AppTFG

Jaime Ramos Romero
Javier Bastarrica Lacalle

Grado en Ingeniería Informática
Facultad de Informática

UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo de Fin de Grado
Madrid, Junio 2015

Directores:
Samer Hassan Collado
Pablo Ojanguren

Autorización de Difusión y Uso

Autorizo a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor, tanto la propia memoria, como el código, la documentación y/o el software desarrollado.

Jaime Ramos Romero
Javier Bastarrica Lacalle

Madrid, Junio 2015

Índice general

Índice de figuras

Índice de cuadros

Abstract

Abstract en inglés.

Keywords:
Apache Wave, Collaboration, Android, Apps, Real Time, Politics

Resumen

Abstract en español.

Palabras Clave:
Apache Wave, Colaboración, Android, Apps, Tiempo Real, Política

Capítulo 1

Introducción

1.1. Objetivos del Proyecto

Los objetivos son los siguientes:

- 1ª parte: Migración de Wave a Android
  - Estudiar la implementacion actual de Wave en Java y GWT.
  - Adaptar la implementación para hacer uso de las caracteristicas nativas de Android.
- 2ª parte: Creación de una aplicación Android
  - Evaluar posibles ideas de aplicación y estudiar su viabilidad.
  - Evaluar con usuarios su usabilidad.
  - Implementar la aplicación.
  - Testear y evaluar con usuarios el resultado.

1.2. Estructura del Documento

- Capítulo 2 - Migración de Wave a Android: Descripcion de la tecnologia Wave y de la Metodología utilizada para la migración a Android.
- Capítulo 3 - AppTFG:
- Capítulo 4 - Resultados y Conclusiones:
- Capítulo 5 - Trabajo Futuro:

Capítulo 2

Migración de Wave a Android

2.1. Estado del Arte

—- PENDIENTE —-

2.2. Wave

2.2.1. Google Wave

Ideado y presentado en 2009 por ingenieros de Google [1], Wave es a la vez un protocolo de comunicaciones [2] y una plataforma web de código libre, que permiten a sus usuarios comunicarse y colaborar entre sí en tiempo real (Ver sección 2.3.1) y de forma federada (Ver sección 2.3.1) a través de Internet. Inicialmente fue desarrollado con el objetivo de integrar en una sola plataforma servicios ampliamente utilizados como son el correo electrónico, las redes sociales y la mensajería instantánea. Pese al gran entusiasmo generado entre la comunidad de desarrolladores tras su anuncio, en el año 2010 Google anuncia el abandono del proyecto [3] debido a su poca acogida entre los desarrolladores y a que decide reorientar el uso de la tecnología hacia sus plataformas de edición de documentos Google Docs [4] y a su red social Google + [5]. Es en este momento cuando el desarrollo libre del proyecto pasa a manos de la Apache Software Foundation bajo el nombre de Apache Wave.

2.2.2. Apache Wave

Al cambiar de manos su desarrollo en 2010, la tecnología pasa a formar parte de la incubadora de la fundación Apache [6] como software de código libre bajo licencia Apache [7]. Así, se produce el desarrollo de Wave In a Box (WIAB) (Ver sección **??**), plataforma que integra un cliente web sencillo y una implementación de un servidor Wave que cualquiera puede descargar y desplegar en su ordenador.

2.3. Tecnologias y Caracteristicas de Wave

2.3.1. Caracteristicas de Wave

Como plataforma de código libre desarrollada para ser utilizada en red, Wave hace uso de distintas tecnologías y protocolos bien conocidos. Entre sus características más destacadas están las siguientes:

Federación

El Protocolo Wave [2] fue desarrollado para utilizar un modelo federado [8] [9] de comunicación basado en la tecnología XMPP [10] [2]. Se trata por tanto de un modelo descentralizado en el que cualquiera de los participantes en la conversación es libre de actuar tanto como servidor como cliente sin que ello afecte a su participación en la conversación. Además, a diferencia de otras tecnologías (como el correo electronico) en las que cada participante almacena su propia copia de la conversación y cada vez que hay cambios se debe transmitir la conversación entera a todos los participantes, Wave tiene la ventaja de que actúa de forma que es el servidor de la conversación el único que almacena la copia entera y se encarga de calcular los cambios que se han producido para transmitir solamente dichos cambios por la red a los participantes, con las consiguientes ventajas en términos de latencia que ello conlleva.

Consistencia en tiempo real

El Protocolo Wave [2] utiliza la tecnología de Transformaciones Operacionales (OT) [11] para garantizar la consistencia en la comunicación en tiempo real entre los participantes. Es decir, cualquier cambio producido por cualquiera de los participantes en la conversación se transmite automáticamente y en tiempo real al resto de los participantes sin pérdida de información y garantizando que los cambios se muestran en el estricto orden en el que se produjeron sin errores [12].

Escalabilidad

Wave fue desarrollado como un protocolo de alta escalabilidad que permite gestionar la existencia de una gran cantidad de conversaciones y participantes sin que por ello se resienta la productividad del sistema.

