JADE Programming for Android

Integrated Master's in Informatics Engineering

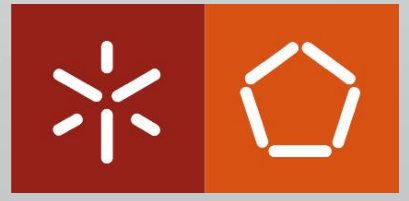
Intelligent Agents

2018/2019

Synthetic Intelligence Lab

Filipe Gonçalves

César Analide

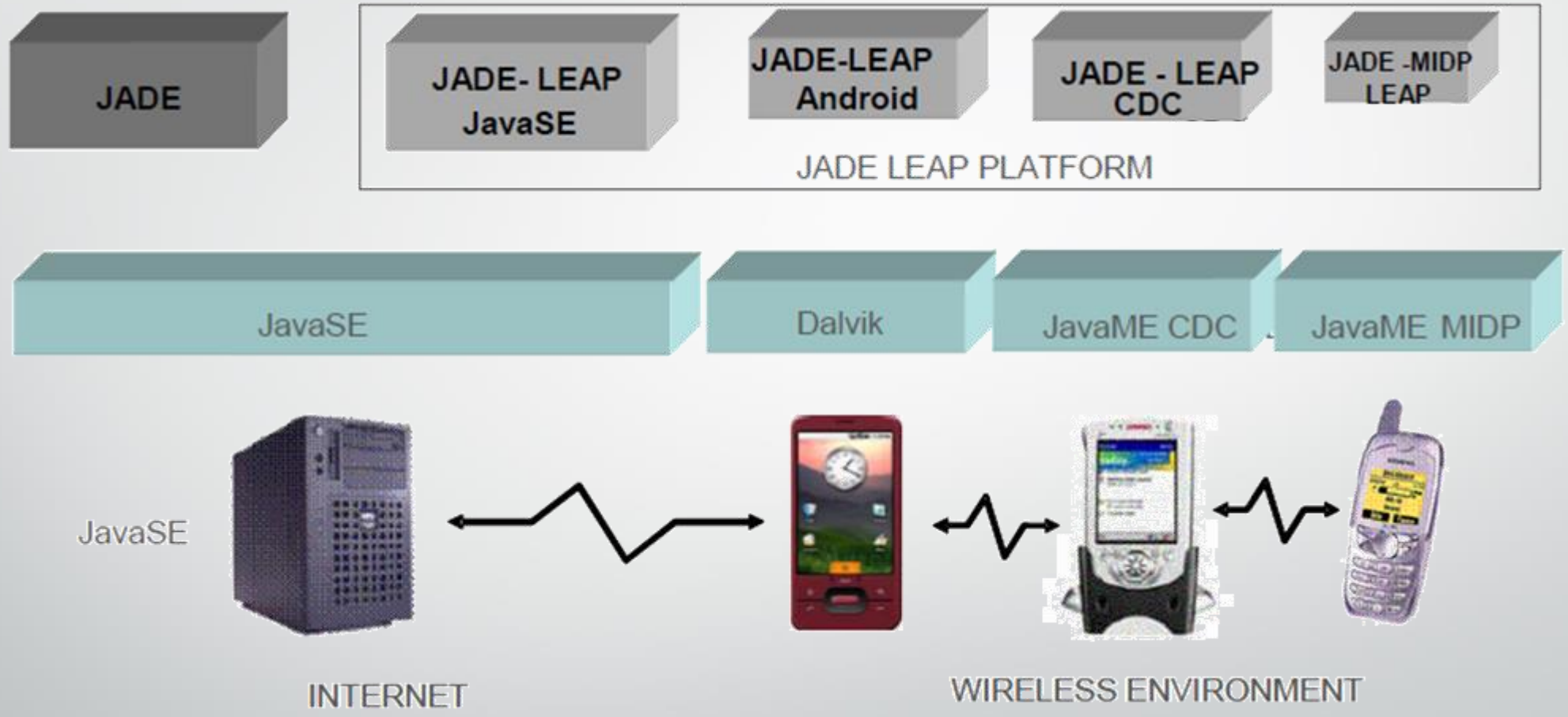


ISLab

JADE-LEAP

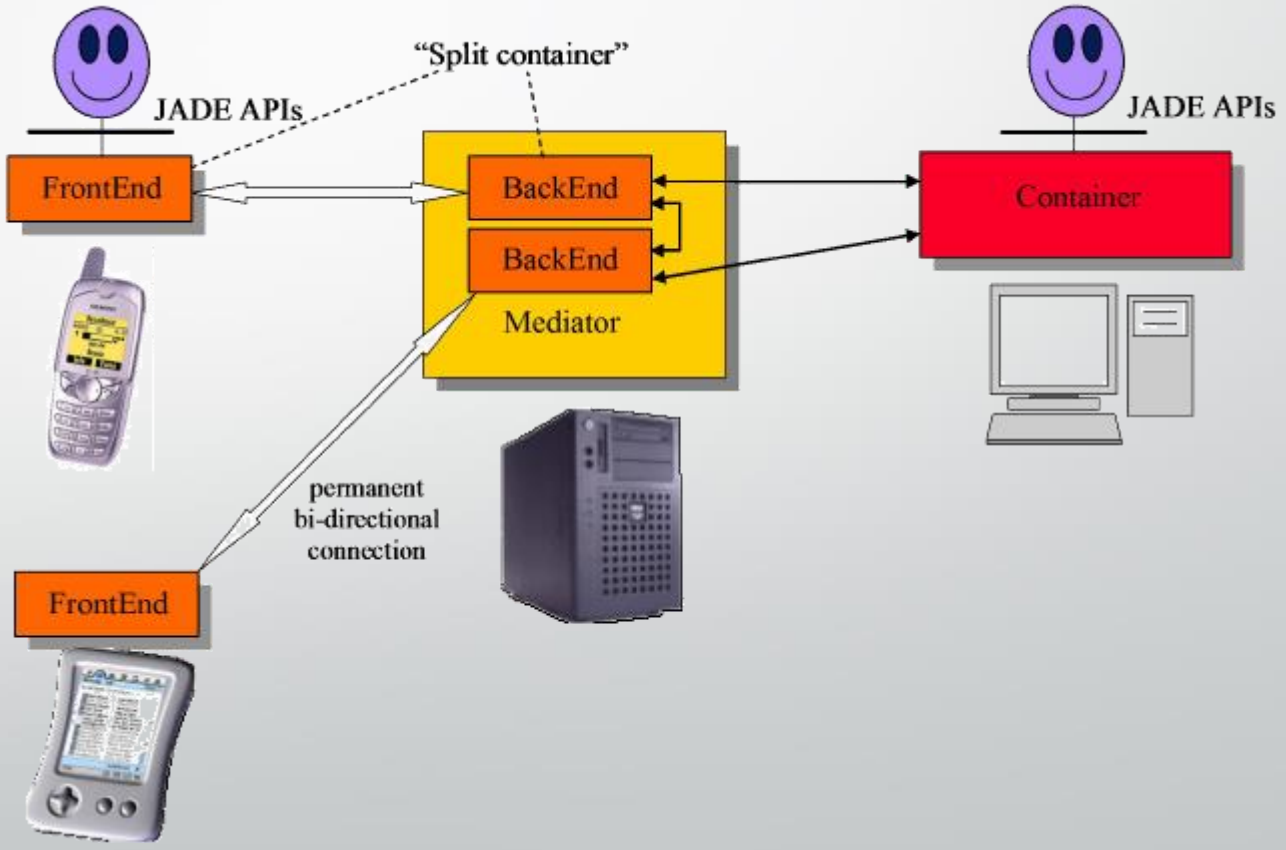
- The JADE run-time was originally designed to address a wide class of devices ranging from full featured servers to mobile phones (e.g. Android, MIDP, .Net Framework, etc.)
- The adoption of JADE in mobile environments dates back to early 2000's with LEAP (Lightweight and Extensible Agent Platform), providing the first implementation of JADE on the Java-enabled telephones of the time.
- 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.
  - jade.tilab.com/dl.php?file=LeapAddOn-4.5.0.zip

Homogeneous Layer JADE APIs



LEAP Architecture

- By activating the LEAP add-on, a JADE container is split into a **front-end running** on the **mobile terminal** and a **back-end running** on the **wired network**.
- **Mediator**: in charge of instantiating and maintaining a set of back-ends, by managing the workload.
- **Each front-end** is linked to its **corresponding back-end**, through a permanent bi-directional connection.



