Proyecto 2

Carlos Gerardo Acosta Hernández
Andrea Itzel González Vargas
Luis Pablo Mayo Vega

Redes de Computadoras
Facultad de Ciencias, UNAM

Índice

1. Especificación de requerimientos

1.1. Enunciado del problema

De acuerdo con el modelo TCP/IP, *la Capa de Aplicación* es la de mayor importancia y en la que se sustenta todo el desarrollo de redes de computadoras pues está compuesta por los protocolos y demás servicios encargados de manejar, intercambiar o decodificar los datos que los usuarios se envían a través de distintos hosts para comunicarse. En otras palabras, se busca que dos o más procesos en distintas computadoras puedan ser capaces de intercambiar información y de esta manera otorgar una mayor capacidad de procesamiento y mejor rendimiento del que se tendría con un único host.

Sin embargo, la *Capa de Aplicación* no puede trabajar sola, siendo mediante un proceso de encapsulación en cada una de las capas, que los datos(PDU) se envian a las capas inferiores, añadiendo cada una de las capas información que le concierne para que sus protocolos puedan manejarlos.

Como sabemos, protocolos como HTTP, DNS, FTP, SMTP o DHCP tienen cada uno una estructura distinta ya bien definida y estandarizada en los articulos de *Request for Comments* publicados, pero eso no significa que sean los únicos protocolos disponibles en la *Capa de Aplicación*. La gran ventaja de esta capa es su adaptabilidad para que se desarrollen protocolos de acuerdo a las necesidades de la aplicación sobre la que se usarán.

1.2. Objetivo de la aplicación

Esta aplicación fue creada con el objetivo de implementar un protocolo de la capa de aplicación en el que el usuario, conectado del lado del cliente, solicite un pokémon a capturar. El servidor elegirá aleatoriamente alguno de los pokémon que estén en su base de datos, se lo ofrecerá al usuario y, si éste acepta capturarlo, también aleatoriamente se indicará si logró capturarlo o no.

El usuario también podrá ser capaz de consultar su Pokédex, con los nombres de los pokémon que ha capturado y la imagen de cada pokémon que seleccione.

El protocolo que hemos diseñado se enfocará en las acciones del usuario y el servidor que involucren una comunicación entre ambos, cada una con un tipo de mensaje específico. Es decir, si un usuario quisiera capturar un pokémon, solo tendría que enviar un mensaje con el código que el servidor entienda como "quiero capturar un pokémon"sin considerar otros aspectos como el nombre del pokémon o el tamaño de la imagen que contiene a tal pokémon.

En ese sentido, diseñar nuestro propio protocolo nos permite tener mayor control sobre la tasa de transferencia de los datos dentro de la aplicación, disminuyendo los costos de la comunicación y mejorando el desempeño del programa.

1.2.1. Casos de uso

Actor	Caso de uso	Descripción
Pokentrenador	Iniciar sesión	El usuario se conecta con el servidor y se le muestra el menú principal de la aplicación
Pokentrenador	Capturar pokémon	El usuario solicita al servidor que le muestre un pokémon y decide si intenta capturarlo (con un número finito de intentos) o no
Pokentrenador	Consultar pokedex	El usuario hace una consulta para buscar un pokémon en su pokédex
Pokentrenador	Cerrar sesión	Cierra la conexión con el servidor

