Sistema de notificación push para instrumentos de diagnóstico

Óscar Jara Pérez

Resumen— Este proyecto ha sido propuesto por Werfen Systelab, una empresa internacional relacionada con los instrumentos de diagnóstico médico. Werfen Systelab dispone de distintos instrumentos de laboratorio. Este trabajo de final de grado quiere integrarse con el instrumento de diagnóstico que lleva por nombre ACLTop, un instrumento de hemostasia. El objetivo principal de este proyecto consiste en comunicar el instrumento ACLTop con otro proyecto de Werfen llamado HemoHub. Esta comunicación es requerida para poder mostrar la información de la que ACLTop dispone en HemoHub de manera centralizada. Gracias a este proyecto, un técnico de laboratorio podría comprobar el correcto funcionamiento de los instrumentos y los resultados que estos ofrecen desde HemoHub sin la necesidad de ir instrumento a instrumento para este cometido. En este proyecto se ha diseñado y desarrollado un gestor de servidores HTTP en los que se puede registrar distintos mensajes. La implementación de este proyecto da soporte al servidor que se encarga de implementar los mensajes para comunicarse con HemoHub, pero también se han desarrollado herramientas que servirán a Werfen Systelab en futuros proyectos.

Palabras clave—Instrumento de diagnosis, ACLTop, HemoHub, comunicaciones, servidores, HTTP, diseño modular.

Abstract—This project has been proposed by Werfen Systelab, an international Company related to medical diagnostics instruments. Werfen Systelab has some different laboratory instruments. This Project aims to integrate with one diagnostic instrument, which is called ACLTop, an hemostasia instrument. The main objective of this project is to communicate ACLTop with another Werfen project called HemoHub. This communication is required to display the ACLTop information on HemoHub. Thanks to this Project, a laboratory technician could check the status of the instruments and the results it offers without the need to go instrument to instrument for this purpose. On this project it has been designed and developed a HTTP server manager that can register different messages. The implementation of this project, supports the server that is responsable for implementing the messages to communicate ACLTop with HemoHub, but also it have been developed tools that will help Werfen Systelab in future projects.

Index Terms— Diagnostic instrument, ACLTop, HemoHub, communications, servers, HTTP, modular design.

1 Introducción

En este documento se describe el proceso de desarrollo que se ha llevado a cabo para el proyecto de final de grado: Sistema de notificación push para instrumentos de diagnóstico.

El tema de este proyecto de final de grado ha sido propuesto por Werfen, una gran empresa internacional relacionada con los instrumentos de diagnóstico para uso médico. Esta empresa está compuesta por cuatro departamentos principales: Instrumentation Laboratory, Biokit, Inova Diagnostics y Systelab. Se encargan de la construcción del hardware, el desarrollo de software, la creación de los materiales químicos (reactivos) necesarios para el uso de sus instrumentos de diagnóstico de laboratorio, del transporte de materiales, etc. Cuentan, además, con un gran abanico de instrumentos de diagnóstico, cada uno enfocado a realizar un tipo de pruebas médicas en concreto. Cada instrumento de diagnóstico dispone de una pantalla propia con una interfaz de usuario intuitiva que permite a los técnicos de laboratorio utilizar estos instrumentos. Estas pantallas también se utilizan para mostrar los resultados obtenidos por el instrumento de diagnóstico.

Systelab, el departamento de Werfen que se encarga del desarrollo de software, actualmente entre los muchos proyectos que tiene entre manos, está realizando un proyecto para centralizar la información de los instrumentos de diagnóstico de un laboratorio en una única pantalla, con el objetivo, de que un técnico de laboratorio pueda

- *E-mail de contacto: oscar.jara@e-campus.uab.cat*
- *Mención realizada: Ingeniería de Computadores.*
- *Trabajo tutorizado por: Miquel Àngel Senar (Departamento de Computadores)*
- *Curso 2015/16*

supervisar todos los instrumentos de manera centralizada. Ese proyecto lleva por nombre HemoHub.

Werfen Systelab ha propuesto para este proyecto de final de grado, planificar y desarrollar el software que permitirá comunicar mediante mensajes a Hemohub con uno de los instrumentos de diagnóstico más avanzados de Werfen, ACLTop, con el objetivo de mostrar la información del instrumento de diagnóstico ACLTop en HemoHub.

Se busca que este proyecto sea transparente, flexible y estándar, puesto que formará parte del software que el instrumento ACLTop tendrá en los laboratorios. Esto implica que deberá seguir unas pautas de diseño determinadas compatibles con la lógica de diseño del instrumento para facilitar su mantenimiento y actualización.

