Protocolos de Comunicación

Trabajo Especial

Grupo 3

Juan Marcos Bellini - 52056

Diego de Grimaudet de Rochebouët - 55821

Juan Li Puma - 55824

Abstract

Este documento describe el Trabajo Especial de la materia Protocolos de Comunicación para la cursada del segundo cuatrimestre de 2016.

Requirements Language

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Índice

Implementaciones

Para este Trabajo Práctico Especial se implementó, como pedía el enunciado, un servidor proxy para el Extensible Messaging and Presence Protocol (XMPP), con ciertas funcionalidades extra que el servidor de origen XMPP no proporciona por defecto. Con un enfoque en performance, se implementó un proxy que funciona con clientes ya existentes.

Proxy XMPP

El corazón del proyecto consiste de un servidor proxy XMPP concurrente. Éste escucha en un puerto particular, se configuran los clientes para conectarse al proxy como si fuese un servidor, y para el cliente lo demás es transparente. El centro de la aplicación son los "handlers" que son los que se encargan de decidir qué hacer con el input/output que hay. Se decidió dividir los handlers en un sistema de herencia para cada funcionalidad. Se tiene el server/cliente que cada uno habla con el cliente o el servidor respectivamente y se encarga de hacer la negociación. Y después se tiene el read/write que se encarga de actuar de *proxy* después.
Para poder interpretar los xml se uso la libreria aalto que permite indicar en qué parte del xml se encuentra, si es el principio de un tag, etc...

Protocolo de Administración

Para controlar el comportamiento del proxy en tiempo de ejecución, como solicita el enunciado, se implementó un protocolo. Éste es un protocolo basado en texto que se usa para conectarse al proxy por un puerto distinto a aquel en el que éste escucha conexiones XMPP, y manejar una sesión en la cual el usuario puede autenticarse, consultar métricas y cambiar configuraciones. Al ser basado en texto, el usuario puede conectarse con cualquier herramienta de red, por ejemplo *netcat*, y leer y escribir texto normalmente. Los comandos que expone el protocolo se describen a continuación en ABNF.

Sintaxis

```
commands    = ptclC / helpC / authC / mtrcC / cnfgC / blckC / unblckC
              / mplxC / l337C / unl337C / langC / userC / logoutC / quitC

ptclC       = "PTCL" LF
```

```
helpC       = "HELP" LF

authC       = "AUTH" SP admin-name SP password LF

mtrcC       = "MTRC" [SP ( metric-name / "ALL")] LF

cnfgC       = "CNFG" LF

blckC       = "BLCK" SP user_jid LF

unblckC     = "UNBLCK" SP user_jid LF

mplxC       = "MPLX" SP ( username / "DEFAULT") SP ( ( server-name port ) /
              "DEFAULT") LF

l337C       = "L337" LF

unl337C     = "UNL337" LF

langC       = "LANG" [SP "DEFAULT"] *(SP lang) [SP "DEFAULT"] LF

lang        = 1*ALPHA

userC       = "USER" SP action LF

action      = ("CREATE" SP username SP password [SP priviledge])  /
("DELETE"
              SP username) / "UPDATE" SP username [SP PASS SP password] [SP
              "PRIVILEDGE" SP priviledge])

priviledge  = *DIGIT

logoutC     = "LOGOUT" LF

quitC       = "QUIT" LF

username    = 1*ALPHA

password    = 1*(ALPHA/DIGIT)

server-name = FQDN de un host al cual conectarse

user_jid    = JID definido en el RFC de XMPP
```

Cualquier implementación del protocolo DEBE implementar todos estos comandos.

Códigos de Respuesta

Para todo comando el servidor DEBE responder con un código y un mensaje (posiblemente vacío). Las respuestas del servidor DEBEN ser de la forma:

```
answer = answer-code SP DQUOTE answer-message DQUOTE LF
```

El answer-code es un código que representa el tipo de respuesta. Todas las respuestas están divididas en tres grupos. Las respuestas de éxito para cuando la operación se realiza exitosamente. Las respuestas de error de usuario, para cuando la operación falla por culpa del usuario. Y las de error de servidor, usadas cuando el problema fue culpa del servidor.

Éxito Axx

Esta clase de código de respuesta indica que la acción que pidió realizar el usuario fue recibida, entendida y aceptada por el servidor y que se pudo efectuar sin problemas.