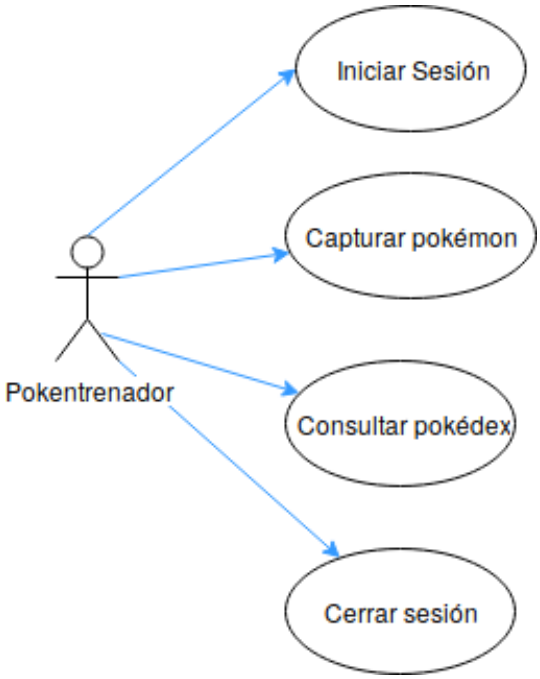
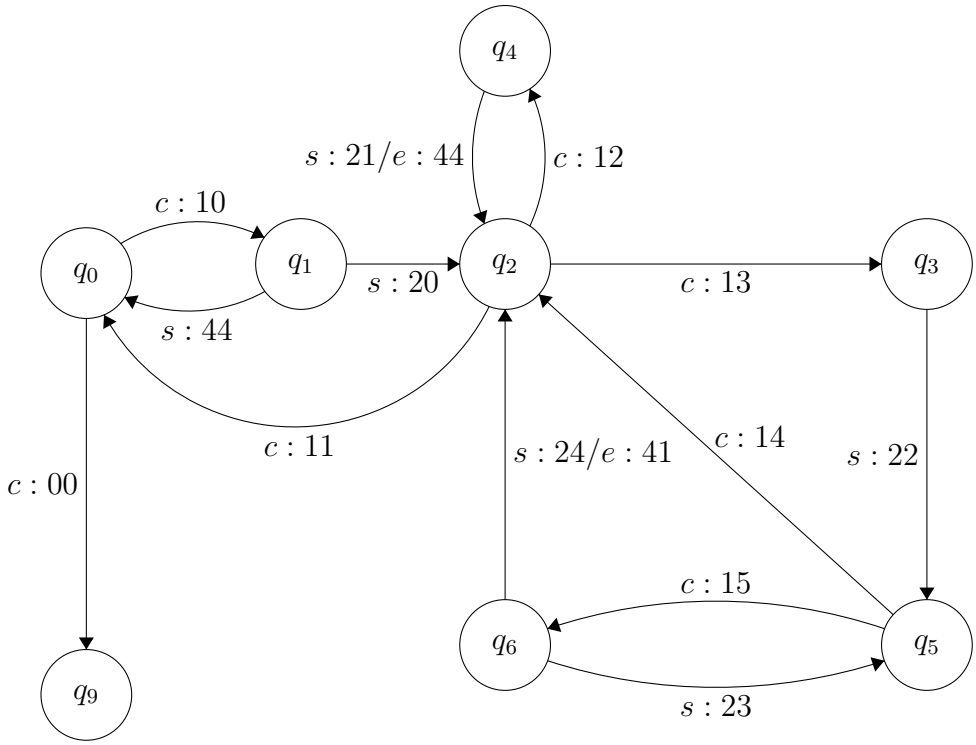


Figura 1: Diagrama de casos de uso de la aplicación

2. Diseño del protocolo

2.1. Máquina de Estados Finita

Máquina de Estados Finita para el protocolo de la capa de aplicación.



2.2. Descripción de los estados

Estado	Descripción
q_0	Conexión establecida, inicio de aplicación.
q_1	Inicio de sesión.
q_2	Menú de juego.
q_3	Solicitud de captura de *pókemon*.
q_4	Búsqueda de un pókemon en la *pókedex*.
q_5	Aparición de un *Pókemon* salvaje.
q_6	Intento de captura de *pókemon*.
q_9	Cierre de conexión.

