

Universidad Nacional de La Plata

Facultad de Informática

Programación Distribuida 2014

=====

Trabajo Final

ReactiveJade

Integrando JADE con una aplicación móvil React Native

Coordinador

Prof. Dr. Fernando Tinetti

Disertante

APU Lucio Flavio Di Giacomo Noack

Febrero 2019

0 Resumen

El presente informe describe la incorporación y utilización del framework JADE en una aplicación móvil desarrollada con el framework React Native, de manera tal de poder evaluar su instalación, configuración y ejecución en entornos heterogéneos que incluyen dispositivos móviles. La implementación propuesta como caso de estudio involucra a un agente que reporta el estado de la plataforma cada n segundos, reflejando el estado de la misma a medida que se agregan o quitan contenedores, aportando información acerca del dispositivo sobre el que se ejecuta cada contenedor, y reportando aquellos casos en el que el agente no puede migrar hacia algún contenedor.

1 Introducción

Los dispositivos móviles constituyen un mercado cada vez más significativo para el desarrollo de aplicaciones. Sólo durante el mes de Enero de 2018 se registraron 3.700.000.000.000 usuarios únicos de Internet desde dispositivos móviles, representando el 80% del uso de Internet a nivel global. El surgimiento de frameworks novedosos que facilitan significativamente el desarrollo de aplicaciones móviles, como es el caso de React Native, también influyen en la proliferación de éstas. En este sentido, resulta imperativo pensar o repensar las tecnologías vigentes y emergentes en un contexto de dispositivos heterogéneos, donde la movilidad y la portabilidad son requerimientos fundamentales.

La plataforma Java, a través de su Java Virtual Machine (JVM), ofrece soluciones para los problemas de portabilidad en una amplia variedad de dispositivo, a partir de su eslogan “Write Once, Run Anywhere”, que establece que el software desarrollado con Java puede ejecutarse en cualquier dispositivo capaz de correr una JVM. Sin embargo, la existencia de derivaciones de la Java Virtual Machine hacen que esto no siempre sea cierto; por ejemplo, Android, el sistema operativo para dispositivos móviles más popular, implementa su propia versión de la JVM: la Dalvik Virtual Machine (DVM). Esta nueva implementación agrega funcionalidades propias del dominio de problemas que enfrenta un sistema operativo móvil, a la vez que restringe o quita algunas de acuerdo al criterio de los desarrolladores, principalmente tomando en cuenta aspectos de seguridad y privacidad.

El framework JADE implementa un middleware para simplificar el desarrollo de sistemas distribuidos multi-agente de acuerdo a las especificaciones de FIPA. Resultan sumamente atractivas las propuestas de FIPA en general, y de JADE en particular. Esta última propone una API concisa, desarrollada en Java, para la administración de plataformas,

contenedores, agentes, mensajería y comportamientos, permitiendo al desarrollador un mayor nivel de abstracción en función del problema a resolver, representándolo con las entidades provistas por el framework y enfocándose en **qué** comunicar y no en **cómo** hacerlo, delegando esto último al middleware.

Desde esta perspectiva surge el interrogante: ¿es posible desarrollar una aplicación compatible con Android utilizando JADE? De acuerdo a los desarrolladores, sí, y en función de esta respuesta ponen a disposición el trabajo del proyecto LEAP, que es compatible con el desarrollo de aplicaciones para Android en sus primeras versiones, pero que presenta problemas al integrarlo con las últimas, además de la falta de funcionalidades que puedan hacer atractivo a un desarrollo actual.

En base a este contexto y con el interrogante aún sin resolver, nos proponemos cursar y documentar una experiencia de uso de JADE en un entorno móvil moderno, integrándolo con las últimas versiones de Android dentro de una aplicación desarrollada con React Native, con el objetivo de demostrar que es posible realizarlo y, además, que resulta atractivo el desarrollo de sistemas distribuidos de esta forma.

2 Propuesta y desarrollo

2.1 Propuesta

La propuesta de este trabajo consiste en el desarrollo de una aplicación móvil que utilice un agente de JADE capaz de moverse entre los contenedores de la plataforma consultando información sobre cada dispositivo y reportándolo al contenedor origen. Para ello, integraremos el desarrollo original de este agente con una aplicación móvil desarrollada con React Native que permita configurar, iniciar y detener un contenedor, además de instancias del agente previamente mencionado.

El desarrollo cuenta con dos etapas: una etapa de implementación del agente que reporta el estado del hardware del contenedor, y una etapa de integración con la aplicación React Native. A fin de simplificar su referencia, llamaremos HardwareSniffer al agente, y ReactiveJade a su integración.

2.2 Desarrollo

2.2.1 HardwareSniffer

El desarrollo de HardwareSniffer involucra la definición de un agente capaz de recorrer los distintos contenedores de su plataforma y consultar datos sobre el estado del hardware del dispositivo asociado. Debe reportar la carga promedio del sistema, la cantidad de memoria física total y libre, la cantidad de memoria virtual total y libre, el nombre del sistema operativo y de la implementación de la máquina virtual. Para ellos, desarrollamos las siguientes clases e interfaces.