LEAP add-on Features

- The split-container mechanism presents a number of features:
  - 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 are **handled by back-ends only**.
    - E.g. to retrieve the address of the container where an agent is currently running.
  - 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.

JADE Android

- JADE containers on Android devices (using ***JadeAndroid.jar***) are all deployed in split mode and they can all take advantage of the mentioned features.
- Though applications intended to run in an Android environment are fully written in Java, the Jade Android application model is quite different with respect to that of normal Java applications (package ***jade.android***).
- Android service that bundles JADE for Android is called ***MicroRuntimeService***, responsible for configuring the JADE environment and for starting and stopping the JADE runtime when required.
- In order to make a service communicating with other Android application components, the ***Context.bindService()*** method must be used to bind with the ***MicroRuntimeServiceBinder***.

JADE Android Code

- The first operation to activate the JADE runtime from an Android activity is to retrieve a ***MicroRuntimeServiceBinder*** object using a subclass of ***ServiceConnection***

```
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 by means of the Context.bindService()

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

JADE Android Code

- Having retrieved the ***MicroRuntimeServiceBinder*** object it is now possible to start a JADE split container

```
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
      ...
    }
});
```

- Host and port where the main container is running (as well as other configuration options) must be specified in a ***Properties*** object.

- All operations are **asynchronous** and the result is made available by means of a ***RuntimeCallback*** object

JADE Android Code

- Once the JADE runtime is up and running it is possible to start an agent:

```
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
      ...
    }
  });
```

- The application context of the current Android application is passed to the agent as first argument (allows the agent to access the Android API)

Chat Client Agent Example

- External components can trigger agent tasks by invoking methods of O2A (Object-to-Agent) interface mechanism
- ***Chat Client Agent*** Example: (Leap/android/demo/src)

```
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

1) To expose an ***Agent*** to an ***O2A interface***

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

2) Android activity can retrieve the ***O2A interface*** exposed by the ***Agent***

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

Intent Class

- ***Intent*** object (package ***android.content)*** is used to notify the user through the GUI
  - E.g. a new chat message has been received

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

- For the Android application to register a receiver for the intents sent by ***Agents***, Android requires an object of a subclass of ***BroadcastReceiver***

```
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");
      ...
    }
}
```

