Design of a FIPA compliant agent platform for limited devices

Guillermo Diez-Andino Sancho, Rosa M García Rioja, and
M^{a} Celeste Campo Vázquez

Department of Telematic Engineer
University Carlos III de Madrid.
Avd. Universidad 30 28911 Leganés Spain
{gdandino, rgrioja, celeste}@it.uc3m.es.

Abstract. Recent advances in micro-electronic and wireless technologies have fostered the proliferation of small devices with limited communication and processing power. We think that agent technology will be of great help in pervasive systems development. Pervasive systems are inherently dynamic, with devices continuously coming and going. Agents are autonomous software entities that can interact with their environment, and therefore they adapt well to such frequent changes. However, the use of Multi-Agent Systems (MAS) in pervasive environments possess important challenges, and it is necessary to adapt their design to meet these challenges.
The first part of this paper describes the design of a FIPA-compliant agent platform, adapted to the limited devices that work in an ad-hoc network. The authors have presented their proposals to the Working Group FIPA Adhoc, and have been included in the white paper elaborated by this group previous to the standarization. The second part describes the agent platform implementation in real devices, using the Java 2 Micro Edition technology.

1 Introduction

Recent advances in micro-electronic and wireless technologies have fostered the proliferation of small devices with limited communication and processing power. The most important advances are new personal devices, such as PDA or mobile phones, although there are few more devices embedded in the environment around us, in an invisible way, which allows the physical environment around us to adapt itself to our preferences and necessities transparently. This new age on computing is what Mark Weiser described in the article entitled "The Computer for the 21st Century" in 1991 [1] as pervasive computing.

In this kind of environments exists an enormous variety of devices with different hardware limitations to execute applications. In addition, the called ad-hoc networks are formed thanks to wireless communication protocols, where devices can come and go spontaneously. Devices limitations and the ad-hoc network characteristics clearly show the inefficiency of porting traditional solutions to these new systems [2]. Furthermore, most of these new devices are not multiproposal ones but they provide a set of concrete services. When they form an ad-hoc network among them it is possible to compound services in an intelligent way to offer a new service coming up to user expectations transparently, as Weiser wished.

In this scene, software applications for limited devices have to adapt themselves to the memory and processor power restrictions of the device where they are executed and to an intermittent communication with a changing quality. More over, they need to be autonomous to reach the user's goals but without the need of being continuously interacting with it. They have to be able to move around other systems to collect information or to execute tasks that device limitations do not allow performing locally or the direct connectivity doesn‘t allow to reach it. The paradigm of distributed computing adapted to these characteristics is called "Agent".

Our current job is based on using the mobile agents paradigm in ad-hoc networks with middleware technology to develop services in ad-hoc networks formed by limited devices which communicate directly without the need of a central system.

Taking the FIPA standard as a reference, we have developed some additional features to complement the LEAP mobile agent platform. The chosen technology to support the viability of our design is the Java version for limited devices, J2ME. We will use the new added LEAP features in applications using location such as those in MavHome [3].

In section 2 we review briefly the FIPA standard. Then, in section 3 we justify and propose some simplifications to the LEAP architecture in order to adapt it to the characteristics of the ad-hoc networks and the restrictions imposed by the limited systems where it will be implemented. In section 4, we introduce concisely the J2ME software platform and we analyse the viability to implement the designed architecture on it, describing part of the functionality we have already developed. To end up, we enclose in section 5 some conclusions and future lines of our job.

2 FIPA standard

Foundation for Intelligent Physical Agents [1] (FIPA) started its activities in 1995 with the aim to standardize different aspects related to agents technology and multiagent systems. Nowadays, this standard can be considered the most widely recognized and globally extended. Furthermore, it has became a reference to follow when coming to agent based developments.

FIPA specifications are grouped into three types: *component*, which is in charge of standardizing all basic technologies related to agents, *informative* which describes possible solutions to applications using agents in a concrete domain, and *profiles*, which are combined specifications as a *component* which allows validating when an implementation is standard compliant.

[1] http://www.fipa.org

The reference FIPA model of agents platform is described in *FIPA Agent Management Specification* [4]. The functionality of each component is also described. The components are:

- **Agent Management System** (AMS): manages the life cycle of agents, local resources and communication channels. It also provides too a white pages service permitting to locate by the name of the agent.
- **Directory Facilitator** (DF) : provides a yellow pages service, that allows to locate by the agent's capabilities and not its name.
- **Agent Communication Channel** (ACC): manages sending messages among agents in the same platform or in different platforms, and allows migration of agents.