- **HardwareSniffer.** Es una clase abstracta que define los métodos que debe satisfacer una implementación de HardwareSniffer, garantizando que todos serán capaz de brindar la información solicitada. Los métodos para obtener la cantidad de memoria virtual total y libre, el nombre del sistema operativo y de la implementación de la máquina virtual son comunes a todas las versiones de las máquinas virtuales, por lo que pueden ser implementadas en esta clase mediante la consulta de propiedades de las clases Runtime y System.
- **HardwareSnifferManager.** Es una clase abstracta que implementa un Singleton con la instancia de implementación de HardwareSniffer propia del sistema operativo en el que se encuentra el contenedor del agente. Esto es un desarrollo clave en el proyecto, ya que apunta a resolver las diferencias entre la JVM y la DVM de Android.
- **HardwareSnifferManagerImpl.** Es la implementación del HardwareSnifferManager propia de cada sistema operativo. que instancia y devuelve el HardwareSniffer correspondiente. Existe una implementación general, y otra propia de Android.
- **LinuxHardwareSniffer.** Es una clase que implementa los métodos para obtener la información en dispositivos con sistema operativo Linux. Accede a la información disponible en los pseudoarchivos /proc/loadavg y /proc/meminfo.
- **AndroidHardwareSniffer.** Es una clase que implementa los métodos para obtener la información en dispositivos con sistema operativo Android. Si bien Android se basa en Linux, los desarrolladores han decidido bloquear por motivos de seguridad el acceso a la información del archivo /proc/loadavg (no así de /proc/meminfo) desde el API Level 24 (Android 7.0 - Nougat), por lo que no es posible realizar ninguna consulta programática relacionada con los procesos, incluida la carga promedio del

sistema. Es por ello que para salvar esta situación devolvemos un valor fijo 14,14 hasta que el equipo desarrollador del sistema operativo ofrezca una alternativa.

- **HardwareSnifferAgent.** Es una clase que implementa un agente que tiene como misión recorrer los contenedores de la plataforma, consultar información sobre el contenedor, y reportarlos en el contenedor origen. Para ello, realiza el siguiente proceso.
 - Consulta al AMS los contenedores disponibles en la plataforma y prepara un itinerario, excluyéndose del mismo.
 - Agregar un comportamiento cíclico `HardwareSnifferBehaviour`, que gestiona los movimientos entre contenedores de acuerdo al itinerario suministrado.
 - Elabora un reporte `HardwareSnifferReport` luego de moverse a un nuevo contenedor.
 - Al concluir con el itinerario, vuelve al origen.
 - Elabora un reporte en el contenedor origen.
 - Notifica el reporte a sus suscriptores.
 - Espera 10 segundos e iniciar nuevamente el proceso.
- **HardwareSnifferBehaviour.** Es una clase que implementa un comportamiento cíclico (`CyclicBehaviour`) y gestiona los movimientos del agente entre contenedores. Si bien originalmente se había implementado como un `TickerBehaviour`, resulta conveniente utilizar un comportamiento cíclico ya que de esta forma el recorrido se realiza tan rápido como lo permita la plataforma. El comportamiento acepta una lista de contenedores a recorrer, y los elimina en función del recorrido. Cuando la lista está vacía, vuelve al origen. Adicionalmente, maneja el caso excepcional en el cual la plataforma sólo tiene un contenedor (y por lo tanto, no se ejecuta el método *afterMove* del agente), reconociendo esta situación y finalizando el recorrido.
- **HardwareSnifferReport.** Es una clase que implementa una instancia serializable para permitir que el agente traslade su listado de reportes en su recorrido por la plataforma. Incluye los datos del estado del dispositivo, además de la fecha de generación del reporte y el nombre del contenedor. Adicionalmente, notifica aquellos contenedores a los que el agente no pudo trasladarse por diversos motivos.
- **HardwareSnifferReporter.** Es una clase que implementa métodos para mostrar en consola los reportes generados, siendo suscriptor de las notificaciones enviadas por el agente.

2.2.2 ReactiveJade

Este paquete incluye aquellas clases e interfaces necesarias para integrar el desarrollo JADE con la aplicación React Native. Principalmente se encarga de gestionar la interacción entre el agente JADE y su entorno, y el módulo de React Native y el entorno Android, de manera tal que sean independientes entre sí. Esta interacción se realiza mediante un sistema de suscripciones, donde se provee una interface que cada implementación debe satisfacer para poder notificarse de los eventos emitidos por los agentes compatibles con ReactiveJade en las distintas etapas de ejecución de sus tareas. Las clases e interfaces implementadas son las siguientes.

