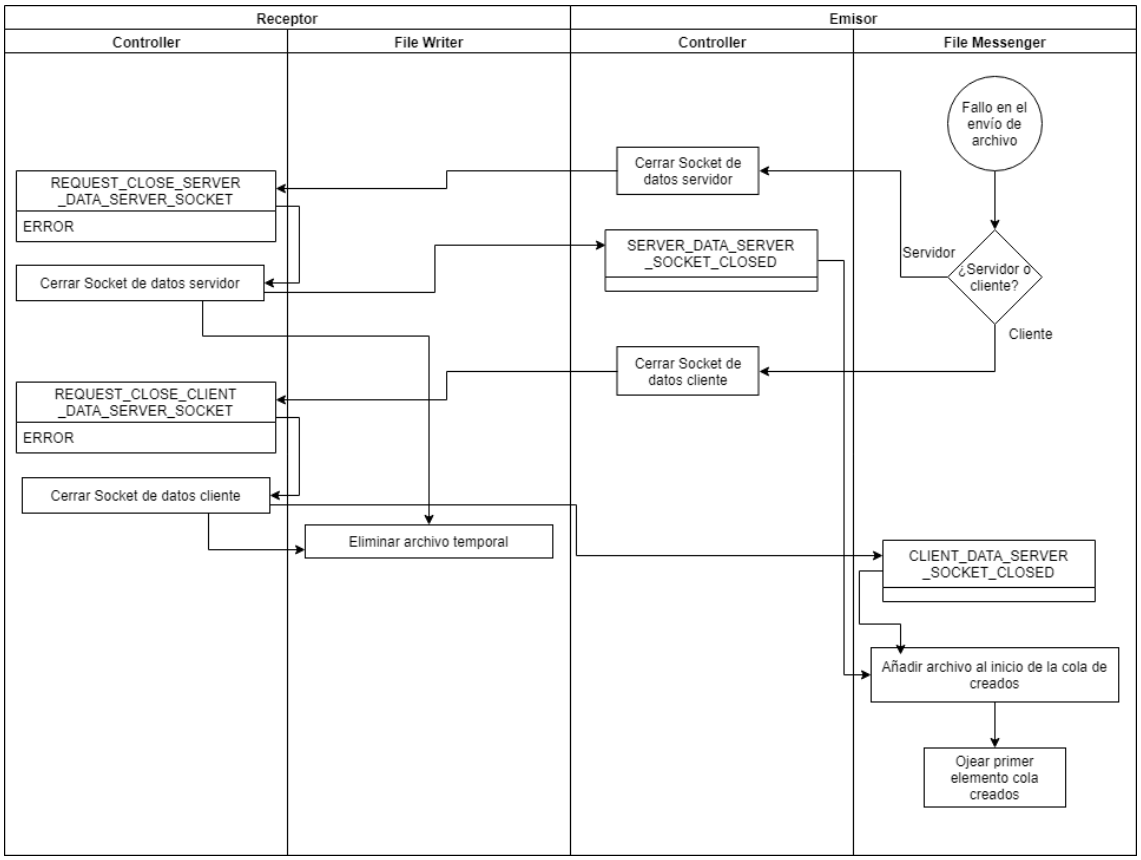

Ilustración 6. Diagrama de flujo: Fallo en el envío de archivo

2.3.6.3 Cancelar envío de archivo

Desde que un archivo es añadido al *pool* hasta que es enviado, se establece un periodo de tiempo, en el cual, el archivo puede ser eliminado; esto puede suceder en el mismo transcurso en el cual ha sido el receptor notificado de que un archivo va a ser enviado, por lo que habría creado el archivo temporal a la espera de la recepción de datos.

De haber sido eliminado el archivo, se notifica al receptor para que elimine el archivo temporal, así como consuma del pool el archivo en cuestión. A continuación, se procede a cerrar el Socket de datos y comprobar si hay más archivos para enviar.