A00 OK
El pedido funcionó. Esta respuesta es la respuesta genérica que debe ser usada en la mayoría de los casos como respuesta cuando la operación es exitosa. Si un comando no tiene especificado un código de retorno en caso de éxito, se usa éste por defecto.

Error de Cliente Bxx

La clases de mensajes Bxx se usa cuando el error es generado por el usuario.

B00 UNAUTHORIZED
Este código debe ser devuelto cuando el usuario usa un comando que requiere autenticación para ser usado. El usuario necesita autenticarse de manera exitosa usando el comando AUTH antes de poder usarlo.

B01 FORBIDDEN
El usuario está autentificado pero su usuario no tiene suficientes privilegios como para hacer esta operación.

B02 FAILURE

El usuario hizo un comando que fue comprendido por el server y que el usuario estaba permitido de efectuar pero por los parámetros dados por el usuario lo que tenía que hacer el comando falló. Esto se usa por ejemplo cuando el usuario intenta autenticarse con un usuario/contraseña errónea.

B03 UNEXPECTED COMMAND
Este codigo tiene que ser retornado cuando el servidor entiende el comando que se está queriendo efectuar pero no es el momento adecuado para efectuarlo porque el servidor se encontraba en un estado en el cual este comando no era permitido.

B04 WRONG NUMBER OF PARAMETERS
Retornado cuando el usuario hace un commando pero la cantidad de parámetros no es válida para ese comando

B05 PARAMETERS MALFORMED
Retornado cuando el usuario hace un comando con la correcta cantidad de parámetros pero la sintaxis de los parámetros no es la esperada. Por ejemplo si se esperaba un número y se envía un letra.

B06 UNKNOWN COMMAND
Retornado cuando el server no comprende el comando

B07 TOO MANY REQUESTS
Retornado cuando el mismo cliente hace demasiado pedidos en una corta cantidad de tiempo. Es RECOMENDADO limitar un usuario a una cantidad de comandos dado una cantidad de tiempo.

B08 NOT FOUND
Retornado cuando el servidor entiende completamente el pedido, y es un pedido que permite elegir cierto recurso entre varios que el servidor provee, pero el recurso pedido es uno que el servidor no contiene. Por ejemplo cuando se pide una métrica no implementada.

B09 POLICY VIOLATION
Retornado cuando el usuario viola alguna política definida por el servidor.

Errores de Servidor Cxx

Son los códigos de error que indican que el servidor entiende que hubo un error y que fue por culpa suya.

C01 INTERNAL SERVER ERROR

Retornado cuando hay un error inesperado que provoca que el pedido no pueda ser procesado.

C02 SERVICE UNAVAILABLE
Retornado cuando el servidor está demasiado ocupado por razones internas y entonces no puedo procesar el pedido

C03 PROTOCOL VERSION NOT SUPPORTED
Retornado cuando el usuario pide un protocolo que no está soportado por el servidor

C04 COMMAND NOT IMPLEMENTED
Retornado cuando el servidor comprende el comando que le piden efectuar pero este no ha implementado ese comando.
El answer-message es un mensaje que se usa para dar información extra al usuario sobre el éxito o falla de lo que se haya intentado hacer. Es tambien usado por los comandos que retornan información para retornar la información pedida. Si formato del mensaje de respuesta es especificado por el comando entoncesse DEBE seguir ese formato y no se puede proveer informacion adicional.De lo contrario no es imbligarotorio que el mensaje no este vacio pero es RECOMENDADO usarlo para darle al usuario informacion adicional.

Comandos

PTCL
El comando PTCL se DEBE usar únicamente al comenzar la conexión y se usa la para indicar la versión del protocolo que se debe usar. En caso de especificar se DEBE usar la versión 100. Si se usase el comando cuando ya se han usado o intentado usar otros comandos, se DEBE retornar el error "B03" (UNEXPECTED COMMAND)

HELP
Muestra una lista con todos los comandos soportados por el servidor proxy.

AUTH
Permite autenticarse. En caso de que las credenciales no matcheen con un usuario/contraseña se debe retornar el error "B02 FAILURE".

