Programación Distribuida y Tiempo Real

Trabajo Práctico Final

Agentes JADE sobre dispositivos Android

Año: 2017

Alumnos:
Bolino, Patricio
Cuitiño, Alfonso

Índice

Introducción

JADE (Java Agent DEvelopment Framework) es un framework open source para aplicaciones con agentes peer-to-peer. Permite desarrollar sistemas distribuidos sin necesidad de que todas las máquinas participantes tengan el mismo sistema operativo, ya que JADE ejecuta sobre la JVM.
El proyecto LEAP (Lightweight and Extensible Agent Platform), en sus comienzos a principios de la década del 2000, logró llevar JADE a los dispositivos móviles que ejecutaban Java. Hoy en día, existe el LEAP add-on para JADE, que reemplaza partes del núcleo de JADE de manera tal que permite ejecutarlo en dispositivos Android, MIDP, la plataforma .NET, entre otros.
Originalmente JADE y JADE-LEAP eran implementaciones paralelas y no podían interoperar. A partir de JADE 4.0, se unificaron en una única plataforma, por lo que puede usarse de manera transparente.
Dada la proliferación actual de dispositivos Android conectados a internet, nos pareció una idea interesante combinarlo con las herramientas de agentes JADE, para implementar un sistema distribuido no solo conceptualmente, sino también geográficamente, y entre dispositivos tan variados y dispares como los teléfonos Android, y computadoras ejecutando Linux y Windows.

Decisiones de diseño

En este trabajo intentamos replicar el ejercicio de la práctica de agentes en el cual había que programar un agente que recorra periódicamente una secuencia de computadoras y reporte información al lugar de origen. Dicha información estaba compuesta por el tiempo total del recorrido, la carga de procesamiento de cada computadora, la cantidad de memoria disponible y los nombres de cada equipo.

Para este proyecto decidimos crear un sistema multiagente en el cual las “computadoras” son los celulares que envían información a la aplicación de escritorio que se ejecuta en una computadora. La información enviada consta de: nombre, fecha, hardware (marca del procesador), versión del SDK de Android, display (nombre para mostrar definido por el fabricante), nombre del operador (ej: Movistar, Personal, Claro), porcentaje carga de la batería, uso del CPU, cantidad de memoria libre, latitud, longitud, altura, un mensaje para indicar la conexión con el sistema (entrada, envío y salida) y el último SMS recibido.

El agente que recorre y obtiene toda la información está representado en la aplicación de escritorio. Para este caso decidimos que el agente reciba la información enviada por todos los terminales móviles, la muestre en pantalla y la guarde en un

archivo de log. Consideramos que no era buena idea que el agente salga a recorrer buscando la información por varios motivos:

1- La información es enviada constantemente por los celulares en un intervalo de tiempo configurable. Por lo tanto, si la cantidad de equipos conectados es lo suficientemente grande y el intervalo pequeño (ej 1 segundo) el agente podría perder información mientras está recolectando.

2- Los móviles tienen la posibilidad de desconectarse del sistema e ingresar nuevamente de forma no predeterminada. Además, nuevos equipos pueden ingresar en cualquier momento. Por lo tanto, no podemos saber de antemano que equipos recorrer y si utilizáramos alguna lista con los equipos conectados antes de que el agente salga a recorrer, corremos el riesgo de que si un equipo se desconecta se produzcan errores en la aplicación.

A diferencia de cómo desarrollamos en su momento el ejercicio de la práctica, para este trabajo utilizamos muchas herramientas que nos provee la propia librería de JADE como son los mensajes ACL para las conexiones y el envío de la información, los behaviours para determinar el comportamiento de los agentes al enviar y recibir la información y las ontologías que permiten la comunicación de objetos como contenido de los mensajes.

Resumen de lo desarrollado

El Main Container de JADE se ejecutará sobre una máquina con Linux o Windows, siendo el servidor para los demás containers de los terminales con Android.
Toda la comunicación entre los agentes desarrollados se hace usando una Ontología con su respectivo vocabulario y clase InfoMensaje. También se exploraron las opciones de comunicación con mensajes ACL y objetos serializables.

