VNC++, control remoto desde Android

TRABAJO FIN DE GRADO

CURSO 2012-2013

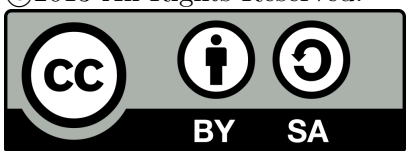
Óscar Crespo Salazar
Gorka Jimeno Garrachón
Luis Valero Martín

Director
Antonio Sarasa Cabezuelo

Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2013

Licencias



Agradecimientos

Agradecemos a Johannew Shindelin por la completa librería para VNC que creó y además distribuirla de forma libre. Y por supuesto por su inestimable y desinteresada ayuda en el Mailing-list de la propia librería, tanto de él como de Christian Beier.

También agradecer a la web stackoverflow `http://stackoverflow.com/`, concretamente la sección de Android, por su certera ayuda con esas inevitables dudas que surgen al desarrollar código desde cero.

También, como no, agradecer a los desarrolladores de Google, que a través de su web `http://developer.android.com/index.html` y ciertos ejemplos clave, han permitido culminar este proyecto.

Mención especial para Axel Amigo, buen amigo nuestro, que nos ayudó poniendo voz al vídeo explicativo del proyecto, a Raúl Crespo, por hacer el montaje que tanto nos ha satisfecho a todos, y por supuesto, a Sergio Barrasa por el fantástico logo realizado para la aplicación.

Por último agradecer a Antonio Sarasa Cabezuelo, director del proyecto, por la confianza depositada en nosotros y su inestimable ayuda durante todo el proceso. También agradecemos a todos los profesores de la Facultad de Informática de la Universidad Complutense de Madrid por proporcionarnos durante todos estos años los conocimientos necesarios para el desarrollo de este proyecto.

Abstract

Nowadays there are a lot of VNC based applications for Android. Nonetheless, none of these applications are Free Software and are implemented using the NDK. The point of this project is to use the NDK potential to make a VNC based application and try to obtain a better performance than the other applications implemented using only the SDK and all the source code will be Free Software.

The protocol to comunicate with a VNC server is called RFB. To implementing this protocol in this project has been used the libVNCServer library. This library facilitates the control of RFB. Without libVNCServer it would be more difficult to make our own library to manage the protocol. JNI is used as the framework to comunicate the native code with Java. The technologies, as well as the process used to built this software is going to be explained along this report.

Keywords

VNC, Android, C++, C, NDK, SDK, JNI, RFB.

Resumen

Hoy en día hay una gran cantidad de aplicaciones basadas en VNC para Android. Sin embargo, ninguna de estas aplicaciones son Free Software y se implementan utilizando el NDK. El objetivo de este proyecto es utilizar el potencial NDK para hacer una aplicación basada en VNC y tratar de obtener un mejor rendimiento que el resto de las aplicaciones implementadas utilizando sólo el SDK y todo el código fuente será Free Software.

El protocolo para comunicarse con un servidor VNC se llama RFB. Para la implementación de este protocolo en este proyecto se ha utilizado la librería libVNCServer. Esta librería facilita el control de RFB. Sin libVNCServer habría sido más difícil hacer nuestra propia librería para gestionar el protocolo. JNI se utiliza como framework para comunicar el código nativo con Java. Las tecnologías, así como el proceso utilizado para construir este software se explicará a lo largo de esta memoria.

Palabras clave
VNC, Android, C++, C, NDK, SDK, JNI, RFB.

Autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

Óscar Crespo Salazar

Gorka Jimeno Garrachón

Luis Valero Martín

Índice

Índice de Figuras

Capítulo 1

Introducción

El desarrollo de las redes de telecomunicaciones ha permitido que progresivamente el control remoto de un ordenador pase de meras terminales de texto a los escritorios remotos de hoy en día. La tecnología de escritorio remoto permite la centralización de aplicaciones que normalmente se ejecutarían en el entorno de usuario. Gracias a esta tecnología el entorno de usuario (cliente) se convierte en una simple terminal de entrada/salida. Todos los eventos de pulsación de teclas y movimientos del ratón se transmiten desde el cliente a un servidor donde la aplicación los procesa como si se tratasen de eventos locales. La imagen de la pantalla se devuelve al cliente y de esta manera se puede controlar desde el terminal cliente la máquina donde se encuentra la aplicación servidor.

Este proyecto se centra únicamente en la realización de una aplicación cliente para Android. De tal manera que la aplicación servidor no está desarrollada en este proyecto. Aunque más adelante se profundizará en cuales son los objetivos concretos de este proyecto, conviene aclarar que el objetivo era desarrollar un cliente VNC para Android, que utilizará el NDK y que fuera de software libre, ya que en la actualidad no existe ninguno que cumpla estos dos requisitos.

Existen diferentes protocolos de comunicación entre la aplicación cliente y la aplicación servidor. Este proyecto se ha desarrollado utilizando el protocolo de comunicación Remote Frame Buffer (RFB), utilizado por todos los clientes y servidores VNC. De esta manera la aplicación será compatible con cualquier servidor VNC instalado en cualquier terminal.

Antes de profundizar más en cómo se han afrontado los objetivos del proyecto conviene establecer ciertos conocimientos básicos relativos al dominio de estas aplicaciones. Comenzaremos explicando algunos conceptos que se han utilizado antes, tales como VNC, RFB, Android, etc.

1.1. Conceptos

1.1.1. Escritorio remoto

Un escritorio remoto es una tecnología que permite a un usuario controlar desde un terminal gráfico (cliente) otro terminal gráfico (servidor). X-Window fue el primer escritorio remoto desarrollado por el Massachusetts Institute of Technology (MIT) en la década de los ochenta [1].

1.1.2. VNC

VNC son las siglas en inglés de Virtual Network Computing [2]. Es un sistema de escritorio remoto que utiliza el protocolo Remote Frame Buffer (RFB) para controlar remotamente otra máquina. Está basado en una arquitectura cliente-servidor, donde la máquina con la aplicación cliente es la que controla a la máquina con la aplicación servidor. VNC transmite los eventos de ratón y teclado desde el cliente al servidor y éste devuelve las actualizaciones de pantalla en la otra dirección a través de una red de comunicaciones.

VNC es independiente del sistema operativo, es decir, un cliente VNC ejecutado en una máquina Windows podría controlar un servidor ejecutado en una máquina Linux. En definitiva, se podría controlar cualquier sistema operativo desde cualquier sistema operativo siempre que ambos sean compatibles con VNC.

1.1.3. RFB

RFB son las siglas en inglés de Remote Frame Buffer. Es un protocolo de comunicación para el acceso remoto a interfaces gráficas de usuario. Como trabaja a nivel de framebuffer es aplicable a todos los sistemas de ventanas (X11, Windows, Macintosh). RFB es el protocolo usado en VNC [3].

1.1.4. Android

Android es un sistema operativo basado en Linux, diseñado principalmente para Smartphones y Tablets. Inicialmente fue desarrollado por Android Inc, pero finalmente Google lo compró en 2005. Fue presentado en 2007 junto con la fundación Open Handset Alliance y el primer móvil con sistema operativo Android salió al mercado en 2008.

La mayoría del código de Android está liberado bajo una licencia Apache, licencia libre y de código abierto. No obstante no está liberado completamente, por lo tanto no se puede decir que Android sea un sistema operativo libre. Uno de sus puntos fuertes es la facilidad para desarrollar aplicaciones gracias a las herramientas que proporciona Google. Estas aplicaciones normalmente están escritas en Java y se desarrollan utilizando el Android SDK, pero también se pueden construir aplicaciones escritas en C/C++ usando el Android NDK [4].

1.1.5. Android SDK

SDK son las siglas en inglés de Software Development Kit. El SDK proporcina las APIs y herramientas de desarrollo necesarias para construir, testear y depurar aplicaciones para Android [5].

1.1.6. Android NDK

NDK son las siglas en inglés de Native Development Kit. El NDK es un conjunto de herramientas que permiten desarrollar parte de la aplicación para Android con lenguajes de código nativo como C o C++ [6].

1.1.7. JNI

JNI son las siglas en inglés de Java Native Interface. Es un framework de programación que permite a código Java llamar y ser llamado desde apliaciones nativas y librerias escritas en otro lenguajes como pueden ser C, C++ o ensamblador [7]. JNI proporciona todo lo necesario para que la comunicación entre el SDK y el NDK sea posible. Gracias a él es posible construir aplicaciones en Android utilizando código nativo.

1.1.8. Tight encoding

Tight es un tipo de codificación más compactada, es muy eficiente y está optimizado para conexiones con poco ancho de banda. Tight incluye tanto la eficiencia de compresión sin pérdidas, como opcionales formas de compresión con perdidas para JPEG [8].

1.1.9. ZRLE encoding

ZRLE, es un tipo de codificación compacta y eficiente. Está optimizado para conexiones con poco ancho de banda y utiliza la librería ZLib para comprimir los datos.

1.1.10. Git

Es un software de control de versiones diseñado por Linux torvals[9][10]. Se centra en la velocidad y la eficiencia. Inicialmente fue construido para el desarrollo del Kernel de Linux pero desde entonces cada vez más proyectos lo han ido adoptando.

1.1.11. WireShark

WireShark es un software analizador de protocolo de red (sniffer) licenciado bajo GPLv2 [11]. Es uno de los sniffer más importantes y completos, en él contribuyen expertos en redes de todo el mundo.

1.1.12. Free Software

Free Software del inglés software libre. Es el software que permite al usuario compartir, estudiar y modificar [12]. Todo sotware libre debe garantizar las libertades de usar el programa con cualquier propósito, estudiar y modificar el programa, distribuir copias del programa y, por último, la libertad de hacer públicas las modificaciones que se hayan realizado.

1.1.13. GPL

GPL son las siglas en inglés de General Public License [13]. Es una licencia de código libre redactada por la Free Software Fundation. El código bajo esta licencia es código libre ya que esta licencia garantiza las cuatro libertades de éste, que son: usar el programa con cualquier propósito, estudiar y modificar el programa, distribuir copias del programa y por último la libertad de hacer públicas las modificaciones que se hayan realizado.

1.2. Motivaciones

La motivación principal para la realización de este proyecto era el desarrollo de algo nuevo. Así, junto con nuestro interés por el desarrollo de aplicaciones Android usando código nativo, nos dimos cuenta de que no existía ninguna aplicación de escritorio remoto que usase esta tecnología. La realización de este proyecto suponía un desafío de cara a la interconexión entre el código en C++ y Java ya que para ello fue necesario aprender a hacer compilaciones cruzadas, a parte de también aprender a utilizar JNI.

1.3. Objetivos

Los objetivos de este proyecto son principalmente dos, el desarrollo de una aplicación de escritorio remoto para Android usando el NDK y que dicha aplicación en su totalidad sea software libre.

El objetivo de realizar la aplicación no era sólo el hecho de construir una aplicación que funcionase correctamente. La aplicación tenía que ser usable y eficiente. Esto último lo hemos logrado, ya que gracias al uso de código nativo se ha conseguido que la aplicación, no sólo sea eficiente si no que tiene niveles de eficiencia comparables con los de la aplicación oficial de los creadores del protocolo. En cuanto a la usabilidad, la aplicación proporciona muchas opciones como, por ejemplo, establecer conexiones cifradas y enviar combinaciones de teclas.

En cuanto al software libre, todo el código ha sido licenciado bajo una licencia GPL versión 3, por lo tanto el software de la aplicación garantiza las cuatro libertades que todo software debe tener para ser considerado como tal.

Consideramos importante y por eso está en esta sección de objetivos, el carácter libre del software, ya que el software libre supone un avance muy importante con respecto al mundo del software privativo, creando una comunidad de usuarios y programadores que interactúan entre sí mejorando el desarrollo del software.

Capítulo 2

Estado del arte

Como referencias actuales para realizar este proyecto, y obtener ideas de aplicaciones enfocadas en el mismo campo, se han utilizado los siguientes clientes para Android:

2.1. VNC Viewer(RealVNC)

VNC Viewer(RealVNC)[14] perteneciente a la compañía RealVNC. Inicialmente VNC surgió en el Olivetti & Oracle Research Lab (ORL3), siendo desarrollado por Tristan Richardson (inventor), Andy Harter (director del proyecto), Quentin Stanfford-Fraser y James Weatherall. Unos años después, varios de estos desarrolladores crearían RealVNC con la idea de seguir con este proyecto software. Al no ser una aplicación de código abierto, es difícil saber el tipo de implementación que lleva a cabo.

Posee una interfaz gráfica muy usable y un rendimiento óptimo. Para el control del ordenador utiliza la pantalla a modo de touchpad, esto es, el movimientos en la pantalla son los movimientos que hará el ratón. Esto proporciona un manejo lento pero extremadamente preciso.

Por otro lado RealVNC no almacena las constraseñas sino que pregunta al usuario por ella cuando conecta a un servidor que la requiera.

Ha sido una de las aplicaciones de referencia, tanto por su interfaz como por su óptimo rendimiento. Como ya se ha dicho con anterioridad es de código cerrado, así que fue imposible trabajar sobre ella.

2.2. Android-vnc-viewer(AndroidVNC)

Android-vnc-viewer(AndroidVNC)[16] es un programa libre y de código abierto de control remoto para dispositivos Android. Es un proyecto desarrollado a partir de una rama inicial de otra aplicación llamada tightVNC. Esta bajo licencia GPLv2 y tiene un repositorio activo en google `http://code.google.com/p/android-vnc-viewer/`.

