Towards on embedded agent model for Android mobiles

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, mrebollo, carrasco, vinglada}@dsic.upv.es

ABSTRACT

This paper presents, a new agent model "specially" designed for the new *Android*[1] Google SDK, where the *Android* mobile phone can be considered as a software agent. This agent model has an approach more practical than theoretical because it uses abstractions which makes possible its implementation on differents systems. The appearance of *Android* as an open system based on Linux has created new expectations for agent implementation. Agents may run in different hardware platforms, one approach useful in Ubiquitous Computing to achieve intelligent agents embedded in the environment. This vision can be considered a real intelligent ambient.

Keywords

Agent model, embedded agent, *Android Google*.

1. INTRODUCTION

The *Ubiquitous Computing* or *Pervasive Computation* [9] is a paradigm in which the technology is virtually invisible in our environment, because it has been inserted in the ambient with the objective of improving people's life quality, creating an *intelligent ambient* [5]. In the *Pervasive Computation*, awareness is becoming an habitual characteristic of our society with the appearance of electronics devices incorporated in all class of fixed or mobile objects (Embedded system), connected by means of networks to each other. It is a paradigm in which computing technology becomes virtually invisible as a result of being embedded computer artifact's into our everyday environment [6].

One approach to implement pervasive computing is to embed intelligent agents. An intelligent agent is a hardware or (more usually) software-based computer system that has the following properties: autonomy, social ability, reactivity and pro-activeness [12]. Embedded-computers that contain these agents are normally referred to as *embedded-agents* [11]. Each embedded agent is an 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 to implement the intelligent mechanisms used for these embedded agents due to the limited processing power and memory capability of embedded computational hardware, the lack of tools for the development of embedded applications and the lack of standardisation. By these challenges and other found problems [8], a remarkable difference between the conceptual agent model and the implemented or expected agent has been detected.

Today, the appearance of the *Android* SDK made by Google as a platform for the development of embedded applications in mobile phones creates a new approach for implementing embedded intelligent agent. Android is an open source platform and the application development is made with a new Java library (Java *Android* library) that is very close to Java for personal computers (J2SE) [1]. Furthermore, there exists the possibility that the *Android* Linux Kernel can be migrated to other platforms or electronic devices, allowing to 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, and how the *Android* SDK allows the implementation of this agent model. This will show 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 details agent implementation in *Android*. Finally, conclusions are shown in section 4.

2. AGENT PLATFORM INDEPENDENT MODEL

Major challenges in pervasive computing include invisibility or unawareness, proactivity, mobility, privacy, security and trust [5]. In such environments, hardware and software entities are expected to function autonomously, continually and correctly.

[1]Android is trademark of Open Handset Alliance, where Google is a member

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3945
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3945

Traditionally, agents have been employed to work on behalf of users, devices and applications [11]. The agents can be effectively used to provide transparent and invisible interfaces between different entities in the environment. Agent interaction is an integral part of pervasive (intelligent) environments because agents acquire and apply effectively knowledge in their ambient.

At the moment, there is a large amount of agent models that provide a high-level description of their components and their functionality. To define the agent model presented in this paper, some of the most used and complete agent model proposals have been studied. The first step was to extract their common features and adapt it to the new proposal. In this way, Tropos [3], Gaia [13], Opera [7], Ingenias [10] and AML [4] have been considered, because they are some of the most commonly used.

An agent model provides an abstract vision of its main components and their existing relationship. The approach to build the agent model uses the MDA (Model Driven Architecture) philosophy, which divides the different models into two classes: a set of platform independent models (PIM) and another set closely related with the supporting platform (platform specific model - PSM). It is a way to develop applications which allows us to separate the logic of the application from the platform used to its implementation. This philosophy, used in "classic software", is also valid for agent development. Figure 1 shows the agent model presented in this paper, that is called *APIM* (Agent Platform Independent Model).

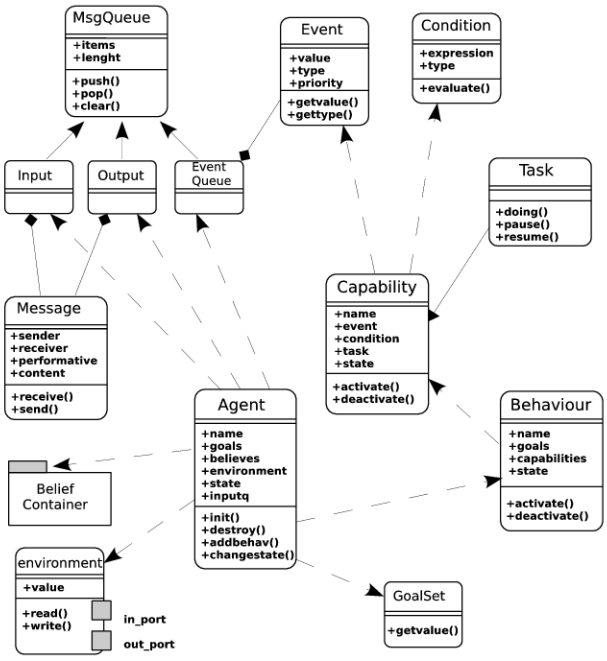

Figure 1: *APIM* structure.

