Agents on the Move: JADE for Android Devices

Federico Bergenti
Università degli Studi di Parma
43124 Parma, Italy
Email: federico.bergenti@unipr.it

Giovanni Caire, Danilo Gotta
Telecom Italia S.p.A.
10148 Torino, Italy
Email: {giovanni.caire,danilo.gotta}@telecomitalia.it

***Abstract*—In this paper we describe the current state of the development of *JADE add-on for Android,* a platform module that enables the deployment of JADE agents on Android devices. First, we motivate the research and we describe the rationale of the project. Then, we detail the coarse-grained architecture of the platform module and we discuss the underlying design decisions. The API of the module is also discussed to detail how programmers use it to host JADE agents on mobile devices. Finally, we briefly outline conclusions on the presented work.**

I. Introduction

JADE (*Java Agent DEvelopment framework*) is the popular open-source framework that facilitates the development of interoperable multi-agent systems [1]–[3]. JADE is the core of *WADE* (*Workflows and Agents Development Environment*) (see, e.g., [4]–[7]) and it is one of the most widely used tools for the development of agent-based systems.

Among the most notable uses of JADE in an industrial setting, Telecom Italia is currently using it to deploy in the field mission critical applications that have direct influence on the work of thousands technicians and millions customers. Just to cite an example, *WANTS* (*Workflow and AgeNTS*) [8] has been used for several years to implement a mediation layer between network elements and *OSS* (*Operations Support Systems*) for the nationwide broadband access network of Telecom Italia.

The core design decisions that characterize the architecture of JADE and that are the ultimate responsible for its proven modularity have already allowed JADE to fit the constraints of mobile environments. The adoption of JADE in mobile environments dates back to early 2000's with *LEAP* (*Lightweight and Extensible Agent Platform*), a EU-funded project that provided the first implementation of JADE on the Java-enabled telephones of the time [9], [10].

Today, the results of LEAP are returned to the community of JADE developers by mean of the *LEAP add-on* for JADE, available open-source from the official JADE Web site [3]. The LEAP add-on replaces parts of the JADE core module to form a modified runtime environment called JADE-LEAP (or *"JADE powered by LEAP"*) that can be deployed on a wide range of mobile devices.

The possibility of hosting JADE agents on Android devices described in this paper is offered as an improvement of the LEAP add-on, and today the LEAP add-on can be configured in the four different ways described below. Though different internally, all configurations of the LEAP add-on provide the same API, thus offering a homogeneous layer over a diversity of devices and networks.

In details, the four configurations of the LEAP add-on are enumerated as follows:

- *android*—to host JADE agents on devices that supports Android 2.1 (or later);
- *midp*—to host JADE agents on devices supporting MIDP 1.0 (or later);
- *pjava*—to host JADE agents on devices supporting J2ME CDC or the obsolete PersonalJava; and
- *dotnet*—to host JADE agents on Microsoft .NET Framework version 1.1 or later.

It is worth noting that JADE and JADE-LEAP were originally two different platforms that did not interoperate. It was not possible to attach a JADE-LEAP container to a JADE main container, and vice-versa. The introduction of version 4.0 of JADE merged JADE and JADE-LEAP into a single platform with full interoperability. This is the reason why we dropped the name JADE-LEAP and today we simply refer to JADE and its configurations for different classes of devices.

In our opinion, the combination of the rich expressiveness of the communication among JADE agents—the IEEE FIPA ACL and interaction protocols—with the power and widespread adoption of Android brings a significant value to the development of innovative distributed and decentralized applications.

II. JADE for Android

The JADE run-time was originally designed to address a wide class of devices ranging from full featured servers to mobile phones. In order to properly address the memory and processing power limitations of mobile devices, and the characteristics of wireless networks in terms of bandwidth, latency, intermittent connectivity and address volatility, and at the same time in order to be efficient when executed on wired network hosts, JADE can be configured to adapt to the characteristics of the deployment environment. The JADE architecture, in fact, is completely modular and, by activating specific modules, it is possible to meet different requirements in terms of connectivity, memory and processing power.

More in details, the LEAP add-on for JADE is in charge of optimizing all communication mechanisms when dealing with devices with limited resources and connected through wireless networks. By activating this add-on, a JADE container is *split*, as depicted in Figure 1, into a *front-end* running on the mobile terminal and a *back-end* running on the wired network. A suitable architectural element on the wired network,

