International Journal of Computer Science & Applications
Vol. 2, No. 1, pp. 15 -24

Towards Engineering a Location-Aware Wireless Multi-agent System

Ignacio Nieto Carvajal, Mercedes Valdés, Juan A. Botía Blaya,
Pedro M. Ruiz, Antonio F. Gómez Skarmeta.
Email: {inieto, mvaldes, juanbot, pedrom, skarmeta}@um.es
Universidad de Murcia, Spain.

Abstract

Physical location of users in a multi-agent system is crucial in order to provide them with context-awareness services. A context-aware system can answer to the requests of its users in a more effective way, being able of consider the conditioners that affect them. In this paper we focus in how to provide a FIPA multiagent system with a generic location service for users. We discuss several alternatives to obtain location-awareness and propose a concrete solution to this issue.

1. Introduction

As the need for mobility of users in a information systems increases, we are challenged to adapt applications and services to be aware of the physical situation of them. Moreover, inside a context-aware system, physical location information receives even more importance, as it determines the physical services the users will have access to and helps knowing the environment they are surrounding by. We present here a general approach to the engineering of a location-aware FIPA + JADE based multiagent system (MAS). This approach will offer a high level interface in order to ignore the physical conditioners of the architecture that sublies. We do that by firstly present a deep study of several alternatives and secondly the solution we have adopted.

We are focused here in a concretehe MAS architecture in which there is a number of user agents for each physical user. The number of user agents by user depends on the number of terminals the user simutaneously uses to stay attacched to the system (i.e. a PDA, a laptop or a desktop computer). Besides user agents, there are also others like the *Locator Agent* (that provides location services to users). Users interact with the system by means of a graphical user interface, configured to adapt itself to the boundaries of the device that is being used. For example, figure 1 shows a particular application of our architecture, the SMAUG system, running on a Sharp Zaurus SL-5500 PDA under linux.

The architecture we use follows FIPA specifications, keeping their guidelines on platform architecture and communication structure and interaction. Moreover, the concrete implementation of SMAUG follows FIPA by means of the JADE platform, which provides an open source, FIPA-compliant, Java implemented agent system. The system works on certain environment. It requires a wireless infrastructure to properly offer a ubiquitous surrounding, connected to a common network containing all the servers required by the system (like the database server or the location service). Developed under Java, SMAUG works over several computer devices, including workstations, laptops, PDAs, and even less powerful devices like for example cell phones.

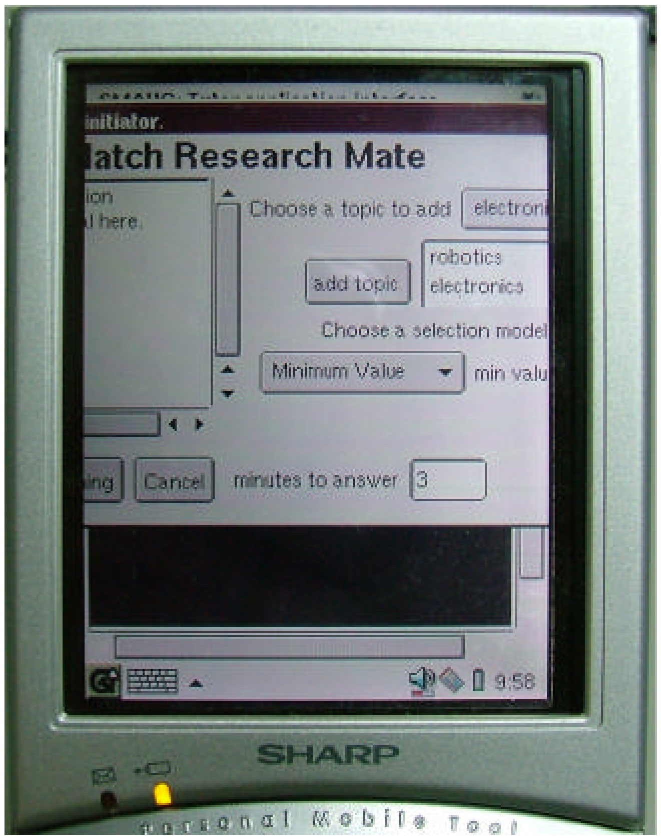

Figure 1: User interaction with SMAUG running on a PDA.