- **ReactiveJadeAgent.** Es una clase abstracta que hereda de Agent del paquete JADE, e implementa la interface ReactiveJadeEventEmitter, la cual establece el requerimiento de implementar la notificación a sus suscriptores. Además, requiere la implementación del método *onMoveError*, que delega en el agente la responsabilidad de actuar frente a un error al moverse entre contenedores.
- **ReactiveJadeEvent.** Es una clase que permite gestionar la comunicación de eventos entre emisor y suscriptores. Permite asignar un nombre al evento para la posterior discriminación de suscriptores, y los parámetros adjuntos como una instancia de ReactiveJadeMap.
- **ReactiveJadeEventEmitter.** Es una interface que establece la implementación de un método que notifique a los suscriptores la ocurrencia de un evento.
- **ReactiveJadeMap.** Es una clase que implementa una extensión de la clase HashMap, con clave String y valor Object. Es una implementación alternativa compatible con WritableMap, la clase que permite realizar la comunicación entre el código Java y el código JavaScript en una aplicación React Native
- **ReactiveJadeSuscribable.** Es una interface que establece la implementación de un método de recepción de eventos ReactiveJadeEvent.
- **ReactiveJadeSubscription.** Es una clase que modela la suscripción de instancias suscriptoras a los eventos emitidos por instancias de emisores. Recorre la lista de suscriptores y notifica el evento.
- **ReactiveJadeSubscriptionService.** Es una clase que enumera los Singleton de suscripciones, creándolas cuando sea necesario, y delegando la resolución de las notificaciones.

Adicionalmente, la implementación para Android cuenta con la siguiente clase auxiliar.

- **ReactiveJadeMapConverter.** Es una clase auxiliar que permite convertir una instancia de ReactiveJadeMap en WritableMap, con el objetivo de simplificar el código en el módulo React Native.

2.2.3 React Native

Para la implementación de la aplicación móvil con React Native se utilizaron los paquetes react-native-device-info, que suministra información sobre el dispositivo, y react-native-elements, que brinda componentes para la interfaz gráfica.

La aplicación cuenta con una pantalla principal desde la cual se puede acceder a la configuración de los parámetros de la plataforma y el contenedor, especificando la dirección IP y el puerto, y el nombre deseado de contenedor. A partir de estos parámetros, al presionar el botón “Start Container” se intenta crear un nuevo contenedor en la plataforma. En caso de éxito, se muestran los botones “Stop Container” (que detiene al contenedor y todos sus agentes contenidos) y “Start Agent” (que inicia un agente HardwareSnifferAgent en el contenedor local). Si el inicio del agente es exitoso, comienza con la ejecución de su comportamiento, recorriendo los contenedores de la plataforma. Al finalizar cada recorrido, se agregar un nuevo listado de reportes al listado de recorridos, donde se muestra en un modal la información sobre cada contenedor.

A fin de simplificar la elaboración de este informe, adjuntamos capturas de la aplicación en el apartado de instrucciones de uso.

2.2.4 Scripts

Con el objetivo de simplificar la incorporación de cambios y ejecución del proyecto, se facilitan scripts que incluyen instrucciones para la compilación, copia de archivos, y ejecución de entornos preconfigurados. En las carpetas standalone/reactivejade y standalone/hardwaresniffer se encuentran los scripts build.sh, que compilan cada paquete (siendo el segundo dependiente del primero), y dentro de standalone/bin se encuentra otro script build.sh que auna a los anteriores, además de scripts start-* que inician una instancia de JADE con determinadas configuraciones predefinidas.

2.3 Notas adicionales

Es importante considerar una serie de aspectos referidos al desarrollo en contraste con la propuesta original.

2.3.1 Manejo de errores

El framework JADE no cuenta con características *out-of-the-box* para el manejo de los errores que pueda enfrentar un agente al moverse o clonarse entre contenedores. Todos los casos de error son capturados y tratados internamente por el middleware, principalmente por la clase `jade.core.mobility.AgentMobilityService`, y únicamente se notifican al usuario desarrollador por medio del logger.

Para solventar esta situación y poder implementar un reporte de errores mínimo, se ejecutaron varias estrategias:

- Se analizó el código fuente del framework para observar qué errores arrojan los métodos relacionados con la movilidad, *doMove* y *doClone*. Estos métodos no arrojan errores, y están anotados con FIXME para su posterior implementación. Es por ello que se reescribieron los métodos para poder propagar las excepciones, sin embargo todas son capturadas por el `AgentMobilityService` y notificadas por medio del logger.
- Se implementó un handler para capturar los mensajes del logger, interpretarlos y notificar los errores. Sin embargo, estos mensajes son genéricos; en el caso del método *doMove* y sus invocaciones asociadas, si bien se especifica el contenedor destino no se identifica al agente, por lo tanto, es imposible asociar un error con un agente.
- Se implementó una solución *ad-hoc*. Los agentes saben cuál es el último contenedor al que intentaron moverse. Al querer moverse nuevamente, se verifica si el último intento se corresponde con la ubicación actual: si no son iguales, entonces ocurrió un error al moverse y se notifica al agente, el cual elabora un informe de error. Esta solución está funcionando correctamente, y en el setup de cada agente se crea un contenedor *fakeContainer* con información incorrecta, para verificar la generación de los informes de error.

2.3.2 Migración de agentes entre plataformas

La migración de agentes entre plataformas involucra desarrollos que quedan fuera del alcance de este trabajo. Con esta implementación, no es posible que un agente pueda moverse a un contenedor registrado en una plataforma foránea. Para realizar dicha implementación puede utilizarse el trabajo realizado por el Inter-Platform Mobility Project.