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

2.1 Agent

An **Agent** has an identifier and a public *name*. The environment is represented by means of its relationship with *Environment*, allowing to define input and output ports to communicate with the outside. Agent's knowledge base is kept in its *belief* set and its *goal set*. The agent has two messages queues, *Input* and *Output*, to communicate with the outside, which temporally store incoming and outcoming messages respectively. Besides messages, the agent can be aware of event arrival, being stored in *EventQueue*. Lastly, the agent has a *State*, related with its life-cycle and with its visibility by 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 that, can be kept active (ready) any *Capability* related with the current situation, avoiding overload the agent with unnecessary knowledge.

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. So, those methods, data, events or messages that are not related with the current agent stage should not be considered. In this way, the agent's efficiency in problem-solving process is improved. A *Behaviour* has a *Name* to identify itself. It also has associated a *Goals set* that may be used either as activation or maintenance conditions. Lastly, it includes a *state* indicating its current activation situation. It is important to remark that more than one *Behaviour* may be active at the same time.

2.3 Capabilities

An *event* is any notification arriving to the agent informing that something that may be of interest for the agent has happened in the environment or inside the agent. It may have caused the activation of a new *Capability*.

The *Tasks* that the agent knows how to fulfill are modelled as **Capabilities**. *Capabilities* are stored inside the *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 have to be executed when the event arrives and the indicated condition is fulfilled. It is also indicated the *State* the *Capability* has. Only the *Capabilities* belonging to current active *Behaviours* are executed.

All the *Capabilities* of an active *Behaviour* will be in a state marked as *Active*. When an event arrives, the *Capability* state 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* return to *Active* again and it remains waiting the arrival of new events. When a

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3945
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3945

Behaviour becomes *Inactive*, all its *Capabilities* stop their execution and change their state to inactive. It is assumed that all the *Capabilities* of an active *Behaviour* can be concurrently executed, so that the system have to provide the needed methods to avoid deadlocks and inconsistencies during their execution.

2.4 Tasks

The last component of the agent model is the **Task**. *Tasks* are the elements containing the code associated to the *Capabilities* of the agent. One *Task* in execution belongs only to 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. It is not defined any method of recovering nor resuming of interrupted *Tasks*. On the other hand, the agent must have some mechanism of "safe stop" to avoid the agent to fall in inconsistent states.

3. IMPLEMENTING APIM IN ANDROID

The developing of *APIM* in *Android* was made using *Android* building block APIs (the API Version m5-rc14) [2]. There are four main components to model *APIM* agents: **Agent**, **Behaviour**, **Capability** and **Task** [1]. Table 1 shows the *Android* blocks used for building components of the *APIM* model and another necessary components.

The design presented can be seen as an interface to implement the agent according to the users requirements and needs. This interface uses specific components provided by the API, as previously commented. Thereby this model inserts a new layer in the *Android* system architecture[1]. This new layer, called *Agent interface*, modifies the architecture as it is seen in the figure 2.

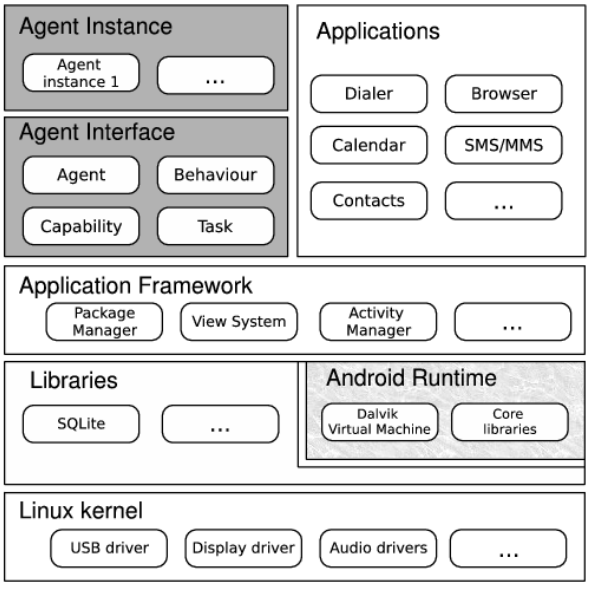

Figure 2: Agent interface in *Android* System Architecture.

3.1 Agent

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

To implement the *APIM* model, some methods of *Service* class have to be overloaded. The `onCreate()` method allows to initialise the agent variables. After the `onStart()` method is executed that enables the agent components. The agent is executed until the user decides to stop its execution. In this moment, the user call `selfstop()` or `stopService()` method, allowing effectively to terminate the agent execution. Every agent component is stopped and destroyed (*Tasks*, *Capabilities* and *Behaviours*).