The remainder of this paper is organized as follows. In section 2., technical possibilities available to provide with location services are analysed. Section 3. exposes the concrete implementation we have finally decided to adopt. Last section 4. presents some conclusions and proposes a frame for future works.

2. Alternatives in the search of context awareness

In this section we review several alternatives we have studied as possible options for implementing the context-awareness of our system. Some have revealed better than others, and we include a short comparisson of their pros and cons. In every situation, we will have a mobile device (such as a PDA or laptop) equipped with a 802.11[7] wireless card that gets access to the network by means of one or more access points. We have several of these access points distributed in different places of the building, so we can guarantee a full area of coverage to our users.

2.1 Current Java developed libraries

One of the first questions that arises when designing our location system is: "Is there a Java developed solution that already does this? ". The short answer is "No". Let’s take a closer look at some of these developed kits and libraries.

The first one we must check out is the Java 2 Micro Edition[5,4] (J2ME). J2ME, in conjunction with The Mobile Information Device Profile (MIDP)[4] and Connected Limited Device Configuration (CLDC) [5] provides a solid Java platform for developing applications that run on devices with limited memory, processing power, and graphical capabilities. Unfortunately, this API works at a very high level of abstraction, and offers no way of accessing to low-level information, essential to get information about the user location.
There is a bunch of developed commercial solutions based on Java. This tools has serious drawbacks, such as dependency of a software supplier and costs of acquisition and maintenance. Also, this tools are not specifically thought for our system, so they contain a great level of functionality we don't need and may lack some functions we need.
There is also independent alternatives in academic circles, such as the Aura Project[8]. Nevertheless, there seems not to be a complete academic solution out there easily and freely available.
Thus, no already designed solution seems to suit our needs. We need to implement our own solution for locating our users. This solution must integrate perfectly with the SMAUG system, being versatile enough to be reused in other applications.

2.2 Signal Strength measurement

One of the methods we can use to obtain the location of the users is to measure the signal of the user's mobile device with respect to every access point. Each access point has an operational range in which the signal follows certain static rules. The nearer the device stays to the access point, the stronger the signal goes. Also, the structural architecture of the building affects the given signal levels. Solid objects such as walls, doors and furniture reduce the signal coverage area. Nevertheless, in a stable environment, the strength distribution of concrete zone keeps constant, and we can measure it. We can store all this information and do periodical checks to see if the situation has changed. Once we know the signal coverage of every access point, we can ask the device for the signal it receives from each.

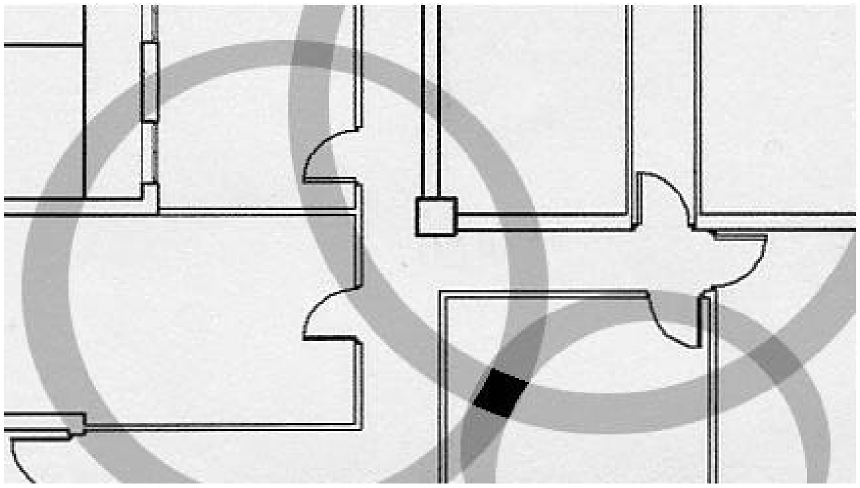
Figure 2: Measuring the signal strength.