2.3.3 Compatibilidad

Por último, debido a las constantes cambios implementados por el grupo de desarrolladores de Android, la aplicación móvil generada sólo es compatible con las versiones actuales de este sistema operativo, teniendo como requerimiento mínimo poseer Android 7.0 - Nougat (API Level 24), disponible desde el año 2016.

3 Compilación e instalación

A continuación se describen los pasos para compilar e instalar la aplicación. El proyecto cuenta con dos desarrollos: el agente JADE y la aplicación móvil, siendo la segunda dependiente de la primera. Para la realización de estos pasos utilizaremos una computadora con sistema operativo Linux.

3.1 Compilación

3.1.1 Agente HardwareSnifferAgent

1. Descargar el repositorio ReactiveJade disponible en la siguiente URL de GitHub: <https://github.com/daniffig/ReactiveJade>. Esta descarga puede realizarse utilizando **git**, o como archivo comprimido .zip. En este último caso, es necesario descomprimir este archivo.
2. Moverse hacia la carpeta ReactiveJade donde se encuentra el código fuente del proyecto.
3. Dentro de la carpeta **standalone** encontraremos el código fuente del agente HardwareSnifferAgent. La aplicación cuenta con scripts para compilar fácilmente los archivos y sus dependencias. Para ello, ejecutamos el comando **sh standalone/bin/build.sh**

4. Las clases compiladas son empaquetadas en contenedores .jar y almacenadas en la carpeta **standalone/lib**, junto con la biblioteca JADE. Adicionalmente, el código fuente necesario por la aplicación móvil es copiado en la carpeta **android/app/src/main/java/com/reactivejade**. No es necesario que el desarrollador modifique estas clases. También se copian los contenedores .jar necesarios en **android/app/libs**, para ser identificados como dependencias por la aplicación React Native.
5. Para iniciar una instancia de JADE utilizando la IP de la red local, podemos ejecutar el comando **sh standalone/bin/start-platform-home.sh**. Este script cuenta con una versión alternativa que inicia automáticamente un agente HardwareSnifferAgent en el contenedor principal, y se invoca ejecutando el comando **sh standalone/bin/start-platform-home-with-agent.sh**.
6. Para modificar y agregar nuevas configuraciones a la instancia, pueden realizarse sobre los scripts mencionados en el punto 5.

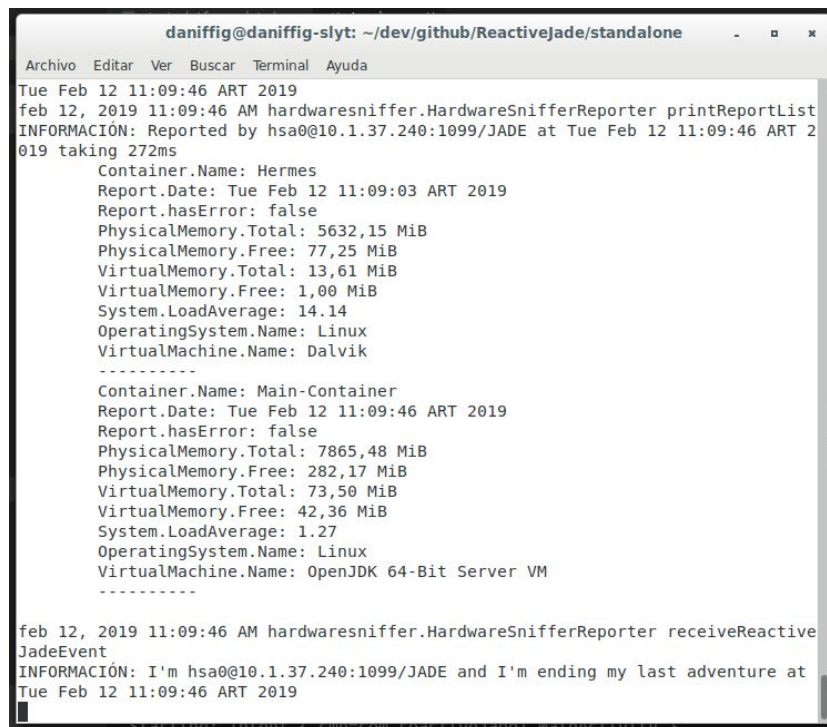
3.1.2 Aplicación móvil

1. Desde la carpeta descargada en el punto 1 de la sección anterior, contenedora del código fuente, ejecutar el comando **npm install** para resolver y descargar las dependencias del proyecto.
2. La compilación de un proyecto React Native implica la creación de un APK firmado. Esto implica generar un archivo de firma personalizado y asociarlo al proyecto. Los pasos para realizar dicho proceso pueden consultarse en la siguiente URL: <https://facebook.github.io/react-native/docs/signed-apk-android>
3. Una vez configurado el archivo de firma, moverse a la carpeta **android** y ejecutar el comando **./gradlew assembleRelease**. Es importante moverse a la carpeta **android**, de otro modo no encuentra la tarea assembleRelease.
4. El APK se genera en el archivo **android/app/build/outputs/apk/release/app-release.apk**, el cual debemos copiar a nuestro dispositivo móvil e instalar como una aplicación con la configuración **Otros orígenes del software**, ya que no proviene de la tienda de aplicaciones oficiales de Android.