Modelo Wave

Además de definir el protocolo del que hace uso Wave, Google definió un Modelo de Datos Conversacional [13] que refleja la arquitectura de los datos que componen las conversaciones en Wave. Así, a grandes rasgos, podemos ver dichas conversaciones como documentos XML sobre los que los usuarios participantes (cualquiera es libre de unirse a una conversación en cualquier momento) actúan creando nuevos elementos o modificando los ya existentes. Este modelo de datos define una nomenclatura propia para los elementos que componen esta tecnología [14] [9]:

- **Wave**: Conjunto de wavelets (conversaciones).
- **Wavelet**: conjunto de documentos de una conversación y sus participantes.
- **Blip**: documento con el contenido de un mensaje en la conversación. Un blip puede tener otros blips dentro de él y los blips pueden ser publicados o no en función de si su visibilidad se extiende o no al resto de participantes de la conversación respectivamente.
- **Manifiesto conversacional**: documento con metadatos que definen la estructura de una conversación.
- **Hilo conversacional**: conjunto de Blips consecutivos que forman parte de una conversación.
- **Extensiones** [15]: pequeñas aplicaciones que se ejecutan dentro de una Wave y aportan nuevas funcionalidades que no forman parte del modelo conversacional básico. Pueden ser de dos tipos:
  - **Gadget**: aplicación que se ejecuta en el contexto de una Wave y en la que todos sus usuarios participan.
  - **Robot**: aplicación que participa en una Wave a modo de usuario automatizado e interactúa con el contenido pudiendo modificarlo y responder a eventos por acciones de otros usuarios reales.

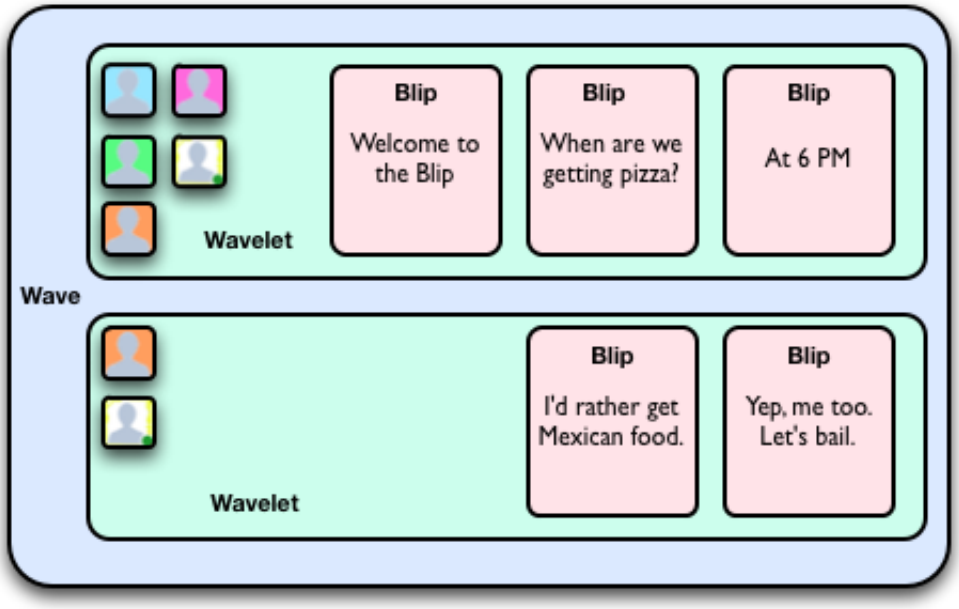

Figura 2.1: Modelo Conversacional de Wave

2.3.2. Servidores Wave

Wave in a Box

Wave In a Box (WIAB) [16] es el nombre de la implementación de un servidor Wave desarrollado por la Apache Software Foundation tras pasar el proyecto a sus manos en el año 2012. Al igual que el resto del código de la tecnología que heredó de Google, está implementado en Java usando OpenJDK [17]. La instalación trae consigo un cliente web desarrollado en Javascript usando el framework Google Web Toolkit [18]. Este cliente web sirve como prueba de concepto de las funcionalidades básicas del Modelo Conversacional de Wave, pudiendo gesionar waves, usuarios y extesiones. Actualmente cualquiera puede descargar y desplegar WIAB en su ordenador siguiendo los pasos que nos proporcionan en su wiki [19]. La aplicación se distribuye en forma de código fuente, accesible entre otras formas desde su repositorio de GitHub [20]. Existen asimismo servidores de prueba ya desplegados en Internet sobre los que se puede observar el funcionamiento de WIAB [21].

