Developing intelligent agents on the
Android platform

Jorge Agüero, Miguel Rebollo, Carlos Carrascosa, Vicente Julián
Departamento de sistemas informáticos y computación
Universidad Politécnica de Valencia
Camino de Vera S/N 46022 Valencia (Spain)
{jaguero, mreollo, carrasco, vinglada}@dsic.upv.es

Abstract

Nowadays, agents may run on different hardware platforms, which is a useful approach in Ubiquitous Computing in order to achieve intelligent agents embedded in the environment. This approach can be considered the vision of an Intelligent Ambient. In this paper, a new agent model “specially” designed for the recent Android\(^1\) Google SDK is presented, where the Android mobile phone can be considered as a software agent. This agent model has an approach which is more practical than theoretical because it uses well-known abstractions which allow the proposed model to be implemented on different systems. The appearance of Android as an open system based on Linux has signalled new hope in the implementation of embedded agents. Finally, the proposed model abstractions used to design the Android agent have been employed to implement a simple example which shows the applicability of the proposal.

\textbf{keywords:} Agent architecture, agent model.

1 Introduction

\textit{Ubiquitous Computing} or \textit{Pervasive Computation} [12] is a paradigm in which the technology is virtually invisible to our environment, because it has been inserted into the ambient with the objective of improving people’s quality of life, creating an intelligent ambient [7]. In Pervasive Computation, awareness is becoming common characteristic of our society with the appearance of electronic devices incorporated into all kinds of fixed or mobile objects (Embedded system), connected to each other via networks. It is a paradigm in which computing technology becomes virtually invisible as a result of computer artifacts being embedded in our everyday environment [8].

One approach to implement pervasive computing is to embed intelligent agents. An intelligent agent is a hardware or (more usually) software-based computer system which has the following properties: autonomy, social ability, reactivity and pro-activeness [13]. Embedded-computers containing these agents are normally referred to as \textit{embedded-agents}[11]. Each embedded agent is an

\(^1\)Android is trademark of Open Handset Alliance, of which Google is a member
autonomous entity, and it is common for such embedded-agents to have network connections allowing them to communicate and cooperate with other embedded agents, as part of a multi-embedded agent system.

The challenge, however, is how to manage and implement the intelligent mechanisms used for these embedded agents, bearing in mind the limited processing power and memory capacity of embedded computational hardware, the lack of tools for the development of embedded applications and the lack of standardisation. These challenges and other known problems [9], a remarkable difference between the conceptual agent model and the implemented, or expected, agent has been detected. For example, it is widely known that Java is a language which is frequently used in the development of agents, but the difference between Java for personal computers (J2SE) and Java embedded devices (J2ME), produces big changes in the implemented agents. This problem is often solved by adding new middleware layers, but the functionality of the agent is reduced on many platforms [7].

But now, with the arrival of the SDK Android made by Google as a platform for the development of embedded applications in mobile phones, a new approach for implementing embedded intelligent agents has been created. Android is an open source platform and the development of the applications is made with a new Java library (Java Android library), which is very similar to Java for personal computers (J2SE) [3]. Furthermore, the Android Linux Kernel could possibly be migrated to other platforms or electronic devices, allowing such agents to be executed in a wide variety of devices.

To sum up, the basic idea is to present an agent model that can be designed using components or abstractions that can be deployed on any programming platform, such as the Android SDK, which allows such an agent model to be implemented. This will demonstrate the feasibility of implementing embedded agents using these abstractions, reducing the gap between the design of embedded agents and their implementation. The rest of the document is structured as follows. Section 2 describes the proposed agent model. Section 3 briefly explains the main components of the Android Platform. Section 4 details agent implementation in Android. In section 5 a simple example demonstrating the viability of implementing the model in the Android SDK is shown. Finally, the conclusions of the present work are expounded in section 6.

2 Agent Meta-Model

The main problem to define a platform-independent agent model is to select the appropriated concepts that will be included in the model and that will be used to build the different features and classes of agents. At the moment, there is a large amount of agent models that provide a high-level description of their components and their functionalities, but they need to be changed and manually implemented when applied to specific agent platforms. To define the agent model presented in this paper, some of the most used and complete agent model proposals have been studied. The purpose of this study was to extract their common features and adapt them to the current proposal. In this way, TROPOS[5], GAIA[14], AUMIL[4], INGENIAS [10] and AML[6] have been considered. So, the proposed process allows to do the analysis and design of the system according to different well-known methodologies (corresponds to the CIM). Then,
the obtained design will be transformed in terms of our proposed meta-model corresponds to the PIM. The main components and basic concepts employed in the meta-model are summarized in Table ???. Moreover, the relationships between these main components are shown in Figure 1. The main components and basic concepts employed in the meta-model are summarized in Table ???. This meta-model is called agent-π (agent-PI: agent Platform Independent).