IMPORTANTE: El APK generado a partir de estas instrucciones de uso está disponible en la carpeta dist del proyecto, firmado por el autor de este trabajo.

3.2 Configuración y uso

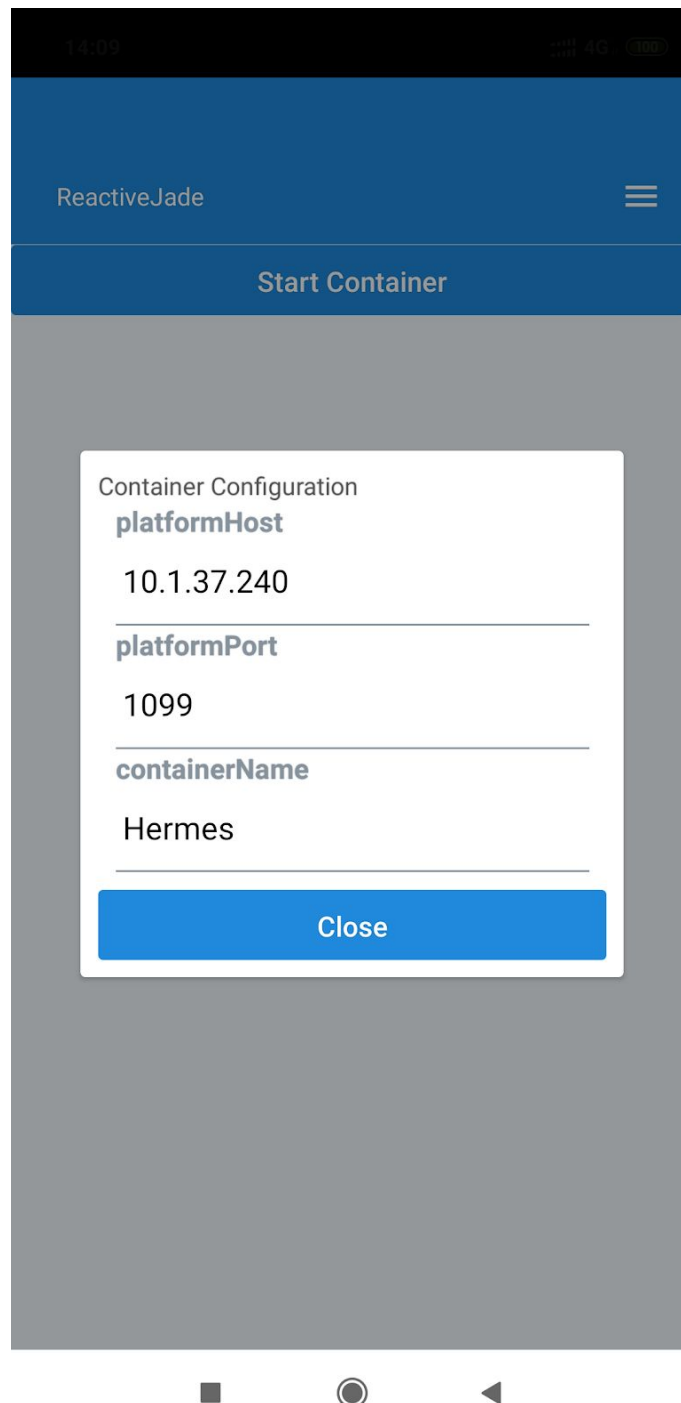
El agente HardwareSnifferAgent no requiere configuraciones adicionales más allá de aquellas propias de cualquier instancia de JADE, que quedan determinadas por el contexto de ejecución y las preferencias del usuario. Los reportes confeccionados por el agente se muestran en consola.

A screenshot of a terminal window titled 'daniffig@daniffig-slyt: ~/dev/github/ReactiveJade/standalone'. The terminal displays two reports from the HardwareSnifferReporter. The first report is for a container named 'Hermes', showing system metrics like physical and virtual memory, system load, and operating system details. The second report is for a container named 'Main-Container', showing similar metrics. The terminal also shows some log messages and a timestamp.