Diagrama de flujo

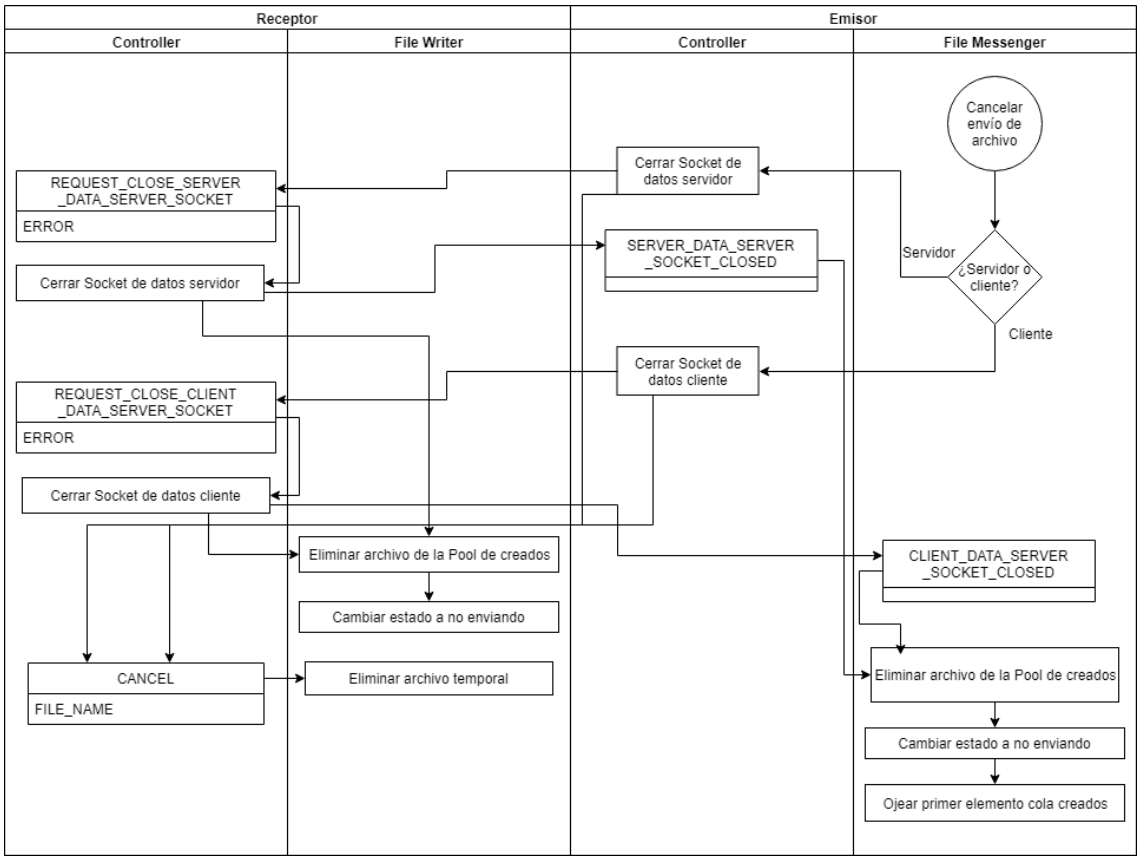

Ilustración 7. Diagrama de flujo: Cancelar envío de archivo

2.3.6.4 Fallo en la recepción del archivo

Si el archivo no ha sido enviado correctamente, ya sea por un fallo en el envío, diferente peso, el MD5 no concuerda, etc. el receptor elimina el archivo tanto del sistema como del pool, inicia el proceso de cierre del Socket de datos con error, por lo que el archivo no es eliminado del pool del transmisor y notifica al transmisor de que el envío no ha sido correcto, añadiendo éste el archivo de nuevo a su cola.