This approach supposes we have special hardware or software added to the devices in order to obtain this information. Usually, a concrete signal level corresponds to a circular radius centered on the access point, so we need three (in some cases two) or

more access points to exactly determine the location of the user. Figure 2 shows how the signal strength of three access points can be measured to obtain the exact location of the user. Depending on the situation, two access points could give us an ambiguous location.
This method is very accurate, given enough access point and a correct initial measurement of the signal levels. Its main drawback is that it requires special hardware and software in the clients in order to obtain the signal level they receive from the access points, as not all devices and wireless cards offer this functionality. Also, changes in the signal levels of the access points (due to the inclusion of furniture or the physical change of an access point) could led us to incorrect measurement and, therefore, to an incorrect evaluation of the user location.

2.3 Location through SNMP

SNMP (Simple Network Management Protocol)[2] is an application-layer protocol that facilitates the exchange of management information between network devices. It is part of the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol suite. SNMP enables network administrators to manage network performance, find and solve network problems, and plan for network growth. Basically, SNMP provides a set of rules that allows a computer to get statistics from another computer across a network.
A SNMP system has three essential components: managed devices, agents, and network-management systems (NMSs).

- Managed devices are network nodes that contains an SNMP agent that gets and stores information from the device, and sends or makes this information available to one or more NMSs through SNMP. In our case, the managed devices are the access point of the network. Nowadays, most access points offers some kind of SNMP support. This support is not always as extended and standardized as we would like for our purposes, as we will see later.
- The agents are usually software modules devoted to management that resides in a managed device.
- The NMSs are network nodes that collects the information from the managed devices. They process, analyze and manage this information. In this vision, our NMS will be the servers of the SMAUG location system.

We have two ways of implementing a location system via SNMP, by means of periodical queries to the managed devices or through SNMP traps.

2.3.1 SNMP queries

The agent contained in a managed device collects many management information. In the case of the access points, this information can be the list of hosts associated to the access point. Thus, every time a user request the system the location of some other user, we can ask our access points for it. We can even request this information periodically and store it in a database, so we can easily recover it at will, answering more quickly to the user petition and keeping the network less congested.
The queries will be done by the servers of the system, and the information will be retrieved from the access points, so the clients will only have to request the information. This makes the operation invisible to the users, one of its main advantages. Also, this approach requires no additional software or hardware in the clients, and only SNMP

software in the servers, which can be obtained freely for most platforms, as SNMP is a well known standard.

2.3.2 SNMP traps

SNMP also includes a mechanism called "SNMP traps". These traps consists in messages a SNMP agent can send to the NMSs when something happens. The mechanism is similar to the event system. An event is fired in the observed device, and the agent sends a notification to the observers of this event. In our case, this events could be the attachment and leaving of the users. In such a system, when a user connects to the access point, the access point will send a trap to our server informing of the situation, so the server can store that the user is connected in an internal representation or an external database. Also, when a user gets disconnected from an access point, this will send a disconnection trap to the same server, so it can store that the user is not associated to this access point anymore.
The advantages of this method is that it don't require the periodical scanning of the network. Messages are only delivered when a user connects/disconnects. Thus, less traffic is generated. Also, we have the certainty that the user location we get is up to date.
Both approaches have, however, the same drawback. Although SNMP is an standardized protocol, not every access point supports it in the same way. The access point information is usually stored in enterprise-private variables, and, depending upon the access point, it can't be retrieved. Also, the trap system is highly vendor-specific, with specific traps that can be sent or not depending upon the concrete product. This makes very difficult the task of working with SNMP in an open system like SMAUG, where several devices must interact independently of their architecture or specific model.

2.4 Location trough authentication

RADIUS[1] is a widely used protocol in network environments. It is commonly used for embedded network devices such as routers, modem servers, switches, etc. It is currently the "de-facto" standard for remote authentication. It is very prevalent in both new and legacy systems. RADIUS provides a robust and centralized user administration, and its present in a way or another in most network devices, such as access points and bridges.
Concretely, most access points provides the option of authenticating the devices accessing it with the RADIUS system, by means of one or more authentication servers. If we use this authentication service, we can register in the RADIUS server the devices that access the system and from which access points. This will be very valuable not only for location purposes, but for security ones.
Unfortunately, this requires a great level of configuration at an architectural level, and a system where the users are known and rarely change. In SMAUG, we plan to have many different users that will access the system by means of an unknown number of changing devices, preventing us of using this method as a viable alternative for determining user location.