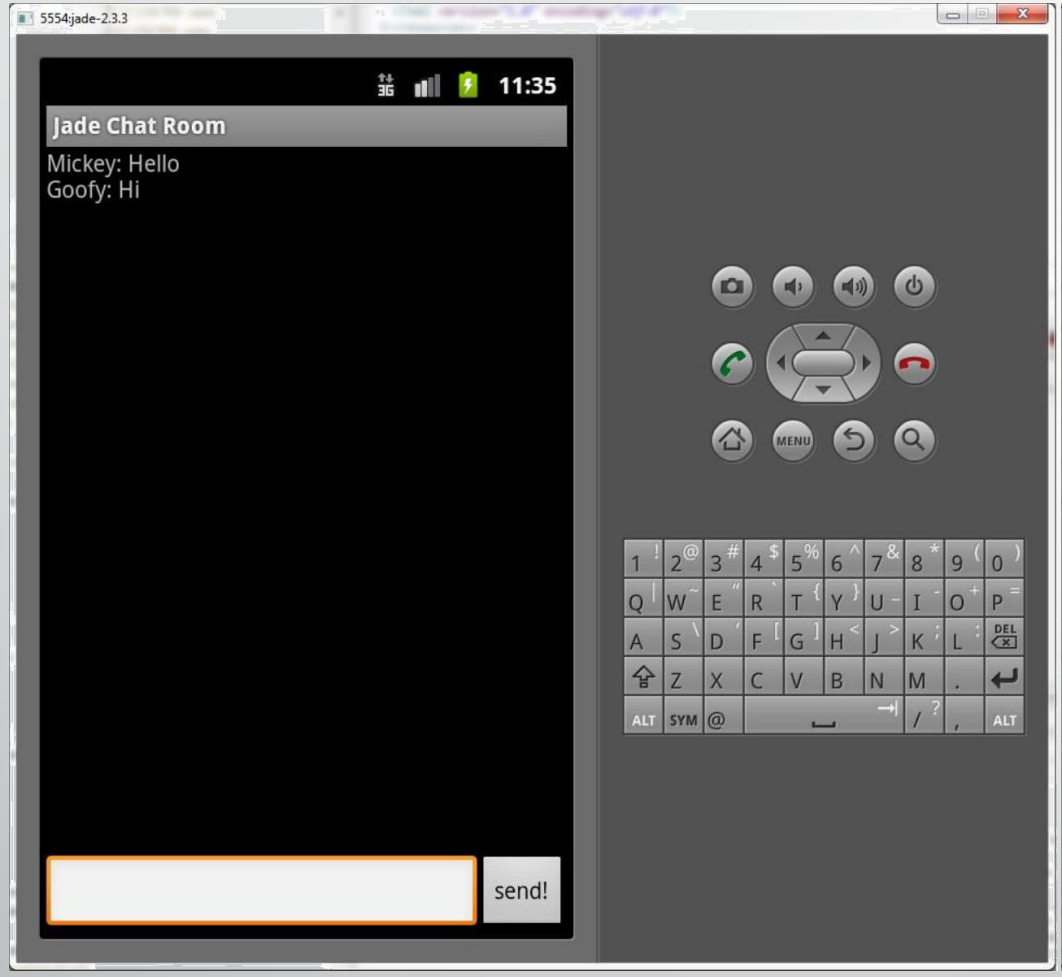
Intent Class

- An Android activity can then register a receiver to intercept intents broadcast from an ***Agent***