The agent interface designed has several methods that allow to implement the *APIM*, but there are two methods that are important to mention: the `init()` method, where the user may write the code necessary to initialise the agent, and the `run()` method, that activates roles that the agent has to play (active the *Behaviours*). The `init()` is executed within *Service's* `onStart()`, that 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.

3.2 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 mean of an *IntentReceiver* class of the *Android* APIs. This base class receive intents sent by events of the *Android* platform. An *IntentReceiver* have to be dynamically registered to treat intents, using `registerReceiver()` method.

The *IntentReceiver* will be running on the main agent thread. The Receiver will be called when any intent arrives and it matches with the intents filters, i.e. bind an intent to 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 handlers (of intents). For example, a role may be activated as a special Agent *Behaviour* when the battery phone 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 to *remove* the *Capabilities*: `add(capability)` and `remove(capability)`.

3.3 Capabilities

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

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3945
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3945

Table 1: The *Android* components used in the *APIM* model.

APIM Components	*Android* Components	Overloaded methods
Agent	Service + Activity	onCreate(), onStart(), onDestroy()
Behaviour	IntentReceiver	registerReceiver(), onReceiveIntent()
Capability	IntentReceiver	registerReceiver(), onReceiveIntent()
Task	Service	onStart(), onDestroy()
Goals	Intents	IntentFilter()
Events	Intents	IntentFilter()
Beliefs	Contentprovider	–
ACL Communications	http	–

A *Capability* is always running an *IntentReceiver*. When an intent arrives and the condition is fulfilled, then the code in `onReceiveIntent()` method is considered to be a foreground process and will be kept running by the system for manipulating the intent. In this moment then the *Task* is launched. The *Capability* interface designed has one important method for matching a *Task* to its corresponding *Capability*, the `addTaskRun(task)` method.

3.4 Tasks

Finally, *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 initialise *Task* variables when it is launched the `onStart()` method allows to execute the user code, throughout a call to a `doing()` method that has to be overloaded by the programmer. Now, the main method of *Task* interface is `doing()`, in where the user writes the Java program to be executed.

4. CONCLUSIONS

A general agent model to build intelligent agents in *Android* platform has been presented in this paper. This model can be easily adapted to almost any hardware/software platform or architecture. Moreover, the agent model has been implemented and tested in the new *Android* platform. The agent interface designed makes possible the implementation of embedded agents according to the users requirements.

The use of the *Android* platform shows the utility and proves the feasibility of designing platform independent agents. Moreover, the use of the proposed model abstractions for the *APIM* agent reduces the gap between the theoretical model and its implementation.

As future works, the services the agent can deliver will be enriched and enhanced from this first version. At the moment, the prototype has been developed using an *Android* emulator. A more precise evaluation of the proposed agent architecture will be donde when the first mobile phone using *Android* system will be launched.

5. REFERENCES

[1] Android SDK, An Open Handset Alliance Project, Web Site, http://code.google.com/android/, January 2008.

[2] Android SDK, Download the Android SDK, Web Site, http://code.google.com/android/download_list.html, , January 2008.

[3] Castro J., Kolp M. and Mylopoulos J., A Requirements-Driven Software Development Methodology, Conference on Advanced Information Systems Engineering, 2001.

[4] Cervenka R. and Trencansky I., The Agent Modeling Language – AML. Whitestein Series in Software Agent Technologies and Autonomic Computing, 2007, ISBN: 978-3-7643-8395-4.

[5] Cook D. and Sajal K. Das, How smart are our environments? An updated look at the state of the art. Pervasive and Mobile Computing, Volume 3, Issue 2, 2007.

[6] Davidsson P. and Boman M., Distributed monitoring and control of office buildings by embedded agents. Information Sciences, Volume 171, Issue 4, 2005

[7] Dignum V., A model for organizational interaction: based on agents, founded in logic. PhD Dissertation, Utrecht University, 2003.

[8] Doctor F., Hagras H. and Callaghan V., A type-2 fuzzy embedded agent to realise ambient intelligence in ubiquitous computing environments. Information Sciences, Volume 171, Issue 4, 2005

[9] European Research Consortium for Informatic and Mathematics (ERCIM NEWS), Special: Embedded Intelligence. Number 67, October 2006

[10] Gomez Sanz J. J., Modelado de Sistemas Multi-Agente. PhD Thesis, Universidad Complutense de Madrid, 2002, Spain.

[11] Hagras H., Callaghan V. and Colley M., Intelligent embedded agents. Information Sciences, Volume 171, Issue 4, 2005

[12] Wooldridge, M and Nicholas R. Jennings, Agent Theories, Architectures, and Languages: a Survey, in Wooldridge and Jennings Eds., Intelligent Agents, Berlin: Springer-Verlag, 1995

[13] Zambonelli F., Jennings N. and Wooldridge M., Developing Multiagent Systems: The Gaia Methodology, ACM Transactions on Software Engineering and Methodology, Vol. 12. p. 317-370, 2003.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3945
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3945