Proyecto Integrador de Redes y Sistemas Operativos: Cliente de LEGO

A.A. Arévalo Alvarado[1], A. Badilla Olivas[1],
G. Rivera Hernández[1], J. P. Chacón González[1]

E.C.C.I., Universidad de Costa Rica.

Contributing authors: ariel.arevalo@ucr.ac.cr;
anthonny.badilla@ucr.ac.cr; geancarlo.riverahernandez@ucr.ac.cr;
jean.chacongonzalez@ucr.ac.cr;

Resumen

En esta investigación se presenta el desarrollo e implementación de un cliente en `C++` que utiliza el protocolo HTTPS para solicitar y obtener figuras de *LEGO* específicas desde un servidor. El proyecto se llevó a cabo siguiendo una arquitectura modular similar al patrón MVC y se desarrollaron ocho clases para gestionar la comunicación, el almacenamiento y la representación de las figuras. Además, se propone un protocolo para la comunicación entre servidores intermediarios y servidores de piezas. Se describen los resultados de las pruebas realizadas y se concluye con una reflexión sobre la complejidad de desarrollar un programa con funcionalidad de red y las expectativas para las etapas futuras del proyecto.

Keywords: HTTPS, servicio de red, protocolos de comunicación

1. Introducción

Esta investigación presenta el desarrollo e implementación de un cliente que utiliza el protocolo HTTPS para solicitar figuras de *LEGO* específicas a un servidor y obtener las piezas necesarias para construir el objeto solicitado. El proyecto se llevó a cabo en varias etapas, de las cuales la construcción del cliente, así como la definición de un eventual esquema de comunicación entre servidores intermediarios en el servicio de figuras, representan la primera. El documento describe la metodología utilizada

para lograr estos objetivos y presenta los resultados obtenidos. El programa en funcionamiento fue desarrollado para entornos Linux y fue probado con éxito según los requisitos del proyecto.

2. Desarrollo

2.1. Cliente

El desarrollo de este proyecto se llevó a cabo en `C++` siguiendo una arquitectura modular en la cual se intenta adecuadamente separar responsabilidades en los diversos componentes. Además, la arquitectura se podría caracterizar como una variante de *MVC* de la siguiente forma:

- Modelo: Existe una clase que representa una figura de *LEGO*, esta acompañada de la capa de datos, la cual consiste en un repositorio para las figuras.
- Vista: Tenemos únicamente una clase de bitácora, o registro, la cual despliega a la consola la información de las figuras.
- Controlador: La única funcionalidad presente para ser parte del controlador consiste en la solicitud de una figura del repositorio y su impresión en consola.

Para el desarrollo de este proyecto se elaboraron ocho clases:

2.1.1. Logger

Se encarga de toda la impresión de la información que debe desplegar el programa, información tanto de errores (con un *traceback* del lugar en el que ocurrió el error), como de excepciones. Así mismo, se encarga de llevar el control del tiempo que cada método tarda en completar su ejecución.

2.1.2. Figure

Representa una dada figura de *LEGO*. Contiene tanto el nombre como las partes de cada figura, así como la funcionalidad necesaria para ser construida a partir del HTML obtenido del servidor de piezas.

2.1.3. FigureController

Dirige el proceso de obtener la figura deseada por el usuario e imprime la información de la misma. Solicita esta figura por nombre al repositorio, el cual encapsula la obtención de la misma.

2.1.4. FigureRepository

Maneja el suministro de figuras. Esto lo hace abriendo comunicación con el servidor de piezas para obtener los datos de cada figura que se le solicita. Para abrir comunicación hace uso del cliente HTTPS, y seguidamente hace una solicitud GET.

2.1.5. HttpsClient

Abre comunicación con un recurso y *host* dado. Para abrir y mantener una conexión hace uso del socket, y elabora las solicitudes necesarias conforme con el protocolo HTTPS.