Posee una interfaz gráfica pobre y una gestión de conexiones poco clara. Para el control remoto del ordenador ofrece una amplio abanico de posibilidades cubriendo cualquier posibilidad al respecto.

En el apartado de la seguridad AndroidVNC almcena las contraseñas en la Base de Datos sin cifrar siendo vulnerable a ataques,

Al ser una aplicación de sofware libre hemos podio ver y aprender mucho de su código y ha sido una de las aplicaciones de referencia ya que ofrece un rendimiento muy bueno. Nos fue imposible trabajar sobre él ya que está construida totalmente sobre SDK y nuestro objetivo es utilizar la potencia del NDK.

2.3. MultiVNC

MultiVNC[15] es un proyecto de software libre bajo licencia GPLv2, basado en android-vnc-viewer y desarrollado por Christian Beier. Incluye soporte para varias codificaciones, así como el uso de Zeroconf (Zero Configuration Networking) para ayudar a crear de forma automática una red IP sin necesidad de configuraciones o servidores especiales.

Al igual que AndroidVNC posee una interfaz pobre y, aunque mejor, una gestión de conexiones no demasiado claro. Para la representación utiliza aceleración por hardware mediante GPU. Respecto al control utiliza un sistema similar a RealVNC.

En el aspecto de las contraseña presenta un funcionamiento semejante a AndroidVNC.

Esta aplicación ha tenido una relevancia menor en el proyecto ya que no nos convencía su rendimiento en móviles de baja gama, se ha usado para observar como resolvían ciertos aspectos y como comparativa de rendimiento.

Capítulo 3

Entorno de desarrollo y banco de pruebas

3.1. Entorno de desarrollo

Todo el proyecto ha sido desarrollado bajo el Sistema operativo Linux Mint Debian Edition 64 bits, bajo las opciones de escritorio tanto Mate como Xfce.

La principal herramienta de desarrollo utilizada ha sido Eclipse Juno, con los entornos de desarrollo integrados tanto para C/C++ como para Java, ya que el proyecto ha necesitado de estos lenguajes.

- JDK: versión 1.6 64 bits.
- SDK: versión android-sdk-r21.1-linux.
- NDK: versión actual android-ndk-r8d, aunque a lo largo del proyecto fue necesario actualizar a esta versión más reciente.
- Android: versión mínima de desarrollo Android 4.0 (API 14). Aunque en un inicio se optó por desarrollar para Android 2.3.3 (API 10) por tener mayor cuota de mercado, nos dimos cuenta al ir desarrollando que la 4.0 nos daría mayores beneficios futuros.
- Git: sistema de control de versiones para el trabajo en equipo.
- Egit: plugin de Eclipse usado para interactuar con el repositorio público del proyecto entre los distintos integrantes.
- Doxygen: plugin que genera la documentación del código final, ya que consideramos más completo que el javadoc de serie.
- ADT: Android Developers Tools versión 22.0.1, plugin de Eclipse para el desarrollo de aplicaciones Android.
- LaTeX: herramienta usada para la composición del texto de la memoria.

Gracias estas herramientas hemos podido trabajar sobre un entorno unificado, ya que eclipse daba todas las posibilidades sin salir de él. Por un lado permitía escribir el código en Java y C/C++ desde la misma herramienta, así mismo, permitía el desarrollo sobre Android gracias al plugin ADT y la gestión del código mediante Git de una forma sencilla gracias a Egit.

3.2. Banco de pruebas

Las pruebas de depuración de la aplicación se han realizado en dos terminales móviles con distintas resoluciones y especificaciones generales para observar el impacto tanto en terminales de gama media/baja, como terminales de alta gama.

- Samsung Galaxy S3 con Android 4.2.2 Jelly Bean.
- Samsung Galaxy Mini 2 con Android 4.1.2 Jelly Bean.

Como servidores VNC se han usado:

- x11vnc, con codificación SSL y sin ella.
- RealVNC, con clave y sin ella.

Capítulo 4

Tecnologías fallidas

En este apartado se describirán algunas de las tecnologías que se pensaron usar en un principio y que por un motivo u otro al final se desecharon.

4.1. Simple DirectMedia Layer

Simple DirectMedia Layer [17][18], o más comúnmente conocida por sus siglas SDL, es una librería desarrollada en C y licenciada y distribuida bajo licencia LGPL, proporciona una serie de funciones para la gestión de dibujos e imágenes en dos dimensiones. Escrita originalmente por Sam Latinga.

Durante el proceso de desarrollo del proyecto, una de las tecnologías por las que se optaron fue SDL, dada las facilidades y potencia que ofrecía para el tratamiento y renderizado de la imagen. Sin embargo, durante la inclusión del mismo en el proyecto surgieron dos problemas que nos decidieron a abandonarlo.

El primer motivo fue que la única versión continuada y con soporte para Android era SDL 2.0, la cual se encontraba en fase de desarrollo y las funciones principales podían cambiar en mitad del desarrollo. Existía una versión no oficial y descontinuada de la misma, que era SDL 1.2, pero entre los errores conocidos estaban el uso del modo vídeo, que era esencial en el proyecto, y dado que estaba descontinuado, nunca se hubiera arreglado.

Por otro lado, el rendimiento obtenido, una vez acoplado al proyecto, fue muy pobre, llegando al caso de que el servidor expulsaba la conexión. Además dado el funcionamiento de SDL, se dibujaba sólo la parte que se veía de la pantalla, la usabilidad al moverse por la misma, teniendo que redibujar continuamente la imagen, hizo inviable su uso.

4.2. SherlockBar

SherlockBar[19] es una librería de código libre para poder usar el patrón de diseño de estilo Holo de Android en versiones anteriores a la aparición de este patrón. En la versión anterior el diseño necesitaba de esta librería para ceñirnos al estilo de las aplicaciones del momento. Como consecuencia a nuestra decisión de portar de la versión 2.3.3 a la 4.0 no fue necesaria la utilización de esta librería.

Durante el desarrollo del proyecto encontramos errores de compatibilidad en el NDK, ya que el funcionamiento de los hilos que eran válidos en uno, no lo eran en el otro.

Debido a la creciente cuota de mercado de la versión 4.0 y la desaparición prematura de terminales 2.3.3, se decidió abandonar el soporte para ésta última. Por ello ésta librería carecía de sentido.

Capítulo 5

Tecnologías usadas

5.1. Librerías de terceros

5.1.1. LibVNCServer

LibVNCServer[20] es una librería multiplataforma de software libre, distribuida bajo la licencia GPL versión 2 o más reciente, desarrollada por Johannes E. Schindelin y escrita en C. Está compuesta en dos partes, por una parte la que está destinada a la creación de un servidor, conocida como LibVNCServer, y por otra, la que está destinada a la creación de un cliente, conocida como LibVNCClient. Para el desarrollo del proyecto se ha usado la parte perteneciente al cliente.

LibVNCServer implementa el protocolo de comunicación RFB (Ver apartado 1.1.3) y añade funcionalidad extra para facilitar el uso del mismo. LibVNCServer se encarga de la codificación y decodificación de datos, así como, el aplanamiento de los mismos para su envío. Implementa protocolos de seguridad para encriptar la comunicación por SSL mediante la librería GNUTLS. Así mismo LibVNCServer utiliza el estándar VNC con lo cual es compatible con la gran mayoría de programas que lo usan.

La librería soporta cualquier codificación de imagen usada por un servidor VNC, entre las que destacan, debido a su gran rendimiento en procesado y bajo coste en envío de datos:

- ZRLE.
- Tight.

Ambas codificaciones utilizan la librería zlib para comprimir los datos de los píxeles en el envío, pero Tight, a diferencia, hace un proceso previo de tratamiento de imagen, de tal forma que el posterior envío de datos es más ligero.

Tight hace un tratamiento de la imagen mediante la librería LibJPEG o, en caso de poderse, LibJPEG-Turbo, siendo esta última una versión avanzada de la primera, de tal forma, que esta preparada para el uso de procesadores vectoriales y su consecuente ganancia de rendimiento.

Así pues, Tight es un codificación preparada para entornos de red pobres ya que tiene un consumo de red bajo. Además en el envío de regiones de la imagen, algo que VNC está constantemente realizando, tiene un rendimiento superior. Sin embargo, Tight puede

tener pérdidas en la calidad de la imagen.

ZRLE aún siendo una codificación preparada para conexiones lentas, es inferior en este punto, sin embargo, al no hacer un tratamiento previo de la imagen no tiene pérdidas en la misma.

La aplicación usa, a diferencia de otras aplicaciones, como codificación máxima Tight, sin embargo, uno de los problemas lo encontramos en no poder usar LibJPEG-turbo. Aunque fue compilada sin problemas para Android, los procesadores móviles actuales carecen de cálculo vectorial, con lo cual no funcionó, con la consiguiente pérdida de rendimiento.

Para el uso de esta librería en Android fue necesaria la compilación de otras librerías que son usadas por LibVNCServer. Para poder compilarlas fue necesario el uso del NDK (Native Development Kit) de Android, ya que todas ellas se encuentran escritas en C/C++. La librerías que fueron compiladas para Android son las siguientes:

- Libnettle.
- Libhogweed.
- Libgmp.
- Libtasn.
- Libgntuls.
- Libgctypt.
- Libjpeg.

Entre las diferentes opciones de compilación que ofrece LibVNCServer, ésta fue compilada con todas ellas, es decir, con LibJPEG necesaria para un óptima codificación de los datos de la imagen, y con GNUTLS para poder encriptar los datos mediante SSL.

5.1.2. SlidingMenu

SlidingMenu[21] es una librería de código abierto de Android, distribuida bajo la licencia Apache versión 2.0, desarrollada por Jeremy Feinstein. Según se cita en el repositorio del autor es una librería que "permite a los desarrolladores crear fácilmente aplicaciones con menús como los que se hizo popular en Google+, YouTube y aplicaciones de Facebook móvil".[21] SlidingMenu se utiliza actualmente en muchas otras aplicaciones Android, como son:

- Foursquare.
- Rdio.
- Evernote Food.
- Plume.
- VLC for Android.
- Mantano Reader.

Todas estas aplicaciones marcan el diseño y entorno de uso presente y futuro en Android, y éste nuevo menú es el que se está implantando.

Se decidió la utilización de esta librería ya que el menú desplegable clásico dejará de tener validez en los nuevos terminales móviles y versiones Android. Además es fácil de integrar en proyectos Android ya existentes.

En el uso de esta librería, se han modificado una serie de parámetros que se consideran útiles para la aplicación.

El menú se despliega desde la izquierda dentro del dispositivo móvil cuando ya se controla la imagen que provee cierto servidor. El margen izquierdo sobre el que se produce el evento de despliegue se tuvo que reducir, ya que quitaba opciones de manejo sobre la imagen en el borde más izquierdo del terminal. Para realizar este cambio fue necesario modificar una constante que marcaba el margen, en la clase CustomViewBehind de ésta librería, concretamente reducir el valor de dips.

La librería permite también la personalización a la hora de realizar el desplegado, manejando el grado de visibilidad y sombreado según el nivel de despliegue, así como los márgenes laterales que deja el menú una vez extendido.

En este menú se han introducido todas las opciones de control y envío de eventos a la imagen.

5.1.3. SQLite Android

SQLite[22] es un motor de bases de datos muy popular en la actualidad por ofrecer características tan interesantes como: su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y por ser de código libre. Éstas son unas de las grandes razones por las que Android usa esta librería software para almacenar los datos.

En este proyecto se da uso a esta tecnología para guardar las conexiones creadas por el usuario. La estructura elegida ha sido muy sencilla, para facilitar la rápida recuperación de datos. La base de datos se crea y se da uso en la clase ConnectionSQLite, encargada de todo el control persistente. En dicha clase se crea la base de datos DBConnections con una única tabla para almacenar las conexiones denominada Connections cuyos campos son:

- **NameConnection:** campo en el que se almacena el nombre de la conexión, que sirve como identificador único a la hora de modificar y realizar consultas.
- **IP:** almacena la IP de la conexión establecida.
- **Port:** puerto de la conexión (por defecto 5900).
- **Psw:** almacena la contraseña si el usuario ha optado por esta opción (por defecto la contraseña es vacía). Por razones técnicas ha sido necesario almacenarla sin ningún tipo de encriptación, ya que es necesario comparar con la clave del servidor. En una versión anterior, usábamos una reducción criptográfica de tipo MD5, pero al no poder comparar la clave del servidor con esta codificación, fue necesario quitarla, y almacenar en plano. Esta opción no nos convencía, pero era la única forma de pasar la clave al servidor.

- **Fav:** indica si la conexión es de tipo favorito o no. En la base de datos las variables de tipo booleanas (true/false) se almacenan como 0 (false) ó 1 (true). Según el valor de este campo, la conexión deberá aparecer en la pestañas de favoritos o no.
- **ColorFormat:** almacena la calidad de color establecida en la conexión. Las opciones son:
  - **1.** super-alta.
  - **2.** alta.
  - **3.** media.
  - **4.** baja.

5.2. Aplicación

5.2.1. Interfaz Holo

Debido a la multiplicidad de diseños dentro de las aplicaciones en Android, con esta interfaz se ha intentado marcar un estándar que sirva como diseño de control y manejo de las nuevas aplicaciones a partir de la 4.0. De esta manera, se facilita al usuario la interacción con todas las aplicaciones ya que comparten el formato de manejo y estilo, compartiendo iconos y las funcionalidades que se entienden de estas pequeñas imágenes. Por esta razón en el proyecto se incluyen los iconos y estándares de diseño marcados por Google, usando los iconos con las distintas resoluciones provistas en Android Developers [23].