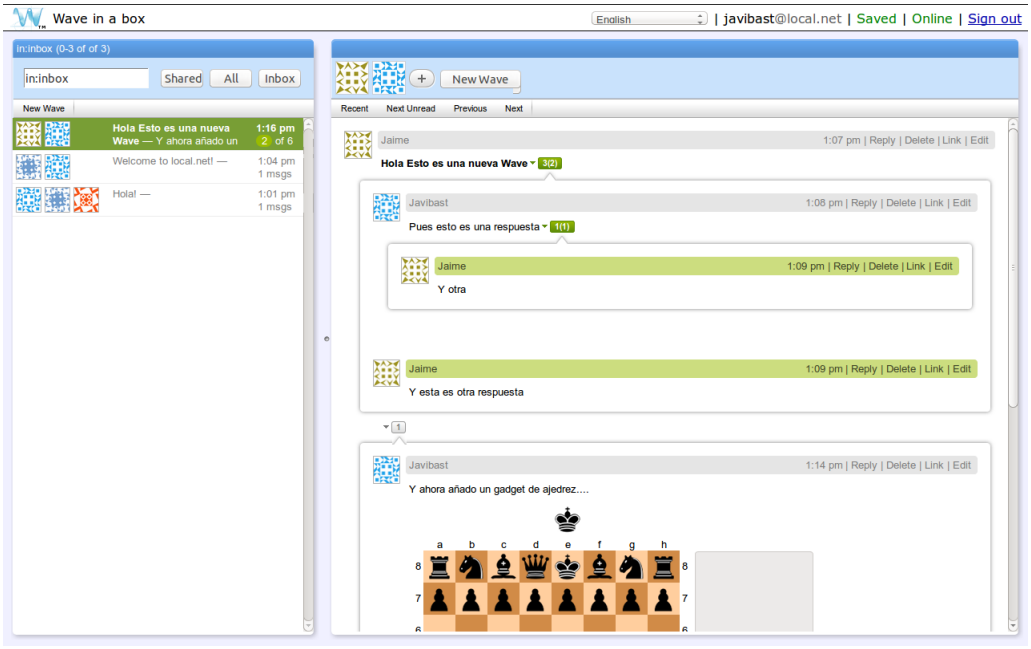

Figura 2.2: Cliente Wave In A Box

SwellRT

Como parte del proyecto europeo P2PValue [22] existe SwellRT, un fork de WIAB que amplía las características de éste último añadiendo un nuevo modelo de datos (Modelo de Datos Colaborativo) más allá del Modelo de Datos Conversacional de Wave original. Proporciona también un API escrito en Java que permite trabajar sobre los datos de ese nuevo modelo en forma de tres tipos básicos: mapas, listas y strings. Es por tanto un framework de colaboración en tiempo real que basa su funcionamiento en Apache Wave y cuyo principal popósito es permitir la integracion de la tecnología Wave en otras aplicaciones, que podrán compartir objetos (de los tipos antes mencionados) de forma federada y en tiempo real. Su código fuente está disponible en GitHub [23], así como sus instrucciones de instalación (Ver el Readme en GitHub).

Para este proyecto se ha usado el framework SwellRT como base para la migración de la tecnología de Apache Wave a la plataforma Android [24]. Se pretende con esto que SwellRT haga uso de las funcionalidades nativas de Android.

2.4. Metodología de Migración

2.4.1. Objetivo

El framework de SwellRT utiliza un servidor WIAB y el protocolo Wave, ambos desarrollados en Java. El **SDK de Android** [25] es compatible con Java, así que a priori la implantación del servidor no supone problemas en los dispositivos móviles. Sin embargo, existe un problema con el API de SwellRT, ya que el lado del cliente fue desarrollado en Javascript usando el framework GWT. Android no soporta de forma nativa estas tecnologías, así que es necesario estudiar el código de SwellRT para sustituir todo el código que haga uso de Javascript/GWT por código compatible con Android. El objetivo de esta parte del proyecto es conseguir que un cliente desplegado en Android sea capaz de conectarse e interactuar con un servidor Wave sin problemas.

2.4.2. Plataforma: Entorno de Desarrollo, Construcción y Depuración

Existen dos entornos de desarrollo (IDE) recomendados por Google para desarrollar en Android: Eclipse [26] y Android Studio.[27] Eclipse es un entorno de desarrollo genérico que, mediante plugins, permite extender sus funcionalidades para desarrollar en diversas plataformas y lenguajes. Android Studio es un IDE basado en el entorno de desarrollo Java IntelliJ IDEA [28] adaptado para trabajar con todas las funcionalidades de Android. En el momento de empezar con la migración Android Studio se encuentra en fase beta de desarrollo, pues Google pretende convertirla en el IDE de desarrollo oficial para Android. Mientras no se lanza la version final de Android Studio, Google recomienda utilizar Eclipse para desarrollar en Android, y las guias para desarrolladores Android estan escritas para Eclipse. En consecuencia tomamos la decision de utilizar el entorno de desarrollo Eclipse para la migracion de SwellRT a Android.

Eclipse

El IDE de Eclipse [26] soporta el desarrollo con Android a través del plugin **ADT (Android Development Tools)** [29], que integra en un solo paquete todas las herramientas necesarias para desarrollar, construir y depurar el código de la aplicacion fácilmente.