Figure 1: Summarized agent-π meta-model

The highest-level entity to be considered is the agent. At this level, organizations of a higher order, group rules or behaviour norms, are not taken into account in this work.

2.1 Agent

An Agent has an identifier and a public name. The Environment is represented by means of its relationship with the ambient or surroundings, allowing the definition of input and output ports for communicating with the outside. The agent’s knowledge base is kept in its Belief set and its Goal set. The agent has two message queues, Input and Output, to communicate with the outside, and they temporarily store incoming and outgoing messages respectively. Besides messages, the agent can be aware of event arrival, which is stored in EventQueue. Lastly, the agent has a State, related its life-cycle and its visibility to other agents.

With regards to the problem-solving methods, the agent has a set of core components. -The Capabilities- which represent the know-how of the agent and follow an event-condition-action scheme. To improve the efficiency of the agent, Capabilities are grouped into Behaviours that define the roles the agent can play. By doing so, any Capability related to the current situation can be kept active (ready), preventing the overloading of agent.
2.2 Behaviours

A set of Behaviours is defined in the agent to distinguish between different environments and attention focuses. Basically, Behaviours are used to reduce and delimit the knowledge the agent has to use to solve a problem. Therefore, those methods, data, events or messages that are not related to the current agent stage should not be considered. In this way, the agent’s efficiency in the problem-solving process is improved. A Behaviour has a Name to identify itself. A Goals Set is also associated to it, which may be used either as activation or maintenance conditions (see Figure 2(a)). Lastly, a state indicating its current activation situation. More than one Behaviour may be active at the same time.

2.3 Capabilities

The tasks that the agent knows how to fulfill are modeled as Capabilities. Capabilities are stored inside Behaviours and they model the agent’s answer to certain events. A Capability is characterised by a Name that identifies it, its trigger Event, an activation Condition and the Task that has to be executed when the event arrives and the indicated condition is fulfilled (see Figure 2(b)). The State of the Capability is also indicated. Only Capabilities belonging to current active Behaviours are executed.

An event is any notification received by the agent informing it that something that may be of interest has happened in the environment or inside the agent. This may have caused the activation of a new Capability.

Figure 2: (a) Behaviours in agent-π, (b) Capabilities in agent-π.

All of the Capabilities of an active Behaviour will be in a state marked as Active. When an event arrives, the state of the Capability changes to Relevant and its activation condition is evaluated. If this condition is fulfilled, the state passes to Applyable and the associated Task begins its execution. When this Task ends, the Capability returns to Active again and it awaits the arrival of new events. When a Behaviour becomes inactive, all of its Capabilities stop their execution and change their state to inactive. It is assumed that all of the Capabilities of an active Behaviour can be concurrently executed, so that the system has to take the necessary steps to avoid deadlocks and inconsistencies during their execution.

2.4 Task

The last component of the agent model is the Task. Tasks are the elements containing the code associated to the agent’s Capabilities. One Task in execution
belongs to only one Capability and it will remain in execution until its completion or until the Capability is interrupted because the Behaviour it pertains to is deactivated. No recovery or resumption method for interrupted Tasks has been defined. On the other hand, the agent must have some kind of "Safe Stop" mechanism to prevent it from falling into inconsistent states.

3 Android Google: A new platform for mobile devices

Android is a software stack for mobile devices which includes an operating system, middleware and key applications. Developers can create applications for the platform using the Android SDK [3]. Applications are written using the Java programming language and run on Dalvik\(^2\), a custom virtual machine designed for embedded use, which runs on top of a Linux kernel. The main components of the Android operating system are:

- **Applications**: Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts and more. All applications are written using the Java programming language. Every Android application runs on its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently.

- **Application Framework**: Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework).

- **Libraries**: Android includes a set of libraries used by various components of the Android system. For example, some of the core libraries support the playback and recording of many popular audio and video formats, and also the core Web browser engine and SQLite for maintenance database.

- **Android Runtime**: Android includes a set of core libraries which provides most of the functionality for the Java programming language. Android Runtime provides abstract components for creating applications.