5.2.2. JNI-NDK

Native Development Kit (NDK), de Android es el SDK mediante el cual se pueden desarrollar aplicaciones nativas para Android, esto es, aplicaciones escritas en C/C++ y compiladas a través de GCC.

Por otro lado JNI, es una interfaz proporcionada por el NDK para la interacción de la parte Java con C/C++.

El proyecto se desarrolla en dos partes. Por un lado la parte Java (Ver sección 5.2.1), que es la delegada de la interfaz gráfica, GUI (Graphics User Interface), y por otra la parte nativa, desarrollada en C++ (Ver sección 5.2.2) y encargada de la conexión, así como tratamiento de píxeles.

El lugar que nos ocupa ahora es el de la parte nativa, también llamada, parte C++. Para la correcta concurrencia entre Java y C/C++, el proyecto ha sido desarrollado de tal forma que cuando se hace la petición de conexión con el servidor, cuya petición es hecha por el usuario desde Java, la aplicación se separa en 2 hilos altamente diferenciados y cuya sincronización es fundamental.

Por un lado encontramos el hilo perteneciente a la GUI de Android, el cual se encargará de interactuar con el usuario. Por otro lado, encontramos el hilo de C++, el cual es creado al iniciarse la conexión y que será el encargado de atender las peticiones hechas por el servidor.

En este punto la aplicación se encuentra bajo unas limitaciones importantes:

- **El uso de memoria:** La memoria que es capaz de reservar y manejar C/C++ dentro de la máquina virtual de Dalvik (JVM) es limitado, es decir, si la parte nativa quisiese crear un vector de grandes dimensiones en Java no podría y para ello tendría que cambiar de hilo y situarse en Java.
- **Rendimiento:** Las invocaciones de funciones de la parte nativa a Java no gozan de una gran velocidad, por ello si la parte nativa quisiese actualizar una gran densidad de información, la forma de actuar sería acceder, mediante JNI, directamente a los datos de la JVM y modificarlo, sin que Java intervenga.

  También encontramos el problema de una velocidad dispar en la ejecución, es decir, la parte nativa se ejecuta a una velocidad y la parte Java a otra, siendo ésta última significativamente inferior. Por ello la parte nativa debe esperar a que Java haya realizado sus procesos antes de proseguir.
- **Concurrencia:** Las invocaciones a Java que son realizadas desde C++ se hacen en un hilo diferente de invocación, y que es creado en cada invocación independiente al resto. Éstas interrumpen el funcionamiento lineal de Java, como por ejemplo, en la notificación de un cambio en la pantalla.

La parte nativa también es la delegada de realizar una de las tareas más críticas de la aplicación, el tratamiento de píxeles. Cuando el servidor envía la notificación de que hay cambios en la pantalla, la aplicación recoge esa información y la debe trasladar a Java, que es la que se encargará de representarlo.

La información de la imagen, que se almacena en el framebuffer de RFB, se recoge como un puntero al tipo uint8_t (uint8_t*), y la parte nativa se encarga de procesarla, ya que, como veremos a continuación, es un proceso pesado y C++ proporciona un rendimiento más óptimo.

Para explicar el proceso hay dos cosas a tener en cuenta:

- **Tamaño del vector:** El tamaño del vector, el cual es bastante grande ya que se calcula con la fórmula:
  Altura*Anchura*BytesPerPixel (bytes por píxel o su abreviación bpp).
  Suponiendo un caso real como podría ser un equipo con una pantalla FullHD (1920x1080), y máxima calidad de imagen, que es 4 bpp sería:

  1920*1080*4 = 8294400.

  Nos encontraríamos con un vector de 8294400 posiciones.

- **Recorrido del vector:** Aunque la representación de la información es un vector lineal, en realidad el recorrido del mismo se realiza como si de un vector bidimensional se tratase. De tal forma que para pasar de una fila a otra se haría:

  Anchura*bpp

  En cualquier caso, nos encontraríamos con un recorrido del orden $O(n^2)$ (coste cuadrático).

Para recoger la información de la imagen, se recorre el rectángulo que ha sufrido cambios y se copia directamente en el vector final. Para hacer la copia se usa la función de C *memcpy*, dada su eficiencia copiando bloques de bytes.

5.2.3. Canvas para la representación

Para la representación de la imagen se ha usado Canvas para Android, usando sus funciones de dibujado de Bitmap. Los dos métodos de Canvas usados para dibujar el bitmap son:

```
drawBitmap(int[] colors, int offset, int stride,
           int x, int y, int width, int height,
           boolean hasAlpha, Paint paint)
```

Mediante este método es posible dibujar un bitmap a partir de su información de píxeles, almacenada en colors, se usa para representar los datos de la imagen según se modifican.

```
drawBitmap(Bitmap bitmap, Rect src, RectF dst, Paint paint)
```

Mediante éste dibujamos un bitmap, pasándole el bitmap ya creado como parámetro.

Cada uno de los métodos anteriores es usado en dos situaciones y contexto distintos, es necesario usar ambos, como veremos más adelante, para una óptima representación. Los dos contextos distintos a los que hace frente la aplicación son:

- **Estado normal:** sucede siempre que hay una actualización en la imagen y siempre y cuando el usuario no se esté desplazando por la pantalla o haciendo zoom. En este caso la imagen se representará mediante su vector de píxeles.

- **El usuario se mueve por la imagen o hace zoom:** cuando el usuario se desplaza a través de la imagen o hace zoom, es necesario crear un bitmap con la información actual de la imagen, ya que en caso contrario, obtendríamos un pésimo rendimiento por dos motivos:

  - Si usásemos la representación a partir de su información de píxeles, en cada progresión por la imagen haría falta redibujar la imagen, consiguiendo un movimiento o zoom lento.
  - Como estaríamos continuamente redibujando la imagen, y además, en cada representación Android crearía un bitmap temporal, provocaríamos un alto consumo de CPU.

De tal forma, al crear un bitmap, el movimiento por la imagen es fluido. El único punto a considerar es la optimización de cuando crear y destruir este bitmap, ya que la creación de bitmaps es un proceso pesado y que requiere de muchos recursos.

Así pues los puntos en los que se destruye o se marca para destruir el bitmap son:

- Cuando el usuario haga click sobre la imagen, en este punto se considera que el usuario intenta interactuar con el servidor, por lo tanto, es necesario destruir el bitmap y mantener actualizada la imagen en todo momento.
- En el caso de que el usuario deje de moverse por la pantalla, al menos, durante medio segundo o deje de hacer zoom. En este caso es conveniente destruir la imagen en pos de conseguir una imagen actualizada del servidor. El motivo por el cual se espera medio segundo, es con el fin de no interrumpir el movimiento del usuario, es decir, si se destruyese el bitmap en el momento que levanta el dedo de la pantalla, al volver a moverse se produciría una lentitud hasta que se volviese a crear el nuevo bitmap. Además así, optimizamos la creación de bitmap y reducimos el uso de CPU.

El problema que encontramos frente a otras soluciones, a la hora de actualizar la imagen, es el hecho de que la parte nativa (C++) actualiza los datos de la imagen directamente en el vector, si por el contrario, todo se hubiese desarrollado en Java, se podría haber actualizado la información sobre un bitmap ya creado, de tal forma que se ahorraría el consumo de CPU derivado de la creación, aunque se perdería rendimiento en la actualización.

Capítulo 6

Especificaciones

En este capítulo se describe la construcción de la aplicación. Primero se expondrán las especificaciones de requisitos tanto externos, como funcionales. Después en el siguiente apartado se explicarán los diagramas UML del código, junto con explicaciones de todo el flujo del programa.

6.1. Especificación de requisitos

Los requisitos de la aplicación son:

6.1.1. Interfaces externos

En esta sección se indican los requisitos externos a la aplicación.

Interfaces de usuario:

- La interfaz de uso proporciona todo lo necesario para controlar la aplicación en su totalidad.

Interfaces de hardware:

- Se debe disponer de un dispositivo móvil y un ordenador.

Interfaces de software:

- La versión de Android del dispositivo móvil debe ser la 4.0 o superior.
- En la máquina remota se debe disponer de una aplicación VNC de tipo servidor.

Interfaces de comunicación:

- Tanto el cliente como el servidor deben disponer de una conexión a Internet o estar ambos en la misma red local.

6.1.2. Funciones

Los siguientes requisitos funcionales definen las acciones que deben tener lugar en el software procesando las entradas y generando las salidas.

Crear conexión:

- Prioridad: Alta.
- Estabilidad: Alta.
- Descripción: Crea una nueva conexión y la añade a la lista de conexiones.
- Entrada: Nombre de la conexión, IP, puerto, contraseña y calidad de imagen.
- Salida: La conexión se añade a la lista y se conecta.
- Origen: Usuario.
- Destino: Aplicación.
- Necesita: Acceso a la Base de Datos.
- Acción: Crea una nueva conexión.
- Precondición: Nombre de conexión no repetida, IP válida y puerto en un rango válido.
- Postcondición: La conexión se añade a la lista de conexiones.
- Efectos laterales: N/A.

Editar conexión:

- Prioridad: Media.
- Estabilidad: Alta.
- Descripción: Se editan los datos de una determinada conexión.
- Entrada: Nombre de la conexión, IP, puerto, contraseña y calidad de imagen.
- Salida: La conexión con los datos modificados.
- Origen: Usuario.
- Destino: Aplicación.
- Necesita: Acceso a la Base de Datos.
- Acción: Edita los datos de una conexión.
- Precondición: IP válida y puerto en un rango válido.
- Postcondición: Los datos de la conexión se modifican.
- Efectos laterales: N/A.

Borrar conexión:

- Prioridad: Media.
- Estabilidad: Alta.
- Descripción: Se elimina una conexión.
- Entrada: Evento usuario.
- Salida: La conexión se elimina de la aplicación.
- Origen: Usuario.
- Destino: Aplicación.
- Necesita: Acceso a la Base de Datos.
- Acción: Elimina una conexión.
- Precondición: N/A.
- Postcondición: La conexión se borra de la aplicación.
- Efectos laterales: Si la conexión está en favoritos se borrará también ahí.

Ver información de la conexión:

- Prioridad: Baja.
- Estabilidad: Alta.
- Descripción: Muestra la información de una conexión.
- Entrada: Evento usuario.
- Salida: Los datos de la conexión.
- Origen: Usuario.
- Destino: Usuario.
- Necesita: Acceso a la Base de Datos.
- Acción: Muestra los datos de una conexión.
- Precondición: N/A
- Postcondición: Se muestran los datos de una conexión.
- Efectos laterales: N/A.

Añadir conexión a favoritos:

- Prioridad: Baja.
- Estabilidad: Alta.
- Descripción: Añade una conexión a la lista de favoritos del usuario.
- Entrada: Evento usuario.
- Salida: La conexión se añade a la lista de favoritos.
- Origen: Usuario.
- Destino: Aplicación.
- Necesita: Acceso a la base de datos.
- Acción: Añade la conexión a la lista de favoritos.
- Precondición: N/A.
- Postcondición: La conexión se añade a favoritos.
- Efectos laterales: N/A.

Borrar conexión de favoritos:

- Prioridad: Baja.
- Estabilidad: Alta.
- Descripción: Borra una conexión de la lista de favoritos.
- Entrada: Evento usuario.
- Salida: La conexión se elimina de favoritos.
- Origen: Usuario.
- Destino: Aplicación.
- Necesita: Acceso a la base de datos.
- Acción: Elimina una conexión de la lista de favoritos.
- Precondición: N/A.
- Postcondición: La conexión se borra de favoritos.
- Efectos laterales: N/A.

Ver lista de conexiones:

- Prioridad: Alta.
- Estabilidad: Alta.
- Descripción: Se muestran todas las conexiones recientes.
- Entrada: Evento usuario.
- Salida: La lista de conexiones recientes.
- Origen: Usuario.
- Destino: Usuario.
- Necesita: Acceso a la base de datos.
- Acción: Se muestran las conexiones recientes.
- Precondición: N/A
- Postcondición: Se muestra la lista de conexiones recientes.
- Efectos laterales: N/A.

Ver lista de conexiones favoritas:

- Prioridad: Baja.
- Estabilidad: Alta.
- Descripción: Se muestran todas las conexiones favoritas.
- Entrada: Evento usuario.
- Salida: La lista de conexiones favoritas.
- Origen: Usuario.
- Destino: Usuario.
- Necesita: Acceso a la base de datos.
- Acción: Se muestran las conexiones favoritas.
- Precondición: N/A
- Postcondición: Se muestra la lista de conexiones favoritas.
- Efectos laterales: N/A.

Abrir menú principal:

- Prioridad: Baja.
- Estabilidad: Alta.
- Descripción: Se abre el menú principal.
- Entrada: Evento usuario.
- Salida: Se muestra el menú.
- Origen: Usuario.
- Destino: Usuario.
- Necesita: N/A.
- Acción: Se muestra el menú principal.
- Precondición: N/A
- Postcondición: Se muestra el menú principal.
- Efectos laterales: N/A

Conectarse:

- Prioridad: Alta.
- Estabilidad: Alta.
- Descripción: Se conecta al servidor especificado en los datos de la conexión.
- Entrada: Datos de la conexión y evento usuario.
- Salida: Conexión con el servidor.
- Origen: Usuario.
- Destino: Usuario.
- Necesita: N/A.
- Acción: Se conecta con el servidor remoto.
- Precondición: El servidor tiene que tener ejecutándose una aplicación VNC.
- Postcondición: El cliente se conecta con el servidor.
- Efectos laterales: N/A.