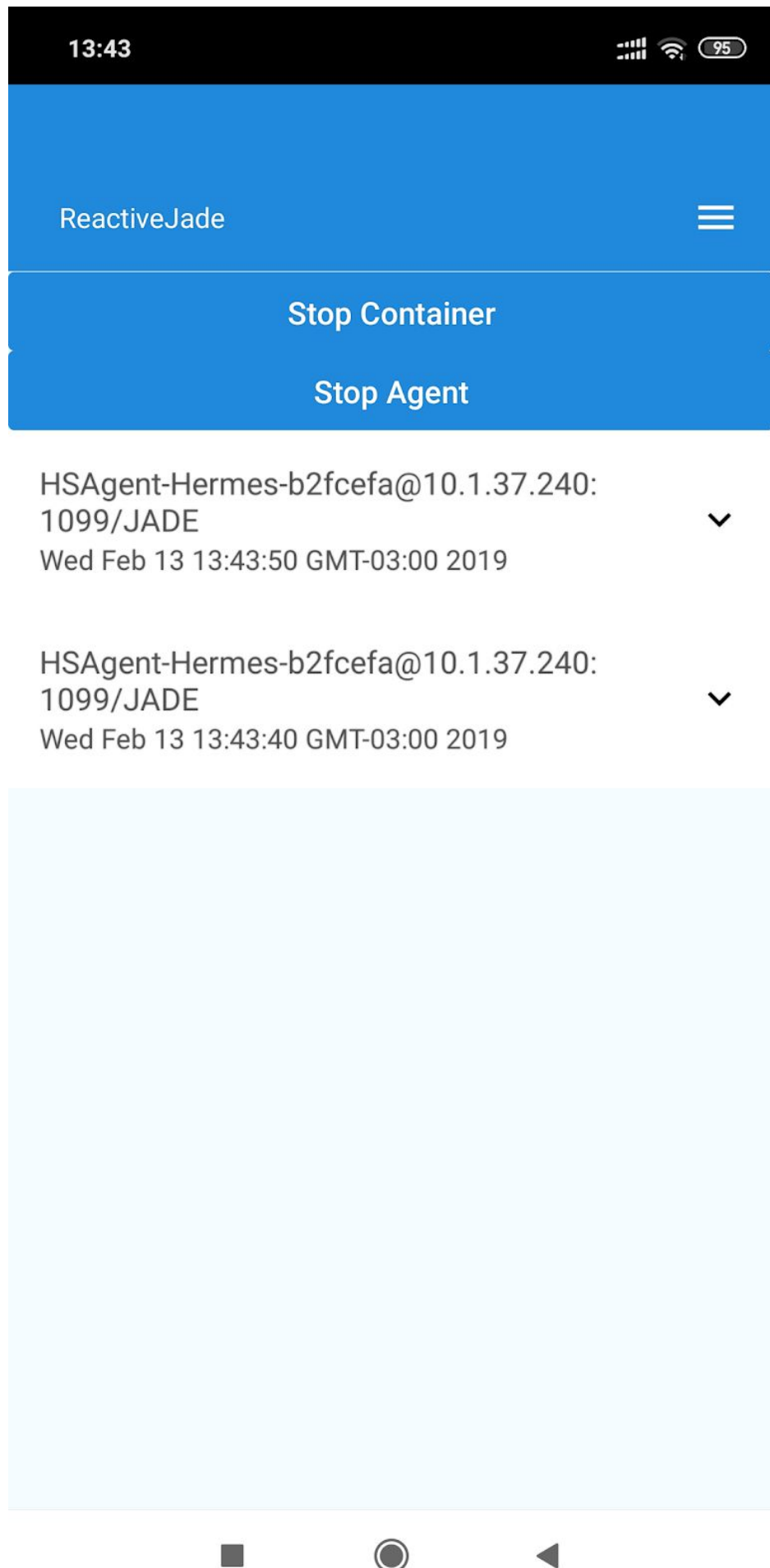
```
daniffig@daniffig-slyt: ~/dev/github/ReactiveJade/standalone
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Tue Feb 12 11:09:46 ART 2019
feb 12, 2019 11:09:46 AM hardwareSniffer.HardwareSnifferReporter printReportList
INFORMACIÓN: Reported by hsa0@10.1.37.240:1099/JADE at Tue Feb 12 11:09:46 ART 2019 taking 272ms
Container.Name: Hermes
Report.Date: Tue Feb 12 11:09:03 ART 2019
Report.hasError: false
PhysicalMemory.Total: 5632,15 MiB
PhysicalMemory.Free: 77,25 MiB
VirtualMemory.Total: 13,61 MiB
VirtualMemory.Free: 1,00 MiB
System.LoadAverage: 14.14
OperatingSystem.Name: Linux
VirtualMachine.Name: Dalvik
-----
Container.Name: Main-Container
Report.Date: Tue Feb 12 11:09:46 ART 2019
Report.hasError: false
PhysicalMemory.Total: 7865,48 MiB
PhysicalMemory.Free: 282,17 MiB
VirtualMemory.Total: 73,50 MiB
VirtualMemory.Free: 42,36 MiB
System.LoadAverage: 1.27
OperatingSystem.Name: Linux
VirtualMachine.Name: OpenJDK 64-Bit Server VM
-----
feb 12, 2019 11:09:46 AM hardwareSniffer.HardwareSnifferReporter receiveReactiveJadeEvent
INFORMACIÓN: I'm hsa0@10.1.37.240:1099/JADE and I'm ending my last adventure at Tue Feb 12 11:09:46 ART 2019
```

La aplicación móvil cuenta con un panel de configuración del contenedor accesible desde un botón ubicado en el vértice superior derecho de la pantalla del dispositivo. En el panel podemos configurar la dirección IP y el puerto de la plataforma, y el nombre del contenedor. Por defecto, este último toma el valor del nombre del dispositivo para las conexiones Bluetooth.