El diseño del software de este proyecto, se ha visto modificado durante el transcurso de su desarrollo para cumplir con los objetivos cambiantes y aportar nuevas y mejores funcionalidades. El anterior diseño respondía a un problema específico más concreto, mientras que la nueva aproximación que se ha propuesto tiene un enfoque más general. Gracias a este nuevo diseño, Werfen Systelab podrá añadir en un futuro, nuevos módulos que formen parte del sistema diseñado, para obtener así nuevas funcionalidades.

En las siguientes secciones de este documento, se explicará la función, el software y la estructura del instrumento ACLTop para entender qué puede aportar este proyecto a un instrumento de laboratorio. A continuación, se explicará el diseño global y los módulos del software que forman este proyecto. Veremos también una sección dedicada a la validación y las pruebas que se han realizado para comprobar el correcto funcionamiento del sistema. En la última sección veremos las conclusiones que se han extraído de la realización de este proyecto y posibles planes de futuro.

2 INSTRUMENTO DE LABORATORIO: ACLTOP

Como ya se ha visto anteriormente, el propósito de este proyecto de final de grado, es integrar el software generado en el instrumento de laboratorio ACLTop. Para entender algunos conceptos de los que se hablarán más adelante es importante conocer el ámbito donde se desarrolla esta aplicación.

Usualmente este instrumento se suele encontrar en grandes laboratorios como los de los que se disponen en los hospitales. El objetivo de este instrumento es realizar pruebas de hemostasia sobre muestras de sangre de pacientes para detectar cualquier tipo de anomalía. Se suele encontrar en grandes laboratorios puesto que es una máquina destinada a ofrecer gran cantidad de resultados por hora. Cara laboratorio aplica sus normas para el uso, pero el instrumento se suele utilizar en turnos de ocho horas, dos veces al día. Hay ocasiones en las que, por cantidad de análisis a realizar, se puede llegar a utilizar durante todo el día.

Para que este funcione, a parte de las muestras a analizar, es necesario que disponga de diversos tipos de materiales químicos (llamados reactivos) para la realización de distintos análisis. Estos reactivos, se mezclan con la muestra de sangre en cantidades exactas y esta, se analiza. También es necesario tener cubetas, sirven internamente para depositar las muestras de los pacientes para hacer distintas pruebas si fuera necesario.

Para que un técnico de laboratorio haga uso del instrumento, es necesario que se sitúe delante e interactúe con él, de manera que toda la información relacionada con el instrumento se mostrará por su propia pantalla, ya sean resultados o errores.

2.1 Hardware del instrumento

Tal y como se muestra en la figura 1, el instrumento ACLTop dispone de dos partes principales:

- El cuerpo del instrumento: es la parte que contiene todo lo relacionado con el hardware para el análisis médico: secciones para poner las muestras de los pacientes, brazos robóticos para aspirar el contenido de las muestras, motores para mover la muestra por dentro del instrumento, zonas seguras para mantener los elementos químicos, etc. Todo el hardware del que dispone el instrumento es a medida, por lo que, en su totalidad, hace uso de hardware controlado a bajo nivel.
  El proyecto no tendrá iteración directa con esta parte del instrumento.
- La pantalla: junto con un ordenador con hardware especifico. Es la que se encarga de la lógica de funcionamiento del instrumento. Este ordenador ejecuta la aplicación principal y utiliza el instrumento como herramienta a través de diversas conexiones para llevar a cabo tareas que tiene programadas o tareas que los técnicos de laboratorio mandan ejecutar.
  Es en esta sección dónde el proyecto se va a integrar.

Figura 1: Instrumento ACLTop.

Aunque el proyecto se integre en una única sección esto no implica que no se pueda acceder a datos del instrumento, puesto que, la sección del ordenador que lo con-

trola tiene acceso a los datos del mismo. Útil, como se verá más adelante, para el intercambio de mensajes entre ACLTop y HemoHub.

2.2 Software del instrumento

Es importante conocer también el software que utiliza el instrumento para poder amoldar el desarrollo del proyecto a las necesidades.
Puesto que el proyecto sólo tendrá interacción directa con la parte de la pantalla, es necesario saber que la aplicación principal que controla la lógica del sistema está desarrollada en lenguaje C++.

3 Objetivos del proyecto

Como ya se ha explicado, este proyecto busca ser transparente, flexible y estándar. Por transparente, entendemos que el proyecto tiene que estar implementado siguiendo la lógica del software que dispone ACLTop para facilitar su integración. También se entiende que éste tendrá que estar documentado y en todo momento hará un uso correcto del lenguaje de programación. Por flexible, entendemos que el proyecto debe ser capaz de adaptarse a diferentes situaciones y que en todo momento sea capaz de funcionar bajo cualquier circunstancia establecida anteriormente. Y por estándar entendemos que el proyecto, se ha de realizar siguiendo los estándares, para de esta manera, asegurar un correcto soporte de las tecnologías utilizadas de cara a la introducción de futuros cambios o mejoras y permitiendo, además, que sea compatible con cualquier sistema sin importar el lenguaje de éste. Dejando así, las tecnologías privativas, fuera del alcance de este proyecto.