Desconectarse:

- Prioridad: Alta.
- Estabilidad: Alta.
- Descripción: Se desconecta del servidor al que se estaba conectado.
- Entrada: Evento usuario o desconexión del servidor
- Salida: Desconexión con el servidor.
- Origen: Usuario o servidor.
- Destino: Usuario.
- Necesita: Estar conectado.
- Acción: Se desconecta del servidor remoto.
- Precondición: Se debe estar conectado para poder desconectarse.
- Postcondición: El cliente se desconecta del servidor.
- Efectos laterales: N/A.

Hacer click:

- Prioridad: Alta.
- Estabilidad: Alta.
- Descripción: Se envía la orden de pulsar el botón izquierdo del ratón.
- Entrada: Evento usuario.
- Salida: N/A.
- Origen: Usuario.
- Destino: Aplicación.
- Necesita: Estar conectado.
- Acción: Se envía la orden de hacer click.
- Precondición: Se debe estar conectado.
- Postcondición: Se envía la orden de hacer click.
- Efectos laterales: N/A.

Pulsar botón derecho:

- Prioridad: Alta.
- Estabilidad: Alta.
- Descripción: Se envía la orden de pulsar el botón derecho del ratón.
- Entrada: Evento usuario.
- Salida: N/A.
- Origen: Usuario.
- Destino: Aplicación.
- Necesita: Estar conectado.
- Acción: Se envía la orden de hacer click en el botón derecho.
- Precondición: Se debe estar conectado.
- Postcondición: Se envía la orden de hacer click en el botón derecho.
- Efectos laterales: N/A.

Moverse por el escritorio:

- Prioridad: Alta.
- Estabilidad: Alta.
- Descripción: Movimientos por la imagen del escritorio del servidor.
- Entrada: Evento usuario.
- Salida: Cambio de la imagen.
- Origen: Usuario.
- Destino: Usuario.
- Necesita: Estar conectado.
- Acción: Se muestran diferentes partes del escritorio según el movimiento.
- Precondición: Se debe estar conectado.
- Postcondición: Se muestra una nueva imagen del escritorio.
- Efectos laterales: N/A.

Abrir menú de conexión:

- Prioridad: Alta.
- Estabilidad: Alta.
- Descripción: Se abre el menú de opciones.
- Entrada: Evento usuario.
- Salida: Se muestra el menú.
- Origen: Usuario.
- Destino: Usuario.
- Necesita: Estar conectado.
- Acción: Se muestra el menú de conexión.
- Precondición: Se debe estar conectado.
- Postcondición: Se muestra el menú de conexión.
- Efectos laterales: Se oculta una parte de la pantalla.

Abrir teclado:

- Prioridad: Media.
- Estabilidad: Alta.
- Descripción: Se muestra el teclado para poder enviar teclas.
- Entrada: Evento usuario.
- Salida: Se muestra el teclado en pantalla.
- Origen: Usuario.
- Destino: Usuario.
- Necesita: Estar conectado.
- Acción: Se muestra el teclado.
- Precondición: Se debe estar conectado.
- Postcondición: Se muestra el teclado en pantalla.
- Efectos laterales: Se oculta una parte de la pantalla.

Enviar pulsaciones de teclas:

- Prioridad: Media.
- Estabilidad: Alta.
- Descripción: Se envía pulsacion de tecla.
- Entrada: Una pulsación de tecla.
- Salida: N/A.
- Origen: Usuario.
- Destino: Aplicación.
- Necesita: Estar conectado.
- Acción: Se envía la orden de pulsar una determinada tecla.
- Precondición: Se debe estar conectado.
- Postcondición: Se envía la orden de pulsar una tecla.
- Efectos laterales: N/A.

Enviar combinaciones de teclas:

- Prioridad: Media.
- Estabilidad: Alta.
- Descripción: Se envían combinaciones de teclas.
- Entrada: Una pulsación de una combinación de teclas.
- Salida: N/A.
- Origen: Usuario.
- Destino: Aplicación.
- Necesita: Estar conectado.
- Acción: Se envía la orden de pulsar una combinación de teclas.
- Precondición: Se debe estar conectado.
- Postcondición: Se envía la orden de pulsar una combinación de teclas.
- Efectos laterales: N/A.

Centrar la imagen:

- Prioridad: Baja.
- Estabilidad: Alta.
- Descripción: Se centra la imagen en la esquina superior izquierda del escritorio remoto y se deshace el zoom.
- Entrada: Evento usuario.
- Salida: Se mueve la imagen.
- Origen: Usuario.
- Destino: Usuario.
- Necesita: Estar conectado.
- Acción: Se centra la imagen.
- Precondición: Se debe estar conectado.
- Postcondición: Se centra la imagen en la esquina superior izquierda.
- Efectos laterales: N/A.

Hacer zoom en la imagen:

- Prioridad: Alta.
- Estabilidad: Alta.
- Descripción: Se hace zoom en una determinada zona del escritorio.
- Entrada: Evento usario,
- Salida: Se hace zoom en la imagen.
- Origen: Usuario.
- Destino: Usuario.
- Necesita: Estar conectado.
- Acción: Se hace zoom en la imagen del escritorio.
- Precondición: Se debe estar conectado.
- Postcondición: Se hace zoom.
- Efectos laterales: N/A.

6.2. Diagramas de clases

Esta sección se centra en la explicación de los diagramas UML y en la del funcionamiento del código. La sección se divide en otras dos subsecciones, una para explicar la parte escrita en Java y otra para explicar la parte escrita en C/C++.

6.2.1. Parte escrita en Java

Primero describiremos las Activities principales del programa. ActivityTabs (Ver Figura 6.2) es la Activity principal del programa. Esta Activity es la encargada de llamar tanto a las Activities de configuración, creación y edición de conexión, como a la Activity encargada de comunicarse establecer y atender la conexión.

NewConnectionActivity (Ver Figura 6.2) es la encargada de crear una nueva conexión. Es llamada desde ActivityTabs. Una vez rellenados los campos se comprueba que sean correctos, se guardan en la base de datos y se realiza la conexión llamando a CanvasActivity.

EditionActivity (Ver Figura 6.2) es la encargada de modificar los datos de una conexión. También es llamada desde ActivityTabs y una vez rellenados los campos comprueba que sean correctos y actualiza la conexión en la base de datos.

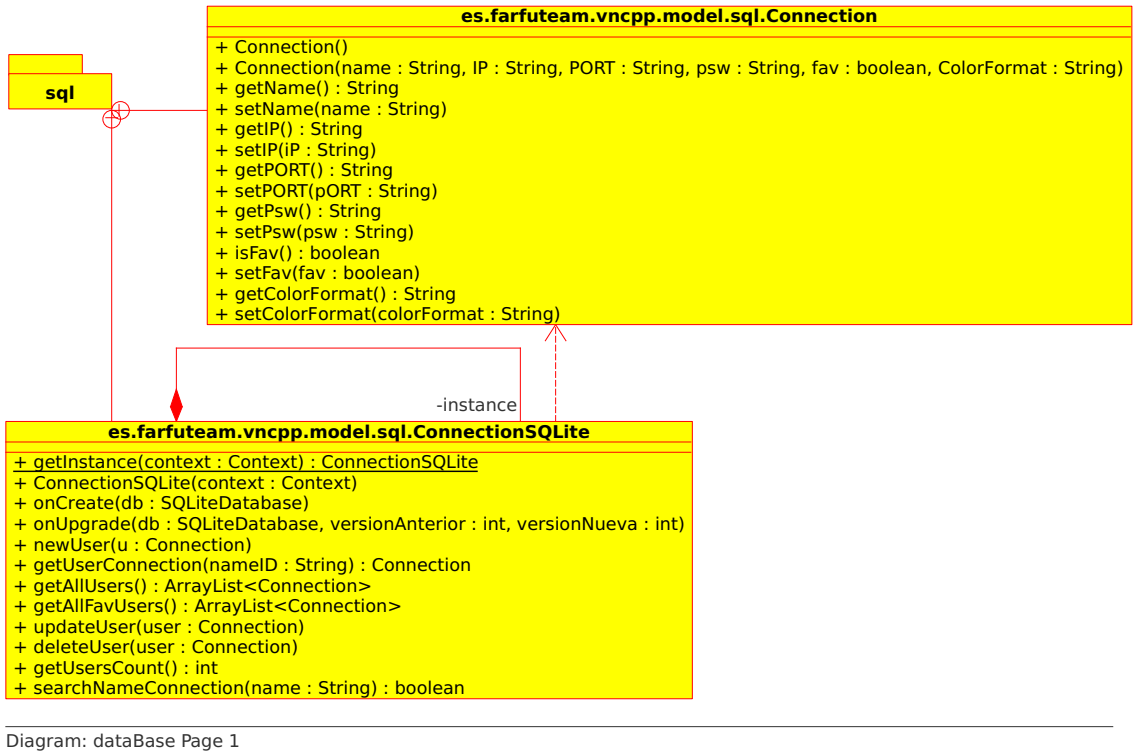


Figura 6.1: Diagrama Base de Datos

La clase encargada de acceder a la base de datos (SQLite) es ConnectionSQLite (Ver Figura 6.1). Esta es llamada desde diferentes Activities como NewConnectionActivity y EditionActivity. La clase Connection se utiliza como clase transfer con la información de una conexión.

ConfigurationMenu crea el menú de configuraciones. Como las clases anteriores es también llamada desde ActivityTabs. Si el usuario cambia alguna configuración, ConfigurationMenu llama al singleton de Configuration y le pasa los valores que se hayan modificado.

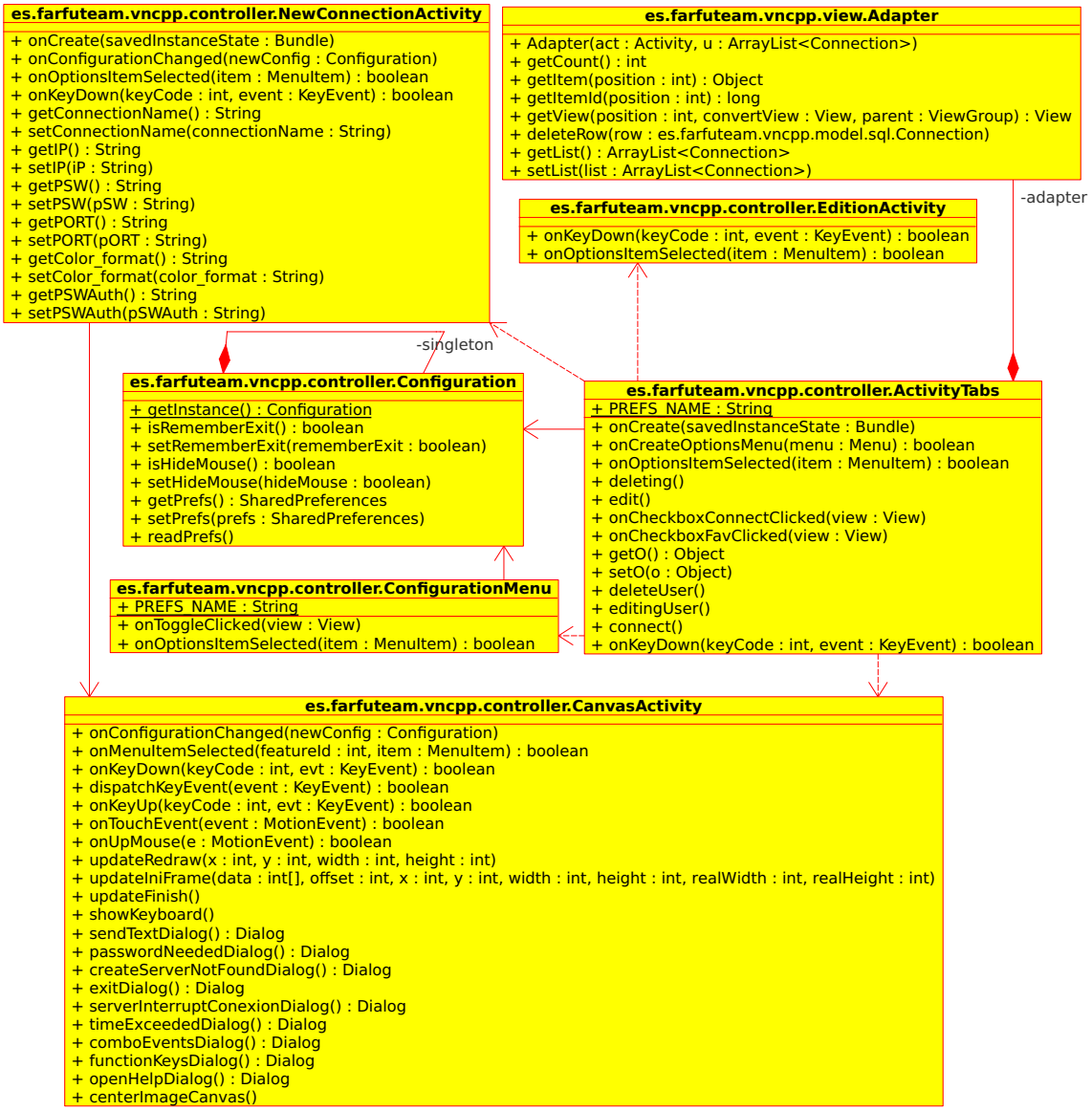


Figura 6.2: Diagrama Activity Principal

Configuration es un singleton que se encarga de acceder al fichero de preferencias. ActivityTabs en el método onCreate llama a esta clase para que se instancie y cargue los datos de este fichero. Luego Configuration podrá ser llamado desde ConfigurationMenu si el usuario decide cambiar algo en la configuración de la aplicación.