Diagrama de flujo

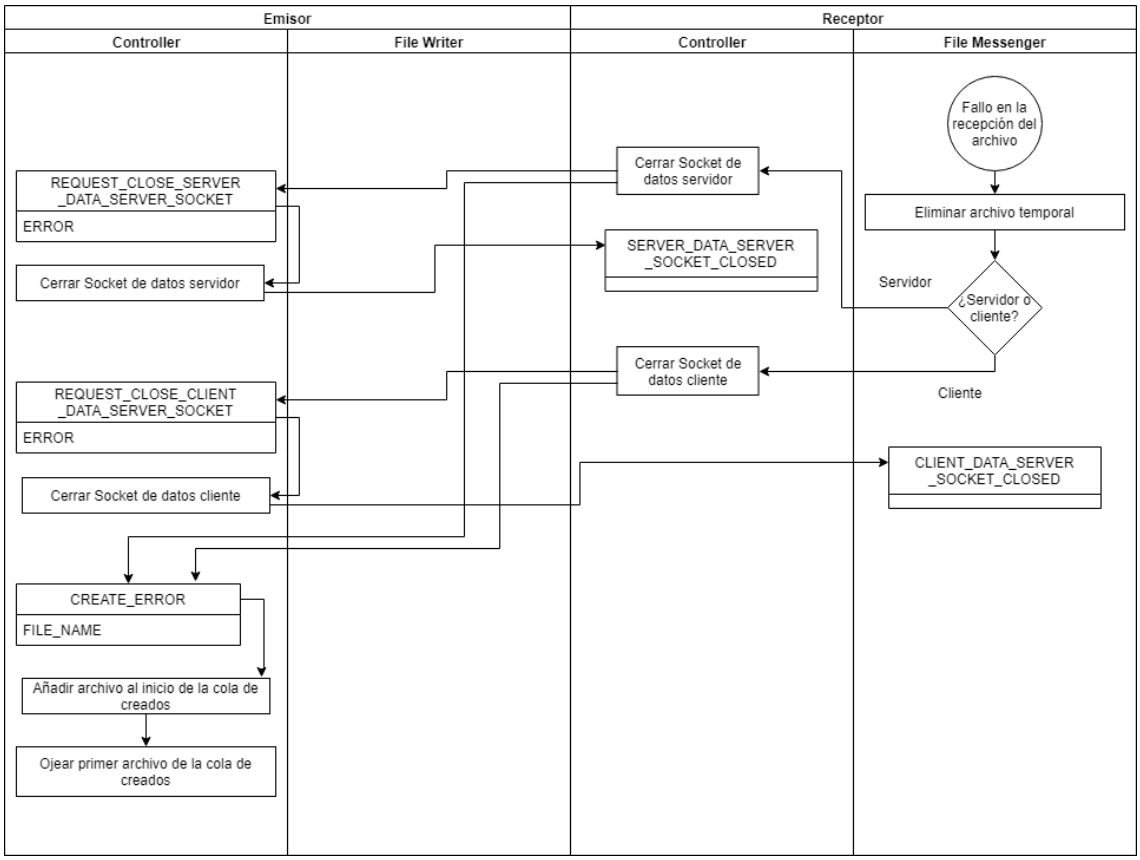

Ilustración 8. Diagrama de flujo: Fallo en la recepción de archivo

2.3.6.5 Archivo eliminado

Cuando el Watcher Service detecta un archivo eliminado, se comprueba que ese archivo no está almacenando en el pool de archivos eliminados, si no es así, lo almacena y registra en la cola de archivos eliminados.

Al añadirse al pool de archivos eliminados, el Listener notifica a la clase encargada de comunicar al receptor. Ésta clase comienza el proceso de notificación al receptor, consumiendo el archivo de la cola de eliminados y enviando el nombre del archivo bajo el comando de eliminar.

El receptor recibe el comando, añade el archivo a su pool de archivos eliminados y elimina el archivo, de ser eliminado correctamente, consume el archivo del pool de archivos eliminados y notifica al transmisor, entonces este elimina el archivo de su pool de archivos eliminados y prosigue a esperar nuevos eventos por parte del Watcher Service.