2.3. Descripción de los mensajes en la comunicación cliente-servidor

Código	Segmento	Descripción
00	`|long|code|`	Termina la conexión para un cliente.
10	`|long|code|name|`	Solicitud de inicio de sesión del cliente. El parámetro `name` se refiere al nombre de usuario del cliente.
11	`|long|code|`	Solicitud de cierre de sesión del cliente.
12	`|long|code|name|`	Acceso y consulta a la *Pokédex* mediante el nombre de un pokémon.
13	`|long|code|`	Acceso a la opción de captura.
14	`|long|code|`	El usuario rechaza el pokémon salvaje ofrecido por el servidor.
15	`|long|code|attempts|name|`	Intento de captura del pokémon salvaje ofrecido.
20	`|long|code|`	Inicio de sesión del cliente exitoso.
21	`|long|code|long_n|name|long_img|img`	Resultado exitoso de la Pokédex con imagen del pokémon.
22	`|long|code|attempts|name|`	Selección aleatoria de un pokémon salvaje para el cliente. Incluye el máximo número de intentos.
23	`|long|code|attempts|name|`	Pokémon no capturado, disminución del número de intentos.
24	`|long|code|long_n|name|img|`	Pokémon capturado con imagen incluída.
41	`|long|code|`	Error: Máximo número de intentos de captura alcanzados.
44	`|long|code|`	Error: Consulta infructuosa.

2.4. Diseño de la base de datos

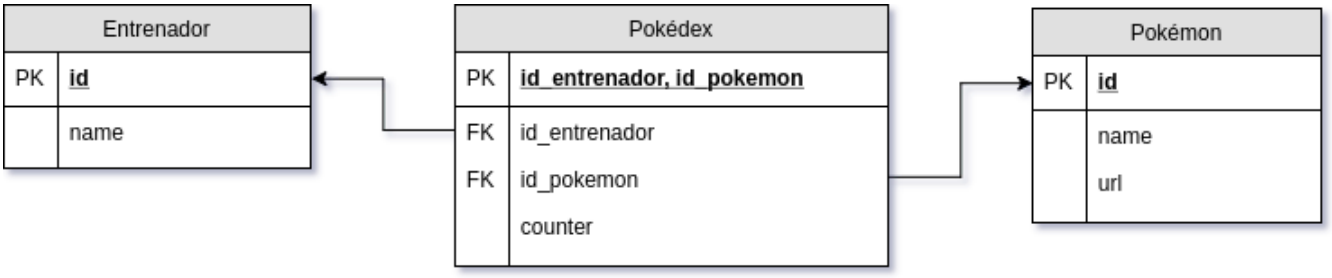


Figura 2: Diagrama de la base de datos relacional.

3. Implementación del protocolo

3.1. Especificación del ambiente de desarrollo

Lenguaje de programación:	Java	**versión:**	1.8
Herramienta de construcción de software:	Apache Ant	**versión:**	1.9.9
Sistema Manejador de Base de Datos:	sqlite	**versión:**	2.8.17
Driver de conectividad JDBC:	sqlite-jdbc	**versión:**	3.16.1
Sistema manejador de versiones:	GIT	**versión:**	2.14.1
Pruebas Unitarias:	JUnit y Hamcrest	**versión:**	4

3.2. Estructura del proyecto

3.2.1. Árbol del directorio (preinstalación)

```
p02
├── build.xml
├── documentos
│   ├── graphic
│   │   ├── casosdeuso.png
│   │   └── ...
│   ├── proyecto2.pdf
│   └── proyecto2.tex
├── lib
│   ├── hamcrest-core.jar
│   ├── junit.jar
│   └── sqlite-jdbc-3.16.1.jar
├── man
│   └── proyecto2.1
├── src
│   ├── redes
│   │   ├── ClienteHilo.java
│   │   ├── ConexionBD.java
│   │   ├── Controlador.java
│   │   ├── Estado.java
│   │   ├── FabricaMensaje.java
│   │   ├── Imagen.java
│   │   ├── Mensaje10.java
│   │   ├── Mensaje12.java
│   │   ├── Mensaje15.java
│   │   ├── Mensaje21.java
│   │   ├── Mensaje22.java
│   │   ├── Mensaje23.java
│   │   ├── Mensaje24.java
│   │   ├── MensajeGenerico.java
│   │   ├── MultiThreadPrueba.jav
│   │   ├── Pokentrenador.java
│   │   ├── Pokeservidor.java
```

```
│   │   ├── Proyecto2.java
│   │   └── test
│   │       └── ControladorTest.java
│   └── sql
│       ├── hi.db
│       └── poke_app_db.sql
└── static
    ├── poke_script.py
    ├── requirements
    ├── Squirtle.png
    └── ...
```

3.2.2. build.xml

Este archivo contiene las directivas de operación para el sistema de construcción ant. Es mediante su lectura que la estructura del proyecto da lugar a la compilación del código y el enlace entre sus dependencias.