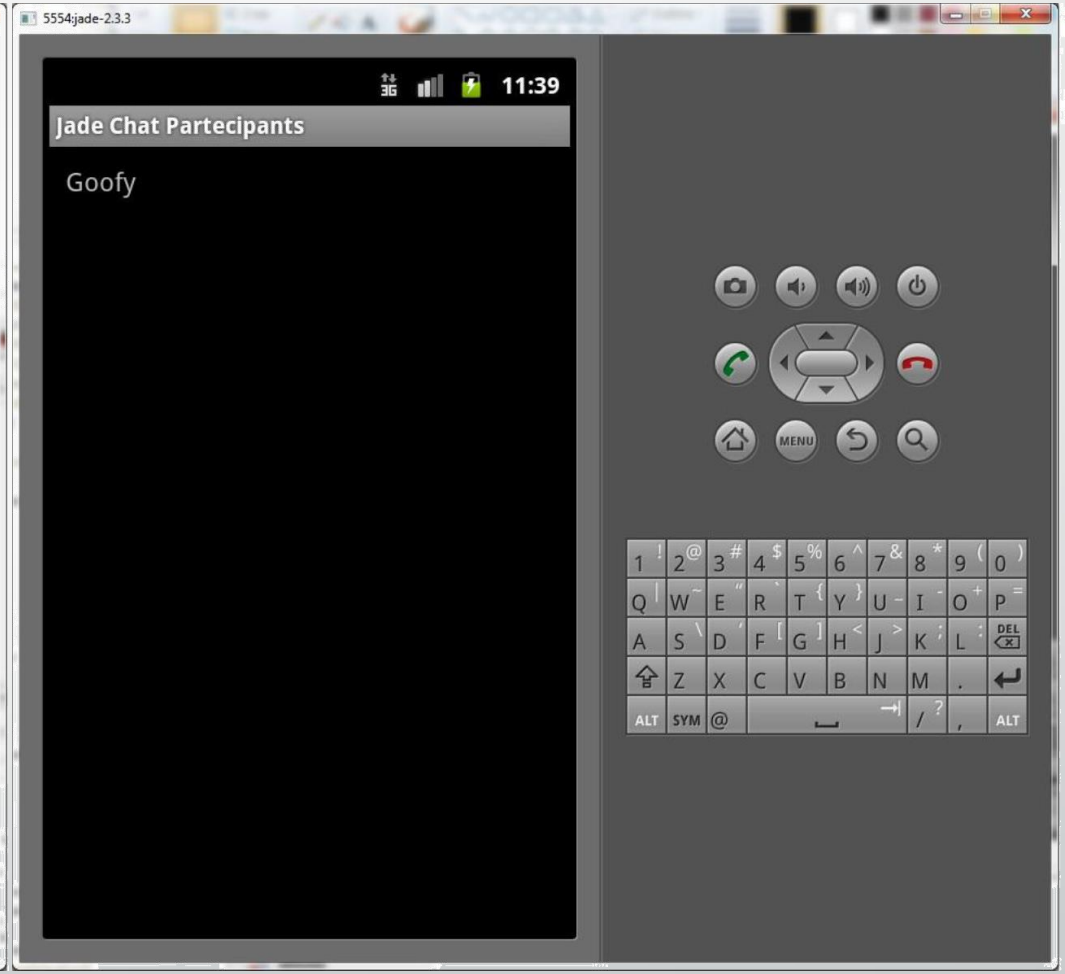
```
MyReceiver myReceiver = new MyReceiver();
IntentFilter filter = new IntentFilter();
filter.addAction("jade.demo.chat.REFRESH");
registerReceiver(myReceiver, filter);
```

- ***Verification of Code Available:***
  - ***LeapAddOn\leap\android\demo\src\chat\client\gui\MainActivity.java***
  - ***LeapAddOn\leap\android\demo\src\chat\client\gui\ChatActivity***
  - ***LeapAddOn\leap\android\demo\src\chat\client\gui\ChatClientAgent.java***

Chat Client Agent



The chat form



The participants list

Conclusions

- Android developers can leverage the features that JADE provides to **simplify** the development of **decentralized** and **distributed applications**
  - Possibility of combining the **expressiveness of IEEE FIPA communication** with the **power of Android**
- By means of JADE, an Android application can easily **embed agents** and therefore become **part of a wider distributed system,** including other mobile devices
- JADE for Android provides an **interface** that allows an application to **start** a local ***Agent***, **trigger behaviours** and **exchange** application-specific objects with ***Agents***
- It is possible to **discover** remote peers carry out complex conversations, **exploit** JADE ontologies to handle structured messages, **perform background activities** according to the behaviour composition model and take advantage of all features of JADE.

Bibliography

- Bergenti, F., Caire, G., & Gotta, D. (2014, September). Agents on the Move: JADE for Android Devices. In *WOA* (Vol. 1260).
- Gotta, D., Trucco, T., Ughetti, M., Semeria, S., Cucè, C., & Porcino, A. M. (2008). JADE Android Add-on Guide.
- 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.
- F. Bellifemine, G. Caire, and D. Greenwood, Developing multi-agent systems with JADE. Wiley Series in Agent Technology, 2007.
- JADE (Java Agent DEvelopment framework) web site. [Online]. Available: http://jade.tilab.com

JADE Programming for Android

Integrated Master's in Informatics Engineering

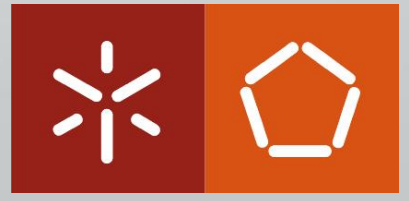
Intelligent Agents

2018/2019

Synthetic Intelligence Lab

Filipe Gonçalves

César Analide



ISLab