Adapter es una clase que sirve para configurar cada elemento de la lista de conexiones y la lista de favoritos.

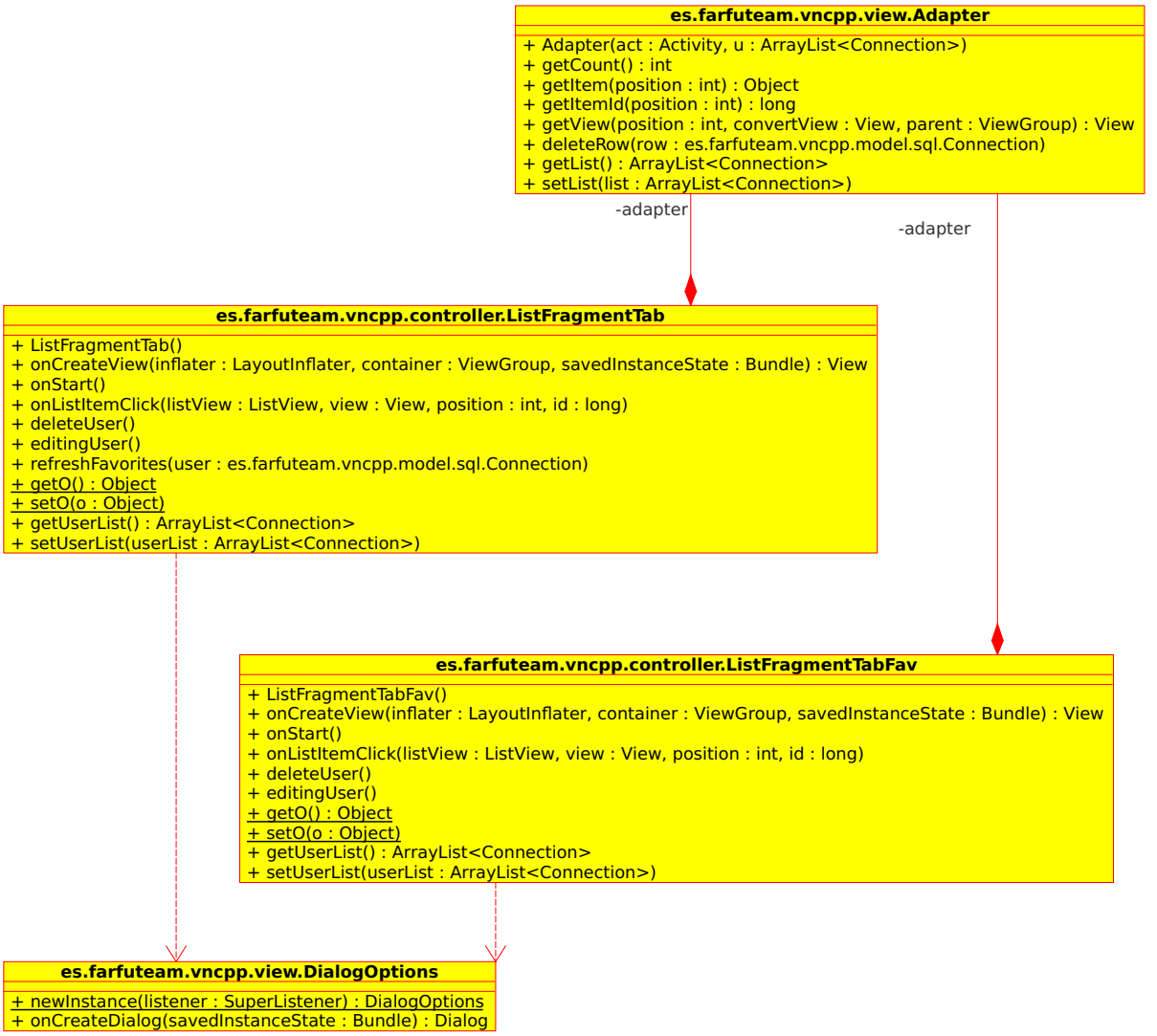


Figura 6.3: Diagrama Listas de Conexiones

Las clases ListFragmentTab y ListFragmentTabFav (Ver Figura 6.3) son las encargadas de mostrar las listas de conexiones y conexiones favoritas respectivamente.

La clase SlideListFragment (Ver Figura 6.4) se encarga de construir el menú lateral y es llamada desde CanvasActivity.

CanvasActivity (Ver Figura 6.4) es la encargada de capturar todos los eventos que se produzcan en el cliente, mostrar la imagen del escritorio y comunicarse con la parte nativa. Para dibujar la imagen se apoya en la clase CanvasView (Ver Figura 6.4). Para Comunicarse con la parte nativa se apoya en VncBridgeJNI (Ver imagen 6.5). La comunicación entre el VncBridgeJNI y el Canvas Activity se realiza mediante el patrón observer (Ver figura 6.4 y 6.5).

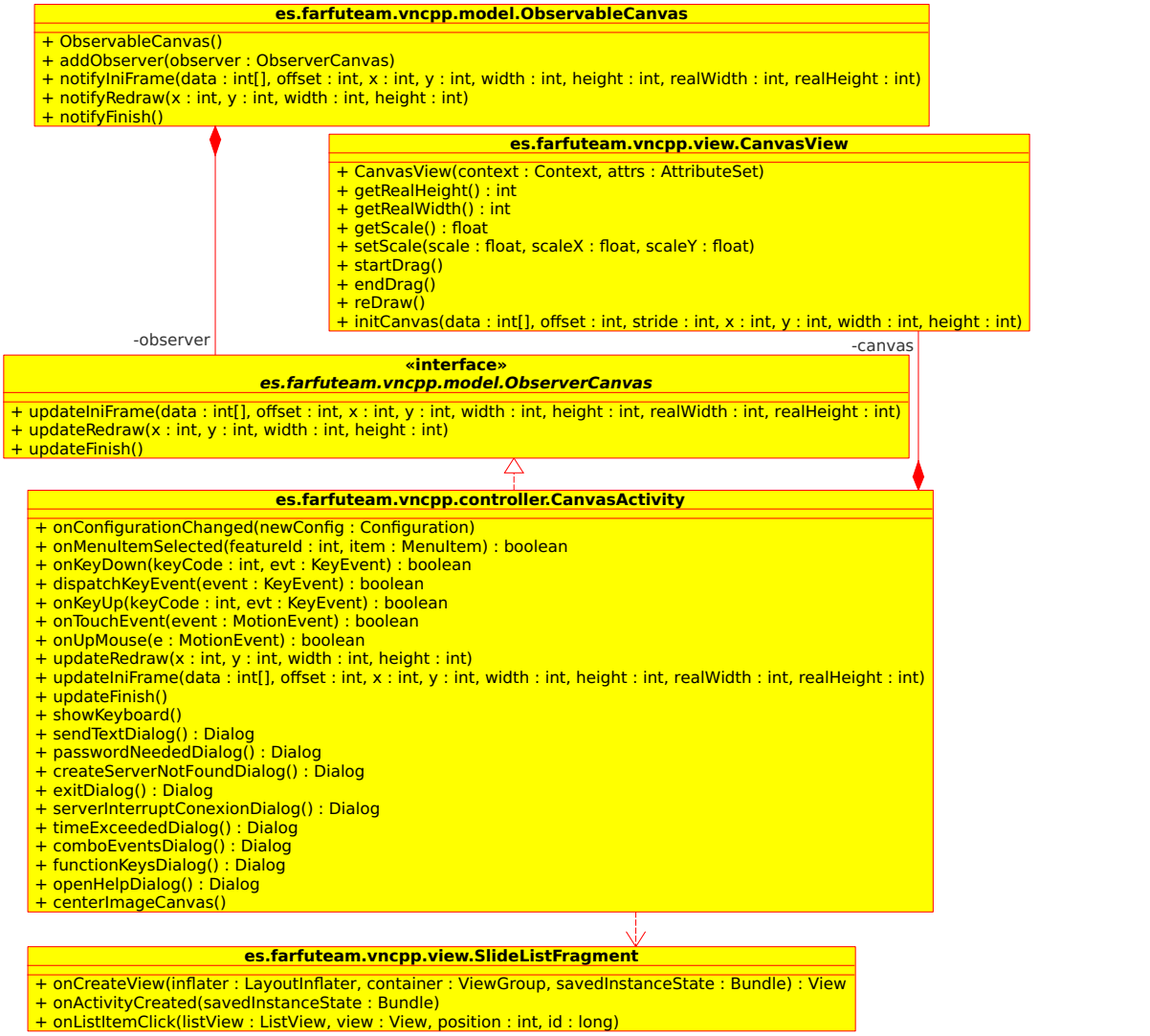


Figura 6.4: Diagrama Canvas

La representación de la imagen se ha realizado con canvas, en la clase CanvasView (Ver Figura 6.4). Para dar mejor sensación de fluidez a la hora de moverse por la imagen se utiliza un Bitmap auxiliar en lugar de siempre el mismo. Cuando el programa entra en modo drag, se crea el Bitmap auxiliar y se inicializa con el valor actual de la imagen. Si el usuario está moviendose por la imagen (modo drag) el método *onDraw* de canvas mostrará el Bitmap auxiliar.

```
if(!drag){
   canvas.drawBitmap(data,offset,stride,x,y,width,height,false,null);
}
else{
   canvas.drawBitmap(bitmap, bitmapRect, bitmapRect,null);
}
```

En el momento que el usuario deja de moverse salta un timer creado en un hilo aparte, de tal manera que durante 0.5 segundos la imagen no se destruye (Ver sección 5.2.3).

La actualización de la imagen nos llega desde el código nativo (Ver sección 6.2.2). Cuando llega una notificación desde C++ de que la imagen ha sido modificada, el canvas se invalida y se redibuja.

Si CanvasActivity detecta un evento de Scroll (movimiento por la pantalla), se llama al método *doScroll*. Este método calcula cuantos píxeles se ha movido en los ejes X e Y, comprobando que no se salga de la imagen. Una vez calculados se llama al método de canvas *scrollTo* para que modifique la imagen.

```
if(moveX != 0 || moveY != 0){
  realX = realX + (int)moveX;
  realY = realY + (int)moveY
  canvas.scrollTo((int)(realX*scaleFactor),
                            (int)(realY*scaleFactor));
}
```

Si CanvasActivity detecta un evento de Scale (zoom), se recalcula el factor de escala y dependiendo de si ha sido positivo o negativo se llama a, el método *scrollTo* de canvas directamente o a el método *doScroll*.

```
if(isDo){
  if(factor > 1){
    int moveX = (int)(x*scaleFactor);
    int moveY = (int)(y*scaleFactor);
    if(x < 0){
      moveX = 0;
      y = 0;
    }
    if(y < 0){
      moveY = 0;
      y = 0;
    }
    realX = (int)x;
    realY = (int)y;
    canvas.scrollTo(moveX,moveY);
  }else if(factor<1 ){
    int auxX = 0;
    int auxY = 0;
    if(x * scaleFactor > canvas.getRealWidth()){
      auxX = -100;
    }else{
      auxX = -10;
    }
    if(y * scaleFactor > canvas.getRealHeight()){
      auxY = -100;
    }else{
      auxY = -10;
    }
```

```
        doScroll(auxX, auxY);
    }
}
```

VncBridgeJNI es la clase que proporcina los métodos para comunicarse con la parte nativa. Es capaz de llamar a funciones nativas tales como iniConnect, mouseEvent, keyEvent, etc., de esta manera se establece la comunicación de Java a C/C++ (Ver sección 5.2.2). Las notificaciones llegan desde el código nativo a través de la interfaz ObserverJNI.