Android SDK [25]: paquete que integra el conjunto de herramientas necesarias para desarrollar en Android. Entre estas herramientas destacan las siguientes:

- **Librerías con el API** de Android y **Documentación** asociada [30]

- **Android Virtual Device Manager (AVDM)** [31] herramienta para gestionar la creación, modificación, ejecución y eliminación de emuladores en Android. Un **emulador** [32] es una máquina virtual que ejecuta una determinada versión de Android. Permite desplegar un dispositivo movil en el ordenador que imita las características software y hardware de uno real para poder hacer pruebas de desarrollo sin necesidad de poseer un dispositivo con Android.

- **Android SDK Manager** [33] herramienta para gestionar las versiones de SDK y herramientas asociadas instaladas. Android se encuentra actualmente en la versión 5.1 (API 22), pero un desarrollador puede elegir desarrollar para una versión anterior si lo estima necesario, por lo que puede descargarse por separado dicha versión y mantener varias API si lo necesita.

- **Dalvik Debug Monitor Server (DDMS)** [34] herramienta que provee las características de entorno de depuración para las aplicaciones en desarrollo.

Teniendo en cuenta la distribucion actual de versiones instaladas en dispositivos Android [35] (Ver figura 2.3) se ha decidido realizar la migración de SwellRT con el API 19 de Android (Version 4.4 "KitKat"). El emulador desplegado para las pruebas de desarrollo utilizará por tanto Android 4.4 .

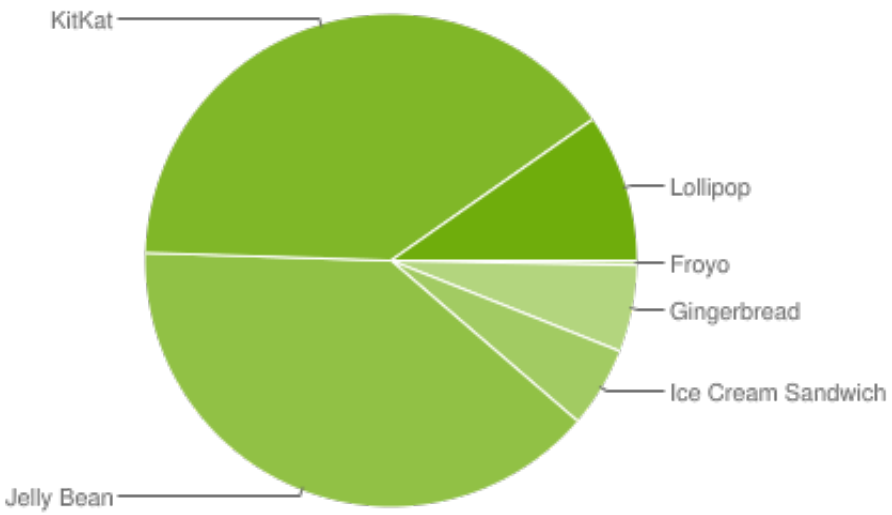

Figura 2.3: Distribución Actual de Versiones Android (Fuente: Google)

Sin embargo existe un problema con la construcción y depuración del código de SwellRT en Eclipse. Android **limita el número de métodos máximos de una aplicacion a 65K** [36] por cuestiones de eficiencia. Para evitar esta limitación, durante el proceso de construcción el SDK de Android utiliza, entre otras, una herramienta llamada **ProGuard** [37]. Esta herramienta se encarga de optimizar el código de la aplicacion buscando remover clases que no se utilizan y ofuscando el código para prevenir la ingeniería inversa. En el caso de SwellRT, el código posee un gran número de clases java necesarias para desplegar el servidor y el cliente de la herramienta, por lo que es necesaria dicha optimización de código realizada por ProGuard. El sistema de compilación de aplicaciones de Android tiene dos formas: compilacion de la aplicacion en modo debug (para hacer pruebas cuando todavía se encuentra en fase de desarrollo) y en modo release (la aplicación se encuentra en su versión final y se empaqueta y se firma digitalmente para lanzarla al público). En el caso de Eclipse, ProGuard solo se ejecuta cuando se construye en modo release, por lo que cuando se intenta compilar una aplicación con tantas clases como SwellRT mientras se desarrolla (modo debug) el sistema da error y no se puede compilar el código para probarlo en el emulador.

La solución que encontramos fue desarrollar en Eclipse (por las facilidades que el entorno proporciona para escribir código) pero realizar el proceso de construcción del código por consola de comandos, ya que en este caso sí que se puede compilar la aplicación en modo debug utilizando ProGuard.

Proceso de Construcción por Consola

Para construir la aplicación por consola de comandos, Android utiliza la herramienta Apache Ant [**?**] para automatizar el proceso de construcción [**?**]. Es importante asimismo tener definida la variable de entorno JAVA_HOME con la ruta de acceso al JDK de java instalado en la máquina. Conviene también, por comodidad a la hora de trabajar con la consola, añadir al PATH del sistema las rutas a la carpeta donde esta el SDK de android (/sdk) y dentro de esta ruta añadir asimismo rutas a las carpetas /tools y /platform-tools.