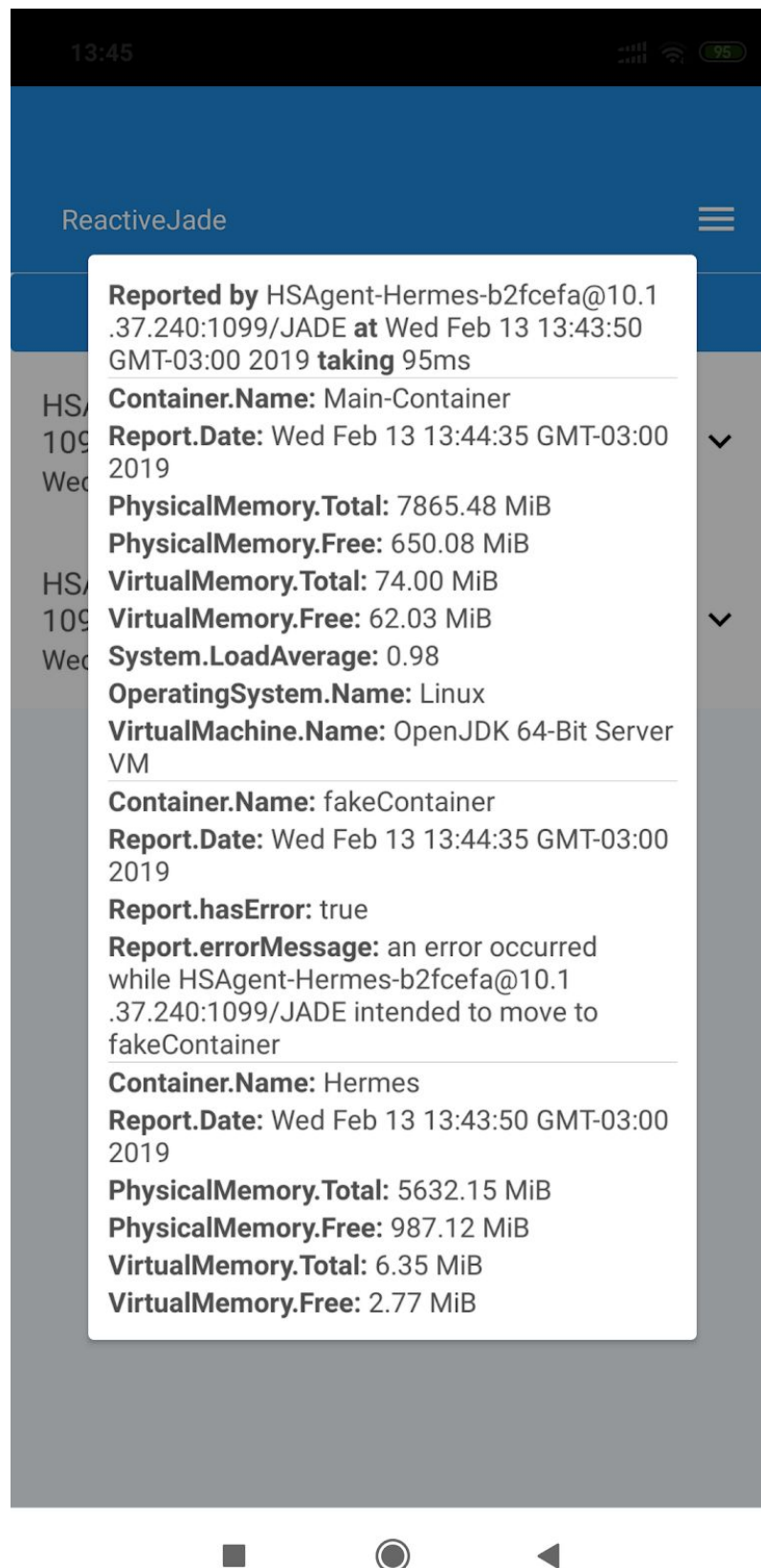
A continuación podemos ver una captura del panel de configuración.