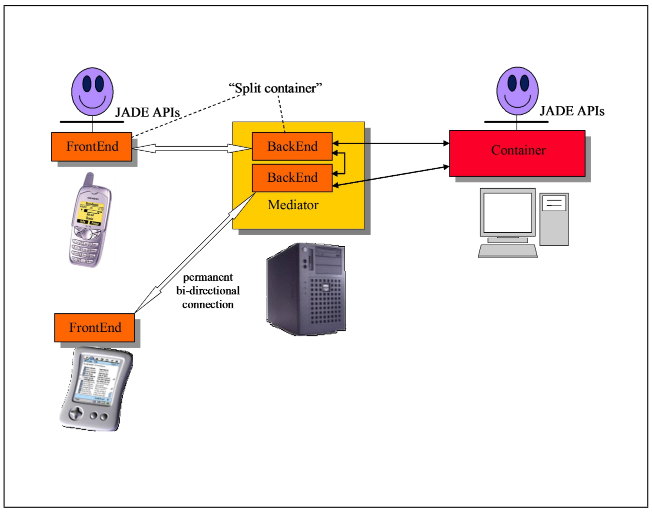

Fig. 1: The JADE split container mechanism for mobile environments as provided by the LEAP add-on.

called *mediator*, is in charge of instantiating and maintaining the back-ends. To better face high workload situations, it is possible to deploy several mediators, each of them managing a set of back-ends.

Each front-end is linked to its corresponding back-end through a permanent bi-directional connection. It is important to note that it makes no difference at all, to application developers, whether an agent is deployed on a conventional container or on the front-end of a split container, since the available functionality and the APIs are exactly the same.

The split-container mechanism has a number of features, as enumerated below.

- The back-end masks to other containers the current IP address dynamically assigned to the wireless device, thereby hiding to the rest of the multi-agent system a possible change of IP address.
- The front-end is able to detect connection losses with the back-end and re-establish the connection as soon as possible.
- Both the front-end and the back-end implement a *store-and-forward* mechanism: messages that cannot be delivered due to a temporary disconnection are buffered and re-transmitted as soon as the connection is re-established.
- Many management messages among containers—e.g., to retrieve the address of the container where an agent is currently running—are handled by back-ends only.
- Part of the functionality of a container is delegated to the back-end and, as a consequence, the front-end becomes extremely lightweight in terms of required memory and processing power. In details, the JADE run-time memory footprint, in a MIDP 1.0 environment, is around 120 KB, and it can be further reduced to 50 KB by using the so called *ROMization* technique, i.e., by compiling JADE together with the JVM.

The configuration of JADE for Android presented in this paper is based on the infrastructure that the LEAP add-on provides. JADE containers on Android devices are all deployed in split mode and they can all take advantage of the aforementioned features. In particular, the management of the volatility of the wireless network connection, as implemented by means of the first three features in the list above, is of peculiar interest because Android does not really provide an effective support for it.

III. How to Use JADE for Android

Android has a very peculiar approach to the development of applications and it also introduces some new concept, and a new nomenclature, that JADE for Android is demanded to respect to ensure acceptability from the community of Android developers. This is the reason why we designed JADE for Android with an extensive use of Android-specific concepts and with a strict adherence to the guidelines for Android development, as suggested by the official documentation.

In details, we decided to wrap JADE for Android into a specific *Android service*. An Android service is an application component that can perform long-running operations and that does not provide a user interface. Other application components can start a service and it will continue to run in the background even if the user switches to other applications.

The Android service that bundles JADE for Android is called `MicroRuntimeService` (in package `jade.android`) and it is responsible for configuring the JADE environment and for starting and stopping the JADE runtime when required.

In order to make this service communicating with other Android application components (e.g., application activities), the *binder* mechanism provided by the Android application model has been used. The binder is a user-defined interface exposed by an Android service when another Android component binds to that service through the `Context.bindService()` method. In our case the binder, namely `MicroRuntimeServiceBinder` (in package `jade.android`) includes methods to manage the life cycle of the JADE runtime and to communicate with the JADE agents hosted in the service.

The first operation to activate the JADE runtime from an Android activity is to retrieve a `MicroRuntimeServiceBinder` object using a subclass of `ServiceConnection` (in package `android.content`), as follows.

```
serviceConnection = new ServiceConnection() {
  public void onServiceConnected(
    ComponentName className, IBinder service) {
    // Bind successful
    microRuntimeServiceBinder =
      (MicroRuntimeServiceBinder) service;
  }

  public void onServiceDisconnected(
    ComponentName className) {
    // Bind unsuccessful
    serviceBinder = null;
  }
};
```

The newly created `microRuntimeServiceBinder` object is used to bind to the service—and to create it if needed—by means of the `Context.bindService()` method, as follows:

```
bindService(
  new Intent(getApplicationContext(),
    MicroRuntimeService.class),
  serviceConnection, Context.BIND_AUTO_CREATE);
```

Having retrieved the `MicroRuntimeServiceBinder` object it is now possible to start a JADE split container as shown in the code snipped below:

```
Properties pp = new Properties();
pp.setProperty(Profile.MAIN_HOST, host);
pp.setProperty(Profile.MAIN_PORT, port);
pp.setProperty(Profile.JVM, Profile.ANDROID);

serviceBinder.startAgentContainer(pp,
  new RuntimeCallback<Void>() {
    @Override
    public void onSuccess(Void thisIsNull) {
      // Split container startup successful
      ...
    }

    @Override
    public void onFailure(Throwable t) {
      // Split container startup error
      ...
    }
});
```

As usual in JADE development, the host and port where the main container is running—as well as other configuration options—must be specified in a `Properties` object.