Existen dos formas de realizar la construcción en modo debug de una app:

1 - Sin tener previamente lanzado un emulador o conectado al ordenador un dispositivo android en modo debug [**?**]:

En este caso es necesario construir la aplicación y luego lanzar el emulador para después instalar la aplicación en él. Para construir la aplicacion en modo debug nos vamos a la carpeta raíz de nuestro proyecto y ejecutamos el siguiente comando:

```
$ ant clean debug
```

Esto nos generará una aplicacion instalable en el directorio /bin del proyecto bajo el formato que Android usa para sus aplicaciones (.apk). El siguiente paso es ejecutar un emulador o conectar un dispositivo android por USB. Para ejecutar un emulador, abrimos otra consola y utilizamos el siguiente comando:

```
$ android avd
```

Lo que nos despliega la herramienta Android Virtual Device Manager (Ver Seccion 2.4.2) para que elijamos/creemos el emulador que queremos ejecutar. Podemos elegir multitud de parámetros [**?**] para el dispositivo que emula (resolución y tamaño de pantalla, de memoria Ram, elementos hardware emulados, etc.) siendo lo más importante elegir un API (versión de Android) que se corresponda con el API que hemos elegido para nuestra aplicación (en nuestro caso API 19). Es recomendable también elegir una imagen del sistema que use un procesador con arquitectura Intel x86, ya que si elegimos la opción por defecto de ARM (los dispositivos móviles actuales usan procesadores ARM) la ejecución del emulador se ralentiza mucho al tener que emular una arquitectura de procesador distinta a la suya (los ordenadores actuales usan arquitectura Intel x86 en su mayoría). Esto únicamente afecta

al rendimiento del emulador, la aplicación es independiente de la arquitectura que haya por debajo.

Una vez lanzado el emulador/dispositivo móvil, procedemos a instalar la aplicación en él ejecutando el siguiente comando en la primera consola (en la que construimos la aplicacion):

```
$ adb install XXXX.apk
```

Siendo XXXX la ruta a donde se encuentra el .apk de la aplicación que previamente hemos construido (/bin). La herramienta ADB (Android Debug Bridge) [**?**] es la que permite la comunicación entre el proceso de la consola de comandos y el emulador/dispositivo móvil. Es importante destacar que si se tienen varios emuladores/dispositivos moviles en ejecucion/conectados hay que especificar en cual se quiere instalar la aplicación añadiendo al comando lo siguiente: **-s emulator -YYYY** siendo esto último el identificador del emulador que podemos encontrar en el título de la ventana del emulador.

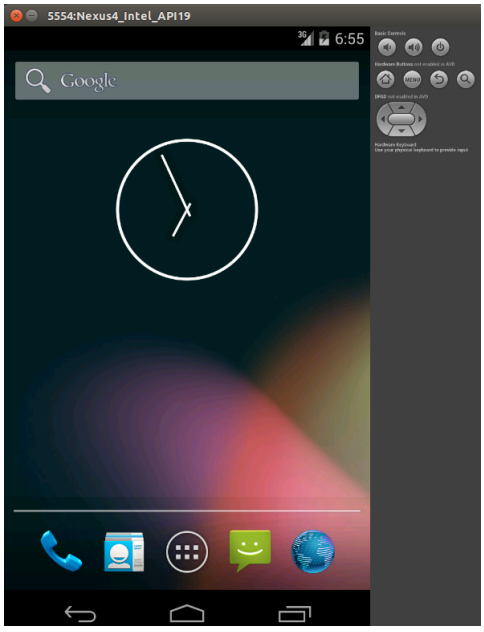

Figura 2.4: Emulador Android API 19

De esta manera podemos probar la aplicación, que será lanzada en el emulador/dispositivo una vez termine su instalación.

2 - Teniendo un emulador previamente lanzado (ver sección anterior para ver cómo se lanza) o un dispositivo móvil ya conectado por USB:

En este caso es todavía más sencillo el proceso de construcción. Nos vamos a la carpeta raíz del proyecto y podemos compilar e instalar la aplicación con un solo comando:

```
$ ant debug install
```

Es importante destacar que este comando solo funciona si tenemos un único emulador o dispositivo conectado, de lo contrario habrá que utilizar el método anterior.

Proceso de Depuración

Una vez instalada una aplicación, podemos depurar su código en ejecución usando la herramienta DDMS del ADT en conjunto con la vista de Debug de Eclipse. Pero antes hay que especificar qué aplicación queremos depurar de las que puedan estar instaladas en el dispositivo o emulador.

En el caso del emulador debemos lanzar la aplicación llamada “Dev Tools” y abrir el menú “Developer Options”. Dentro de este menú habilitaremos las opciones de “USB debugging” y de “Wait for debugger”. Además pulsaremos sobre “Select Debug app” y seleccionaremos la aplicación que queremos depurar.

En el caso de un dispositivo Android debemos ir a los Ajustes del dispositivo y seleccionar el menú de Opciones de Desarrollador. Aquí habilitamos las opciones de ”Depuración de USB”(si no esta habilitada ya) y de .Esperar al depurador”. Además pulsamos donde pone ”Seleccione una aplicación para depurarz elegimos la aplicación que queremos depurar.