Como ya se ha comentado anteriormente, el software necesita ser desarrollado en lenguaje C++ para la correcta integración con el instrumento. Es importante apuntar que mientras que ACLTop está desarrollado en C++, HemoHub está implementado en Java, y eso tiene ciertas implicaciones a la hora de enviar mensajes, como veremos en las siguientes secciones.

En un laboratorio suelen haber muchos instrumentos, con la implantación de este proyecto en ACLTop se busca que un técnico de laboratorio sea capaz de acceder a la información de todos estos desde un solo punto.

Un esquemático simple del diseño conceptual del proyecto podría ser el que se muestra en la figura 2.

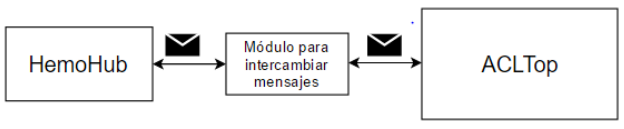

Figura 2: Diseño conceptual del proyecto.

3.1 Los mensajes

Conociendo todo lo anterior, Werfen Systelab propuso un conjunto de mensajes objetivo que utilizarían ACLTop y HemoHub para intercambiar información. A continuación, se muestran los mensajes principales.

- Intercambiar la información de los resultados obtenidos en la ejecución del análisis sobre la muestra de un paciente.
- Intercambiar la información del estado del instrumento. El estado indica en la fase que se encuentra el instrumento: parado, inicializando, realizando análisis, etc.
- Intercambiar la información de los sensores del instrumento. El instrumento ACLTop dispone de varios sensores interiormente que permiten el correcto funcionamiento. Algunos de ellos, por ejemplo, indican la temperatura operativa del instrumento, muestran cual es el nivel de los materiales químicos (reactivos) restantes, indican si quedan suficientes cubetas, etc.
- Intercambiar las posibles alarmas que muestre el instrumento. Las alarmas indican las incompatibilidades que han surgido, por ejemplo, que el técnico no ha situado correctamente los reactivos en la ubicación correcta.

Cabe mencionar que aparte de los mensajes mostrados anteriormente, Werfen Systelab propuso otros mensajes secundarios en un principio, pero estos, debido a la evolución de otros proyectos en la empresa, han sido destinados a otro software en donde su uso era más oportuno.

4 Diseño del sistema

El diseño de este proyecto debe ser capaz de cumplir con los requerimientos que se propusieron o que venían impuestos. Para ello, se decidió dotar al proyecto de cierta modularidad con la finalidad de facilitar entre otras cosas, la integración del sistema y la flexibilidad.

El concepto del diseño del proyecto capaz de cumplir los objetivos es el siguiente: un gestor de servidores que es capaz de crear tantos servidores como sea necesario, en los que se puede registrar qué función (usualmente, contactar con otro módulo) realizar en función de la petición recibida ya sea antes o después de la creación del servidor.

El diseño propuesto se divide en dos grandes módulos o bloques: el módulo que se encarga de la comunicación entre ACLTop y HemoHub (servidor y cliente) y el módulo que se encarga de serializar los mensajes para que ambos extremos se entiendan.

4.1 Tecnologías utilizadas

Una gran parte del tiempo de la planificación de este proyecto se destinó entre otras cosas a la investigación, selección y prueba de las tecnologías que mejor se adaptaban a este proyecto. Las tecnologías estándar que eran más comunes en proyectos de este tipo y que mejor se

adaptaban a las circunstancias dadas, fueron las siguientes:

- Para llevar a cabo la transmisión del mensaje se decidió que se utilizaría HTTP como protocolo de transporte. Puesto que es un estándar muy extendido y utilizado y se adaptaba a la necesidad de comunicar una aplicación implementada en C++ con otra en Java. También se decidió utilizarlo por la facilidad y lógica que los verbos HTTP (GET, POST, PUT, DELETE…) ofrecen.
- Para serializar el cuerpo del mensaje que se quería enviar, se decidió utilizar JSON. Fue escogido debido a la estructura interna del software de ACLTop, estructuras complejas orientadas a objeto de C++, Java es incapaz de leer estructuras binarias de ese tipo. Utilizamos JSON entonces, para serializar los objetos y los datos que queremos enviar o recibir. De esta manera, ya sea Java o cualquier otro lenguaje, se puede interpretar el contenido del mensaje sin más problemas.