Por el lado de Linux y Windows, se implementaron 2 agentes:

El AgenteAdmin que implementa la interfaz SubscriptionManager y se encarga de registrar/deregistrar a los agentes que se conectan/desconectan al sistema. Lo instanciamos junto con el MainContainer de JADE.

El AgenteDesktop, se ejecuta junto con una ventana de interfaz gráfica representada por la clase ControlPanel (figura 1). Este agente se encarga de escuchar todo lo que los demás agentes envían, mostrando la información en pantalla, en un cuadro de texto y ubicando a los agentes en un mapa provisto por Google Maps. A su vez, escribe toda la información recibida en un archivo de log.

Por el lado de Android, se implementó una aplicación con dos pantallas. La primera es la de configuración, donde se pueden setear los parámetros para la comunicación entre los agentes (figura 2). La segunda, es la pantalla de comunicación, donde se visualiza toda la información que los agentes van enviando. Al comenzar, se instancia un container de JADE, y un AgenteMobile (figura 3). Este agente es el encargado de enviar y recibir su información y la de los demás agentes conectados al sistema.

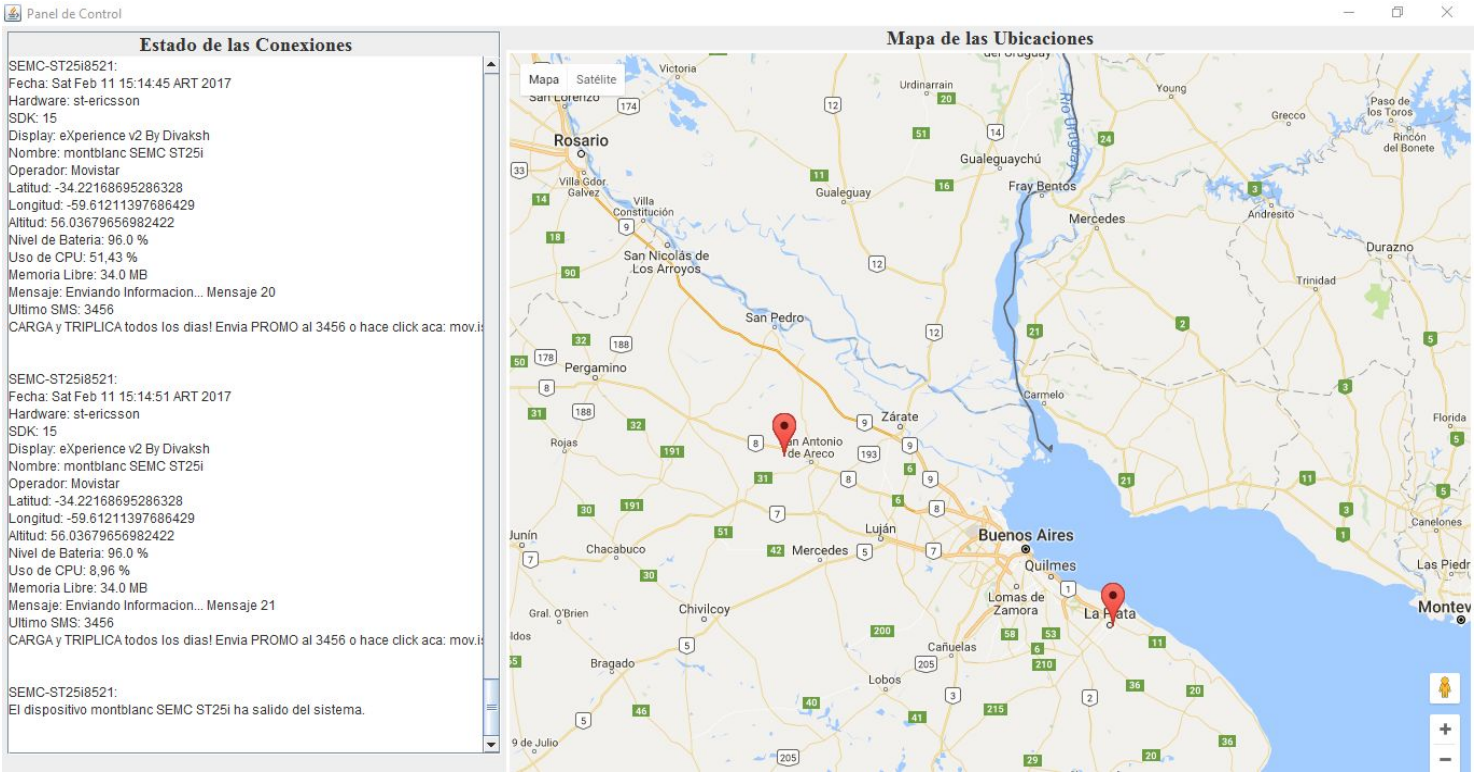
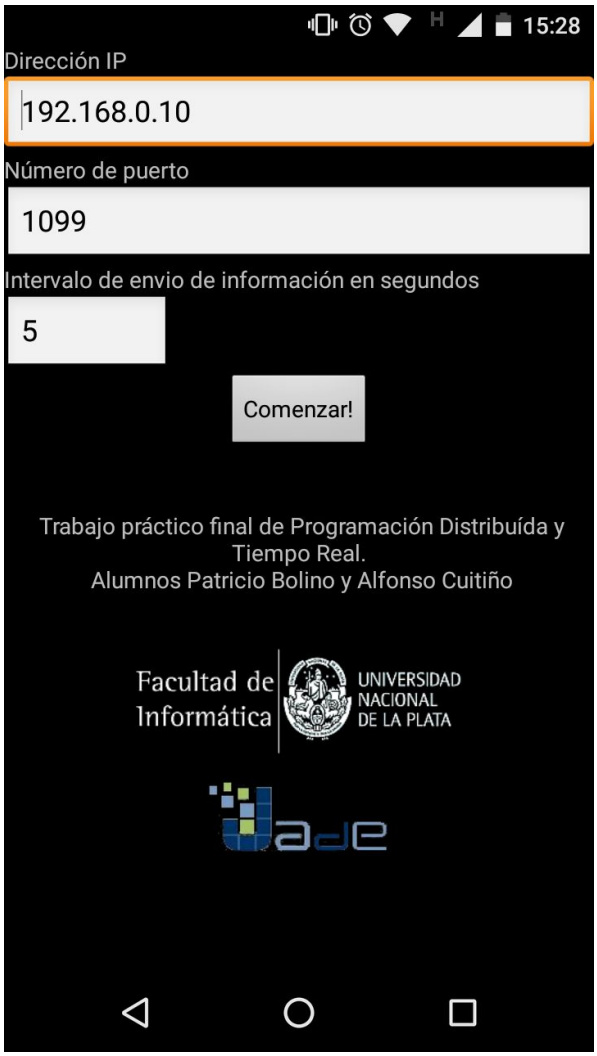
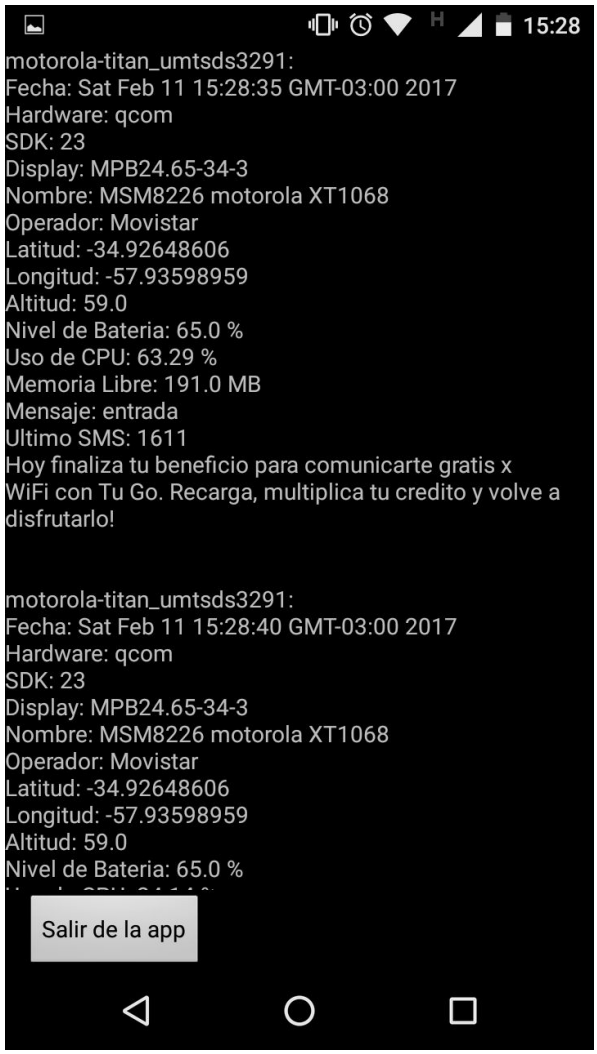