MTRC
Accede a las métricas. Si se usa el comando sin ningún parámetro, el servidor DEBE retornar como mensaje auxiliar la lista de todas las métricas separadas por un espacio de la forma:
`*(metric SP) LF`
Si se proporciona un parámetro, el servidor responde con el valor de la métrica, si existe. De no existir, responde con código B08. En el caso particular de que el nombre de métrica sea "ALL"

(case-sensitive) el servidor DEBE responder con una lista de todas las métricas con su respectivo resultado al lado de la forma:
`*(metric SP metricResult SP) LF`

CNFG
Retorna todas las configuracion que hayan sido efectuadas como mensaje auxiliar que DEBE ser de la forma:
`"L337" SP ("ON"/OFF") SP "#" SP "BLCK" SP ("NONE" / 1*(username SP)) SP "#" SP "DEFAULT" SP server-name SP port SP # "MPLX" SP`

BLCK

El comando BLCK bloquea a un usuario el servidor proxy para que este ya no pueda enviar ni recibir mensajes (stanzas "message"). De intentar enviar un mensaje se le notificará al usuario con un error.

UNBLCK

El comando UNBLCK le quita la condición de bloqueado al usuario que se le indique, permitiendo que pueda volver a mandar mensajes.

MPLX

Este comando permite al administrador de multiplexar a un usuario al servidor default o algun otro que no sea el default o de cambiar el default.
Si el primer parámetro es <username> entonces se DEBE multiplexar a ese usuario a lo que sea indicado a continuación.
Si el primer parámetros es "DEFAULT" entonces se debe cambiar el servidor default por lo que sea indicado a continuación.
Para determinar el servidor destino se usan los siguiente parámetros. Si el segundo parámetros es "DEFAULT" entonces significa que DEBE ser multiplexado al servidor que esta indicado como default.
De lo contrario, se debe tomar el servidor destino como el segundo parámetro y el tercer parámetro el puerto destino.
Si se efectua la action MPLX DEFAULT DEFAULT se debe responder como que la operación se realizó exitosamente y no se debe cambiar el servidor DEFAULT.

L337

El comando L337 indica al servidor se ponga en modo "leet" haciendo que a todas las stanzas "message" que pasan por el servidor, a los caracteres dentro del del tag <body>, se les efectúen las siguientes transformaciones:

* a por 4 (cuatro)
* e por 3 (tres)
* i por 1 (uno)
* o por 0 (cero)
* c por < (menor)
* t por 7 (siete) - adicional nuestro, no está especificado en el enunciado

Las transformaciones para los caracteres que son letras se efectúan tanto para las letras en minúscula como para las letras en mayúscula.

UNL337

El comando UNL337 indica al servidor salga del modo "leet", lo que implica que los mensajes ya no se transformarán al pasar por el <body>.

LANG

El comando LANG permite cambiar el idioma en el cual el servidor va a responder en los mensajes auxiliares.
Si uno usa a LANG si ningún parámetro, este devuelve una lista de todos los idiomas que soporta el servidor proxy. Escrito según la ISO 639-2.
Si se pasa algún parámetro, el servidor buscará entre los distintos <lang> que le pasaron y eligirá el primero que matchee con uno que soporte y usará ese como lenguaje para los mensajes auxiliares.
Si se pone como primer parámetro "DEFAULT" no solo se elige el idioma en el cual el servidor le responderá en esta conexión los mensajes con ese idioma, sino que también se pondra a ese idioma como el default del servidor con el cual responderá a menos que se cambie el lenguaje.
Si al final se pasa la palabra DEFAULT entonces se entenderá que en caso de que ninguno de los <lang> matchee con alguno del servidor se debe elegir el lenguaje default del servidor en vez de retornar error.
El lenguaje NO debe aplicarse a los nombres de los comandos. El lenguaje debe reflejarse en los mensajes de error y PUEDE reflejarse en los nombres de las métricas.
El servidor deberá retornar como mensaje-auxiliar el lenguaje que se ha elegido.

LOGOUT

Logout permite cerrar session. Ese comando NO DEBE cerrar la conexión establecida. además, tanto si el usuario estaba logueado como si no, el resultado debe ser que el usuario quede no logueado.

QUIT

Quit permite cerrar la conexión de manera elegante. Debe ser usado cuando el usuario quiere cerrar la conexión.

USER

Este comando permite crear, eliminar o modificar cuentas de usuario del protocolo de administración. Es recomendable que para realizar esta acción se requiera un nivel de privilegios más alto que el usuario commun. Es recomendable que exista un usuario que pueda realizar estas acciones.