Una vez hecho esto, cada vez que ejecutemos la aplicacion saldrá un mensaje de advertencia y se quedará esperando a que conectemos un depurador para continuar con su ejecución. Para esto, nos vamos a Eclipse y abrimos la vista de DDMS. Aquí nos aparecerá, entre otras cosas, un espacio con todos los procesos en ejecución en el dispositivo/emulador. Localizamos el proceso de nuestra aplicación y pulsamos sobre el bichillo verde para conectar el

depurador a ella. Llegados a este punto la aplicacion continua su ejecución en el emulador y aparece un escarabajo verde al lado del proceso de la app en la ventana de DDMS, que indica que se esta depurando ese proceso. Es entonces cuando podemos abrir la vista de depuración de Eclipse y proceder a trabajar con breakpoints para depurar y estudiar el código con el fin de solucionar errores.

2.4.3. Migración: Identificación y Solución de Problemas

El objetivo de esta parte del proyecto es conseguir que el cliente de SwellRT se pueda desplegar en Android para asi conseguir que se conecte al servidor WIAB que tambien incluye. Para ello lo primero que haremos será desplegar el servidor en nuestro ordenador clonando el repositorio de GitHub de SwellRT y siguiendo los pasos descritos en el Readme del proyecto [23]. Para comprobar que el servidor se ha instalado correctamente, podemos ejecutarlo por consola (ver Readme) y abrir un navegador web con la dirección http://localhost:9898. Si nos aparece una ventana de Login de WIAB es que ya tenemos un servidor WIAB corriendo en nuestro ordenador. Creamos entonces un usuario y contraseña de prueba. Este paso es importante ya que la aplicación Android intentará conectarse contra este servidor mientras estemos haciendo pruebas de desarrollo.

A continuación crearemos un proyecto Android en Eclipse e incluiremos en él todas las clases de SwellRT. Uno de los componentes principales de Android a la hora de desarrollar son las **Actividades** [**?**], que representan las pantallas que se le muestran al usuario y que responden a su interacción programáticamente. Por tanto, crearemos una nueva actividad principal (waveAndroid.java) que se ejecutará al lanzar la aplicación y que por el momento intentará conectarse al servidor especificando por código el usuario y contraseña que hemos creado antes en el servidor. Wave realiza este login contra el servidor usando dos tecnologias: HTTP [**?**] y WebSockets [**?**].

Conexion HTTP

Wave fue desarrollado para utilizar el protocolo WebSocket para la conexión al servidor, pero esta tecnología necesita realizar una autenticación HTTP previa. Lo primero que haremos será otorgar **permisos de conexión a internet** a nuestra aplicación. Android utiliza un **sistema de permisos** [**?**] para controlar los privilegios de cada aplicación. Estos permisos se declaran

en el **Manifiesto** de la aplicación [**?**], archivo que declara sus características. Para ello basta con añadir lo siguiente al manifest.xml de la aplicación:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Tambien hay que tener en cuenta que cuando nos encontramos en el emulador no estamos en la misma red que el ordenador en el que trabajamos, por lo que la conexion a la URL http://localhost:9898 no es valida. No obstante, esto tiene facil solucion pues **el emulador de Android define unas direcciones IP de red especiales** [**?**] para este tipo de casos. Basta con sustituir localhost por la direccion 10.0.2.2 para conseguir acceder al servidor WIAB desplegado en el ordenador. La direccion URL sera por tanto: **http://10.0.2.2:9898**.

Lo siguiente que haremos será ejecutar el código de Login del cliente SwellRT para intentar localizar dónde se lleva a cabo la conexion HTTP. Para ello llamamos desde la actividad principal (WaveAndroid.java) al método startSession() de la clase WaveClient.java pasándole el usuario y la contraseña antes creados.

Esto provoca un error de ejecución y la aplicación se cierra. Lo siguiente que hacemos es depurar la aplicación (Ver Seccion 2.4.2) estudiando el LogCat [**?**] (Ver Figura 2.5) para ver donde se produce el error. Descubrimos que el problema estaba localizado en el método login() de la misma clase, que intentaba realizar una **peticion POST HTTP** al servidor utilizando un **RequestBuilder** de la libreria **com.google.gwt.http.client**. He aquí el primer problema: la actual conexión utiliza métodos de GWT/Javascript para hacer la petición post y Android no es compatible con esta tecnología.