3 Platform design

As we have said in the previous section, FIPA defines three functional elements in its architecture that provide a variety of basic services for the establishment of agents on the platform. A FIPA compliant platform should have these components providing the interfaces and the functionality defined in their standards.

Throughout this section we analize the possibility of simplifying this functionality to do the implementation in limited devices easier, as well as the service offered by the platform adapts itself to the changing environment requirements in which is located.

3.1 Agent Management System

Agent Management System (AMS) manages the life cycle of agents, including those related to mobility, that is, it creates, destroys agents and manages the passing of one platform to another. AMS also gives support to a searcher/finder service called "white pages" which locates agents by their name.

This element is essential in an agent platform, and in our current platform we keep it with full functionality described in FIPA standard, but reducing it to a local scope in the platform. Therefore, we have eliminated the chance of extending the search to other platforms, contacting to remote platforms.

3.2 Directory Facilitator

The adaptation of the functionality of DF to limited devices operating in ad-hoc networks is one of the most important topics of researches performanced. The Working Group AdHoc of FIPA focuses its current efforts on adapting the DF. In fact, the proposal described in this section is one being considered in the group [5].

DF is the element of the platform that allows an agent to discover services offered by other agents in the same environment. In the definition of how a traditional DF works the search for remote services is done using the federation of DFs concept.A DF not only has a register of services given by local agents (native one or not), but it can also register other DFs permitting to do wider searches of services which are register in other DFs. Number of hopes among DF federated is limited by a restricted searcher parameter.

Now, we analyse some disadvantages of DFs federation to discover remote services in ad-hoc networks: First, the search for a concrete remote service always means a communication with this system, due to the fact at the beginning we only know one DF existing in that platform, but not its registered services. That implies doing transmission with no success guaranty. Secondly, if searches conditions permit hopes with more than one level, that is, the search in a remote DF can spread to another remote DF federated in it. On the contrary than in traditional networks, it is possible that services registered on DFs can not be reached, so, the solution is no valid.

3.3 Ad-hoc Discovery Agent

To solve the two exposed problems and with the aim of maintaining the functions specified in FIPA000023 with regards to the DF, so that agents can not modify the way they interact with it when discovering services we propose the introduction of an agent in the FIPA architecture in ad-hoc networks. This agent, **Ad-hoc Discovery Agent (ADA)** is a compulsory element with the follow functionality:

– It will register and maintain updated in the DF of the platform where it resides the remote services offered by the ad-hoc network. It means that the DF will have inputs of remote services but not a remote DF. In this way we will minimize the number of transmissions.
– It will propagate the requested remote services to the DF, when it requests it, because the searching conditions do allow it.
– It will announce through the associated discovery protocol, when possible, the services registered on the DF.

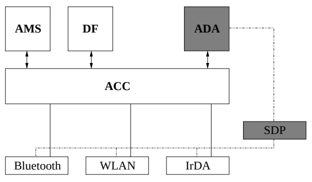

Fig. 1. FIPA ADA introduction

ADA will use a Service Discovery Protocol (SDP) to discover remote services. In this way, it will have to select a service adaptable to the restrictions exposed in the previous section so that an efficient solution in ad-hoc networks can be guaranteed. In [6] an analysis of possible protocols is carried out to finally propose the Pervasive Discovery Protocol (PDP), that has been defined in our working group. The Working Group FIPA AdHoc has published a White Paper in which part of our analysis and description is included [7].

ADA registering in the DF The ADA agent provides a service that has to be registered in the platform DF where it resides. In this way it will always exist an input with the reserved AID corresponding to the ADA in the DF operating in an ad-hoc network.

This agent is registered in the DF establishing in the `:type` parameter of the `service-description` the `ada` value.

When the DF process a `search` and in the search restrictions it is allowed to propagate the searching to the ad-hoc network (it can be reused the `max-depth = 1`) the DF delegates the searching to the ADA.

When the ADA receives through the SDP some new knowledge of a new remote service of the ad-hoc network, it will register it in the DF through a `register` request. As the network changes, these registries will have a `timeout` associated that will manage the ADA so that when it expires a `deregister` request to the DF will be performed. The ADA must only store the corresponding `agent-identifier` and its associated `timeout`.

The ADA can also provide to the SDP (if it applies for it) the local services registered in the DF. It can be achieved by a `search` request over the DF.