Figura 1. La interfaz gráfica generada por la clase ControlPanel.java, mostrando la ubicación de 2 agentes conectados.





Izquierda: Figura 2. La pantalla principal de la aplicación. Al presionar el botón comenzar, se instancia un container con un nuevo AgenteMobile y se pasa a la siguiente pantalla.

Derecha: Figura 3. Durante la ejecución del agente, la información enviada por sí mismo, y la recibida de todos los demás agentes del sistema, se va listando en pantalla.

Estructura del proyecto

Como se describió en la sección anterior, la aplicación consta de 2 partes: el desktop que se ejecuta sobre una computadora y el mobile que se ejecuta sobre los celulares.

La aplicación desktop consta por un lado de los agentes AgenteAdmin y AgenteDesktop que implementan la lógica de la aplicación y por el otro de la interfaz gráfica con el ControlPanel y el mapa. Para iniciar la aplicación tenemos la clase Main que se encarga de arrancar al AgenteDesktop.
La aplicación móvil se compone del AgenteMobile y la interfaz IAgenteMobile para implementar la lógica de la aplicación. Para la interfaz gráfica usamos la clase EstadoInicialAplicacion que setea valores por defecto al instalar la aplicación y los activities MainActivity y ComunicacionActivity que se utilizan para tomar los valores de la vista y armar el contenido de la información.

Ambas aplicaciones utilizan la misma ontología donde se definen los objetos Egreso y Ingreso para administrar las conexiones al sistema y InfoMensaje para comunicar la información de los terminales. También se define el vocabulario AppVocabulary y la ontología AppOntology.

Infraestructura necesaria

Para ejecutar la aplicación desktop se necesita una computadora de escritorio o notebook con sistema operativo Windows o Linux y que tenga instalado Java 1.8 (JRE). Para el caso de que se quiera ejecutar utilizando los archivos fuentes se necesitará el JDK y podría utilizarse el IDE Eclipse.

Para ejecutar la aplicación móvil se necesita uno o más celulares con Android 2.3 o superior. Si se quiere ejecutar desde los archivos fuente, necesitará Android Studio[1] 2.0 o superior.

Dado que es un sistema distribuido, se necesita tener una infraestructura de red que interconecte a todos los dispositivos participantes. Una opción es que cada dispositivo tenga su IP pública, de esta manera todos se “ven” con todos y pueden interconectarse.

[1] https://developer.android.com/studio/

Otra manera es ejecutarla sobre una LAN, por lo tanto la computadora y los terminales móviles deberán estar en el mismo segmento de red. Se puede utilizar un router con NAT y ejecutar la aplicación en la red interna. Esta fue la forma que usamos para el desarrollo, ya que no disponemos de un servidor web en internet que pueda ejecutar el Main Container en Java.