Diagrama de flujo

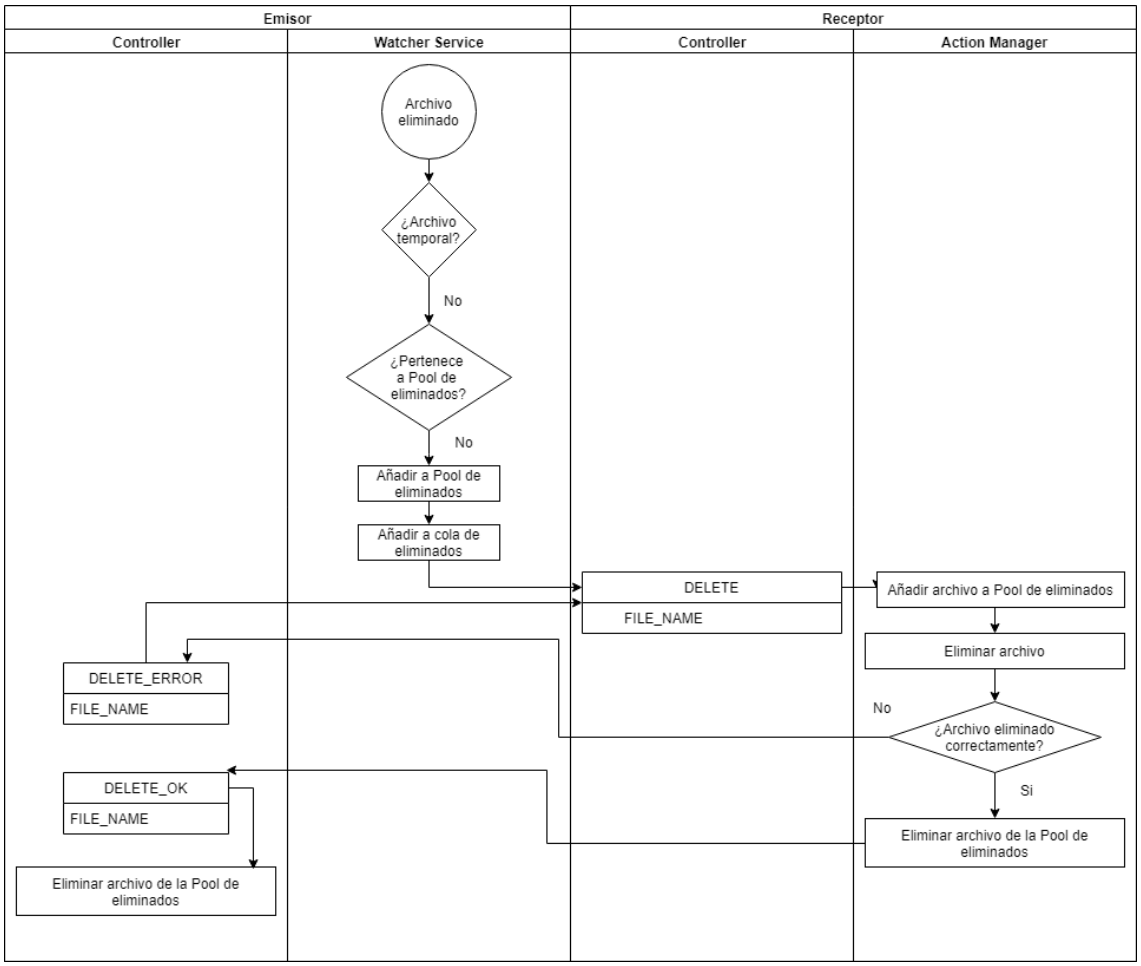

Ilustración 9. Diagrama de flujo: Archivo eliminado

2.3.6.6 Patrón de diseño 'Object pool'

Para mantener el control sobre los archivos modificados, creados o eliminados para que no haya duplicidad, cuando el Service Watcher detecta un evento, registra en el pool el archivo correspondiente si no existía previamente, de esta manera evitamos tramitar dos veces el mismo archivo al receptor, ya que, en determinados casos, el Service Watcher genera más de un evento por cada archivo creado, modificado o eliminado, dependiendo de la gestión de eventos que genera el sistema operativo.

Los archivos se mantienen registrados en el pool hasta ser enviados correctamente.

‘Listener’ sobre objetos añadidos

Cuando un archivo es añadido a los respectivos pools, es notificado mediante un Listener a la clase encargada del envío de archivos; entonces ésta comprueba si el cliente/servidor no se encuentra enviando otro archivo, y si es así, comienza el proceso de envío.

2.3.6.7 Estructura de datos tipo ‘cola’

A la par que los archivos son añadidos al pool para que no haya duplicidad, son añadidos a una estructura de cola para mantener un orden y gestión del envío, siendo consumidos de la cola una vez comienza el envío del archivo.

2.3.6.8 Interfaces

Tanto para el caso de que se requiera cerrar el Socket de datos desde el envío debido a un error, como en la recepción ya sea debido a una recepción correcta o errónea, se ha requerido de instanciar interfaces para comunicar entre las diferentes clases el acceso a los métodos estáticos privados correspondientes al cierre de tales Sockets.

2.3.7 Registro de actividad en la solución

La solución implementa un sistema de registro de toda la actividad realizada, mostrando por consola la información indicada cómo almacenándola en un archivo de texto propio de cada día hábil.

La actividad queda registrada a cuatro diferentes niveles:

{INFO, ERROR, WARNING, DEBUG}

Siendo estos indicados a la hora de llamar a los pertinentes métodos de la clase de registro.

Además, se registra la fecha y hora del evento, así como la zona horaria en la que se ha registrado, el gestor de la excepción, el hilo de ejecución, la clase desde la que se ha ejecutado, si es cliente o servidor, y el mensaje descrito.

Estos son tres ejemplos de eventos registrados:

INFO 2019-05-17 11:02:39.627 GMT+0200 ConfigurationLogger [main] - com.biit.synchronizer.Synchronizer: [Server] Reading setting 'javax.net.ssl.trustStorePassword='.