- **Linux Kernel**: Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

There are four building blocks in an Android application: **Activity**, **Intent Receiver**, **Service** and **Content Provider**. An application does not need to use all of them, but they can be combined in any order to create an application. Each application has a manifest file, called **AndroidManifest.xml**, which lists all of the components used in the application. This is an XML file where you declare the components of your application:

\(^2\)Android Virtual Machine
• **Activity:** The most common of the four Android building blocks. An activity is usually a single process with an interface in an application. Each Activity is implemented as a single class that extends the Activity base class. The Activity displays a user interface, composed of Views, which responds to events.

• **Intent Receiver:** An event handler. It allows the reaction of the application to events (called Intents) to be defined. Examples of these are when the phone rings, when the data network is available or when it’s midnight. Intent Receivers do not display a UI (User Interface), although they may use notifications to alert the user if something interesting has happened. The application does not have to be running for its Intent Receivers to be called; the system will start the application, if necessary, when an Intent Receiver is triggered.

• **Service:** A Service is a long-life code that runs without a UI. It is a process running in the background without interaction with the user for an indeterminate period of time. A good example of this is a media player application, whereby the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen. In this case, a Service will remain running in the background to keep the music going.

• **Content Provider:** Applications can store their data in files, a database or any other mechanism. The Content Provider, however, is useful for sharing data with other Android applications. The Content Provider is a class that implements a standard set of methods in order to let other applications store and retrieve the type of data that is handled by that Content Provider.

4 The Andromeda Platform

**Andromeda** (ANDROid eMbeddED Agent platform)[1, 2] is an agent platform specifically oriented to embedded agents over the Android operating system. The agents developed inside this platform are based on the agent-π meta-model. Android can be seen as a software system specifically designed for mobile devices which includes an operating system, a middleware and key applications. Developers can create applications for the platform using the Android SDK. Applications are written using the Java programming language and they run on Dalvik (the Android Virtual Machine), a custom virtual machine designed for embedded use, which runs on top of a Linux kernel.

The proposed Andromeda platform includes all the abstract concepts of the agent-π meta-model. The implementation was done using the main API components of Android (SDK 1.0). The correspondence between the Android components and the main agent-π abstract concepts are explained below.

---

4.1 Agent

The Agent class is designed to handle the arrival of events. Therefore an agent has to consider the changes to its environment (this may be of interest to the agent) to determine its future actions activating and deactivated the appropriate Behaviours in response to any internal or external situation. In this way, Agent class is implemented as one Android Service.

To implement the agent-π model, some methods of Service class have to be overloaded. The onCreate() method allows agent variables to be initialised. Then the onStart() method is executed, enabling the agent components. The agent is executed until the user decides to stop its execution. At that moment, the user employs the selfstop() or stopService() method, allowing the effective termination of the agent execution. Every agent component is stopped and destroyed (Tasks, Capabilities and Behaviours).

The agent interface designed has several methods that allow the agent-π to be implemented, but there are two methods that it is important to mention: the init() method, where the user may write the code necessary to initialise the agent, and the run() method, which activates roles that the agent has to play (activate the Behaviours). The init() is executed within the Service’s onStart(), which is called when the agent starts for first time. The Agent class can also launch a UI (User Interface), one Activity, to interact with users and to show its internal state and progress. The programming interface is shown in Figure 3.

```java
public class Agent extends Service {
    private AID myAID;
    private Goals mygoals;
    private List<Behaviour> myListBehaviours;
    ...
    public void init()
    private void run()
    public boolean changestate(Behaviour behaviour, boolean cond)
    public void addbehav(Behaviour myBehaviour)
    public void destroy()
    protected void agentDestroy()
}
```

Figure 3: Agent interface of agent-π.

4.2 Behaviour

The Behaviour class works as a container of the Agent Capabilities and it can group as many Capabilities as the user desires. All of them can be activated and deactivated when events arrive. Behaviours are implemented by means of an IntentReceiver class from the Android APIs. This base class receives intents sent by events from the Android platform. An IntentReceiver has to be dynamically registered to treat intents, using the registerReceiver() method. The IntentReceiver will be running on the main agent thread. The Receiver will be called when an intent arrives which matches the intents filters, i.e. bind an intent to an object that is the receiver of the intent.

As the agent may play one or more roles at any moment, the Behaviour class can activate new roles to register the respective handler (of intents). For
example, a role may be activated as a special Agent Behaviour when the phone battery is low. This can be done by an IntentReceiver that receives the intent LOW_BATTERY.