3.2.3. src

En este directorio encontraremos dos subdirectorios o, más propiamente paquetes. En *redes/* se encuentra el código fuente principal del protocolo de aplicación, mientras que en *test/* se encontrarán los archivos de las pruebas unitarias contempladas para algunas de los métodos del proyecto.

3.2.4. static

Aquí deben encontrarse forzosamente las imágenes de los pokémon que estarán disponibles en la dinámica de la aplicación, con un nombre de archivo dado por el formato: `<pokémon>.png`. Además vienen incluídos, un *script* en python que permite volver a descargar en ese directorio las imágenes de los pokémons de la primera generación y su archivo de requerimientos para funcionar (que puede utilizarse mediante la introducción del comando: `$ pip install -r requirements`, previa a la ejecución del *script*, ya sea en un ambiente virtual o directamente sobre el sistema).

3.2.5. documentos

El documento que ahora mismo está leyendo debe encontrarse en este directorio, junto con el código fuente en *LaTeX* que lo genera. Además, el subdirectorio *graphic/* contiene las imágenes utilizadas en el código mencionado para la generación del documento *PDF*.

3.2.6. lib

Como ya se mencionó en la sección 3.1, se utilizaron bibliotecas externas tanto para las pruebas unitarias como para el controlador del *SMBD*. Tales recursos deben encontrarse en este directorio.

3.2.7. man

En este directorio se encuentra el archivo que contiene la información que será agregada al *man pages* (manual) de la distribución de *Linux* durante la instalación.

En la siguiente sección será más evidente la importancia del listado anterior, pues se describirá brevemente su interacción en el proceso de instalación y ejecución del proyecto. Adicionalemente, si se desea explorar la conformación del código a mayor detalle, recomendamos ver la sección 4.1.3.

4. Uso y pruebas del protocolo

4.1. Manual de uso

4.1.1. Compilación

Para compilar el proyecto y generar un recopilado de ejecutables de la *JVM*, un archivo *JAR*. Basta con ejecutar *Apache-Ant* sin argumentos, pues se ha especificado el *target* descrito por defecto:

```
[user@host p02]$ ant
```

4.1.2. Linux man pages

Si es preferible que las instrucciones generales de uso sean agregadas al *Linux man pages* para así mantener accesible la forma de uso del proyecto, preparamos el siguiente *target* en *ant* para este proposito:

```
[user@host p02]$ ant man
```

Cuya ejecución solicitará permisos de superusuario para agregar la página de manual de este proyecto. Una vez ingresada la contraseña de *sudo* será posible visitar el manual desde cualquier ruta mediante el siguiente comando:

```
[user@host ~]$ man proyecto2
```

4.1.3. Generación de documentación

Para explorar la organización programática del proyecto sin necesidad de visitar todos los códigos fuente involucrados en la construcción del proyecto, recomendamos generar la documentación de código *html*, para así hallar la explicación de cada método implementado, sus entradas y salidas y el papel que desempeñan en el protocolo. Para ello, basta efectuar:

```
[user@host p02]$ ant doc
```

Esto generará un directorio *doc/*, que contiene un conjunto de documentos en formato *html* descriptivos de cada clase en el código, relacionados unos con otros a través de hipervínculos que permiten una navegación fluida que no necesariamente es posible en la lectura directa del código.