According to the Android best practices, all operations are asynchronous and the result is made available by means of a `RuntimeCallback` (in package `jade.android`) object.

Once the JADE runtime is up and running it is possible to start an agent as shown in the code snippet below:

```
serviceBinder.startAgent(nickname,
  className,
  new Object[] { getApplicationContext() },
  new RuntimeCallback<Void>() {
    @Override
    public void onSuccess(Void thisIsNull) {
      // Agent startup successful
      ...
    }

    @Override
    public void onFailure(Throwable t) {
      // Agent startup error
      ...
    }
  });
```

It is worth noting that the *application context* of the current Android application is passed to the agent as first argument. This allows the agent to access the Android API as needed. Anyway, this is not sufficient to let the newly created agent interact the with user, and the cleanest way to implement interactions between GUI components (mainly Android activities) and the agent is to use the *O2A* (*Object-to-Agent*) interface mechanism introduced in version 4.1.1 of JADE. This mechanism allows an agent to expose one or more interfaces that can be retrieved by external components. External components can trigger agent tasks by invoking methods of such interfaces.

The following code snippet shows how to define an O2A interface that an hypothetical chat agent could expose to let the application activity: *(i)* pass chat messages from the user to the agent to have them forwarded to other users via respective agents; and *(ii)* retrieve the list of users currently involved in the chat.

```
public interface ChatClientInterface {
  public void handleSpoken(String s);
  public String[] getParticipantNames();
}
```

The `handleSpoken()` method is used by the application activity to make the agent forward a messages to all chat participants.

The `getParticipantNames()` method is used to retrieve the list of users currently connected to the chat, e.g., to have it listed to the user.

The following statement, that should be used in the `Agent.setup()` method, shows how an agent exposes an O2A interface.

```
registerO2AInterface(
  ChatClientInterface.class, this);
```

Similarly, the following statement shows how an Android activity can retrieve the O2A interface exposed by the agent, typically in the `Activity.onCreate()` method.

```
chatClientIf = MicroRuntime.getAgent(nickname)
  .getO2AInterface(ChatClientInterface.class);
```

When working with JADE for Android, it is often the case that an agent needs to proactively communicate with its user via the application GUI. In an hypothetical chat application when a chat agent receives a message from another chat agent, it need to present it to its user via the common chat trace view. The preferred way to implement this kind of interaction is to use the mechanism that Android provides to allow different components of an application to interact. This is based on broadcasting so called *intents* that can be received by interested components. The code snippet below shows how a chat agent can create and broadcast an `Intent` (in package `android.content` object to notify the GUI that a new chat message has just been received.

```
Intent broadcast = new Intent();
broadcast.setAction("jade.demo.chat.REFRESH");
broadcast.putExtra("msg",
  speaker + ":␣" + message + "\n");
context.sendBroadcast(broadcast);
```

Where context is the field where the agent stored the application context received as first startup argument.

In order for the Android application to register a receiver for the intents sent by agents, Android requires an object of a subclass of `BroadcastReceiver` (in package `android.content`), as follows.

```
private class MyReceiver
  extends BroadcastReceiver {
  @Override
  public void onReceive(Context context,
    Intent intent) {
    String a = intent.getAction();
    if (a.equals("jade.demo.chat.REFRESH")) {
      String t = intent.getStringExtra("msg");
      ...
    }
}
```

The code snippets below show how an Android activity can register a receiver to intercept intents broadcast from an agent.

```
MyReceiver myReceiver = new MyReceiver();
IntentFilter filter = new IntentFilter();
filter.addAction("jade.demo.chat.REFRESH");
registerReceiver(myReceiver, filter);
```

A very complete, yet not too complex, example of using JADE for Android is available in a dedicated tutorial downloadable from JADE Web site [3] and it implements an agent-based chat application. The code snippets in this section are largely inspired by such an example.

Even if the full power of JADE can only be exploited by means of specific agents and the use of agent-based messaging, often applications simply need a way to reliably deliver requests to service agents and have access to respective responses. For such cases, JADE for Android provides an helper class called `MicroRuntimeGateway` (in package `jade.android`) that is responsible for managing the lifecycle of the JADE service and of the embedded container.