Comunicación entre agentes

En primera instancia, el enfoque que elegimos fue el de usar los mensajes representados por la clase de JADE ACLMessage[2]. Estos son mensajes del estándar ACL (*Agent Communication Language*) definido por la FIPA (*Foundation for Intelligent Physical Agents*), que idealmente permitirían que los agentes de JADE puedan comunicarse con otros agentes escritos en otros lenguajes y plataformas, que implementen el estándar FIPA. El objeto ACLMessage tiene un campo `content` de tipo String que representa el mensaje. Es la forma más sencilla y fácil de entender para comunicar los agentes. Al principio, desde el agente mobile enviábamos todos los datos concatenados en un solo String. La desventaja se hizo evidente cuando quisimos desglosar los datos para mostrarlos en el agente desktop, teníamos que descomponer el string y parsearlo programáticamente para separar los campos. Fue por esto que pasamos a probar la otra forma de comunicación que provee el ACLMessage, usando el campo contentObject para comunicar objetos que implementen la interfaz serializable de Java.
Usando los métodos `setContentObject` y `getContentObject` se pueden enviar y recibir objetos dentro de los mensajes ACL, casteándolos al momento de recibirlos. El uso de objetos como contenido del mensaje no está aconsejado ya que no cumple con los estándares FIPA.
El problema que surgió fue que al recibir el mensaje ACL, no podía ser casteado al objeto correspondiente, elevando una excepción UnreadableException[3]. No logramos encontrar información muy concreta sobre la causa de esta excepción, más allá de que obviamente la clase “original” y la clase a la que va a ser casteada deben ser exactamente las mismas, lo cual se cumplía. Otra causa más probable es la diferencia entre la versión de Java que se usa en Android y el JRE de Oracle para computadoras de escritorio pudiendo haberse presentado algún tipo de incompatibilidad.
La tercera opción fue la comunicación usando Ontologías, esta opción es la más adecuada puesto que permite la comunicación entre agentes que soporten el

[2] http://jade.tilab.com/doc/api/jade/lang/acl/ACLMessage.html
[3] http://jade.tilab.com/doc/api/jade/lang/acl/UnreadableException.html

estándar FIPA independientemente de las plataformas y los lenguajes utilizados. Para hacer posible este tipo de comunicación se debe definir un objeto que implemente la interfaz Predicate (InfoMensaje) con los atributos que formarán parte de los mensajes ACL y sus respectivos métodos setters y getters. Además, implementamos un vocabulario que es un conjunto de strings que representa el nombre de la clase del objeto junto con el nombre de todos los atributos. Este vocabulario es utilizado en la ontología implementada con el patrón Singleton y donde se hace uso de un Schema de tipo PredicateSchema[4] en donde se configura cada atributo del objeto definido en el vocabulario junto con el tipo de dato que representa ese atributo. Para enviar la información, los métodos behaviour tienen una variable *myAgent* al que se le pide el ContentManager para setearle con el método fillContent el mensaje ACL y los datos a enviar (el objeto InfoMensaje). Para recibir la información, se debe utilizar la variable *myAgent* del behaviour y el método receive para obtener el mensaje ACL, luego con el ContentManager usando el método extractContent, que recibe como parámetro el mensaje ACL, se obtiene el objeto InfoMensaje.

Documentación de las clases

Aplicación Desktop

AgenteAdmin

Hereda de Agent e implementa la interfaz SubscriptionManager. Es el agente encargado de gestionar las conexiones de los terminales al sistema usando un Behaviour SubscriptionResponder[5]. Se instancia al ejecutar el script de inicio.

AgenteDesktop

Hereda de Agent y se crea al iniciar la aplicación. Se encarga de administrar las suscripciones al sistema, con un CyclicBehaviour llamado AdministradorDeSubscripcion declarado como clase interna, usando los predicados de la ontología Ingreso y Egreso. Mantiene una lista de los agentes conectados.

También se encarga de recibir la información enviada por los terminales y loguear dicha información en el archivo de texto. Esto lo hace con otro CyclicBehaviour llamado InformacionRecibida, cada vez que llega un mensaje, se lo manda a la interfaz gráfica por el método mostrarInformacion, y al archivo de log con el método loguearInformacion.