2.5 Access-point low-level strategy

The last alternative we discuss (and the one we have chosen to implement) is obtaining the information we need from the own device. If we can access to low level information of the device, such as the MAC address of the access point we are connected to, or the name of that access point, we can use this information to access a database containing all this data and thus obtain the exact location we are in, that will lead us to the context information.

Most devices and operating systems provides a way of getting this information. For this solution to work properly, we need to access the wireless device information in a clean and transparent way. We need a common interface that doesn't bother about the way this data are obtained. In our case, this is done by supplying native libraries that are read trough the Java Native Interface (JNI)[6]. The native methods, written in low level languages (such as C) pass the low level information to the JNI, and we access this information from our Java system in a transparent way.

The main drawback of this method is the extra work required. We need to write a native implementation for every system we will work on (mainly for linux workstations, windows PCs and PDAs based on Strong-ARM processors). Once we have this, though, we can provide a high level interface in Java that access to this information.

3. Access-Point MAC Address based solution

We have decided to implement a solution based on consulting low level information from the device in order to get the location information. In this section we explain the structure of this alternative.

3.1 Retrieving the low level information

In the SMAUG system, we must distinguish between static and mobile devices. Static devices know their own location, as this is configured at the beginning and never changes. Mobile devices don't know this information, but can know some location metadata, such as the access point MAC address. Our Location System possess a special group of agents called "LocatorAgents". The agents from each device send this information to a LocatorAgent by means of a FIPA interaction. The LocatorAgent is responsible for maintaining this information and the agents are responsible for keeping their locations up to date by means of communication interactions with the LocatorAgent. If a user's agent want to know the location of another user, it queries the LocatorAgent directly.

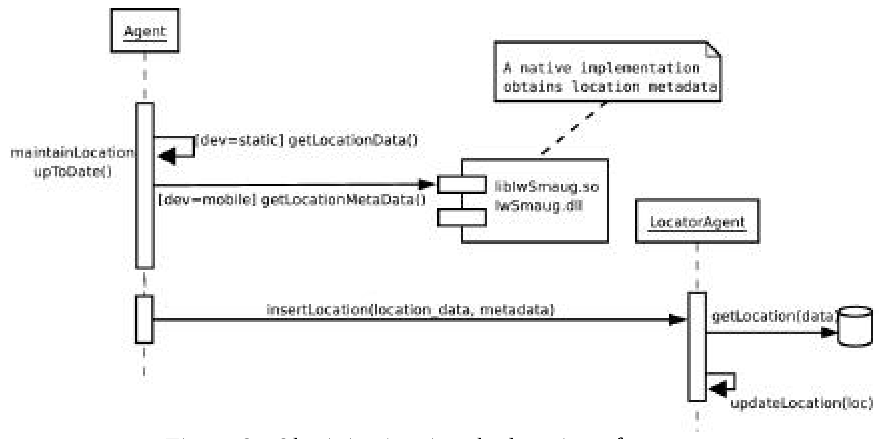
Agent
A native implementation obtains location metadata
maintainLocation upToDate()
[dev=static] getLocationData()
[dev=mobile] getLocationMetaData()
libIwSmaug.so
IwSmaug.dll
LocatorAgent
insertLocation(location_data, metadata)
getLocation(data)
updateLocation(loc)

Figure 3: Obtaining/storing the location of an agent.