This class provides a gateway between an Android application and a JADE multi-agent system and it maintains a single internal *gateway agent* that acts as an entry point to the JADE system. The activation/termination of such an agent (and its underlying split container) are completely managed by the gateway and the application developers do not need to care about them. Moreover, the internal gateway agent is accessible via a simple interface and it acts much like a simple Java object, rather than a full-featured agent.

IV. Conclusion

The motivation of the work presented in this paper is that we believe that Android developers can leverage the features that JADE provides to simplify the development of decentralized and distributed applications. In particular, we think that the possibility of combining the expressiveness of IEEE FIPA communication with the power of Android brings a notable value to the development of innovative applications based on the peer-to-peer paradigm.

By means of JADE an Android application can easily embed agents and therefore become part of a wider distributed system possibly including other mobile devices (not necessarily based on Android). JADE for Android provides an interface that allows an application to start a local agent, trigger behaviors and, more in general, exchange application-specific objects with agents. It is therefore possible to discover remote peers, carry out possibly complex conversations with them, exploit JADE ontologies to handle structured messages, perform background activities according to the behavior composition model and, in general, take advantage of all features of JADE.

We have already experienced with JADE for Android in a recent project called *AMUSE* (*Agent-based Multi-User Social Environment*) (see, e.g., [11]–[13]). AMUSE is a JADE-based open-source platform for mobile social games that features JADE for Android to deploy agents on players' devices, thus exploiting the synergistic combination of powerful mobile devices and the abilities of agents to support decentralized coordination (see, e.g., [14], [15]). The work in AMUSE allowed us to improve the features and the stability of JADE for Android and it also provided complex test cases. Moreover, the adoption of JADE for Android suggested important improvements in the Android-specific API.

Finally, we are in the process of bridging JADE with the recent revival of the research on agent-oriented programming languages with *JADEL* (*JADE Language*), a novel programming language that eases the development of JADE agents.

References

[1] F. Bellifemine, G. Caire, and D. Greenwood, *Developing multi-agent systems with JADE.* Wiley Series in Agent Technology, 2007.

[2] F. Bellifemine, A. Poggi, and G. Rimassa, "Developing multi-agent systems with a FIPA-compliant agent framework," *Software: Practice & Experience*, vol. 31, pp. 103–128, 2001.

[3] JADE (Java Agent DEvelopment framework) web site. [Online]. Available: http://jade.tilab.com

[4] M. Banzi, G. Caire, and D. Gotta, "WADE: A software platform to develop mission critical, applications exploiting agents and workflows," in *Procs. Int'l Joint Conf. Autonomous Agents and Multi-Agent Systems*, 2008, pp. 29–36.

[5] G. Caire, E. Quarantotto, M. Porta, and G. Sacchi, "WOLF: An Eclipse plug-in for WADE," in *Procs. IEEE Int'l Workshops Enabling Technologies: Infrastructures for Collaborative Enterprises*, 2008.

[6] F. Bergenti, G. Caire, and D. Gotta, "Interactive workflows with WADE," in *Procs. IEEE Int'l Conf. Enabling Technologies: Infrastructures for Collaborative Enterprises*, 2012, pp. 10–15.

[7] WADE (Workflows and Agents Development Environment) web site. [Online]. Available: http://jade.tilab.com/wade

[8] F. Bergenti, G. Caire, and D. Gotta, *Large-Scale Network and Service Management with WANTS.* In press, 2014.

[9] F. Bergenti, A. Poggi, B. Burg, and G. Caire, "Deploying FIPA-compliant systems on handheld devices," *IEEE Internet Computing*, vol. 5, no. 4, pp. 20–25, 2001.

[10] F. Bergenti and A. Poggi, "Ubiquitous information agents," *Int'l J. Cooperative Information Systems*, vol. 11, no. 34, pp. 231–244, 2002.

[11] F. Bergenti, G. Caire, and D. Gotta, "Agent-based social gaming with AMUSE," in *Procs.* 5^{th} *Int'l Conf. Ambient Systems, Networks and Technologies (ANT 2014) and* 4^{th} *Int'l Conf. Sustainable Energy Information Technology (SEIT 2014)*, ser. Procedia Computer Science, vol. 32, 2014, pp. 914–919.

[12] F. Bergenti, G. Caire, and D. Gotta, "An overview of the AMUSE social gaming platform," in *Procs. Workshop From Objects to Agents*, 2013, pp. 85–90.

[13] AMUSE (Agent-based Multi-User Social Environment) web site. [Online]. Available: http://jade.tilab.com/amuse

[14] F. Bergenti, A. Poggi, and M. Somacher, "A collaborative platform for fixed and mobile networks," *Communications of the ACM*, vol. 45, no. 11, pp. 39–44, 2002.

[15] F. Bergenti and A. Poggi, "Agent-based approach to manage negotiation protocols in flexible CSCW systems," in *Procs.* 4^{th} *Int'l Conf. Autonomous Agents*, 2000, pp. 267–268.