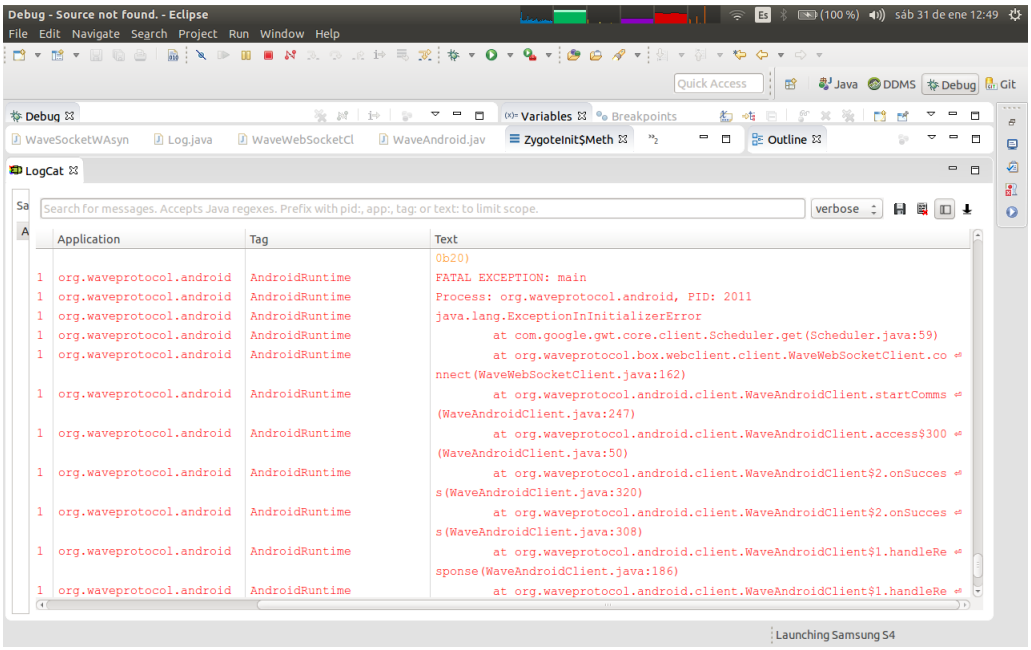

Figura 2.5: Ejemplo de Traza de Error en Logcat

Hay por tanto que encontrar una librería similar compatible con Android que construya una peticion **HTTP POST** y la envie al servidor. La primera opcion que valoramos fue utilizar la **libreria HTTP Apache** [**?**], incluida en el SDK de Android desde sus primeras versiones. Sin embargo, Google recomienda [**?**] a partir del API 10 (Android 2.3 "Gingerbread") utilizar la **libreria HttpURLConnection** [**?**]. Por tanto esta ultima es la que elegimos para la migracion.

De forma simplificada, este seria un esquema de la nueva estructura del login HTTP:

```
import java.net.HttpURLConnection;

private void login(final String user, final String password,
    final Callback<String, String> callback) {

                //Construct the URL String urlStr with the
                    server, user and password parameters

            URL url = new URL(urlStr); //String
            HttpURLConnection connection = (HttpURLConnection
                ) url.openConnection(); //Open the connection
                to the given URL
```

```
            connection.setDoOutput(true); // allow the POST
                connection
            connection.setRequestProperty("Accept-Charset",
                CHARSET);
            connection.setRequestProperty("Content-Type", "
                application/x-www-form-urlencoded;charset="
                 + CHARSET);

            OutputStream out = connection.getOutputStream();
            out.write(queryStr.getBytes(CHARSET)); //Set the
                POST parameters

            if (connection.getResponseCode() != 200) {
                      //ERROR during the connection
                      connection.disconnect(); //Disconnect
                          from the server.
            } else {
                      //Continue with the login process (
                          WebSocket)
                      connection.disconnect(); //Disconnect
                          from the server.
            }
}
```

Sin embargo, aquí no acaba el problema. Por cuestiones de usabilidad y de respuesta a la interacción del usuario, Android establece dos reglas para trabajar con el proceso de la actividad que se le esta mostrando al usuario (llamado **UI Thread**) [**?**]:

- **1. No bloquear el UI Thread**
- **2. No acceder al UI Thread directamente desde otro Thread**

La conexión a un servidor es un proceso susceptible de durar un tiempo variable según las condiciones de la red, lo cual deja la aplicación en espera hasta que se realiza dicha conexión, bloqueando el UI Thread. Por tanto, decidimos usar un hilo (Thread) por separado en forma de **AsyncTask** [**?**] para llevar a cabo la tarea de Login, tal y como recomienda Google hacer para trabajar con conexiones a la red [**?**]. La ventaja por tanto de usar otro hilo para esto es que la actividad principal no se bloquea.

El siguiente es un esquema del AsyncTask encargado del Login:

```
private class LoginTask extends AsyncTask<String, Void,
    String> {
```

```
    @Override
    protected String doInBackground(String... params) { //
        method that executes on the new Thread without
        blocking the UI Thread
      login(params[0], params[1], params[2]); //Do the login
    }

    @Override
    protected void onPostExecute(String result) { //method
        that executes on the UI Thread once doInBackground()
        finishes its execution.

      if (result != null) {
        callback.onLogin(); //Notify the login success using
            the proper callback method
      } else { //The doInBackGround method has had a problem
          and the result of its execution was null
        callback.onError("Wave Login Error"); //Notify the
            login error using the proper callback method
      }
    }
}
```

Es importante tambien destacar que la arquitectura de SwellRT y de Wave esta planteada de manera que utiliza llamadas asíncronas (callbacks) para notificar al resto de la aplicacion del resultado de los procesos de conexión al servidor.