ERROR 2019-05-17 11:02:39.634 GMT+0200 SynchronizerLogger [main] - com.biit.synchronizer.Client: java.io.FileNotFoundException: (No existe el archivo o el directorio)

DEBUG 2019-05-17 14:41:33.895 GMT+0200 SynchronizerLogger [main] - com.biit.synchronizer.Client: [Client] Connecting to server '127.0.0.1' at port '1025'...

Siendo posible gracias a este registro, poder recuperar el proceso que se estaba llevando a cabo hasta el fallo de la aplicación, monitorizando desde el hilo que está lanzando cada acción, la clase que contiene el código en ejecución y la descripción de la acción. Facilitando así la solvencia de errores en nuestra solución.

2.3.7.1 Manejo de excepciones

Como podemos apreciar en los registros ejemplo anteriores, el tercer parámetro indica quien ha gestionado la excepción, ya que para mayor facilidad de comprender el flujo e información que deseamos obtener por parte del registro de actividad, se han implementado diferentes clases que extienden de la clase 'Exception' de Java.

Éstas clases, extienden la funcionalidad de la clase 'Exception', dando la facilidad de identificar y relacionar cada excepción en que proceso de la ejecución han sucedido.

3. Testeo

A continuación, redactaré los test tanto unitarios, de integración, como reales, realizados para la estabilidad y mantenimiento de la solución desarrollada.

Gracias a Maven la implementación de los test tanto unitarios como de integración ha sido más sencilla y automatizada. Aprovechando las diferentes etapas del ciclo de vida que nos ofrece, se ha configurado el archivo de configuración "pom.xml" para crear y/o ejecutar tanto los directorios como archivos temporales para poder testear la solución, así como limpiar todo rastro de éstos archivos tras las pruebas de testing pertinentes.

3.1 Unitarios

Para realizar los test unitarios, antes de ejecutarlos, en la etapa "generate-sources" de Maven, se crea una carpeta en el directorio raíz del proyecto llamada "TempTest" donde se crearán, modificarán o eliminarán los archivos pertinentes para comprobar el correcto funcionamiento de los métodos que trabajan con archivos.

Se han realizado los siguientes test para ello:

- Crear archivo:

En este test comprobaremos la funcionalidad de envío de un archivo cuando éste es creado en la carpeta a sincronizar. Creando un archivo de un tamaño determinado en la carpeta del emisor, y comprobando que ha sido enviado correctamente a la carpeta del receptor, corroborando su peso y MD5 checksum.

- Modificar archivo

En este test comprobaremos la funcionalidad de envío de un archivo cuando éste es modificado en la carpeta a sincronizar. Modificando una cantidad determinada de bits a un archivo ya creado en la carpeta del emisor, y comprobando que ha sido enviado correctamente a la carpeta del receptor, corroborando su peso y MD5 checksum.

- Eliminar archivo

En este test comprobaremos la funcionalidad de eliminar un archivo. Crearemos un archivo en la carpeta del emisor, esperaremos a que sea enviado al receptor, y lo eliminaremos, comprobando que efectivamente ha sido eliminado en la carpeta del receptor.

- Crear varios archivos y eliminar todos:

En este test comprobaremos la funcionalidad de eliminar varios archivos de la clase ActionManager, creando un par de archivos cualesquiera vacíos y llamando al método “ActionManager.deleteAll()” para eliminar todo contenido de la carpeta en cuestión, a posterior comprobar que la carpeta se haya vacía.

- Comparar MD5 SUM con MD5 generado:

Este test comprobará el correcto funcionamiento del método comparador del MD5 de los archivos en cuestión, comparando en este caso el MD5 correspondiente al archivo generado, siempre el mismo; con el MD5 que nuestro método está obteniendo.

- Comparar MD5 generado con MD5 SUM:

Este test comprobara el correcto funcionamiento del método comparador del MD5 de los archivos en cuestión, comparando en este caso el MD5 obtenido del archivo que hemos creado para el test con el MD5 correspondiente al archivo generado, el cual siempre es el mismo.

3.2 Integración

Para realizar los test de integración, en la fase “pre-integration-test” de Maven, generamos 4 carpetas, 2 carpetas que contendrán los ejecutables para cliente/servidor y sus correspondientes archivos de configuración, y 2 carpetas para la sincronización de archivos a las que atenderán respectivamente cliente y servidor.

Seguidamente, se iniciarán los ejecutables cliente y servidor con los respectivos comandos para iniciar como tales respectivamente, y una vez conectados, comenzaremos el testing unitario trabajando sobre las carpetas creadas para la compartición de archivos.

Los test realizados son:

- Crear un archivo en el directorio

En este test crearemos un archivo en la carpeta del cliente y comprobaremos que se ha enviado y creado correctamente a la carpeta del servidor.