The Behaviour interface designed has several methods, but two main methods are provided to add and remove the Capabilities: add(capability) and remove(capability). When the user has to create a new Behaviour, the constructor method must be called, which supplies the Behaviour name and its trigger Event as Behaviour(Name, Event). To illustrate this, Figure 4 shows part of the programming interface implemented.

```
public class Behaviour extends IntentReceiver{
    private List<Capability> myListCapability;

    public void add(Capability mCapability)
    public boolean remove(Capability mCapability)
    public void activate()
    public void deactivate()
    ...
}
```

Figure 4: Behaviour interface of agent-π.

### 4.3 Capabilities

Capabilities are characterised by their trigger Event, an activation Condition and the Task that must be executed when an event arrives, and the indicated condition that is fulfilled. The Capability is implemented by means of an IntentReceiver class from the Android APIs. This base class receives intents sent from events in the Android platform, so that it is similar to Behaviours.

A Capability is always running an IntentReceiver. When an intent arrives and the condition is fulfilled, the code in onReceiveIntent() method is considered to be a foreground process and will be kept running by the system to manipulate the intent. It is at this moment that the Task is launched.

The Capability interface designed has one important method for matching a Task to its corresponding Capability: this is the addTaskRun(task) method. When the user has to create a new Capability the constructor method must be called, supplying the Capability name and its trigger Event as Capability(Name, Event). In Figure 5 part of the programming interface is shown.

```
public class Capability extends IntentReceiver{
    private Condition condition;
    private Boolean state;

    public void activate()
    public void deactivate()
    public void setCondition(Condition condition)
    public boolean addTaskRun(Task nametask)
    ...
}
```

Figure 5: Capability interface of agent-π.
4.4 Tasks

Now, Task class is one special process to run as an Android Service. To implement the Task, some methods of Service class have to be overloaded. The onCreate() method allows Task variables to be initialised when it is launched. The onStart() method allows the user code to be executed, throughout a call to a doing() method that has to be overloaded by the programmer. Now, the main method of Task interface is doing(), where the user writes the Java program to be executed (see the interface in Figure 6).

```java
public class Task extends Service implements Runnable {
    public MsgQueue outputQ
    ...
    public void doing()
    public synchronized void pause()
    public synchronized void resume()
    public void taskDestroy()
    public final void send(Message msg)
    public final Message receive(MessageTemplate pattern)
    public final Message blkReceive(MessageTemplate pattern, long time)
    ...
}
```

Figure 6: Task interface of agent-π.

Finally, the intents are used to model the Goals that activate the Behaviours and Events that allow the Tasks of a Capability to be executed. To manipulate and store the agent Beliefs, the ContentProvider is used as a database. The Communication between agents is implemented creating FIPA ACL messages.

Table 1 shows the Android blocks used for building components of the agent-π model and other necessary components. Thereby this model inserts a new layer in the Android system architecture[3]. This new layer, called Agent interface, modifies the architecture, as seen in Figure 7.

<table>
<thead>
<tr>
<th>agent-π Components</th>
<th>Android Components</th>
<th>Overloaded methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>Service</td>
<td>onCreate(), onStart(), onDestroy()</td>
</tr>
<tr>
<td>Behaviour</td>
<td>IntentReceiver</td>
<td>registerReceiver(), onReceiveIntent()</td>
</tr>
<tr>
<td>Capability</td>
<td>IntentReceiver</td>
<td>registerReceiver(), onReceiveIntent()</td>
</tr>
<tr>
<td>Task</td>
<td>Service</td>
<td>onStart(), onDestroy()</td>
</tr>
<tr>
<td>Events</td>
<td>Intents</td>
<td>IntentFilter()</td>
</tr>
<tr>
<td>Believes</td>
<td>Contentprovider</td>
<td>-</td>
</tr>
<tr>
<td>ACL Communications</td>
<td>http</td>
<td>-</td>
</tr>
</tbody>
</table>

5 Example

An example of two agents talking by means of a chat session is used to show the applicability of this proposal. So, a simplified Chat Session between two agents that send and receive ACL messages is proposed. This simple example is presented with academic goals, to explain and show how to use an agent interface designed in Android platform only. This example does not attempt to illustrate the interaction of a complex agent.