Llegados a este punto, tenemos un proceso de login Http que hace uso de la libreria HttpUrlConnection y de un AsyncTask para realizar esa primera conexión al servidor. Depuramos la aplicación y comprobamos que efectivamente el login http se realiza correctamente (la respuesta del servidor tiene código 200). **Sin embargo la aplicacion aun no funciona correctamente, pues se cierra al intentar ejecutar el código que se encarga del siguiente paso del login: la conexión por WebSocket.**

Conexion WebSocket

Login

2.4.4. Organización y Resultados

Servicio Android

Resultado de la Migracion

Diagramas y Dependencias

Capítulo 3

Creación de aplicación Android (NOMBRE DE APP)

3.1. 1ª Parte: Programas Políticos

3.1.1. Estado del Arte

3.1.2. Intencion

3.1.3. Objetivos

3.1.4. Usabilidad

3.2. 2ª Parte: Propuestas y Comparativas

3.2.1. Estado del Arte

3.2.2. Intencion

3.2.3. Objetivos

3.2.4. Usabilidad

3.3. Tecnologias y Metodologías de la app

3.3.1. Base de Datos

3.3.2. Service REST

3.3.3. Frontend

Capítulo 4

Resultados y Conclusiones

4.1. Discusion de Resultados

4.2. Conclusiones

Capítulo 5

Trabajo a Futuro

5.1. Mejoras

Bibliografía

[1] Inc. Google. Meet Google Wave.
http://googlecode.blogspot.com.es/2009/05/hello-world-meet-google-wave.html.

[2] Inc. Google. Google Wave Federation Protocol Over XMPP.
http://wave-protocol.googlecode.com/hg/spec/federation/wavespec.html.

[3] Inc. Google. End of Google Wave.
https://support.google.com/answer/1083134?hl=en.

[4] Google Docs.
https://drive.google.com/.

[5] Google+.
https://plus.google.com/.

[6] Apache. Apache Wave (Incubating).
http://incubator.apache.org/wave/about.html.

[7] Apache. Apache License 2.0.
http://www.apache.org/licenses/LICENSE-2.0.

[8] Inc. Google. Wave Federation.
http://www.waveprotocol.org/federation.

[9] Google. Google Wave Federation Architecture White Paper.
http://wave-protocol.googlecode.com/hg/whitepapers/google-wave-architecture/google-wave-architecture.html.

[10] Peter Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. 2011.
RFC 6120 Available at http://tools.ietf.org/html/rfc6120.

[11] Understanding Operational Transformation.
http://www.codecommit.com/blog/java/understanding-and-applying-operational-transformation.

[12] Inc. Google. Google Wave Operational Transformation.
http://www.waveprotocol.org/whitepapers/operational-transform.

[13] Inc. Google. Google Wave Conversation Model.
https://wave-protocol.googlecode.com/hg/spec/conversation/convspec.html.

[14] Inc. Google. Google Wave API Overview.
http://www.waveprotocol.org/wave-apis.

[15] Inc. Google. Google Wave Extensions.
http://www.waveprotocol.org/wave-apis/extensions.

[16] Apache. Wave In A Box.
http://www.waveprotocol.org/wave-in-a-box/.

[17] Oracle. OpenJDK.
http://openjdk.java.net/.

[18] Inc. Google. Google Web Toolkit.
http://www.gwtproject.org/.

[19] Apache. Install WIAB.
https://cwiki.apache.org/confluence/display/WAVE/Install+WIAB.

[20] Apache. WIAB Repository.
https://github.com/apache/incubator-wave.

[21] WIAB Server Example.
http://waveinabox.net/.

[22] P2PValue. P2P Value European Project.
http://www.p2pvalue.eu/.

[23] P2PValue. SwellRT, a real-time federated collaboration framework.
https://github.com/P2Pvalue/swellrt.

[24] Inc. Google. Android Developers Site.
http://developer.android.com/index.html.

[25] Inc. Google. Android SDK.
https://developer.android.com/sdk/index.html.

[26] Eclipse Foundation. Eclipse IDE.
https://eclipse.org/ide/.

[27] Inc. Google. Android Studio IDE.
https://developer.android.com/tools/studio/index.html.

[28] JetBrains. IntelliJ IDEA IDE.
https://www.jetbrains.com/idea/.

[29] Google Inc. Eclipse Android Development Tools (ADT).
https://developer.android.com/tools/help/adt.html.

[30] Google Inc. Android API Reference.
http://developer.android.com/reference/packages.html.

[31] Google Inc. Android Virtual Device Manager.
http://developer.android.com/tools/devices/managing-avds.html.

[32] Google Inc. Using the Android Emulator.
http://developer.android.com/tools/devices/emulator.html.

[33] Google Inc. Android SDK Manager.
https://developer.android.com/tools/help/sdk-manager.html.

[34] Google Inc. Android Dalvik Debug Monitor Server.
http://developer.android.com/tools/debugging/ddms.html.

[35] Google Inc. Android Versions Distribution.
https://developer.android.com/about/dashboards/index.html.

[36] Google Inc. Building Apps with Over 65K Methods.
https://developer.android.com/tools/building/multidex.html.

[37] Google Inc. ProGuard.
http://developer.android.com/tools/help/proguard.html.