3.4 Agent Communication Channel

In a multiagent society, agents must cooperate to do their tasks and reach their goals. Traditional communication mechanisms allow two agents to transfer information, but it is not enough when the goal is reaching a social behaviour. That is, some mechanisms are required to add semantic content to the interchanged messages.

FIPA has developed ACL, which stands for Agent Communication Language. ACC in FIPA architecture is the element that gives support to communication between agents using ACL. To simplify it we have done an analisys of ACL, represented in the next section.

ACL is based on **communicative acts**, which are in charge of passing messages, asking for information, negotiations, performing tasks and handling errors.

Sometimes conversations between agents follow some concrete patterns, repeated in occasionally few situations. Taking into account these patterns, FIPA has defined certain **protocols**. A protocol is a pattern used to guide conversations. They are like guided conversations in which each agent knows what message has to send and which they can receive.

Communication among agents The main idea of the communicative acts among agents centers around the request for carrying out an action in another agent.This operation can be done in different ways depending on whether the action is asked for a concrete agent or if the provider of the action is unknown.

When an agent A asks for an action to another concrete agent B, A sends a `request` message. The receiver can accept it or deny the invocation with an `accept` or `refuse` message respectively. If it accepts, it will do the action and inform the other agent with an `agree` message. If there is an error while the action is being performed, the agent which started the communication will receive a `failure` message.

When agent A needs to do an operation but it doesn't know who is capable of doing it or it's known that there are several agents providing this action, a negotiation communication is established. In this case, the communication starts with a `cfp` message, asking for proposals to the agents. The action to do and some conditions to take into account when the action will be performed are specified in the request. Consulted agents send their proposals to the initial agent with `propose`, they also point the conditions of the proposal. If they refuse the message, they send a `refuse` to the transmitter. Agent A can be waiting for a answer to long, so, in order to avoid it a time is set and all messages received once the time has expired are discarded. Agent A studies the different proposal and finally chooses the best one for itself notifying it to the provider agent with a `accept-proposal` message and informing to the rest with a `reject-proposal` message. Once the proposal is accepted the agent which started the communicative act can cancel it if some change over the initial situation takes place. More over, Agent B can answer with an `inform` pointing that the action has been done or with a `failure` message to indicate a fault has occurred `failure`.

Agents also communicate among them to interchange information. That is, the agent starts the request of information sending a `query-if` or `query-ref` message. Later on, agent B can send the requested information (inform),a failure message when a fault has occurred , with not-understood or refusing the request explaining the reason .

The previous explanation refers to communication among agents, but ACL is also used to ask for operations to the elements integrated in the platform, as for example AMS and DF.

Message structure ACL FIPA message contains one or more message elements needed to get an efficient communication among agents. It is composed of an **identifier** of the communicative act which defines the meaning of the message or the action that the transmitter agent asks for with this concrete message. It also includes a sequence of **parameters** defines as a joint of couple key-value which allows to associate with each concrete communicative act all the necessary information

The minimal group of ACL messages As we have seen in the previous section, ACL has a huge variety of messages and each one has about 11 different parameters which turns the language into a so heightweight one to be implemented in a mobile agents platform for limited devices. This requirements has obliged to look for the minimal group of messages to cover all the necessities among agents.

When you need to request for a concrete operation you can do it with `request`, `inform` and `agree` messages. In the same way, you can use them for requesting information to the platform elements.

The negotiation process, explained before, requires lots of messages but it can also be simplified. Agent A sends a `request` to the AMS or DF asking for all the agents which can perform the concrete operation. The elements of the platform send information using `inform`, and in this way the functionality given by `cfp` and `proposal` is reached. The follow step is deciding who is required to perform the action. This task is done internally to each agent. Once it has decided who is going to perform the operation a `request` message is sent.

To sum up,the minimal group of messages to communicate agents in a platform for limited devices is `request`, `inform`, `agree`. `Cancel` can be replaced establishing a request time.

4 Implementation

In parallel to a FIPA compliant platform design based on LEAP, our working group began to evaluate the possibility of implementing this platform on real devices.

This study carried out firstly to select the implementation programming language. Nowadays we can assure that despite the developments performed in specific languages, it has been Java the most used language in the development of mobile agent platforms due to the following advantages: platform independence, secure execution, dynamic classloading, serialization mechanisms, reflexion, ...

All this made us select as a base for our developments the new version of Java for limited devices J2ME, specifically MIDP. J2ME still maintains some of the characteristics making Java a good mobile agent platform programming language, but there are some other features of this language which have been restricted or even disappeared due to security issues. With regards to these changes we can find for example those allowing us to implement code mobility (dynamic class loading, serialization and reflexion). We will later show on this section how have we solved these problems.