[4] http://jade.tilab.com/doc/api/jade/content/schema/PredicateSchema.html
[5] jade.tilab.com/doc/api/jade/proto/SubscriptionResponder.html

En su método setup inicializa el archivo de log, instancia los Behaviours anteriormente descritos, inicializa un ACLMessage con la ontología desarrollada, y finalmente instancia la interfaz gráfica ControlPanel.

ControlPanel

Se encarga de la interfaz gráfica de la aplicación desarrollada en Java Swing, contiene la ventana de texto y el mapa para ubicar a los dispositivos. El mapa fue implementado usando la librería JxBrowser[6], invocando un mapa de Google Maps dentro del componente BrowserView que provee esta librería para Java Swing.

Main

Es la clase principal del sistema y se encarga de arrancar la aplicación creando al AgenteDesktop, asumiendo que ya está ejecutándose el Main Container. Sino, levanta una excepción y termina.

Aplicación Móvil

MainActivity

Es la vista que se utiliza para obtener la configuración inicial (dirección IP, puerto e intervalo de envío) para conectarse al desktop. Al presionar el botón “comenzar”, se encarga de persistir la configuración para futuras ejecuciones, e instanciar el servicio de JADE LEAP, y el container de JADE con el AgenteMobile.

AgenteMobile

Hereda de Agent e implementa la interfaz IAgenteMobile. Es el agente encargado de enviar la información: envía los mensajes de entrada y salida al sistema y la información mientras el terminal está conectado al sistema. También recibe la información enviada desde los otros terminales y administra la suscripción al AgenteAdmin de la aplicación desktop.
Este agente tiene un comportamiento (*Behaviour*) AdministradorDeSuscripcion, que hereda de CyclicBehaviour[7], y que lo registra ante el agente Manager usando un mensaje ACL, y lo mantiene actualizado acerca de cuáles agentes están conectados al sistema (que serán los destinatarios de la información enviada). Los comportamientos *cyclic* viven durante toda la ejecución del agente, por eso sirve para quedar escuchando al manager y que vaya llegando la nueva información.
El siguiente comportamiento es un CyclicBehaviour, InformacionRecibida, que se encarga de recibir la información que envían los demás agentes, y pasársela al ComunicacionActivity para que la muestre en pantalla.
Para enviar la información cada una cantidad de segundos configurable, se usa un TickerBehaviour[8].

---

[6] https://www.teamdev.com/jxbrowser
[7] http://jade.tilab.com/doc/api/jade/core/behaviours/CyclicBehaviour.html
[8] http://jade.tilab.com/doc/api/jade/core/behaviours/TickerBehaviour.html

Y por último, se implementó un OneShotBehaviour[9] llamado MensajeIndividual, que se usa únicamente para avisarle al sistema cuando el agente ingresa o se retira.
A la vez, en el método setup del agente, se encarga de emitir un Broadcast[10] de Android para indicar que el agente ya tiene sus Behaviours y está listo para funcionar. Este broadcast es escuchado por el MainActivity, y al recibirlo inicia la ComunicacionActivity.

ComunicacionActivity

Es la vista utilizada para mostrar la información enviada al desktop y recibida desde los otros terminales. Esto se hace implementando un BroadcastReceiver, una clase interna que se encarga de escuchar el broadcast emitido por el AgenteMobile con clave “actualizar”. Cuando recibe uno de estos broadcasts, muestra la nueva información en el campo de texto.
También es donde se arma el mensaje con la información que se obtiene del celular, a través del método obtenerInformacionDelDispositivo().
Tiene un botón “Salir de la app”. Al presionarlo, se avisa al AgenteMobile que envíe un mensaje avisando que sale del sistema, y que detenga las comunicaciones, y se finaliza la aplicación.

Ontología

AppOntology

Hereda de Ontology y se implementó como un singleton. Agrega PredicateSchemas con los objetos Predicate y para cada uno agrega los atributos configurándolos según el tipo.