4.1.4. Precondiciones

Antes de comenzar la ejecución es importante que de las secciones anteriores se haya efectuado cuando menos la 4.1.1. En caso de querer ahorrar los pasos anteriores y simplemente convenimos que se ejecuten todos, basta con el siguiente *target* de *ant*:

```
[user@host p02]$ ant all
```

También es importante que el archivo de la base de datos de *SQLite* se encuentre bajo la ruta: *src/sql/**hi.db***. De no hallarse ahí, pero sí el esquema de la base de datos en el archivo *src/sql/**poke__app__db.sql***, es posible generar la base de datos ejecutando:

```
[user@host p02/src/sql]$ sqlite3 hi.db < poke_app_db.sql
```

Finalmente, sólo resta verificar que las imágenes a utilizar por la aplicación se encuentren en la ruta adecuada. Por ejemplo, para el pokémon `charmander`, debe existir la imagen *static/**charmander.png***. En caso de no hayarse ninguno, a continuación damos una serie de instrucciones[1] para que el *script* incluído en ese directorio descargue de internet nuevamente las imágenes que la aplicación mostrará.

```
[user@host p02/static]$ virtualenv py_env
[user@host p02/static]$ source py_env/bin/activate
(py_env)[user@host p02/static]$ pip install -r requirements
(py_env)[user@host p02/static]$ python3 poke_script.py
```

Una vez asegurandonos de esto, podemos proceder a la ejecución de la aplicación.

4.1.5. Ejecución de pruebas unitarias

Si se desea incluir nuevas pruebas en el directorio *src/test/* para la verificación de alguna función del protocolo o simplemente correr las ya existentes, ejecutamos lo siguiente:

```
[user@host p02]$ ant test
```

Que compilará y ejecutará tanto el código de las pruebas como el de la aplicación y listará los tiempos, errores y salidas sin errores de cada una.

4.1.6. Ejecución

La ejecución del programa como viene especificado en el manual incluído (ver 4.1.2) se realiza mediante el archivo *JAR* resultado de la compilación y posee una cantidad de argumentos tanto para un cliente como para unservidor. Es importante señalar que decidimos manejar este archivo como uno solo, es decir se utiliza tanto para la ejecución de un cliente como la de un servidor, diferenciando el comportamiento del programa mediante los argumentos.
Para ejecutar la aplicación con sus valores por defecto para cada parte de la comunicación, efectuamos al menos uno[2] de cada uno[3] de los siguientes comandos:

Servidor

[1] Éstas sugieren el uso de un entorno virtual, pero que no es fundamentalmente necesario.
[2] La aplicación soporta la conexión de múltiples clientes.
[3] De preferencia con un servidor ejecutándose previo a la ejecución de cualquier cliente.

```
[user@host p02]$ java -jar proyecto2.jar
```

Cliente

```
[user@host p02]$ java -jar proyecto2.jar -c
```

4.2. Demostración del funcionamiento

Ya se ha visto cómo realizar la ejecución más sencilla del proyecto (4.1.6). En la presente sección mostramos una posible navegación por el programa y sus resultados, apreciables en capturas de pantalla de la aplicación.

4.2.1. Inicio de sesión

Una vez establecida la conexión entre cliente y servidor, el servidor le pedirá al usuario que ingrese su nombre para poder iniciar sesión o cerrar la conexión entre ambos hosts. Si el nombre de usuario ingresado se encuentra registrado en la base de datos de la aplicación, el servidor le mostrará al usuario con sesión iniciada su menú principal de juego.

4.2.2. Uso dentro de la aplicación

Una vez iniciada la sesión del usuario, éste tendrá acceso a un menú principal en el que se muestre las opciones que le otorga la aplicación, como capturar un pokémon, consultar su pokédex o cerrar sesión.

Si el usuario quiere capturar un pokémon, el servidor le mostrará aleatoriamente alguno de los nombres de pokémon registrados y el usuario decidirá si desea o no capturarlo. Si acepta el desafío, tendrá que lograr capturarlo antes de llegar al límite de intentos permitido.

Para consultar su pokédex, el usuario deberá ingresar el nombre del pokémon que quiera buscar. Si el pokémon ya fue capturado, se le mostrará la imagen de tal pokémon y recibirá un mensaje de error si el nombre del pokémon no es el correcto o el usuario no lo ha capturado.

Si el usuario decide cerrar su sesión, regresará a la pantalla de inicio de la aplicación y se le preguntará si desea iniciar sesión de nuevo o cerrar la conexión entre los hosts.