Mobile Agents: A Construction Kit for Mobile Device Applications

Michael Friedrich, Kirsten Terfloth, Gerd Nusser, and Wolfgang Küchlin
WSI for Computer Science
University of Tuebingen
Email: {friedrich,terfloth,nusser,kuechlin}@informatik.uni-tuebingen.de

***Abstract*— Mobile Agents are a well-known programming paradigm nowadays. There is a multitude of research concerning Mobile Agent Systems with emphasize on agent coordination, agent languages and agent migration technology. On the one hand, it is often argued, that Mobile Agents are well-suited for the use in the Internet and especially with Mobile Devices and roaming users. On the other hand, there are only few publications describing the actual implementation of a Mobile Agent System for Mobile Devices.**

In this paper we present our implementation of the Mobile Agent System Okeanos for the use in mobile environments with emphasize in agent routing and forwarding.

1. Introduction

Mobile Devices such as PDAs, mobile phones and notebooks get more and more common these days. More complex applications can be deployed on these Mobile Devices because of their increasing computing power and memory resources. During mobile usage, especially outside the office and not at home, cheap broadband communication is usually not available. Therefore, the standard usage is to connect to the Internet just occasionally and keep connection time low.

In this application context, remote operations like searching the web or retrieving information, should be optimized for these connection constraints. Mobile Agents software objects which can migrate in a network autonomously and support this kind of scenario in several ways:

- Code mobility enables the execution of tasks without a connection to the originating system. An example is sending an agent to search text documents in different locations of a network, while the Mobile Device being disconnected.
- Agents communicate and work locally. A Mobile Agent takes its application logic along during migration and performs its task autonomously. This is an advantage over synchronous C/S-architectures which cease working without a network connection, since Mobile Agents do not depend on this connection to fulfill their tasks.
- Mobile Agents contain fundamental fault tolerance features, because they are built for unreliable environments. The services offered are not guaranteed by the serving agent, if not negotiated otherwise. A Mobile Agent using other agents has to be aware that offered services may disappear without notice.
- Logical mobility of Mobile Agents contributes to their ability to survive in an ever changing environment. There is conceptually no difference between a change due to an agent migration (logical mobility), or the migration of the hosting Mobile Device (physical mobility).

In this paper we present our experience with extending a Mobile Agent system to deploy it on PDAs. The work is structured as follows. First, we give an overview over the implemented system and the basic elements of an agent system. The problems arising with Mobile Agent Systems on PDAs are discussed in Chapter 3. A detailed illustration on the problems of agent routing together with the solution we chose follows in Chapter 4. A sample application is described in Chaper 5 based on our implementation. Chapter 6 compares the related work and the paper concludes with a summary and outlook.

2. Overview

Every Mobile Agent System consists of Mobile Agents and agent execution environments (agent places). These places supply the agent with several vital services:

- *Life* The agent place offers the agent a thread of control, memory and a registration for the agent to be known in the agent world.
- *Communication* allows agents to communicate with each other. Some systems allow remote connections, others allow only local connection.
- *Migration* is vital for the agents to be mobile. The agent requests the agent place to transfer itself to an other place. Serialization and transfer is done by the environment.

Most Mobile Agent System use Java as programming language. The connections are usually based on TCP/IP with sockets or higher level protocols like Java RMI. Furthermore, an interconnection network like the Internet is assumed, where every agent place can connect to another.

The application is divided into multiple agents. Some of them are mobile, others are stationary service agents. As depicted in Figure 1, agents can reside on mobile devices, too, due to their small memory footprint.

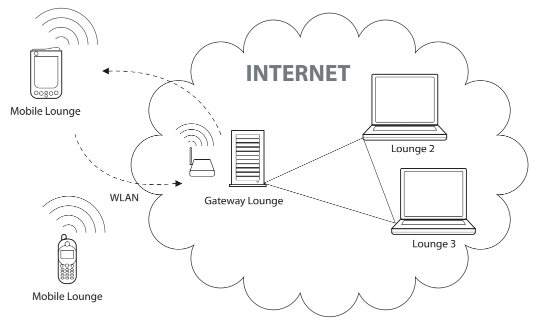

Fig. 1. Devices for hosting mobile agents. Some systems are fully connected, others use gateways to connect to the Internet.

3. Motivation

A. Applications for Mobile Devices

The ”classic” way of building distributed applications is the n-tier Client-Server application[1]. Due to performance restrictions a 2-tier architecture with a fat client and an server-side back end is often not possible. Therefore, a multi-tier architecture with a thin client (e.g. a WWW browser) connects to an application server, which in turn instruments the back-end servers. This application architecture satisfies the performance restrictions on Mobile Devices, but needs a synchronous network connection. When the connection is cut off, the application does not work any longer as there are only viewer components and no application logic on the client device. Therefore, this kind of architecture is not suitable for Mobile Devices.