AppVocabulary

Es una interfaz que contiene un conjunto de strings que representan los nombres de las clases y atributos de los objetos Predicate.

Egreso

Implementa la interfaz Predicate y mantiene una lista de agentes. Se utiliza toda vez que un agente (celular) sale del sistema.

Ingreso

Implementa la interfaz Predicate y mantiene una lista de agentes. Se utiliza toda vez que un agente (celular) ingresa al sistema.

[9] http://jade.tilab.com/doc/api/jade/core/behaviours/OneShotBehaviour.html
[10] https://developer.android.com/guide/components/broadcasts.html

InfoMensaje

Implementa la interfaz Predicate y representa la información que es enviada por los celulares y recibida por la computadora. Esto es, el mensaje, fecha, nombre del hardware, marca, modelo, versión de SDK, geolocalización, operador telefónico, SMS, uso de CPU, uso de Memoria, y batería.

Instrucciones para ejecutar el sistema

Usando los binarios

Primero se debe iniciar el Main Container de JADE, utilizando el script llamado startJade.bat en Windows o startJade.sh en Linux.
El script contiene simplemente el comando:

```
java -cp lib\jade.jar;bin jade.Boot -gui -nomtp manager:agentes.AgenteAdmin
```

Este script inicializa el servidor con un agente Manager, que es el encargado de administrar a los agentes que se conectan y desconectan del sistema.

Una vez hecho esto, pueden usarse el monitor para Linux o Windows, y la aplicación en Android.

Para abrir el monitor, debe usarse el script iniciarMonitor.bat en Windows, o iniciarMonitor.sh en linux. Al igual que el anterior, este script consiste de un solo comando:

```
java -jar jade-monitor.jar
```

El paquete jade-monitor.jar contiene todas las clases desarrolladas y las librerías necesarias (JADE y JxBrowser para el mapa).
En algunas versiones de Windows, puede que haya problemas con los permisos al intentar ejecutarlo. Para evitar esto, se puede usar la consola en modo administrador.

En Android, la aplicación puede instalarse usando el paquete APK suministrado. Puede que haya una advertencia de que la aplicación proviene de “orígenes desconocidos”. Esto se debe a que el paquete no está firmado con un certificado digital. Simplemente hay que activar la opción “permitir orígenes desconocidos” al recibir la alerta.

Compilando el código

El código fue desarrollado en dos partes, un proyecto para el IDE Eclipse, y un proyecto para el IDE Android Studio.
El proyecto de Eclipse se encuentra en el directorio “Desktop”. Al importarlo en Eclipse, se debe ejecutar la clase “Main.java”. Esta es la responsable de instanciar al AgenteDesktop y a la interfaz gráfica. El AgenteAdmin es instanciado al momento de iniciar el main container de JADE, por lo que no es necesario iniciarlo desde Eclipse.
Por otro lado, en Android Studio, al cargar el proyecto, simplemente se puede seleccionar la opción de ejecutarlo. Para esto es necesario habilitar en el celular la depuración USB para instalar el APK.
Automáticamente generará la aplicación e inicia la pantalla principal, MainActivity.

Referencias

JADE Tutorial and Primer. Jean Vaucher and Ambroise Ncho. Université de Montréal https://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html

Documentación oficial de JADE
http://jade.tilab.com/documentation/tutorials-guides/

Federico Bergenti, Giovanni Caire, Danilo Gotta. Agents on the Move: JADE for Android Devices http://ceur-ws.org/Vol-1260/paper9.pdf

Application-defined content languages and ontologies
http://jade.tilab.com/doc/tutorials/CLOntoSupport.pdf

Developing Multi-Agent Systems with JADE. Fabio Luigi Bellifemine, Giovanni Caire, Dominic Greenwood. ISBN: 978-0-470-05747-6

Danilo Gotta. JADE Android Add-on guide.
http://www.agilemethod.csie.ncu.edu.tw/download/agent/JADE_ANDROID_Guide.pdf