Como resultado final se escogieron distintas librerías implementadas en C++ que utilizaran las tecnologías seleccionadas más arriba. El proceso de selección no ha sido una tarea sencilla, los aspectos utilizados para escogerlos han sido, entre otros, si disponían de licencia para poder usar y distribuir el software de manera gratuita, si se mantienen actualizadas de cara a poder solucionar posibles problemas que surjan, o de si tenían dependencias de otras librerías o no. Este último punto es importante, pues si queremos hacer un diseño flexible y modular necesitamos que no tenga dependencias.

4.2 Estructura del diseño

En la figura 3, se muestra un esquemático que representa la estructura de los módulos del software del proyecto.

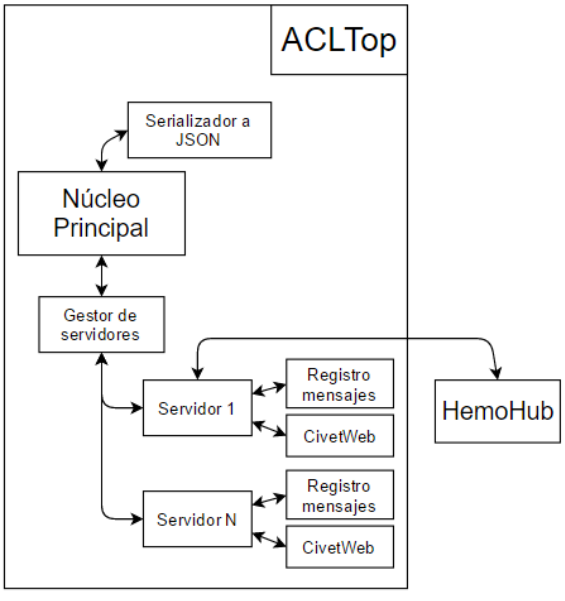

Figura 3: Diseño del proyecto.

De nuevo, en la figura 3 podemos apreciar las dos entidades principales, por un lado, tenemos a HemoHub, un proyecto externo, por otro lado, tenemos el instrumento ACLTop. Como podemos apreciar en la figura, dentro del instrumento, podemos ver sus módulos. El módulo 'núcleo principal' es el que hace referencia a todo el contenido de la aplicación del que ya disponía ACLTop, desde los módulos más importantes hasta la interfaz gráfica. Los otros, son los que se han diseñado en este proyecto para cumplir con los objetivos.

El proceso de recepción ideal de un mensaje de HemoHub hasta ACLTop en el diseño de la figura 3 es el siguiente:

1. HemoHub, que conoce la dirección IP y puerto del servidor, manda una petición HTTP esperando como respuesta un recurso.
2. El servidor que ha recibido la petición, comprueba que ésta esté registrada en el 'registro de mensajes'.
3. El servidor pregunta al 'núcleo principal' por el recurso.
4. Se dispone el contenido del recurso en formato JSON para que HemoHub pueda entenderlo.
5. Se devuelve a HemoHub el mensaje que contiene el código HTTP 200 OK, y el contenido del mensaje en formato JSON.

Más adelante se explicará con más profundidad para que sirve cada módulo y su lógica interna.

4.3 Comunicación

La comunicación es la parte del proyecto más importante y extensa. La librería elegida para las comunicaciones ha sido CivetWeb.

CivetWeb es un servidor HTTP implementado en C++ que actualmente cuenta con el soporte de una comunidad activa que se encarga de dar soporte y perfeccionar ese proyecto. CivetWeb es un proyecto paralelo de Mongoose (un servidor HTTP conocido) que creó la comunidad cuando Mongoose se volvió de pago.

Se ha escogido CivetWeb, porque no necesita dependencias para su uso, porque tiene licencia gratuita y que permite la distribución, y porque es compatible con OpenSSL.

OpenSSL es la librería para seguridad que ha sido escogida para mantener la comunicación entre HemoHub y ACLTop cifrada. Esta se tiene que instalar como si de un ejecutable se tratara. Tiene soporte y es multiplataforma.

Como se puede apreciar en la figura 4, el módulo del servidor, dispone tanto de la capacidad de crear un servidor HTTP utilizando la librería CivetWeb, como la capacidad de guardar los mensajes registrados.

La implementación de esta sección se ha realizado siguiendo el modelo MVC (Modelo-Vista-Controlador) como se podrá ver en las siguientes secciones, para aportar al proyecto una modularidad y facilidad de mantenimiento añadido.