Problemas Encontrados

Durante el diseño y la implementación del proxy surgieron varios problemas. El primero de ellos fue el manejo de las stanzas XML. El primer intento a resolver esto fue modelarlas con objetos que representaran las distintas posibles stanzas. En particular, como existen stanzas que tienen elementos hijos, éstas se declararían como clases internas de las stanzas "madre" y se retornaría el objeto pertinente una vez leídas la stanza madre y todos sus elementos hijos. El problema más grande que surgió con este approach fue que para poder retornar un objeto completo, se debería tener que leer bastante de la red, acumulando los bytes leídos hasta que se cierre la stanza madre. Esto podría fácilmente acumular mucho más que el tamaño de nuestros buffers (como sería el caso en un mensaje muy largo), deshabilitando por completo el proxy ya que no podría ni agregar bytes leídos al buffer, ni vaciar el buffer ya que no llegaría a leer el cierre de stanza. Para evitar tener que solucionar un problema complejo, se decidió procesar y enviar los bytes a medida que llegan al proxy, manteniendo un estado que cambia según se van leyendo stanzas. Esto, si bien nos quita la posibilidad de hacer chequeos de validez (el proxy puede estar enviando XML inválido sin saberlo), esto no es un gran problema ya que el servidor eventual responde con el mismo error con el que respondería el proxy si analizara el tráfico; además, nos da la posibilidad de interpretar mensajes mucho más grandes.

El segundo problema importante surgió de la decisión de usar AALTO como librería de interpretación de XML. Pese a que permite parsear los documentos XML con facilidad, trajo muchos problema ligados a su diseño; por ejemplo, nos vimos obligados a reconstruir todos los elementos XML que Aalto reconoce, un proceso propenso a errores, ya que éste no proporciona la funcionalidad de reconstruir un elemento entero una vez que se leyó su closing tag. Aalto también des-escapa todos los caracteres especiales que reconoce, por lo que nos vimos obligados a volver a escaparlos. Por último, pero no por menos, Aalto no proporciona información de un elemento hasta que haya leído todo su opening tag (no así con el texto interno del elemento), lo que nos obligó a monitorear cuánta memoria podría estar utilizando Aalto y lanzar un error pasado un umbral.

Un tercer problema fue el cómo almacenar los datos leídos. El primer intento era tener una cola "infinita" de bytes que eran pasados a un ByteBuffer de salida. Por cuestiones de optimización y para no depender de una cola de tamaño impráctico, se decidió utilizar un único ByteBuffer de tamaño no limitado (en principio). Se establece un límite de bytes leídos, que al superarse deshabilita la lectura para la conexión en cuestión hasta que se libere lugar. Cabe destacar que la idea de limitar la cantidad de bytes a leer en base al tamaño libre del buffer no servía, ya que al momento de leetear el texto se podrían generar más bytes que los leídos (ej. 'c' → "<"). Por esa razón, la condición de corte es cuando se alcanza o supera el límite; si por ejemplo, queda 1 byte hasta el límite, el próximo caracter a leer es c y leeting está habilitado, los bytes generados en exceso se guardan de todas maneras antes de deshabilitar la lectura para esa conexión. Esta estrategia obviamente introdujo limitaciones en cuanto a cuánto leer de la red, ya que el buffer se puede llenar.

Limitaciones

Nuestra implementación tiene ciertas limitaciones que se describen a continuación:

En primer lugar, el proxy no soporta ningún tipo de encriptación. Tampoco soporta negociaciones "complejas," es decir, al detectar el primer error en la negociación, de cualquier tipo, ésta se considera inválida y se termina en vez de darle la oportunidad al usuario de reintentar. La multiplexación está limitada a tomar efecto al momento de establecer una conexión (es decir, si un cliente está logueado y se lo multiplexa a otro servidor, hace falta que se reloguee para que esto tome efecto) ya que para que tomara efecto inmediatamente haría falta implementar un proceso de negociación adicional - tomando el cuidado de no desconectar al usuario si ocurre cualquier error.

Otra limitación de más bajo nivel surgió con la decisión de usar una cola de ByteBuffers. Mientras esto nos daba más flexibilidad a la hora de generar los mensajes de salida, como se tiene una cola de buffers puede pasar que un buffer esté casi completamente vacío y de todas maneras se tenga que hacer una pasada para imprimir con ese buffer y en la siguiente levantar otro que estaba más lleno.