4.1 Agents technology in J2ME

We can find several proposals trying to get involved J2ME limited devices in mobile agent platforms. On this section we make a review of the most important ones.

LEAP The LEAP project consists on a consortium of companies such as Motorola, British Telecom and Siemens that were pioneer when trying to demonstrate the viability of building agent platforms on limited devices. Its applicability is centered in networks with an infrastructure. Because of that, it does not exist a complete agent platform in mobile terminals and their operation always depends on an intermediate connected system.

We can find other related works such as the one developed in Monash University, in which a mobile agent platform has been developed based on the Palm Operating System.

4.2 TAgentsP: our mobile agent CLDC profile

The previous proposals have seriously contributed with regards to the adaptation of mobile agents platforms in limited devices. LEAP is a valid platform in a network with infrastructure in which it is not necessary to have an autonomous platform in the limited devices, and so on it is possible to depend on a non limited system. In the scenery we want to tackle, an ad-hoc network, it is not a valid approach.

Our initial development and research is centered in providing J2ME with the disappeared Java features that converted it into a good agent platform programming language: object serialization and dynamic classloading.

To achieve this goal and by following the J2ME modular philosophy [8], we have complemented the MIDP profile to build a new profile with the basic functionality of a mobile agent platform. We have called this profile TAgentsP (Travel Agents Profile).

Once provided the core services and with the purpose of building an autonomous mobile agent platform without the need of interacting with non limited devices we have developed a minimal HTTP server which will allow us both agent mobility and direct communication between them.

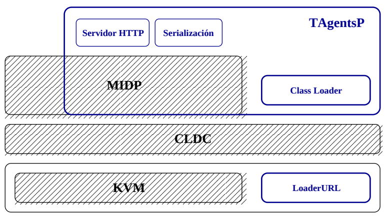

Fig. 2. TAgentsP Architecture

In Figure 2 we can see our proposed architecture. In the next sections we will describe each of the modules we have developed and tested successfully on limited devices: the serialization module and the HTTP server. Later on, we will explain the reason why we have not implemented yet the dynamic classloading mechanism.

TAgentsP implementation One of the first lacks we want to solve in our new TAgentsP profile is the lack of dynamic classloading mechanisms and object serialization that will allow us mobility between limited devices.

Agent mobility is achieved by converting it into a stream of bytes travelling through the network. This stream of bytes will later be reconstructed on the destiny device. This mechanism is called serialization. Even more, when trying to support agents migration, we need to provide dynamic classloading because if it did not exist the targeted device could not execute the received agent and so on reconstruct it.

In the next sections we provide a description of the developed serialization mechanism and the built HTTP server.

J2ME serialization mechanism *Serialization* is a mechanism with which you can convert an object into a stream of bytes representing its state, and so on it can be transported through the network or stored in a persistent way in a file system. This conversion would not be worth if there would not be a later recovery, which is the mechanism called *deserialization*

The serialization is a requirement to support agents mobility due to the fact that it allows converting an agent into something transportable conserving its state. It is also a mechanism necessary when communicating messages between agents living in different platforms.

Due to the lack of a reflexion mechanism in J2ME, by which classes can look into themselves (knowing their own methods, attributes, parameters, constructors), the developed mechanism will have some limitations, not being able to achieve a completely and automatic serialization process in which the class programmer does not have to take part on it.

It can be said that what we are looking for is the basic support that added to the programmer knowledge about the classes he is implementing we can obtain a generic and extensible mechanism to convert objects into streams of bytes, being able to transport them to later recover their state on the targeted devices.

The main problem to cope up with when development of serialization mechanisms in J2ME is the impossibility to know in real time the methods and class attributes in J2ME, as we did in the standard release of Java (remember reflexion).

The serialization we have carried out will make the programmer not to deal with the whole process, the data format and the serialized class recovery process. The only responsibility of the programmer will be the `Serializable` interface implementation as we show next.

The posed solution consists on building firstly a `Serializable` interface which contains two basic methods specifying the operations any serializable class must perform: `writeObject` to serialize and `readObject` to deserialize it. The aim of this interface is to distinguish the classes that do know how to serialize/deserialize themselves.

The next step is the creation of two new classes: `ObjInputStream` and `ObjOutputStream`. These classes will control the serialization/deserialization process by performing the necessary requests to the `writeObject` and `readObject` methods every serializable class implements.