- Crear un archivo en cada directorio

En este test crearemos un archivo en la carpeta del cliente y otro en la carpeta de servidor, posteriormente comprobaremos que ambas carpetas están sincronizadas.

- Eliminar archivo en el directorio

En este test crearemos un archivo en la carpeta del cliente, corroboraremos que ha sido enviado y creado a la carpeta servidor, y entonces lo eliminaremos de la carpeta del cliente, comprobando que ha sido también eliminado de la carpeta del servidor.

- Modificar archivo en el directorio

En este test crearemos un archivo en la carpeta del cliente, corroboraremos que ha sido enviado y creado a la carpeta servidor, y entonces modificaremos el archivo añadiendo texto; posteriormente comprobaremos que tanto el archivo de la carpeta del cliente como el archivo en la carpeta del servidor pesan los mismo y tienen el mismo MD5.

- Crear varios archivos

Se crearán 10 archivos de poco peso en la carpeta cliente, y comprobaremos al terminar que éstos archivos han sido enviados y creados en la carpeta del servidor correctamente.

3.3 Carga

- Crear 200 archivos de poco peso (Min 1KB, Max 5MB)

En éste test crearemos 200 archivos de poco peso en la carpeta del cliente, y comprobaremos al terminar que estos archivos han sido enviados y creados en la carpeta del servidor correctamente.

- Crear 200 archivos de gran peso (Min 20MB, Max 80MB)

En este test crearemos 200 archivos de gran peso en la carpeta del cliente, y comprobaremos al terminar que éstos archivos han sido enviados y creados en la carpeta del servidor correctamente.

- Crear 200 archivos mezclando poco y gran peso

En éste test crearemos 200 archivos mezclando un 10% de ellos de gran peso y un 90% de ellos de poco peso en la carpeta del cliente, y comprobaremos al terminar que éstos archivos han sido enviados y creados en la carpeta del servidor correctamente.

- Crear 200 archivos mezclando poco y gran peso, modificando y eliminando aleatoriamente archivos

En éste test crearemos 200 archivos mezclando un 10% de ellos de gran peso y un 90% de ellos de poco peso aleatoriamente en la carpeta del cliente y en la carpeta del servidor, modificando en el proceso aleatoriamente algunos archivos y eliminando aleatoriamente también archivos; finalmente comprobaremos que ambas carpetas contienen los mismos archivos y han sido sincronizadas correctamente.

Para comprobar que ambos directorios están sincronizados correctamente, cada test espera un tiempo decidido en base a su coste; expirado ese tiempo, se comprueba que el último archivo del test está creado (cada test crea un determinado número de archivos nombrados con un identificador numérico, por lo que sabemos en cada test el nombre del último archivo que se creará), entonces se comprueba que ambos directorios contienen los mismos archivos comprobando sus nombres, peso y MD5.

Todos los test se aseguran al finalizar de borrar todo archivo en las carpetas de sincronización para asegurar de que, en caso de fallo, el siguiente test pueda ser realizado correctamente.

Estos tests nos aseguran una cobertura de sincronización entre ambas carpetas hasta el manejo de doscientos archivos, de un tamaño máximo de ochenta megabytes.

Además de asegurar que, si un archivo es eliminado o modificado durante el envío de otro archivo, será comunicado por igual al receptor y será eliminado o modificado en su defecto.

3.3.1 Jenkins

Jenkins es uno de los servidores de integración continua más empleados actualmente, además de ser gratuito y open-source. Esta herramienta se encarga de realizar las tareas designadas para el proceso de compilado y despliegue de las soluciones de los repositorios Git indicados.

Encargado por tanto en éste proyecto, de llevar a cabo todos los test y evitando la pérdida de tiempo esperando a que estos finalicen, ya que el proyecto integra test de larga duración, como el test de enviar 200 archivos de peso.

3.4 Producción

La solución ha sido desarrollada y testeada bajo Linux en una red interna, pero en el caso real, se ejecutaría bajo Windows y una red privada hospitalaria. Por lo que una vez desarrollada una versión estable de la solución, corroborando que la sincronización de archivos se realiza correctamente; se han realizado pruebas en local con un cliente bajo Windows con el software médico Cosmed OMNIA instalado.