Posibles Mejoras/Extensiones

Entre las posibles mejoras y extensiones que se podrían implementar están la de mejorar el sistema de negociación, para permitir negociaciones más "complejas." También se podría implementar soporte para encriptación para que las conversaciones transiten de manera segura. En este caso, si se habilita el modo l337 el proxy debería tomarse el trabajo de desencriptar todo el tráfico entrante, transformarlo como sea necesario, y volver a encriptarlo (posiblemente con otro mecanismo, ya que las conexiones cliente-proxy y proxy-servidor son independientes).

En cuanto a la limitación de ByteBuffers antes mencionada, se podría mejorar. También se podría agregar más funcionalidad, como por ejemplo:

- Nuevos métodos de l337 o de transformación de texto.
- Silenciado condicional (ej. Por tiempo, o cuando alguien se des/loguea)
- Multithreading

Conclusión

Después de la realización de este trabajo se pudo observar que la implementación de un protocolo existente y la creación de uno no es tan simples como uno podria pensar. Seguir un protocolo implica comprenderlo en su totalidad y para su correcto funcionamiento tiene que lograr seguir todo lo que le pide a la perfección, lo que no es una tarea menor. Y crear un protocolo fue sorprendentemente complicado. Cada decisión tiene un gran peso porque cualquier si esta mal diseñado va a ser muy complicado escalarlo cuando se saquen nuevas versiones.

Guía de Instalación

Para instalar y correr el proxy:

1. Clonar el proyecto
2. Abrir una terminal en la carpeta del proyecto 'chinese-whispers' (donde está el POM)
3. Generar el JAR ejecutable corriendo: `mvn clean compile assembly:single`
4. Correr el proyecto con: `java -jar target/chinese-whispers-1.0-SNAPSHOT-jar-with-dependencies.jar <proxy-port> <admin-port> <default-xmpp-server-adderss> <default-xmpp-server-port>`, reemplazando los valores entre picos con los deseados. Todos los parámetros son obligatorios.

Configuración y Monitoreo

Para configurar y/o monitorear la aplicación:

1. Conectarse al servicio de administración (corre en el mismo host que el proxy en el puerto admin-port especificado en la sección anterior)
2. Autenticarse con `AUTH protos 42`
3. Correr `HELP` o referirse a la especificación del protocolo de administración para ver los comandos disponibles y su sintaxis.

Para la configuración y monitoreo se usa el protocolo antes explicado. Gracias a los comandos L337, MPLX y BLCK se pueden realizar las distintas configuraciones de poner el modo leet, multiplexar un usuario y silenciar a un usuario que cumplen lo especificado en el protocolo. Para monitorear se usa el comando CNFG también antes explicado. Las métricas implementadas fueron:

- Cantidad de bytes leídos/escritos por el protocolo de administración
- Cantidad de bytes leídos/escritos por el proxy
- Cantidad de mensajes silenciados
- Cantidad de accesos

Arquitectura

La aplicación está separada en capas. Hay una capa de conexión, que implementa mecanismos de comunicación TCP. La misma utiliza selectores del paquete *java.nio*, para que los sockets utilizados sean no bloqueantes. Gracias a esto, la aplicación puede funcionar con un único *thread*.

Por encima de dicha capa, se posicionan dos capas (en paralelo) de protocolos de aplicación. La primera implementa parte del protocolo *XMPP* (lo necesario para cumplir con lo pedido en la consiga). En esta capa se implementan los mecanismos para poder actuar como un *proxy XMPP*, procesando los mensajes necesarios. La segunda capa implementa el protocolo de administración/configuración, el cual permite poder configurar el sistema mientras está corriendo.

La tercera y última capa (la capa de aplicación) contiene la lógica de aplicación. Guarda las métricas y valores de configuración del sistema. Además contiene el *main loop* de la aplicación. A continuación se muestra el diagrama de clases.