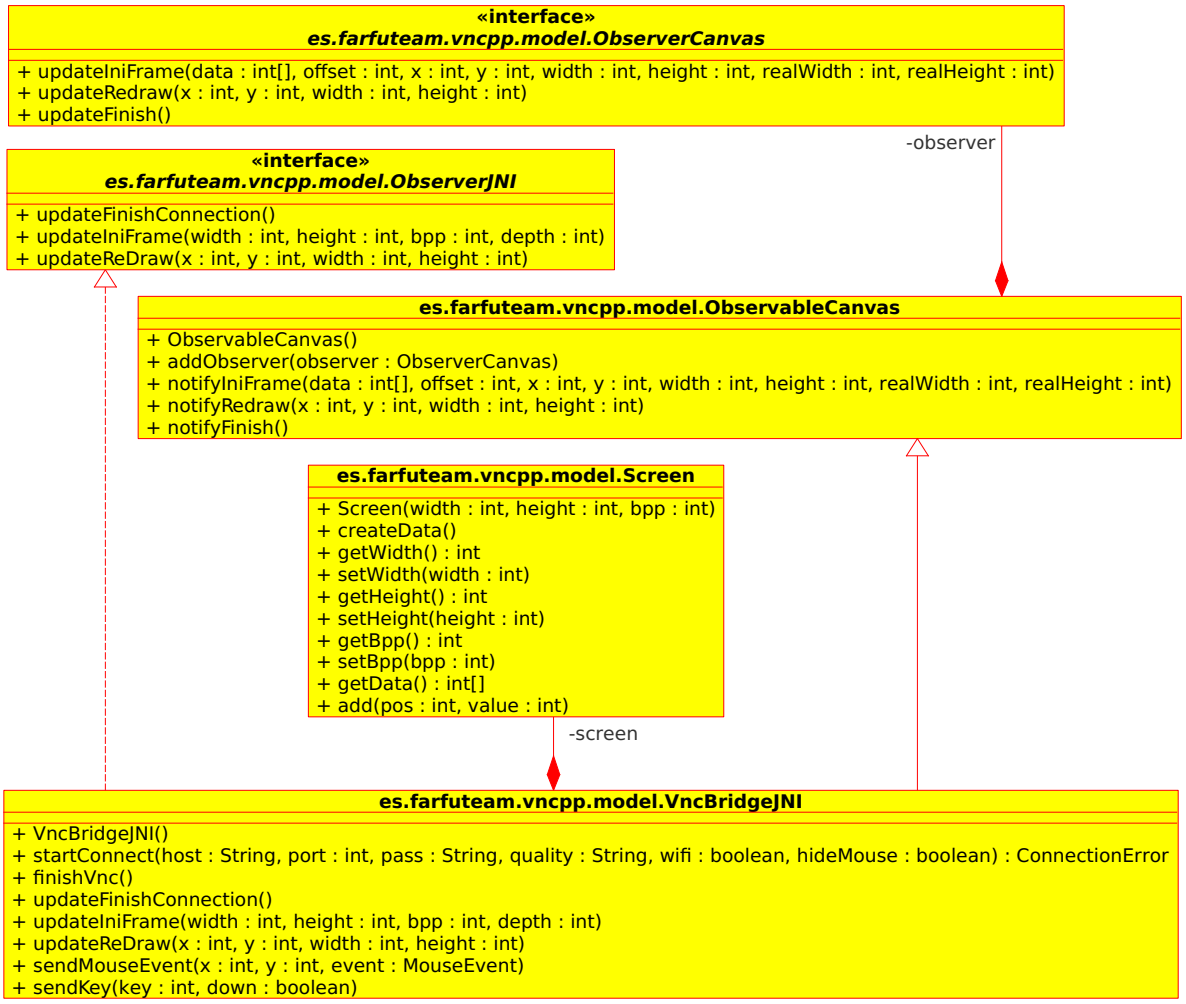


Figura 6.5: Diagrama Modelo

6.2.2. Parte escrita en C/C++

Ahora vamos a describir el código escrito en C/C++. La clase Java VncBridgeJNI realiza las llamadas a la parte nativa a través de las funciones implementadas en el fichero JavaBridge.cpp. Este fichero no se muestra en el diagrama de clases puesto que no es una clase, aún así, su única función es hacer las llamadas propias a la clase Vnc, en otras palabras actúa como puente de Java a C++.

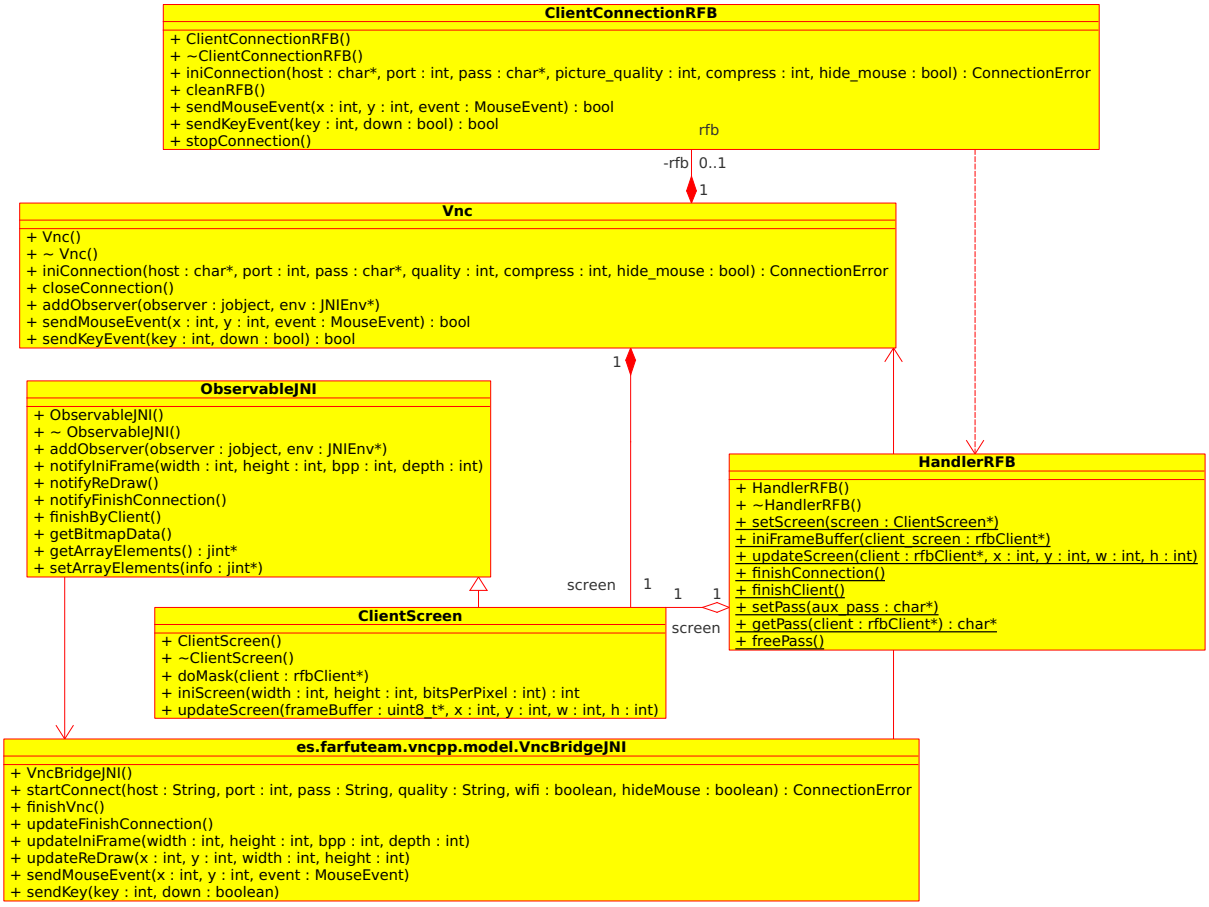


Figura 6.6: Diagrama Conexión

Al inicializar el código nativo se crea la clase Vnc (Ver Figura 6.6) y se añade el observer a la clase ClientScreen para que éste pueda realizar las notificaciones a Java. En la constructora de Vnc se crean las clases ClientScreen, ClientConnectionRFB y se le pasa a la clase HandlerRFB la referencia a ClientScreen para que ésta pueda comunicarse con ella. Vnc retrasmitirá todas las ordenes que le lleguen de JavaBridge a ClientConnectionRFB.

La clase ClientConnectionRFB (Ver Figura 6.6) es la encargada de gestionar la conexión entre el cliente y el servidor utilizando el protocolo RFB. Para establecer una conexión lo primero que se hace es inicializar la estructura rfbClient, en este punto se indica cual es el host, el puerto, la calidad de imagen, etc. Un punto muy importante de la inicialización es indicar cuales son las funciones que se han de llamar cuando el server mande notificaciones. Para ello nos apoyamos en la clase HandlerRFB, de tal manera que, por ejemplo, cuando llega un update de la imagen indicamos que tiene que llamar al método *updateScreen* del handler.

```
clientRFB=rfbGetClient(bitsPerSample,samplesPerPixel,bytesPerPixel);
clientRFB->serverPort = port;
clientRFB->serverHost = host;
clientRFB->programName = "VNC++";
```

```
HandlerRFB::setPass(pass);
clientRFB->GetPassword = HandlerRFB::getPass;
clientRFB->appData.qualityLevel = picture_quality;
clientRFB->appData.compressLevel = compress;
clientRFB->appData.useRemoteCursor = hide_mouse;
clientRFB->MallocFrameBuffer=HandlerRFB::iniFrameBuffer;
clientRFB->canHandleNewFBSize = TRUE;
clientRFB->GotFrameBufferUpdate=HandlerRFB::updateScreen;
clientRFB->FinishedFrameBufferUpdate= HandlerRFB::finishUpdate;
clientRFB->listenPort = LISTEN_PORT_OFFSET;
clientRFB->listen6Port = LISTEN_PORT_OFFSET;
```

Una vez terminada la inicialización de rfbClient se procede a la conexión con el servidor.

```
if(!rfbInitClient(clientRFB,0,NULL)){
  error_connect = NoServerFound;
  if(DEBUG)
    LOGE("No server found");
  clientRFB = NULL;
}
else if( !clientRFB->frameBuffer){
  if(DEBUG)
    LOGE("No Frame Found");
  error_connect = NoFrameFound;
  cleanRfb();
}
```

Si todo va bien se crea un hilo para manejar los eventos del protocolo RFB. Dicho hilo es un bucle infinito a la espera de mensajes que provengan del servidor.

```
while(!aux_this->stop_connection) {
  mes=WaitForMessage(aux_this->clientRFB,timeWait);
  if(mes<0){
    aux_this->stop_connection = true;
    serverOut = true;
  }
  if(mes){
    if(!HandleRFBServerMessage(aux_this->clientRFB)){
      aux_this->stop_connection = true;
      serverOut = true;
    }
  }
}
```

La variable aux_this es un puntero a ClientConnectionRFB. Si no se ha parado la conexión se espera hasta que llega un mensaje del servidor, la función encargada de ello es *WaitForMessage*. Después la función *HandlerRFBServerMessage* es la encargada de comprobar cual es el mensaje que ha llegado y en función de ello, llamar a unas funciones u otras de HandlerRFB.

Para enviar los eventos del ratón o de pulsación de teclas se utilizan las funciones *SendPointerEvent* y *SendKeyEvent* respectivamente. Cabe destacar que para enviar el

evento de pulsación de teclado hay que transformar el código de tecla que nos llega desde Java para que el server sea capaz de interpretarlo correctamente.

```
bool ClientConnectionRFB::sendKeyEvent(int key, bool down){
  rfbKeySym rfbKey = transformToRfbKey(key);
  if(rfbKey != 0){
    SendKeyEvent(clientRFB, rfbKey, down);
  }
}
```

La clase HandlerRFB (Ver Figura 6.6) es una clase completamente estática. Esto se debe a que, dado que LibVNCServer se encuentra escrita en C y los eventos del mismo son punteros a funciones, si no fuesen estáticos los métodos en C++ tienen de forma implícita el parámetro this, que es una referencia a la propia clase. Por el contrario C no es un lenguaje orientado a objetos, por ello ese parametro this no existe en las llamadas. Al declarar estáticos los métodos, el parámetro this desaparece. Sus métodos básicamente redirigen la llama a las funciones de ClientScreen, por eso era necesario pasar la referencia al inicio.

ClientScreen (Ver Figura 6.6) se encarga del manejo de las actualizaciones de la imagen, así como de comunicar todos los cambios a Java. Esta comunicación la hace a través de ObservableJNI mediante la herencia de sus métodos.

Cuando llega un evento de inicializar la imagen desde el handler se llama al método *iniScreen*. En este método se inicializan todos los valores de la imagen y se envía una notificación a Java.

```
int ClientScreen::iniScreen(const int width,
                            const int height,
                            const int bitsPerPixel) {
  if(DEBUG)
    LOGE("JNI iniFrameBuffer");
  this->width=width;
  this->height=height;
  //se pasa de bits a byte
  this->bytesPerPixel=bitsPerPixel/8;
  this->depth = bitsPerPixel;
  //se calcula el espacio total del buffer
  this->size = this->height*this->width*this->bytesPerPixel;
  if(DEBUG)
    LOGE("FIN JNI iniFrameBuffer");
  notifyIniFrame(this->width,
                 this->height,
                 this->bytesPerPixel,
                 this->depth);
  return size;
}
```

Cuando se quiere invocar métodos de Java, para notificar cambios, es necesario colocarse en el entorno de ejecución actual, esto es, en el hilo. Para ello usamos la función *getEnviroment*. Este método comprueba si la variable env, del tipo JNIEnv*, se encuentra apuntando al hilo actual, si no fuese el caso, se coloca en él.

```
void ObservableJNI::getEnviroment(){
  int getEnv = vm->GetEnv((void**)&env,JNI_VERSION_1_6);

  if(getEnv == JNI_EDETACHED){
    vm->AttachCurrentThread(&env,NULL);
  }
}
```

El método notifyIniFrame obtiene el método *updateIniFrame* del entorno Java y lo llama pasándole la información nueva. Después se llama a *GetObjectClass* para obtener la clase VncBridgeJNI y así, poder hacer llamadas a sus métodos. Para hacer una llamada a un método de Java, una vez tenemos la clase, hay que utilizar el método *GetMethodID* para obtener el ID del método que deseamos invocar. Una vez que dispongamos de dicha ID podemos llamarlo usando *CallVoidMethod*.

```
void ObservableJNI::notifyIniFrame(int width,
                                   int height,
                                   int bpp,
                                   int depth){
  getEnviroment();
  if(DEBUG)
    LOGE("Take method iniFrame");
  observer_class  = env->GetObjectClass(this->observer_object);
  //cogemos el ID
  jmethodID updateScreen = env->GetMethodID(observer_class,
                                            "updateIniFrame",
                                            "(IIII)V");
  if(DEBUG)
    LOGE("Launch method iniFrame");
  env->CallVoidMethod(observer_object,
                      updateScreen,
                      width,
                      height,
                      bpp,
                      depth);
  if(DEBUG)
    LOGE("Finish launch method iniFrame");
  env->DeleteLocalRef(observer_class);
}
```