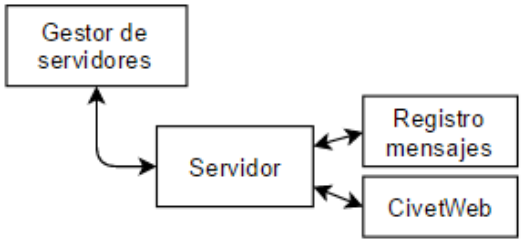

Figura 4: Estructura de los módulos de comunicación.

4.3.1 Registro de mensajes

El módulo que se encarga de guardar los mensajes registrados, se ha utilizado una función estándar para guardar los valores como se muestra a continuación:

```
map<ID, FUNCION > Registro
```

Del campo ID corresponde a un *string* que representa el recurso de una petición HTTP. El campo FUNCTION, es un *Callback* que contiene la función que se quiere ejecutar (usualmente de otros módulos) para un ID concreto.
Por ejemplo, si queremos devolver la versión de nuestro sistema que se encuentra en la función "DevolverVersion" cuando nos llega la siguiente petición HTTP: "HTTP GET /GetVersion", en el contenido del "Registro" debería haber una entrada equivalente a: <"/GetVersion", DevolverVersion>.

4.3.2 CivetWeb

Por lo que respecta al uso de CivetWeb es relativamente sencillo, únicamente tendremos que añadir los archivos de la librería en nuestro proyecto y crear una instancia con unos parámetros de entrada que serán los parámetros que el servidor utilizará como opciones de configuración (un *string*), tal y como se muestra a continuación:

```
CivetServer ServerHTTP(opciones)
```

Con esta sentencia, la librería CivetWeb, levanta un servidor HTTP mientras la instancia no se destruya. Las opciones que nos permite modificar el servidor durante su creación son diversas: el directorio a servir, el puerto, si el servidor utiliza SSL, el número de *threads* que el servidor tendrá, etc.

Como ya se ha comentado, para la creación del servidor que se encarga de HemoHub, se ha utilizado SSL TLS 1.2 como método de seguridad utilizando la librería de OpenSSL. De esta manera las comunicaciones entre HemoHub y ACLTop serán seguras al estar cifradas.

Para gestionar las peticiones que llegan al servidor, se da soporte a las peticiones HTTP GET y POST. Se ha decidido dar soporte sólo a estos dos tipos para mantener la lógica del funcionamiento tradicional de una función: tiene o no parámetros de entrada y devuelve o no un resultado. En ambos casos, tanto parámetros (utilizando POST), como devolver resultado (utilizando POST o GET), se utiliza un JSON como método de intercambio de información.

La implementación de los métodos que se encargan de implementar el GET y el POST dista muy poco una de la otra. Si la petición es GET, se comprueba que el recurso que se pide exista (si en el Registro hay una entrada con esa ID), en caso afirmativo, se ejecuta la función (*callback*) relacionada con esa petición y, por último, se devuelve un mensaje HTTP dónde, en la cabecera llevará un código HTTP y en el cuerpo, el mensaje de respuesta en JSON. Si la petición es POST, el proceso es idéntico, pero se ha de comprobar si se ha recibido un JSON de entrada para utilizarlo dentro de la función de *callback* a la que se llama.

Por lo que respecta a los códigos HTTP mencionados anteriormente, son códigos numéricos que aparecen en las cabeceras de los mensajes HTTP. Estos tienen mucha utilidad puesto que cada número tiene relacionado un significado en concreto. Hay muchos códigos diferentes dentro de HTTP, pero en este proyecto sólo se han implementado los necesarios y los que se consideraban más útiles dadas las circunstancias. El código que se va a devolver como respuesta se puede modificar desde la función *callback*, gracias a que los códigos han sido empaquetados internamente en una estructura de datos. Los códigos implementados son los siguientes:

- "200 OK": la petición ha ido correctamente.
- "204 NO CONTENT": el mensaje de vuelta no lleva un contenido.
- "404 NOT FOUND": el recurso que el cliente ha pedido no se ha encontrado en la lista de mensajes registrados.
- "500 INTERNAL SERVER ERROR": indica que ha habido algún error internamente.

El código de respuesta se ha de especificar en cada uno de los *callbacks* manualmente, excluyendo el código 404 que se genera cuando el ID de la petición no está registrado en el "Registro" y el código 500 que se genera automáticamente si la llamada al *callback* no se puede ejecutar correctamente u ocurre un error interno. Aunque éstas dos últimas también se pueden especificar manualmente si así se desea.