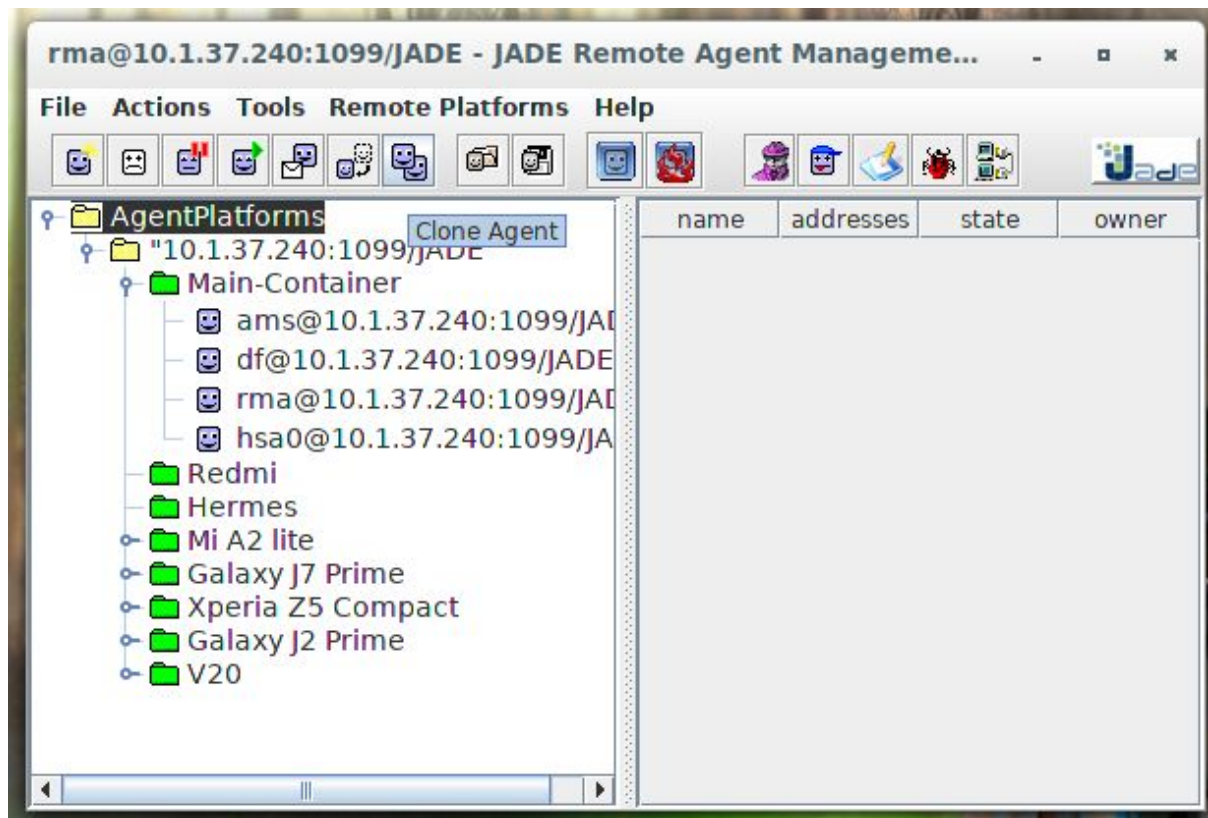
Una vez establecida la configuración de la plataforma y el contenedor, podemos presionar el botón “Start Container” para registrar un nuevo contenedor en la plataforma. Si la creación es exitosa, se habilita el botón “Stop Container” para eliminarlo junto con todos sus agentes contenidos, y otro botón “Start Agent” para crear e iniciar una instancia de HardwareSnifferAgent. Cuando un agente finaliza su recorrido, se agrega al listado de reportes una nueva entrada. A continuación vemos una captura de pantalla de la aplicación.



Por último, presionando sobre cada entrada individual en el listado podemos visualizar el listado de reportes de cada contenedor. A continuación vemos una captura de pantalla de dicha vista.



Adicionalmente, podemos observar el estado de la plataforma desde la interfaz gráfica provista por JADE.



4 Trabajo a futuro

Como resultado de la elaboración de este trabajo, adjuntamos una lista de las tareas a futuro que podrían realizarse para profundizar la investigación.

1. Implementar sniffers para otras plataformas compatibles con JVM y DVM, por ejemplo, Windows, o versiones de Android anteriores a API Level 24.
2. Investigar e implementar una solución para el manejo de errores.
3. Investigar métodos alternativos para obtener la información del dispositivo con sistemas operativo Android, principalmente en el caso de la carga promedio del sistema.
4. Permitir instanciar contenedores principales desde la aplicación móvil.

5 Conclusiones

La implementación de una aplicación móvil utilizando el framework JADE remarca que existe potencial para el desarrollo de software distribuido en estos dispositivos sin la

necesidad de pensar la solución bajo un paradigma cliente-servidor. Los resultados obtenidos a partir de la propuesta inicial demuestran que aún al día de hoy y a pesar de haber lanzado su última versión estable durante el año 2017, la propuesta de JADE sigue vigente y a la expectativa de ser explotada. Sería sumamente interesante el lanzamiento de una nueva versión que contemple la resolución de aquellos pequeños detalles o funciones incompletas que presenta el framework, por ejemplo, en lo que se relaciona con el manejo de errores; no delegándolos al desarrollador (ya que se perdería la esencia original de un middleware autónomo) pero sí permitiéndole conocer su existencia para poder manejarlos correctamente.

6 Referencias

- ReactiveJade. Código fuente, binarios, y documentación.
<https://github.com/daniffig/ReactiveJade>
- JADE documentation. <http://jade.tilab.com/>
- React Native. Build native mobile apps using JavaScript and React.
<https://facebook.github.io/react-native/>
- React Native Elements. Cross Platform React Native UI Toolkit.
<https://react-native-training.github.io/react-native-elements/>
- Tecnologías para el desarrollo de aplicaciones móviles.
https://docs.google.com/presentation/d/1MZaL8zvQSd6M0eQNPik4X9SskNWAvWXMMvNN-FQ_48w/edit?usp=sharing
- Inter-Platform Mobility Project. <https://tao.uab.cat/ipmp/documentation.html>
- React Native. Generating Signed APK.
<https://facebook.github.io/react-native/docs/signed-apk-android>