The `writeObject` method will deal with the writing of the values of the attributes of a certain class into a stream of bytes, obtaining in this way a serializable object. Using the `readObject` method this object can be interpreted allowing the construction of a new object with the information obtained from the serialized object.

If we are going to serialize a certain class, for example `Serializable_Agent` we are going to create firstly an `ObjOutputStream` object. This object will receive the object to serialize,it will build a stream of bytes in which it will write the class name of the object to serialize[2] to finally call the `writeObject` method of the `Agent_Serializable` class that will take care of writing the necessary information (the value of its attributes, types, . . .) into a stream of bytes. In this way it could subsequently be recovered by using the `readObject` method of the same class.

The developed mechanism offers the advantage that if the programmer deals with the fact of writing the serialization/deserialization methods (writeObject y readObject) any J2ME class implementing the serializable interface can be converted into a stream of bytes that will be able to travel through the network from device to device. With a mere call to the readObject method this stream would be converted into a new object with the same state than the serialized object. This process will be completely transparent.

HTTP server in limited devices The TAgentsP developed profile would not be worthy if it did not exist a direct communications mechanism between mobile devices. In our development we have opted to build a minimal J2ME HTTP 1.1 server as a direct communication mechanism due to the fact that it offers an open solution which can be used for many other purposes.

We introduce next the server basic architecture. This architecture is composed of three main modules. The first one, the *configuration* one, will define the main configuration options (number of request to accept, port, server name, supported MIME types, . . .).

The *file system* module will manage the resources stored in the server. One of the problems we have found in the development of this server was the lack of an accessible file system to J2ME so that we could provide the resources requested by the HTTP clients. This module solves it by implementing of a file system over RMS[3] so that we can create and manage (create, delete, list . . .) directories and files on a J2ME device. By using the last module, *request service* all the HTTP requests (GET, POST, HEAD) will be performed.

This server can be used to many other purposes such as: class interchanging between different devices to support agent mobility or direct communication between devices by using HTTP as a transport mechanism of ACL messages.

[2] This operation can be performed by calling the Object.getClass().getName() method supported in J2ME

[3] J2ME persistent storage package that allows the creation and management of binary data registers.

5 Conclusions and future works

In this paper we propose the usage of mobile agent technology as a middleware to develop applications targeted to limited devices that communicate together by wireless protocols, composing the so called ad-hoc network. This is a contribution to pervasive computing due to the fact that it allows the integration of limited and embedded devices in the physical world, in distributed computing.

Taking the FIPA standard as a reference, we have carried out an analysis comprising the different defined functional elements and we have proposed a simplification so that they can be implemented in a limited device. In the case of the DF, it is not only necessary its simplification but its adaptation to the ad-hoc networks restrictions. In this way, the authors take an active part of the Working Group FIPA AdHoc that is nowadays discussing the proposals with regards to the DF, between them, the one proposed here.

Nowadays, we go on working on the design of the platform defining the interface between the ADA agent and the underlying service discovery protocol and the DF. We also go on analysing the impact the simplification of ACL could entail in the agent communication.

As we have seen, we have selected J2ME technology to implement the carried out design. We have designed and implemented a new J2ME profile, TAgentsP, which offers us the code mobility and direct communication between devices that J2ME and LEAP did not offer us. This profile has successfully been tested in real devices. Our working line is centered on the integration of our developments in the LEAP platform.

References

1. Mark Weiser. The Computer for the 21st Century. *Scientific American*, September 1991.
2. Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy Sussman, and Deborra Zukowshi. Challenges: An Application Model for Pervasive Computing. In *Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, 2000.
3. Sajal K. Das, Diane J. Cook, Amiya Bhattacharya, Edwin O. Heierman III, and Tze-Yun Lin. The Role of Prediction Algorithms in the MavHome Smart Home Architecture. *IEEE Wireless Communications*, December 2002.
4. FIPA. *FIPA Agent Management Specification*, March 2002.
5. C. Campo. Directory Facilitator and Service Discovery Agent. Technical report, FIPA Ad-hoc Technical Committee, 2002.
6. C. Campo. Service Discovery in Pervasive Multi-Agent Systems. In Tim Finin and Zakaria Maamar, editors, *AAMAS Workshop on Ubiquitous Agents on embedded, wearable, and mobile agents*, Bologna, Italy, July 2002.
7. WG-AdHoc. Agents in Ad Hoc Environments. A Whitepaper, 2002.
8. Enrique C. Ortiz. A Survey of J2ME Today. Technical report, Wireless Java, 2002.