Para realizar las pruebas en local del proceso completo de importación de datos desde OMNIA necesitamos obtener una licencia y el susodicho software para poder hacer uso de él. Esto no fue difícil ya que previamente se había trabajado con Cosmed y la relación comercial y profesional seguía presente. Esta primera fase de testing en producción ayudó tanto a comprobar que todo el proceso de comunicación entre los diferentes softwares se realizaba correctamente, como a conocer mejor el software Cosmed OMNIA, la composición de los XML y el protocolo para obtener los archivos contenedores de los datos requeridos, así como la extracción de los datos en cuestión.

Una vez obtenido un resultado estable de esta primera fase, pasamos a las pruebas en un entorno real, en base a la disponibilidad de una estación de trabajo libre en el centro hospitalario. Mediante la herramienta TeamViewer, accedimos a dicha máquina para instalar el software de sincronización y realizar el proceso completo de obtención de datos del software médico Cosmed OMNIA instalado en Windows bajo una red hospitalaria. Dado que esta máquina era propiedad particular de Cosmed, no hubo problema alguno con la seguridad del hospital para instalar dicho software.

En esta segunda fase alcanzamos, tras solucionar los problemas encontrados, una versión estable final de la solución software para la sincronización de carpetas; cumpliendo así los objetivos propuestos y una nueva ventana de negocio a comercializar.

3.4.1 Problemas encontrados

En la primera fase de testing en producción, nos encontramos con el primer problema a solucionar bajo una ejecución del cliente en Windows. La consola de Windows en ciertas ocasiones, sin una regularidad exacta, dejaba de mostrar por pantalla la información que debía, siendo esto un problema para depurar en caso de fallo de sincronización o gestión del protocolo; esto se debía a que el modo de "Edición rápida" estaba activado en la consola, generando este problema debido a la gestión interna de la propia consola, bastando con desactivar esta opción para evitar el problema.

En la segunda fase, nos encontramos con un problema mayor, debido a que la conexión de los sockets para el envío de archivos, a veces se veía interrumpida, generando un fallo en la sincronización y en el protocolo. El mayor problema para depurar y solucionar este error. Fue que sucedía aleatoriamente, produciéndose de media dos de cada veinte ejecuciones; generando el fallo posiblemente incluso antes de empezar a enviar el buffer de datos, por lo que el receptor no llegaba a conocer de ello. Esto obligó a modificar el protocolo de sincronización de carpetas, para que fuese el transmisor quien se encargase de gestionar el error, no el receptor como debería de ser; reiniciando el proceso de envío y notificando al receptor para reiniciar el proceso de recepción.

Una vez solucionado estos dos problemas, la solución está lista para pasar al uso con clientes reales.

4. Conclusiones finales

4.1 Objetivos cumplidos

Tras la integración de esta solución software, se ha podido implementar la comunicación de la aplicación web SportMediScore de BiiT Sourcing Solutions con el software médico Cosmed OMNIA. Permitiendo así a los usuarios de SportMediScore la comunicación directa desde la propia aplicación web en lugar de tener que introducir los datos a mano, tanto para importar como para exportar.

Actualmente gracias al desarrollo de esta solución, se han integrado en SportMediScore dos funcionalidades, 'Inicializar visita' e 'Importar examinación'. La primera funcionalidad, permite registrar en el software médico OMNIA los datos del paciente registrado en SportMediScore y crear una visita registrada en el día actual; la segunda funcionalidad, nos permite importar los datos obtenidos tras realizar y registrar la examinación del paciente en el software médico OMNIA al formulario presente en la aplicación web SportMediScore, pudiendo éstos ser modificados de ser requerido. Es decir, tenemos el control de la aplicación de Cosmed desde

SportMediScore, evitando la incomodidad de forzar el intercambio de una aplicación a otra.

4.2 Caso de éxito

El software ha sido desplegado con éxito dentro de un centro médico sin interferir en su seguridad de red. Siendo transparente a esta. La solución puede ser exportada fácilmente a otros entornos.

Hasta ahora 4 médicos con 7 pacientes han utilizado este software. Se ha reducido el tiempo de realización de una cita médica en un 19% gracias a evitar la necesidad de usar dos aplicaciones distintas, y al evitar la copia de datos de una a otra. Esto se ha traducido en la satisfacción de nuestros clientes. Gracias a esto, la empresa BiiT ha cerrado un acuerdo con Orbis Sport que lo convierte en cliente suyo durante, al menos, 2019 y 2020. El software actual está implantado en centro médico deportivo Orbis Sport, un centro médico deportivo independiente localizado en el centro médico Zyuderland en la ciudad de Sittard-Geleen, calle C/ Dr. H. van der Hooffplein 6162BG.