Los módulos de Registro de mensajes y CivetWeb trabajan conjuntamente, pero son independientes. Un servidor podría tener su "Registro" lleno de mensajes y no tener el servicio de CivetWeb activado, y de manera contraria podría tener el "Registro" vacío y el servicio CivetWeb activo (aunque no podría responder a ninguna petición, ya que no tiene mensajes registrados), o incluso mantener el "Registro" con mensajes y poder activar y desactivar el servicio "CivetWeb" cuantas veces se quieran. Esto es así, porque se puede dar el caso que se necesite activar y desactivar un servicio en distintas ocasiones, para el cual solo se necesita registrar las funciones una

vez, de esta manera se incrementa su versatilidad.

4.3.3 Interfaz y gestor de servidores

Como se puede apreciar en la figura 3, un gestor de servidores puede llegar a contener más de un servidor. Esto tiene varias implicaciones, por ejemplo, saber en qué servidor necesitas registrar un mensaje, o saber qué servidor activar o desactivar. Hemos de recordar, que gracias al diseño modular en nuestro sistema podríamos tener distintos servidores activados al mismo tiempo. Este problema, se ha solucionado creando una interfaz que tiene acceso al gestor de servidores, en donde hay definido en un enumerado con el nombre de cada uno de los servidores que podría haber. Por ejemplo, si tuviéramos tres servidores podríamos tener definido algo semejante a:

```
enum InstanciaServ {Ser_1, Ser_2, Ser_3}
```

Este enumerado será un campo obligatorio en todas las funciones de la interfaz mencionada, que esconde, además, la implementación del sistema para que cualquier módulo de ACLTop pueda utilizarlo sin problemas.
Las funciones a las que podemos acceder desde la interfaz de comunicaciones de la implementación y sus parámetros son los siguientes:

`bool CrearServidoresHttp()`
: Inicializa el "Registro" de todos los servidores de manera que hace posible que se puedan registrar mensajes.

`bool RegistrarMensaje(InstanciaServ, ID, FUNCION)`
: En la instancia seleccionada, añade en el "Registro" el recurso ID y el *callback* FUNCION.

`bool BorrarMensaje(InstanciaServ, ID)`
: En la instancia seleccionada, elimina en el "Registro" el recurso ID y su *callback* FUNCION.

`bool HabilitarServidor(InstanciaServ, Dir, Puerto)`
: En la instancia seleccionada, levanta un servidor CivetWeb con el puerto indicado y sirviendo un directorio concreto.

`bool DeshabilitarServidor(InstanciaServ)`
: En la instancia selecionada, para el servidor CivetWeb que está habilitado.

`bool DestruirServidoresHttp()`
: Detiene y elimina todos los servidores habilitados y sus "Registros".

4.4 Serialización

La serialización es el segundo bloque de este proyecto, tiene el objetivo de transformar la información binaria de los sistemas propia de cada lenguaje de programación, a una que sea multiplataforma y todos los lenguajes puedan entender. Se ha hecho uso de JSON para este cometido. La librería en C++ elegida para llevar a cabo esta tarea ha sido rapidJSON, por su estupendo rendimiento [3], su facilidad de uso y porque nuevamente, no tiene dependencias.
El módulo de serialización también es independiente y todo el software del instrumento ACLTop puede acceder a la herramienta para serializar una estructura en formato JSON.

Debido a que se da soporte a mensajes HTTP GET y HTTP POST, es necesario el proceso de serializar una estructura a JSON para devolver un recurso en un mensaje, pero también es importante deserializar un JSON que pueda llegar en un mensaje de entrada a una estructura. Por lo tanto, hay dos grandes funcionalidades, serializar y deserializar.
La estructura que siguen las funciones de serializar es la siguiente:

```
string Serializar(TipoEntrada)
```

Internamente la función usa el parámetro de entrada y lo serializa en un JSON en el formato de salida *string*, utilizando la librería rapidJSON. Cabe mencionar que serializar a JSON se ha de hacer manualmente para cada tipo de entrada que se quiera serializar, por lo que la tarea de implementar una serialización de una estructura compleja no es sencilla.
Por el contrario, la estructura de las funciones de deserializar siguen la siguiente estructura:

```
void Deserializar(String json, TipoSalida)
```

La función escanea el JSON de entrada y para cada uno de los campos los asigna al tipo de salida conocido. De nuevo, encontramos el mismo problema, hemos de conocer la estructura del JSON de entrada para poder asignar correctamente los valores al tipo de salida, y si éste es una estructura compleja, la tarea volverá a ser complicada.

5 Validación y Pruebas

La fase de validación y de pruebas en un proyecto que está involucrado en realizar análisis médicos con personas, posiblemente es la más importante de todo el proyecto.