The implementation of the agent was done in an Android emulator, because currently there are no real phones where applications can run. The first step
of the design process is to identify the roles of the agents. As agents simply send and receive information from each other, we model the agent with only one Behaviour, which is called CHAT. A simple chat session has one Capability where users send information whenever they want and another Capability which awaits the arrival of a message. Therefore, two Capabilities are created: one to transmit a message and the other to receive it (see Figure 8(a)).

Each agent Capability has the mission of sending or receiving messages. It is necessary to remember that a Capability receives intents. When the intent arrives and the condition is fulfilled, the Task is launched. The Capability that sends messages is called SendMsg, and its Task, task_Send, transmits the information when users press the send button (see Figure 8(b)).

The Capability that receives messages is called ReceiveMsg and its Task, task_Receive, waits for the arrival of other agent messages. So, agents are ready to begin the process of communication and the exchange of information in the Chat. Messages will be displayed on the phone screen. Now, to program the agent interface (for this preliminary implementation), proceed as explained below:

- Create one Behaviour with name= CHAT.
Create one Capability for sending messages, with name= SendMsg, and the condition (intent), which wakes it up.

Then add the Task (task_Send) that permits the ACL message to be sent.

Create another Capability for receiving messages, with name= ReceiveMsg, and the condition (intent), which wakes it up.

Then add the Task (task_Receive) that permits the ACL message to be received.

Add these two Capabilities to the Behaviour.

Add the Behaviour, using the addbehav() method. The agent is executed and the messages will be displayed on the emulator screen (see Figure 9).

Figure 9: Chat in the emulator screen.

Figure 10: Task for send Chat messages.

6 Conclusions

A general agent model for building intelligent agents on the Android platform has been presented. This model can be easily adapted to almost any platform.
public class MyAgent extends Agent {
    public void init(){
        //Create one Behaviour
        Behaviour myBehaviour= new Behaviour("CHAT");

        //Create two capabilities and its condition trigger
        Capability myCapabilityTX = new Capability("SendMsg");
        Capability myCapabilityRX = new Capability("ReceiveMsg");

        //Condition and intent trigger of send
        Condition mycondSend = new Condition() {
            @Override
            public boolean expression(Intent intent) {
                if (intent.getAction() == "Android.intent.action.MY_SENDMSG") {
                    return true;
                } else { return false; }
            }
        };
        //Set the conditions different of null
        myCapabilityTX.setCondition(mycondSend);
        myCapabilityRX.setCondition(mycondReceive);

        //Create and add tasks that send and receive the chat messages
        Task myTaskTX = new task_send;
        myCapabilityTX.addTaskRun(myTaskTX);
        Task myTaskRX = new task_Receive;
        myCapabilityRX.addTaskRun(myTaskRX);

        //Add the Capabilities to the Behaviour
        myBehaviour.add(myCapabilityTX);
        myBehaviour.add(myCapabilityRX);

        //Add Behaviour the agent and execute it
        addbehav(myBehaviour);
    }
}

Figure 11: agent-π Agents Chat.

or architecture hardware/software. Moreover, the agent model has been implemented and tested on the Android platform. The agent interface designed allows embedded agents to be implemented according the requirements of the user.

The use of the Android platform demonstrated the utility and probed the feasibility of designing a platform-independent agent. The use of the proposed model abstractions for agent-π agent reduces the gap between the theoretical model and its implementation.

The embedded agent design achieves the functionality required of it. Furthermore, the Android platform promises to be a new platform for implementing novel agent models. This is because Java API is very similar to the Personal Computer version, allowing an embedded agent-based approach to be implemented with even more advanced mechanisms. This is a useful feature in Pervasive Computing. Additionally, as Android platform is a Linux system, there is a high probability that the platform can be migrated to a range of different devices.

As future work, the services that this first version of the agent can deliver will be enriched and enhanced. The prototype has been developed using an emulator for Android. The evaluation of the performance of the agent architecture presented will be carried out when the first mobile phone using the Android system is launched.
While this article was being written a Jade\(^4\) version for Android system was developed. Though the authors have not carried out an in-depth evaluation of Jade in the Android architecture, it must be stated that the agent model described in this paper presents a conceptually different to JADE model, because this model is wholly integrated with Android’s building block and JADE is not.

7 Acknowledgment

This work was partially supported by CONSOLIDER-INGENIO 2010 under grant CSD2007-00022 and by the Spanish government and FEDER funds under TIN2006-14630-C0301 project.

References


---

\(^4\)http://jade.tilab.com/