Para finalizar, el método *updateScreen* de ClientScreen es el encargado de actualizar una porción de la imagen. El inicio de la porción a actualizar viene indicado por los parámetros x e y, siendo w y h la anchura y la altura respectivamente de dicha porción.

```
void  ClientScreen::updateScreen(uint8_t *frameBuffer,
                                 const int x,
                                 const int y,
                                 const int w,
                                 const int h){
  if(DEBUG)
    LOGE("JNI Update");
  int pixel;
```

```
  int pos;
  int pos_col;
  int pos_array;
  getBitmapData();
  jint *info = getArrayElements();
  int real_i,real_j;
  for(int i=0;i < w;i++ ){
    real_i = i + x;
    pos_col = (real_i*bytesPerPixel);
    for(int j=0;j<h;j++){
      real_j = j + y;
      pos = ((bytesPerPixel*width) * real_j) + pos_col;
      memcpy(&pixel,&frameBuffer[pos],bytesPerPixel);
      pos_array = (width *real_j) + real_i;
      info[pos_array] = pixel;
    }
  }
  if(DEBUG)
    LOGE("JNI notify reDraw");
  setArrayElements(info);
}
```

Lo que hace este método es recorrer la porción que ha cambiado con los dos bucles for. Las variables real_i y real_j son las posiciones reales de la porción dentro de la imagen. Como la imagen se representa en un buffer unidimensional es necesario calcular el desplazamiento que se obtiene:

$$pos = ((bytesPerPixel)(width))(real\_j) + ((real\_i)(bytesPerPixel))$$

Una vez tenemos la posición copiamos el píxel con memcpy y lo guardamos en info. El bitmap info es una referencia al bitmap de Java, lo obtenemos a través de los métodos *getBitmapData* y *getArrayElements*, de tal manera que cuando se modifican elementos de dicho bitmap en C se están modificando en Java. Lo único que falta por hacer es actualizar la información a través del método *setArrayElements*. Mediante el que JNIEnv libera el vector del entorno de C/C++ con la función *ReleaseIntArrayElements* y se actualiza el vector de Java.

Capítulo 7

Comparativas

Se describe la comparación del resulatdo final con las aplicaciones que han sido tomadas como referencia en el principio del proyecto. Como ya se indicó en el capítulo dos, usamos como punto de partida tres aplicaciones que abarcan el campo del control remoto móvil.

7.1. Funcionalidad

Nuestro propósito inicial era mejorar ciertos aspectos de estas aplicaciones ya existentes, y en este apartado observamos resultados.

7.1.1. Funcionalidades destacadas compartidas con VNC++

- Hace zoom.
- Muestra el teclado.
- Escribe.
- Envía secuencias de teclas.
- Recuerda conexiones previas.

7.1.2. Funcionalidades diferentes respecto VNC++

- **RealVNC:** como ya se matizó, esta aplicación está basada en software privativo, por lo que no disponemos de mucha información relativa a la implementación interna de este proyecto. Pero algo que nos propusimos desde el principio era crear un software puramente libre ya que pensamos que es así como se llega a un producto final con más opciones de desarrollo y uso futuro.

  Permite, a través de un menú de opciones, desactivar el modo de pantalla completa, así como opción de Copy&paste. También es destacable la diferencia en el manejo, en VNC Viewer al mover el dedo por la pantalla lo que moveremos es el cursor y no la imagen como en nuestro caso.

- **MultiVNC:** libre y de código abierto, usa aceleración hardware OpenGL para el dibujado y el zoom, mientras que en nuestro caso se ha optado por el uso de Canvas. El manejo en la pantalla de MultiVNC es igual que el de RealVNC.

- **AndroidVNC** también es libre y de código abierto, pero como contrapartida tiene una interfaz pobre y además el usuario se ve obligado a tener la pantalla en posición horizontal. En dispositivos con pantallas de reducido tamaño, la información es difícil de acceder, haciéndose el manejo bastante incómodo. En contrapartida ofrece muchos métodos de control, así como, la posibilidad de enviar texto desde un campo de texto.

7.1.3. Mejoras consideradas

Nuestra principal meta, como se indica en la sección de objetivos (Sección 1.3), era lograr usar el potencial del NDK para el uso de VNC, que es la principal desventaja de todos los proyectos software libre estudiados.

Partiendo de esta diferencia, el resto de matices los hemos querido mejorar también, viendo las deficiencias que pudieran tener el resto de aplicaciones, que echamos en falta como usuarios.

Como todas ellas usamos el protocolo RFB, con opciones de pantalla completa, mostrar teclado, etc., con lo que en estos aspectos no mejoramos ni empeoramos nada.

Pero como usuarios hemos intentado hacer un envío más intuitivo de teclas combinadas (tipo Alt-F4), así como diálogos informativos al usuario. Además se ha incorporado la opción de centrar la imagen, con el fin de recolocarse.

En cuanto a la codificación de la imagen, los otros dos proyectos software de referencia usan ZRLE, lo que implica no aplicar ningún tratamiento previo a la imagen con la consiguiente carga en la transmisión de datos. VNC++, sin embargo, usa codificación Tight, que aplica una compresión a la imagen para tener un rendimiento mayor.

Otro logro significativo es que nuestra aplicación soporta SSL (capa de conexión segura), que es un protocolo criptográfico que permite conexiones seguras, que es algo que como usuario siempre se demanda.

7.2. Rendimiento

Para analizar los resultados obtenidos, desde el punto de vista del rendimiento, se han querido analizar diferentes puntos: uso de CPU, tiempo de respuesta y uso de la red.

7.2.1. Entorno de pruebas

Para hacer la mediciones en un entorno adecuado, las pruebas han sido realizadas en el siguiente entorno:

- **Versión Android:** 4.2.2 Jelly Bean.
- **Móvil:** Samsung Galaxy S3.
- **Sistema Operativo en Servidor:** Linux Mint Debian Edition 64 Mate.
- **Resolución del servidor:** 2704x1818.
- **Servidor VNC:** X11vnc.

- **Router:** Xavi 7968.

Aplicaciones a comparar:

- VNC++.
- RealVNC (VNC viewer).
- AndroidVNC.
- MultiVNC.

Para unas mediciones adecuadas, las pruebas se han realizado en una red local cerrada, en la cual, el servidor se encontraba conectado al switch mediante cable de ethernet y el móvil mediante wifi. También se han hecho las pruebas mediante el uso de 3G en móvil, en un entorno equivalente, con la excepción de que en este caso el servidor se encontraba conectado a Internet.

7.2.2. Características de las pruebas

En las pruebas se ha querido someter a las aplicaciones a una situación de estrés , para ello se ha llevado a cabo la reproducción de un vídeo en 1080p a pantalla completa durante aproximadamente 4 minutos (2 minutos en el caso del 3G). Las muestras tomadas a partir de las mismas son aproximadamente de los primeros 180 segundos (80 segundos con 3G) de la aplicación, partiendo desde el inicio total de la conexión. El vídeo se encontraba en reproducción desde el primer momento de la conexión.

7.2.3. Resultados

En el primer punto, uso de CPU, no se ha conseguido ninguna forma fiable de conseguir mediciones, y por otro lado, no todas las aplicaciones utilizan la misma tecnología, por ejemplo, MultiVNC utiliza GPU para la representación. Por ello se ha decidido desechar las pruebas relativas a este punto.

En segundo lugar, tiempo de respuesta. Nos encontramos en un caso análogo al anterior, aunque en este caso si cabe destacar un matiz. En la carga inicial de la aplicación, al mostrar la primera imagen del servidor, en las pruebas con Wifi, RealVNC es sensiblemente más rápido que VNC++ y ambos son notablemente más rápidos que AndroidVNC, el cual a su vez, es sensiblemente más rápido que MultiVNC.

Cabe destacar que en el caso de conexión 3G las diferencias fueron muy significativas, mientras RealVNC y VNC++ mantuvieron tiempos parejos (como ya se ha descrito anteriormente), AndroidVNC y MultiVNC mostraron la primera imagen prácticamente al acabar la prueba.

En cuanto al tiempo de respuesta una vez hecha la carga, nos encontraríamos en rendimientos muy parejos entre todos, nada que destacar en este punto.

Por último, nos encontramos en las pruebas de uso de red, en este punto sí hemos podido obtener resultados fiables y concluyentes, por ello lo detallaremos de forma amplia en el siguiente apartado.

7.2.4. Uso de Red

Para la realización de estas mediciones se ha utilizado el software Wireshark (Ver sección 1.1.10), mediante el cual se ha podido capturar los paquetes, para posteriormente, recoger los resultados.

Antes de proceder con los resultados explicaremos los diferentes campos que componen los resultados:

- **Packets:** número de paquetes.
- **Between first and last packet:** tiempo transcurrido desde el primer y último paquete.
- **Avg. Packet/sec:** promedio de paquetes por segundo.
- **Avg. Packet size:** promedio de tamaño del paquete.
- **Bytes:** bytes totales de la conexión.
- **Avg. bytes/sec:** promedio de bytes por segundo.
- **Avg. Mbit/sec:** promedio de Megabits por segundo.

Dentro de las cuales encontraremos dos columnas:

- **Captured:** cálculos en función del total de paquetes capturados.
- **Displayed:** cálculos en función de los paquetes que se muestran, esto es, con filtros aplicados.

Para el análisis nos fijaremos en la columna Displayed ya que ésta representa los cálculos con el filtro de 180 segundos aproximadamente, como se podrá observar. De entre todos los campos, los que tendrán más relevancia y peso para las pruebas son Avg bytes/sec y Avg. Mbit/sec, ya que el cálculo promedio muestra resultados de mayor relevancia.

Como ya hemos nombrado anteriormente, las mediciones se han hecho en dos situaciones, Wifi y 3G, los resultados son:

Wifi:

VNC++:

Traffic	Captured	Displayed	Marked
Packets	14273	9773	0
Between first and last packet	266,668 sec	180,412 sec	
Avg. packets/sec	53,523	54,170	
Avg. packet size	9106,926 bytes	8988,075 bytes	
Bytes	129983155	87840459	
Avg. bytes/sec	487433,822	486888,260	
Avg. MBit/sec	3,899	3,895	

Figura 7.1: Rendimiento Wifi VNC++

RealVNC:

Traffic	Captured	Displayed	Marked
Packets	49724	27934	0
Between first and last packet	293,211 sec	180,873 sec	
Avg. packets/sec	169,584	154,440	
Avg. packet size	4277,449 bytes	4319,656 bytes	
Bytes	212691858	120665279	
Avg. bytes/sec	725387,796	667127,438	
Avg. MBit/sec	5,803	5,337	

Figura 7.2: Rendimiento Wifi RealVNC

AndroidVNC:

Traffic	Captured	Displayed	Marked
Packets	87256	49688	0
Between first and last packet	305,424 sec	180,127 sec	
Avg. packets/sec	285,688	275,850	
Avg. packet size	2458,271 bytes	2449,087 bytes	
Bytes	214498901	121690248	
Avg. bytes/sec	702298,565	675580,533	
Avg. MBit/sec	5,618	5,405	

Figura 7.3: Rendimiento Wifi AndroidVNC

MultiVNC:

Traffic	Captured	Displayed	Marked
Packets	82344	49327	0
Between first and last packet	396,512 sec	180,202 sec	
Avg. packets/sec	207,671	273,732	
Avg. packet size	2533,726 bytes	2548,249 bytes	
Bytes	208637138	125697460	
Avg. bytes/sec	526180,742	697536,219	
Avg. MBit/sec	4,209	5,580	

Figura 7.4: Rendimiento Wifi MultiVNC

Como se puede observar, los resultados de RealVNC, AndroidVNC y MultiVNC son muy similares, esto es debido principalmente a que todos ellos utilizan la codificación para la imagen ZRLE. Sin embargo, los resultados de VNC++ son muy distintos, debido a que usa Tight.

En primer lugar, se puede observar que tanto AndroidVNC como MultiVNC tienen un tamaño medio de paquetes muy parecido, mientras que los de RealVNC son aproximadamente el doble, y a su vez, los de VNC++ el doble de éste.

Por otro lado, encontramos que el promedio de Mbit/sec de RealVNC, AndroidVNC y MultiVNC es muy parecido pero el de VNC++ es significativamente inferior. Esto se debe a la codificación de la imagen, como ya dijimos anteriormente, la codificación usada por VNC++ es Tight. Esto provoca, al procesar previamente la imagen, que el uso de la red sea inferior.

De estos resultados se pueden concluir principalmente dos cosas:

- Por un lado, que debido a un uso inferior de la red, VNC++ funcionaría mejor en entornos de poca conexión.
- Y por el otro, que debido a que los paquetes son más grandes, VNC++ respondería peor a la pérdida de paquetes, ya que si se pierde un paquete la información a reenviar es superior.