A lo largo del desarrollo del proyecto, para cada módulo y para cada una de sus funciones, se han realizado pruebas unitarias con el objetivo de comprobar el correcto funcionamiento de estas. Las pruebas unitarias son funciones que tienen el objetivo de encontrar un fallo en la implementación. Estas pruebas constaban en parte, en utilizar las funciones implementadas en situaciones esperadas, fuera de rango, erróneas, etc. Para de esta manera, observar si su comportamiento es el esperado.

Errores que afectaban a otros módulos, debido a pequeños cambios, se verían reflejados en la ejecución de las pruebas de unidad y se procedía a su continua reestructuración para volver a superar con éxito estas pruebas.

Las pruebas relacionadas con la implementación del protocolo HTTP, se llevaron a cabo con una herramienta de Google Chrome desarrollada con ese objetivo. La herramienta lleva por nombre DHC [8], y nos permite reali-

zar peticiones con los verbos HTTP que escojamos a una dirección y puerto que nosotros quisiéramos, en nuestro caso la dirección dónde se encontraba el servidor. Esta herramienta nos permite observar también el contenido de la respuesta de nuestro servidor, si esta está bien formada, si tiene todos los campos, si el código HTTP es el correcto, el JSON de salida, etc.

El único problema que tiene DHC en este proyecto, es que necesita que una persona lo utilice. Y eso no es factible para poder ejecutar todas las pruebas de unidad. Por ello, este inconveniente se convirtió en una nueva herramienta que el equipo de ACLTop también podría utilizar en otros módulos. De la misma manera que el objetivo de este proyecto era crear un servidor HTTP, se pensó que, para automatizar las pruebas, se crearía un cliente HTTP. El cliente se ha implementado utilizando una vez más una librería C++. Esta librería lleva por nombre WinHttpClient [9] y está basada en las librerías de comunicaciones HTTP que Windows dispone. Con esta nueva implementación se pudieron automatizar las pruebas unitarias (no sin primero realizar pruebas unitarias sobre el cliente) al mismo tiempo que ofrecer una nueva herramienta para ACLTop. Esta librería se desarrolló siguiendo el concepto de la interfaz de comunicaciones. Se implementaron los verbos HTTP GET, POST y una función que permite descargar archivos, útil para futuros proyectos. La estructura de las funciones GET y POST es la siguiente:

`GetResp(UrlServ, Puerto, Recurso, SSL)`
: Envía una petición GET a la URL y puertos con el recurso indicado. Se puede decidir si utilizar SSL o no. Recibe un *string* con el contenido de la respuesta.

`PostResp(UrlServ, Puerto, Recurso, Datos, SSL)`
: Envía una petición POST a la URL y puertos con el recurso indicado, añadiendo además en el cuerpo del mensaje, el contenido de los datos. Se puede decidir si utilizar SSL o no. Recibe un *string* con el contenido de la respuesta.

Otra de las pruebas realizadas a lo largo del desarrollo del proyecto tenía como objetivo comprobar el uso de memoria del sistema a lo largo de un día de trabajo. Se analizaron dos situaciones. La primera consistía en que el Gestor de Servidores creara y destruyera constantemente un servidor, para ver si esta acción repercutía negativamente en memoria, comprobando así la gestión de recursos. La segunda prueba consistía en tener un servidor habilitado que recibía peticiones constantemente durante un día, para observar si la gestión continua de mensajes provocaba alguna pérdida en memoria. Las dos pruebas se superaron con éxito manteniendo el uso de memoria constante durante toda su ejecución.

Por último, se ha realizado una prueba que intenta recrear el uso real del instrumento simulando al mismo tiempo tres servidores distintos desde el gestor de servidores. Estos servidores implementan tres servicios (clientes) distintos a HemoHub que ACLTop actualmente está utilizando gracias al desarrollo de este proyecto. Durante un periodo de dos horas se ha generado un archivo indicando el momento de recepción de los mensajes de cada servicio para comprobar el movimiento interno de los mensajes de este proyecto en ACLTop en una situación que podría ser real, en la cual cada uno de los tres servicios envía peticiones a un ritmo distinto, de la cual se ha podido extraer la siguiente figura.

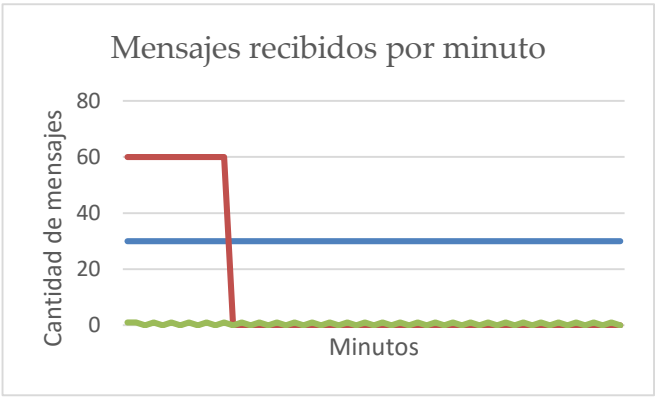

Figura 5: Prueba realizada con tres clientes y un servidor HTTP.