This interaction requires a number of steps, as shown in figure 3. First of all, the agent determines if it's contained in a mobile or static device. If is contained on a static device, the information is a concrete place, configured previously. In other case, the device must call a native method written in a native language and stored in the *IwSmaug* library. All this information is sent to the LocatorAgent. The LocatorAgent then contacts with the database where this information is translated into a concrete location. The LocatorAgent then inserts this location into his internal structure, which contains an association of every agent with a concrete location, where locations are complex objects correlated with each other. The agent that sends the location executes the pseudo-code algorithm shown below:

```
InsertLocation:
deviceType = consultDeviceType();
if (deviceType is a mobile device)
data = metadata (call native method
from libIwSmaug);
else if (deviceType is a static device)
data = direct_location (get
static location);
end if
send data to the LocatorAgent;
```

The Locator agent receives the petition and executes the following pseudo-code:

```
InsertNewAgent:
data = receiveData();
if (data is direct data location)
location = obtainDirectLocation(
database, data);
else if (data is metadata about location)
location = obtainMobileLocation(
database, data);
locationList.insertLocation(location);
```

3.2 The location process

Once we can access, manage and communicate the location information, we can plan a strategy for handling all this information. Our Location System is consists of the stakeholders (the users' agents) and the LocatorAgents. We want to save as much bandwidth as possible, so a *polling* method is not a factible solution. Instead, our system uses a *Last known position* policy, exposed in the figure 4.

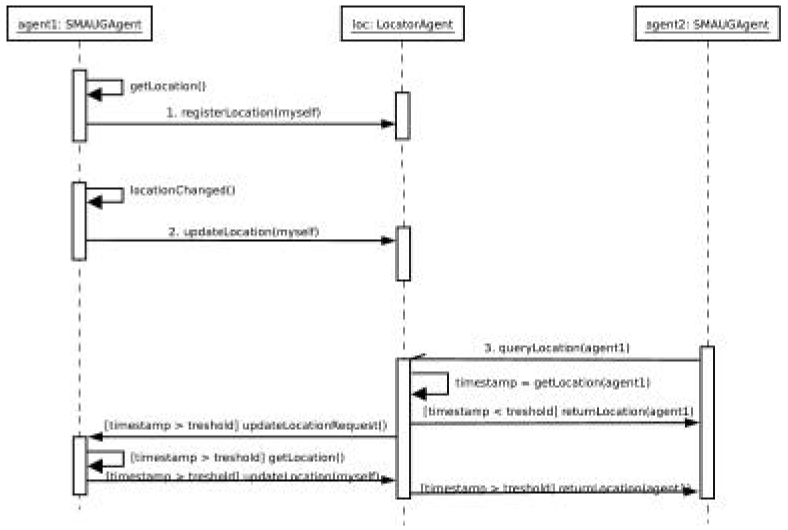
agent1: SMAUGAgent
loc: LocatorAgent
agent2: SMAUGAgent
getLocation()
1. registerLocation(myself)
locationChanged()
2. updateLocation(myself)
3. queryLocation(agent1)
timestamp = getLocation(agent1)
[timestamp < treshold] returnLocation(agent1)
[timestamp > treshold] updateLocationRequest()
[timestamp > treshold] getLocation()
[timestamp > treshold] updateLocation(myself)
[timestamp > treshold] returnLocation(agent1)

Figure 4: Structure of the location process.

In this approach, every agent send on starting its actual position data to the LocatorAgent. The LocatorAgent calculates and stores this information along with a timestamp. If the agent is going to shut down or change its position, it informs the LocatorAgent to delete its entry or update it, respectively. When an agent sends a location query to the LocatorAgent, this searches its location table to determine if the agent is located. If the agent doesn't belong to the table, the LocatorAgent responses that the user agent is not localizable. If there is an entry for the agent, the timestamp is examined. If the timestamp is above a certain *doubt threshold*, we are not sure if the user location is exact (it could have had problems or its machine could have broken down), so we query the agent for its location. In any other case, we are reasonably convinced that the user is on this location. This location information is of course sent back to the requester.

3.3 Sharpening the location system

Once we have the basic capacity of location through a single access point, we can go deeper and take further the accuracy of the SMAUG location system. Developing the low level information retrieving library SMAUG provides, we can establish methods for retrieving information from random access points, such as the signal-to-noise ratio. If the wireless device we are working on and the access points we possess support it, we can query the nearer access points of a device to obtain this information. With this information, we can apply a triangularisation like the one exposed in section 2.2, but without the need for installing additional hardware to the devices or using external measurement applications. Unfortunately, this information could possibly not work on every single device, but we can use it as an auxiliar tool that allows us to sharpen the location measure in the devices that support it.