3G:

VNC++:

Traffic	Captured	Displayed	Marked
Packets	8117	6648	0
Between first and last packet	142,553 sec	80,549 sec	
Avg. packets/sec	56,940	82,534	
Avg. packet size	1486,543 bytes	1517,727 bytes	
Bytes	12066273	10089852	
Avg. bytes/sec	84643,896	125263,907	
Avg. MBit/sec	0,677	1,002	

Figura 7.5: Rendimiento 3G VNC++

RealVNC:

Traffic	Captured	Displayed	Marked
Packets	11039	7062	0
Between first and last packet	122,966 sec	80,423 sec	
Avg. packets/sec	89,773	87,811	
Avg. packet size	1473,957 bytes	1510,797 bytes	
Bytes	16271014	10669251	
Avg. bytes/sec	132321,601	132663,956	
Avg. MBit/sec	1,059	1,061	

Figura 7.6: Rendimiento 3G RealVNC

AndroidVNC:

Traffic	Captured	Displayed	Marked
Packets	9585	7892	0
Between first and last packet	99,501 sec	80,447 sec	
Avg. packets/sec	96,331	98,102	
Avg. packet size	1437,301 bytes	1443,103 bytes	
Bytes	13776531	11388968	
Avg. bytes/sec	138456,276	141570,642	
Avg. MBit/sec	1,108	1,133	

Figura 7.7: Rendimiento 3G AndroidVNC

MultiVNC:

Traffic	Captured	Displayed	Marked
Packets	9269	7056	0
Between first and last packet	146,665 sec	71,301 sec	
Avg. packets/sec	63,198	98,961	
Avg. packet size	1404,548 bytes	1428,624 bytes	
Bytes	13018755	10080372	
Avg. bytes/sec	88764,974	141377,512	
Avg. MBit/sec	0,710	1,131	

Figura 7.8: Rendimiento 3G MultiVNC

Como se puede observar los valores con una conexión 3G son muy parecidos, ello es debido a cómo funciona la propia conexión y sus limitaciones. Así pues se puede concluir que el rendimiento en entornos de 3G de las aplicaciones es equivalente.

Capítulo 8

Conclusiones

8.1. Conclusiones

Como conclusión pensamos que se han alcanzado las metas propuestas en el apartado 1.3. Todo el código ha sido licenciado bajo una licencia libre (GPL versión 3), por tanto el objetivo de crear una aplicación de software libre se ha cumplido sin mayor problema. En cuanto al objetivo más importante, la construcción de la aplicación de escritorio remoto en Android utilizando código nativo, también se ha cumplido, ya que no sólo se ha construido una aplicación con muchas funcionalidades, sino que es una aplicación eficiente, capaz de competir con todas las demás aplicaciones de escritorio remoto para Android que existen en este momento. Cabe destacar una vez más que la gran diferencia y la razón por la que este proyecto es innovador, es en que ninguna de las demás aplicaciones utilizan código nativo, al menos no las de software libre. La imposibilidad de saber como están construidas las aplicaciones privativas nos impide afirmar con total rotundidad que no existe una aplicación de escritorio remoto para Android que utilice el NDK. No obstante, al no existir una de software libre, la innovación sigue estando ahí, ya que hemos trabajado sobre algo que no existe en ningún lado y con unos resultados considerablemente buenos.

8.2. Trabajo Futuro

Como posible trabajo futuro se podría añadir soporte para todos los idiomas, en dos sentidos. Por una lado la traducción de la interfaz a diveros idiomas y, por otro, soporte en el teclado para más idiomas.

Otra cosa que se podría investigar más es la aceleración por hardware, de esta manera se podría conseguir una mayor velocidad a la hora de trabajar las imágenes y con ello se obtendría una mejora del rendimiento global considerable.

También pensar en portar la aplicación a las demás plataformas móviles como Firefox OS.

Bibliografía

[1] Wikipedia Escritorio Remoto: http://es.wikipedia.org/wiki/Escritorio_remoto

[2] Wikipedia Virtual Network Computing: http://en.wikipedia.org/wiki/VNC

[3] Remote Frame Buffer Protocol: http://www.realvnc.com/docs/rfbproto.pdf

[4] Wikipedia Android (operating system): http://en.wikipedia.org/wiki/Android_(operating_system)

[5] Software Development Kit: http://developer.android.com/sdk/index.html

[6] Native Development Kit: http://developer.android.com/tools/sdk/ndk/index.html

[7] Wikipedia Java Native Interface: http://en.wikipedia.org/wiki/Java_Native_Interface

[8] TightVNC, Tight encoding: http://www.tightvnc.com/decoder.php

[9] Wikipedia Git: http://en.wikipedia.org/wiki/Git_(software)

[10] Git: http://git-scm.com/

[11] WireShark: http://www.wireshark.org/

[12] Free Software: http://www.fsf.org/about/what-is-free-software

[13] General Public License: http://www.gnu.org/licenses/gpl.html

[14] Wikipedia RealVNC: http://es.wikipedia.org/wiki/RealVNC

[15] MultiVNC: https://play.google.com/store/apps/details?id=com.coboltforge.dontmind.multivnc&hl=es

[16] Android-vnc-viewer: http://code.google.com/p/android-vnc-viewer/

[17] SDL: http://www.libsdl.org/

[18] Wikipedia Simple DirectMedia Layer: http://es.wikipedia.org/wiki/Simple_DirectMedia_Layer

[19] SherlockBar: http://actionbarsherlock.com/

[20] LibVNC: http://libvncserver.sourceforge.net/

[21] SlidingMenu: https://github.com/jfeinstein10/SlidingMenu

[22] SQLite Android: http://developer.android.com/reference/android/database/sqlite/package-summary.html

[23] Interfaz Holo: http://developer.android.com/design/style/themes.html

Apéndices

Apéndice A

Aquí se presenta una pequeña descripción de cómo se usa la aplicación. Este texto de ayuda se puede ver también desde la propia aplicación accediendo al menú en la ventana principal. Se encuentra tanto en español como inglés, al igual que todos los diálogos y texto de los botones. Es el propio sistema Android, en base al idioma seleccionado por defecto en el sistema operativo, el que selecciona el idioma a mostrar. En caso de no disponer del idioma correspondiente al del sistema, por defecto se mostrarán los texto en inglés.

A.1. Página de ayuda de VNC++

A.1.1. Manejo principal

Para crear una nueva conexión pulse el icono "+" en la barra superior de Acción (Ver Figura A.1). Una vez dentro, es necesario especificar la IP del servidor, así como un nombre de identificación de la conexión. El puerto por defecto es el 5900, pudiéndose cambiar, así como la contraseña, pudiendose dejar en blanco si no se necesita.

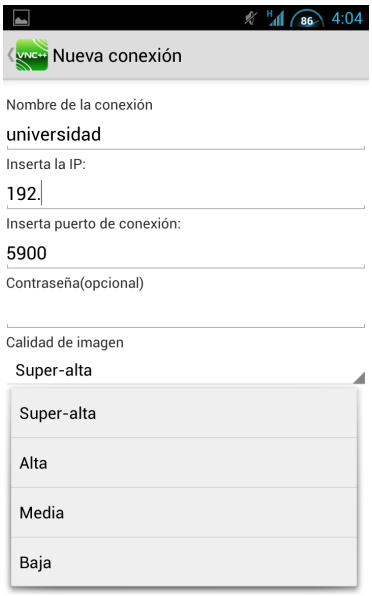


Figura A.1: Nueva conexión

Figura A.2: Vista del escritorio

El desplegable de calidades sirve para indicar la calidad de la imagen. Se puede elegir entre calidad super-alta, alta, media o baja, según el tipo de conexión del usuario ya sea más lenta o rápida, o simplemente por necesidad de cierta calidad de imagen.

En el botón desplegable de opciones en la actividad principal se puede acceder a tres submenús:

- La opción de Configuración: permite, a través de dos switches de selección, marcar si se recuerda o no la opción de confirmar salida, así como ocultar el cursor o no (Ver Figura A.3).
- Manual de uso parecido al que se está narrando (Ver Figura A.4).
- Sección "Acerca de" donde se reflejan los creadores de la aplicación, así como el tipo de licencia y las librerías usadas (Ver Figura A.8).

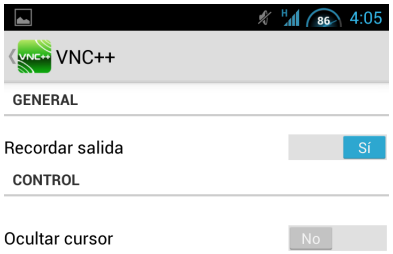


Figura A.3: Configuración

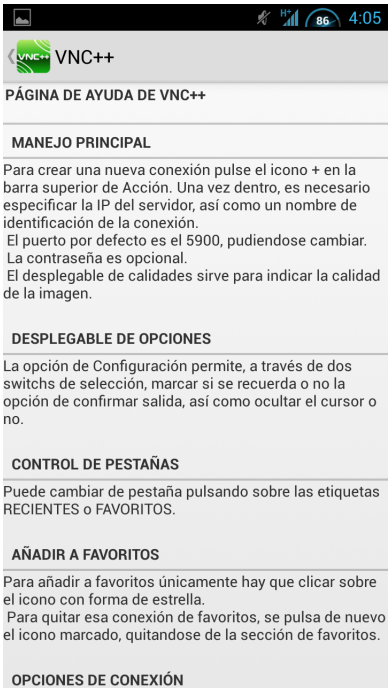


Figura A.4: Manual de uso

A.1.2. Control de pestañas

Se puede cambiar de pestaña pulsando sobre las etiquetas RECIENTES o FAVORITOS, y así ver las distintas conexiones creadas (Ver Figura A.5).

A.1.3. Añadir a favoritos

Para añadir a favoritos únicamente hay que clicar sobre el icono con forma de estrella (Ver Figura A.5).

Para quitar esa conexión de favoritos, se pulsa de nuevo el icono marcado, quitándose de la sección de favoritos.

A.1.4. Opciones de conexión

Al pulsar sobre una conexión de la lista, se presenta un menú de opciones. Desde ahí se puede conectar, mostrar la información de la conexión, editar y eliminar dicha conexión. También se puede conectar pulsando el icono de la derecha con forma de flecha.

A.1.5. Opciones de edición

Se podrá modificar cualquier valor previo de la conexión, a excepción del identificador elegido, del mismo modo que se crea una nueva conexión.

A.1.6. Menú lateral

Una vez cargada la imagen se puede acceder al menú de opciones, tanto desde el botón físico del terminal, como arrastrando desde la izquierda de la pantalla para sacar el menú lateral. Una vez ahí se puede mostrar teclado, mandar teclas, centrar imagen, mostrar una ayuda reducida y desconectar (Ver Figura A.6).

Figura A.5: Pestañas

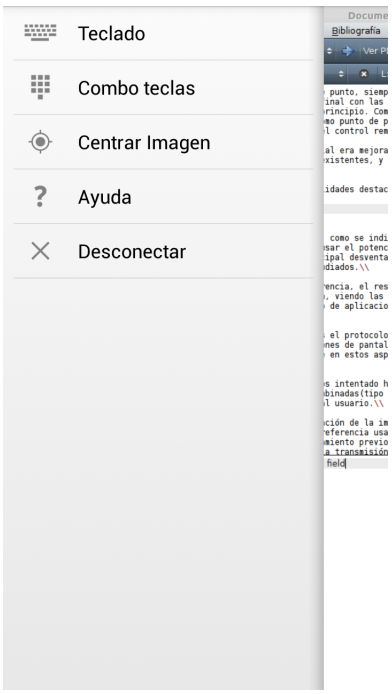


Figura A.6: Menú lateral

A.1.7. Manejo

Para mandar evento click, simplemente haga una pulsación con el dedo.

Para hacer doble click, mantenga presionado el dedo sobre la pantalla durante unos 3 segundos.

Para hacer zoom, coloque un dedo sobre la pantalla, mientras con otro dedo alejas o acercas del primer dedo, a modo de pinza.

A.1.8. Envío de teclas especiales

Para mandar eventos ctrl, alt, supr, shift, basta con seleccionar la tecla Çombo teclas", y en la ventana emergente seleccionar las teclas que se quieran enviar (Ver Figura A.7).

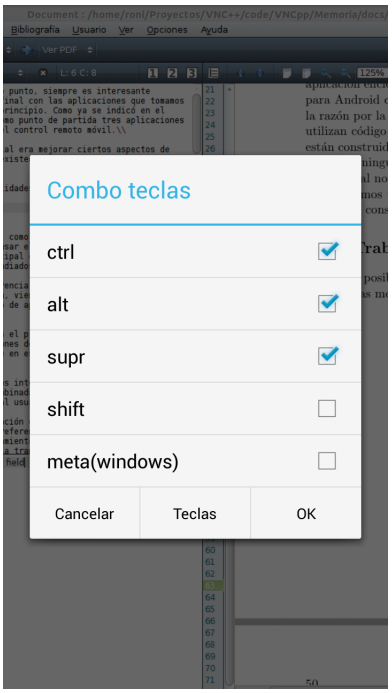


Figura A.7: Teclas especiales

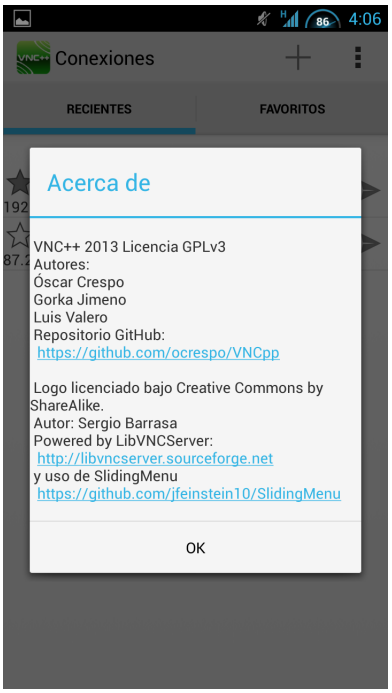


Figura A.8: Acerca de

Las teclas función (F1, F2, etc) y el resto de letras, se pueden seleccionar desde el botón central "Teclas".