El resultado fue el esperado. El servicio señalizado con color rojo en la figura 5 mantuvo un flujo intenso de mensajes, pero después de un tiempo el número de mensajes fue nulo. Este servicio simula el comportamiento que tiene un servicio que envíe peticiones para mostrar la información de los resultados médicos del instrumento. El servicio representado en azul, se mantuvo constante durante todo el análisis. Este simula la petición de un servicio para mostrar las temperaturas de los sensores. Y el último, el servicio señalizado en verde, enviaba una cantidad mínima oscilante de mensajes a lo largo de la prueba. Este último simula un servicio que quiere saber el estado del instrumento.

Esta última prueba demostró que el software desarrollado es capaz de soportar un nivel de carga de trabajo con distintos clientes haciendo peticiones al instrumento.

5 Conclusión

A lo largo del desarrollo de este proyecto se ha planificado, diseñado e implementado un software capaz de cumplir con los objetivos iniciales propuestos. Este proyecto se ha realizado para formar parte del nuevo software del instrumento de Werfen, ACLTop, dotándolo así con nuevas funcionalidades. El objetivo principal de este trabajo de final de grado se podría resumir en la necesidad de la entidad externa HemoHub, otro proyecto de Werfen, para obtener y mostrar la información de la pantalla de cada instrumento utilizando una serie de mensajes entre ACLTop y el propio HemoHub.

Durante este proceso, se ha analizado la estructura de software interna del instrumento de diagnóstico ACLTop, para así, desarrollar un software que se adapte a la metodo-

logía y filosofía que el instrumento sigue. Se han barajado distintas opciones para su diseño e implementación, pero el resultado obtenido implementa y sobrepasa los objetivos iniciales.

Werfen necesitaba implementar únicamente unos mensajes en un servidor HTTP para permitir a HemoHub y ACL-Top comunicarse. Pero con la realización de este proyecto de final de grado, se ha desarrollado un software que es capaz, además: de crear distintos servidores HTTP mediante un gestor de servidores, registrar cualquier tipo de mensaje, un cliente HTTP con diferentes utilidades y una herramienta para serializar a formato JSON. Todo ello, siguiendo un diseño estructurado que permite la modularización para su fácil mantenimiento y futura actualización. De esta manera, Wefen Systelab podrá utilizar estas herramientas en futuros proyectos.

Haciendo referencia a estos, Werfen Systelab, ya tiene en mente nuevos módulos para ACLTop haciendo uso de la implementación de este proyecto.

AGRADECIMIENTOS

Agradecer a Werfen Systelab por enseñar y formarme en la profesión que he elegido, por darme la oportunidad de realizar un proyecto de estas características y por permitirme sentir el entorno laboral de una empresa internacional y de gran competitividad. Agradecer por supuesto a mi tutor Miquel Àngel Senar, por los consejos y esfuerzos para guiarme durante la realización de este proyecto.

BIBLIOGRAFÍA

[1] D. Gourley and B. Totty, HTTP: The Definitive Guide. O'Reilly Media, 2002.

[2] S. Meyers, Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14. O'Reilly, 2014.

[3] P. Farrell-Vinay. (2008). Manage software testing. [Online]. 573p. Disponible en: http://site.ebrary.com/lib/bibliotequesuab/docDetail.action?docID=10218341

[4] Google. (2016, 6 de Junio). FlatBuffers: Benchmarks [Online]. Disponible en: https://google.github.io/flatbuffers/flatbuffers_benchmarks.html

[5] M. Yip (2015). RapidJSON Documentation. [Online]. Disponible en: http://rapidjson.org/

[6] E. Young (2015). OpenSSL Documentation. [Online]. Disponible en: https://www.openssl.org/docs/

[7] Various authors. (2016). CivetWeb Documentation. [Online]. Disponible en: https://github.com/civetweb/civetweb/blob/master/README.md

[8] Reslet (2016). DHC Sercive. [Online]. Disponible en: https://dhc.restlet.com/

[9] Cheng Shi (2010, 21 de Septiembre). A Fully Featured Windows HTTP Wrapper in C++. [Online]. Disponible en: http://www.codeproject.com/Articles/66625/A-Fully-Featured-Windows-HTTP-Wrapper-in-C