4.3 Conclusiones personales

Tras el desarrollo de esta solución he tanto aprendido nuevos conceptos, tecnologías, habilidades y conocimientos como afianzado los que ya conocía.

Empezando por el desarrollo de software en equipo bajo metodologías ágiles en equipo, trabajando sobre mis habilidades sociales, así como mi capacidad de expresión y comprensión.

Continuando por el aprendizaje de uso de nuevos softwares para mejorar y facilitar el proceso de desarrollo tales como Git, Jenkis y Maven.

Y finalizando por la mejoría como ingeniero del software desde el planteamiento del desarrollo de tal, hasta la capacidad de su desarrollo. Aprendiendo a realizar tests tanto unitarios, de integración, como de carga.

Afianzando en todo el proceso, el conocimiento adquirido durante el grado en Ingeniería Informática, tal como el desarrollo bajo marco ágil Scrum, la ingeniería del software en el desarrollo de diagramas de flujo para mayor facilidad de la comprensión de flujo y ejecución del software, y aplicar estructuras de datos tales como colas y pool's para hacer posible el desarrollo de la solución.

Tras el conocimiento adquirido, habría cambiado en el proceso de desarrollo el intento de transmitir los archivos por texto plano mediante cadenas de bytes en un JSON. Puesto que no es una solución viable desde el punto de vista de seguridad, y ofrece una alta posibilidad de fallo en cada transmisión. Debido a la gran cantidad de paquetes con información que requeriríamos de enviar y la alta posibilidad de pérdida

de tales al ser comunicados al receptor; por lo que, si en cada envío requerimos de solicitar de nuevo partes perdidas, el tiempo de envío sería excesivo.

4.4 Trabajo futuro

Tras el desarrollo de ésta solución software, existen posibilidades de mejora queda abierta. Tal como la implementación de la capacidad de ser multiusuario, ofreciendo así la posibilidad de atender a diferentes clientes desde el mismo servidor; ya que actualmente se requiere de un servidor exclusivo por cliente.

También la implementación de nuevas interacciones desde SportMediScore con Cosmed OMNIA; ya que actualmente tan solo están implementadas las funcionalidades de registrar un cliente y obtener sus datos. Tales como modificar los datos del cliente, registrar médicos o iniciar el proceso del test desde SportMediScore, pretendiendo así llegar hasta un punto en el cual no requiramos de la interacción de dos softwares, controlando todo el proceso unificadamente desde SportMediScore.

5. Referencias

[1] Díaz Domínguez, Luis F. y Navarro Huerga, Miguel A. Universidad Alcalá de Henares (UAH), 2014. ISBN: 9788415834366.

[2] Ryan Hodson. Ry's Git Tutorial. RyPress, 2014. ISBN: B00QFIA5OC.

[3] Robert C. Martin. Código Limpio: Manual de estilo para el desarrollo ágil de software. ANAYA MULTIMEDIA, 2012. ISBN-10: 8441532109.

[4] Brian R Jackson. Maven: The Definitive Guide. O'Reilly Media, 2018. ISBN-10: 0596517335.

[5] Alonso Álvarez García, Rafael de las Heras del Dedo, Carmen Lasa Gómez. Métodos Ágiles y Scrum. ANAYA MULTIMEDIA, 2012. ISBN-10: 8441531048.

[6] A. M. Vozmediano. Java para novatos: Cómo aprender programación orientada a objetos con Java sin desesperarse en el intento: Volume 3. CreateSpace Independent Publishing Platform, 2017. ISBN: 1548217859.

[7] Raghuram Bharathan. Apache Maven Cookbook. Packt Publishing, 2015. ISBN: 1785286129.

[8] Richard E. Silverman. Git Pocket Guide. O'Reilly Media, 2013. ISBN: 9781449325862.

[9] Jean-Philippe Gouigoux. Docker. Primeros pasos y puesta en práctica de una arquitectura basada en micro-servicios. ENI, 2018. ISBN: 2409015891.

[10] Sai Srinivas Sriparasa. JavaScript and JSON Essentials. Packt Publishing, 2013. ISBN: 1783286032.

[11] Java Keytool: Consultado el 05/11/2018 en https://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html.

[12] Taiga. Consultado el 08/10/2018 en https://taiga.io/.

[13] Cosmed OMNIA. Consultado el 15/10/2018 en https://www.cosmed.com/en/products/software/omnia-standalone.

[14] Jenkins. Consultado el 09/10/2018 en https://jenkins.io/.

[15] BiiT Sourcing Solutions. Consultado el 08/10/2018 en http://biit-solutions.com/.