3.4 Towards a middleware with location services in heterogeneous networks

Currently, SMAUG system is aimed at working on a wireless network environment, like the one we possess in our facilities. Nevertheless, nowadays there is a wide variety of network technologies available to users, and the actual tendency is to develop a roaming capacity between these networks (this is contemplated in the included in what has been known as "Fourth generation mobile networks". All of this networks (such as the Wi-Fi technology we are using, and also others like UMTS/CDMA2000)[3] possess its own way to locate their users. Also, many other generation technologies can be used along with these.

One of the aims of the SMAUG project is the establishment of a heterogeneus system that doesn't have to rely on any particular architecture or physical device. Thus, the proposed solution needs to be shielded from the peculiarities networking equipment underneath. This means that we need to include a global solution for all this networks into our location-awareness service. Thus, we need to build a concrete API that will serve as a high level interface for all the network location service. We will build a concrete adapter to this API for each network technology used, so we'll end up with a general interface that can be used to access the location service regardless the particular technology in use.

Several networking technologies can be integrated into the location service:

- *Universal Mobile Telecommunications System* (UMTS): UMTS is one of the Third Generation (3G) mobile systems being developed within the ITU's IMT-2000 framework. It is capable of supporting high-speed mobile data services as well as voice. The basic architecture of a UMTS network is similar to that of a GSM network, consisting of mobile handsets, a cellular radio network and a backbone with switches.
- *Global Positioning System* (GPS): GPS is a worldwide radio-navigation system formed from a set of 24 satellites and their ground stations. GPS uses these set of satellites as reference points to calculate positions accurate to a matter of meters. GPS possess some advantages, like being available in the entire globe and its accuracy. Nevertheless, it has its drawbacks. GPS doesn't work inside buildings or on very populated cities, and takes a little time to get the location, due to the necessity of aligning the sufficient number of stations.
- *Wireless Fidelity* (Wi-Fi): Wi-Fi is the short for wireless fidelity and is meant to be used generically when referring of any type of 802.11 network, whether 802.11b, 802.11a, dual-band, etc. The term is promulgated by the Wi-Fi Alliance. Our current work bases around this technology, and we are exploring in expanding it to a global network positioning and location service.

When all this network become integrated in a location service solution, we will reach a total location system where context awareness will be an integral part of the user interaction with computer devices.

4. Conclusion and future work

In this paper we have analyzed several methods for obtaining location services for a multi-agent system. We have proposed a location management approach and we have implemented it in SMAUG, which is a multiagent system developed by ourselves. The proposed approach is based on a low level detection of the access points to which wireless devices are attached, but it does not require any changes to existing networking equipments. Although the proposed approach is integrated with SMAUG, it is modular enough to be adaptable to any other middleware framework.

As future work, we aim at integrating the proposed appraoch with those offered by other underlying networking technologies, as well as to be able to integrate different sources of location information. As long term future work, we plan to extend this location service into a much richer context awareness service providing a much richer set of information to future applications and services.

Acknowledgements

Part of this work has been funded by the Spanish Science and Technology Ministry (MCYT) by means of the 'Ramon y Cajal' work programes as well as the *TIC-2002-04021-C02-01* project.

References

[1] William Allen Simpson Carl Rigney, Allan C. Rubens and Steve Willens. Remote authentication dial in user service (radius), 2000.
[2] Schoffstall Case, Fedor and Davin. A simple network management protocol (snmp), 1990.
[3] 3GPP2 Group. Cdma2000 wireless ip network standard: Introduction, 2003.
[4] JSR 118 Expert Group. Mobile information device profile for javatm 2 micro edition, 2002.
[5] JSR 139 Expert Group. Connected limited device configuration specification, 2003.
[6] Sun Microsystems. Java native interface specification, 1997.
[7] IEEE Computer Society. Ansi/ieee std 802.11, 1999 edition, 1999.
[8] João Pedro Sousa and David Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. *Software Architecture: System Design, Development, and Maintenance.*, 2002.
[9] M. Wooldridge, N. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 15, 2000.