2.1.6. Ipv4SslSocket

Maneja una conexión SSL sobre TCP sobre IPv4. Facilita la apertura de esta conexión, así como la lectura y escritura a través de la misma. Hace uso de `sys/socket.h`.

2.1.7. SslCtxPtr y SslPtr

Consisten en wrappers RAII para los punteros de sistema `SSL` y `SSL_CTX`.

2.2. Protocolo para Servidor Intermediario

El protocolo propuesto consiste opera de la siguiente forma:

- Un servidor de piezas que entra en operación hace *broadcast* (UDP) en el puerto 7777 con su dirección IP y el nombre de las figuras que contiene. Los servidores intermediarios hacen *listen* en el puerto 7777. Al recibir un *broadcast* de un servidor de piezas, actualiza su tabla interna con la dirección IP y las figuras del servidor nuevo.
- Un servidor intermedio que entra en operación hace *broadcast* en el puerto 8888, de su dirección IP. Los servidores de piezas hacen *listen* en el puerto 8888. Al recibir un *broadcast* de un servidor intermediario, responde por el puerto 7777 con su dirección IP y el nombre de las figuras que contiene.
- Un cliente se comunica con algún servidor intermedio por HTTPS, esperando respuesta con el HTML de la figura solicitada, o la respuesta adecuada de HTTPS en su defecto.
- Al recibir comunicación de un cliente, un servidor intermediario se comunica con el servidor de piezas que contenga la figura solicitada según su tabla interna en el puerto 6666 por TCP. Una vez concretada la conexión, el servidor intermediario envía una hilera de texto que contiene el nombre de la figura solicitada, y el servidor de piezas responde con el HTML adecuado, el cual el servidor intermediario reenvía por HTTPS al cliente.
- Si la figura solicitada no se encuentra en la tabla interna del servidor intermediario, o este no logra concretar conexión con el servidor de piezas, el servidor intermediario retorna la respuesta adecuada de HTTPS al cliente, 404, y actualiza su tabla interna con la ausencia del servidor de piezas del caso.

3. Resultados de Pruebas

Resultado o salida al solicitar las piezas de la figura "horse":

```
[29 ms][INFO]: HORSE
5 pieces of brick 2x2 yellow
4 pieces of brick 2x4 white
2 pieces of brick 2x4 yellow
1 pieces of brick 2x6 yellow
1 pieces of brick 2x3 yellow
2 pieces of brick 1x1 yellow eye
2 pieces of brick 2x1 yellow
1 pieces of brick 2x2 white
Total amount of parts for horse is: 18
[29 ms][INFO]: Finished.
```

Resultado o salida al solicitar las piezas de una figura inexistente:

```
[43 ms][ERROR]: Figure is empty, cannot print.
[43 ms][INFO]: Finished.
```

Nota: Este suceso no se maneja en la capa de red debido a que la respuesta de HTTPS recibida es de 200, indistinguible de una exitosa, por lo que el recurso parece existir. Dado esto, el repositorio construye exitosamente una figura, la cual simplemente resulta tener datos vacíos. Por esto, es hasta que se intenta imprimir la figura que se logra identificar que la figura no existe. Se recomienda retornar respuestas adecuadas del servidor de piezas cuando este se desarrolle.

4. Conclusión

Con base en la experiencia de desarrollar un servicio de red utilizando solamente las facilidades dispuestas por el sistema operativo, se ha logrado obtener una renovada apreciación por la multitud de partes en movimiento que cooperan para posibilitar la telecomunicación. Aún por limitada que fuera la experiencia solo de construir únicamente el mecanismo receptor de esta interacción, se ha logrado cultivar un entendimiento más profundo de la complejidad requerida para llevar a cabo un programa que presente funcionalidad de red. Es de la esperanza de este grupo que las etapas restantes de este proyecto sean tan enriquecedoras como esta, y se espera con anticipación erudita la labor que traerá las etapas restantes de esta empresa educativa.