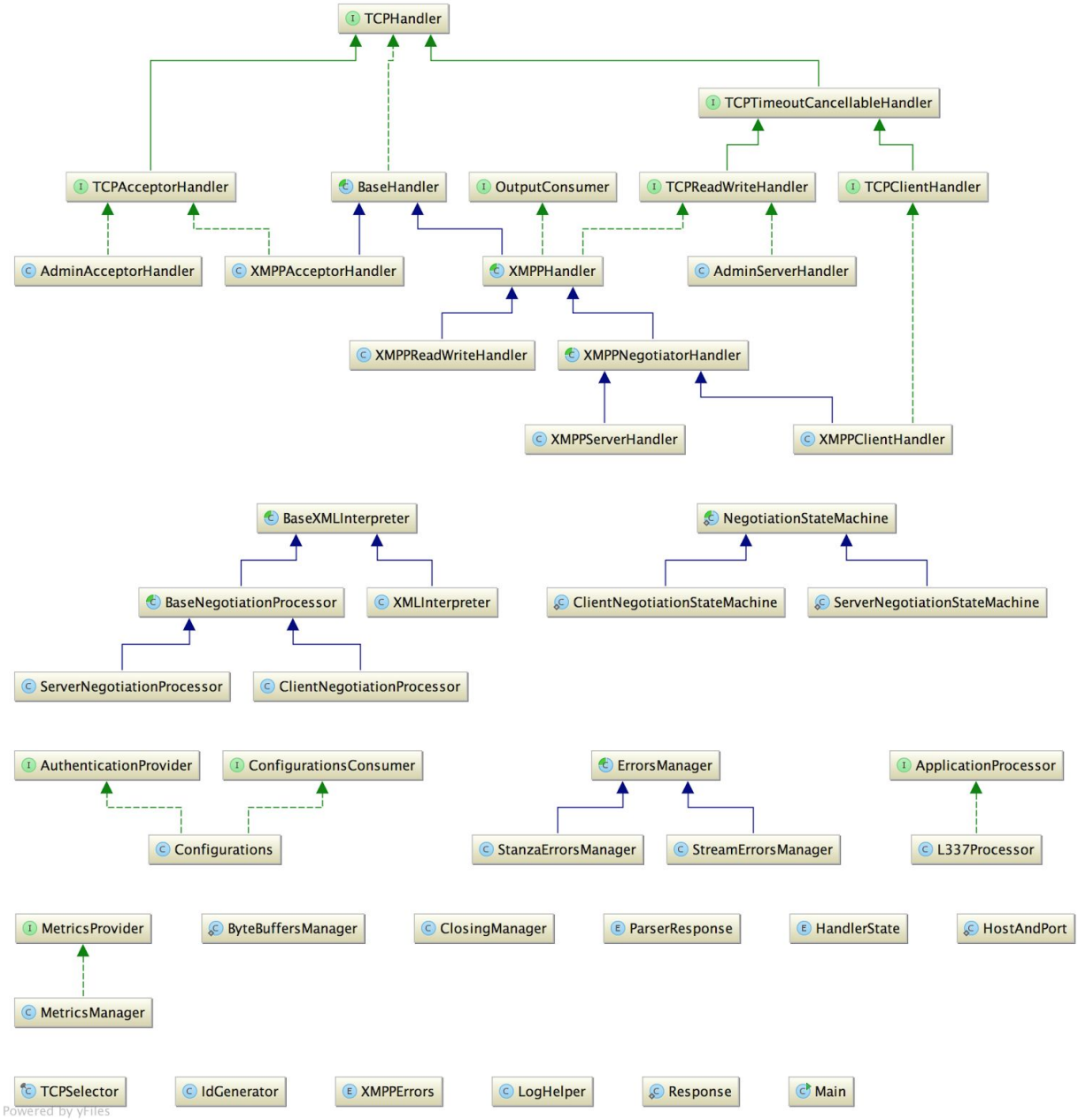

Figura 1: Diagrama UML del proyecto.

Ejemplo de uso del protocolo de aplicación

help
A00 "AUTH LANG HELP QUIT L337 UNL337 BLCK UNBLCK MPLX CNFG MTRC LOGOUT"
AUTH protos 42
A00 "OK"
asd
B06 "UNKNOWN COMMAND"
l337
A00 "OK"
l337
A00 "OK"
unl337
A00 "OK"
l337
A00 "OK"
cnfg
A00 "L337 ON # BLCK NONE # MPLX NONE # DEFAULT 10.1.34.106 3333"
mplx DEFAULT localhost 5222
A00 "OK"
CNFG
A00 "L337 ON # BLCK NONE # MPLX NONE # DEFAULT localhost 5222"
mtrc ALL
A00 "administrationSentBytes 298 readBytes 12091 numSilencedMessages 0
administrationReadBytes 98 numAccesses 2 sentBytes 12173 "
quit
A00 "OK"

Bibliografía

RFC 2119: Key words for use in RFCs to Indicate Requirement Levels
https://www.ietf.org/rfc/rfc2119.txt

ISO 639-1: Language codes
http://www.iso.org/iso/catalogue_detail?csnumber=22109