We propose the use of smart components, which can be transferred to Mobile Devices as needed and also removed during run-time. In our approach, Mobile Agents are used to build flexible applications which overcome the restrictions of Mobile Devices.

B. Challenges

As discussed above, the combination of Mobile Agent Systems and Mobile Devices is promising. However, Mobile Devices have inherent restrictions, which make the implementation of Mobile Agent Systems challenging.

Most Mobile Devices lack of a sufficient power supply for long continued work. The operation time of a Mobile Device depends on its usage profile. PDAs and smartphones are usually used for quick information retrieval tasks, like looking up addresses, appointments or emails. The ”smart”-functions are only used for short durations and the device switches to stand-by afterwards. The power restriction is even higher if the device is connected to a network, because this consumes an additional amount of battery supply. Therefore, continuous network connections cannot be maintained for PDAs and smartphones.

The cpu performance and amount of RAM memory also contributes to electrical power usage. Manufacturers choose a tradeoff between battery run time and computing performance which often leads to less powerful devices. Since memory for Mobile Devices is still rather expensive, most of these devices are limited in

[1] n-tier includes also the 2-tier architecture

RAM, which has to be taken into account when developing applications for Mobile Devices.

Wireless connections, their associated bandwidth and the quality of service which can be provided have a high impact on the way a consumer works with his Mobile Device. High costs for only low bandwidth enforce the connection time to be kept as short as possible. The quality of service for wireless connections depends on the roamed environment. Whereas it is good and can be influenced in indoor office environments, the outdoor performance of wireless networks is affected by geographical conditions and network provider coverage. Furthermore, reconnecting to a network can lead to a change in the network address of a Mobile Device. Network fault tolerance is an important issue for the application.

Evaluating the software side of application development for Mobile Devices reveals not only company specific interface definitions, but also a wide variety of Java runtime environments according to the characteristics of the device. This leads to different APIs so that application implementations for a PC are not automatically portable.

The challenges mentioned above affect the architecture of applications for Mobile Devices, especially distributed applications which require network communication. In the following, we examine this kind of applications closer.

4. Implementation

A. Java Flavor

We chose to use the Jeode Java Virtual Machine, which supports the Personal Java Standard. This corresponds to the former Java 1.1.8 Version, nowadays outdated and replaced by the J2ME CDC Profile[1].

Other industrial standards for Java on Mobile Devices exist like SuperWaba[2]. SuperWaba does not conform to the J2ME standard and and therefore lacks interoperability,as applies to any specialized implementations of the Virtual Machine. They are built to support special features like strong migration. Using these implementations contradicts Java's ”write once – run anywhere” philosophy.

B. Choice of Device

There is a wide variety of Mobile Devices, ranging from embedded devices with minimal cpu, memory, and I/O up to full featured notebook computers with workstation performance. We tried to use the lowest scale devices for our Mobile Agent System Okeanos[3], which makes use of Java RMI and object serialization. The device has to support the Java CDC, and the optional package for supporting Java RMI. Furthermore, we needed a TCP/IP connection and decided to use WLAN (IEEE 802.11b) capabilites. The chosen device is a HP IPAQ 5400 with a 250 MHz XScale processor, 64 MB RAM and Windows PocketPC 2002 as operating system. This rather high-end PDA is a good base for development. Deployment of the Okeanos system may be able on less equipped devices, but with the advantages in computing power, PDAs with this performance will soon be available in the consumer class.

The PDA is connected to the intranet by WLAN and has a static IP-address. The effects when an IP-address is changed due to the usage of DHCP are shown later in this chapter. As mentioned above the Insignia Java Virtual Machine Jeode is used for executing the Mobile Agent System.

C. Agent Communication

Agents communicate only locally in Okeanos. There is no support for remote communication, since this will increase the complexity of the system without generally gaining major improvements. Nevertheless, it can be integrated if needed. Mobile Agents have to travel to remote execution environments to talk to others located there. This behavior fits well into an asynchronous application architecture, which is required with Mobile Devices.

D. Migration

The Okeanos Mobile Agent System uses weak migration for agent transportation. Java has no support for freezing and transferring the execution stack of a thread of control. Therefore, Java-based agent systems can only use weak migration without having to install a special JVM.

When a Mobile Agent wants to migrate to a certain destination, it is serialized and transferred via Java RMI. In case no connection to the destination is currently available, the Okeanos system stores a Mobile Agent until this connection is established again. For some application, waiting too long can be critical. For instance, if the user expects a feedback, even a few seconds can confuse him. The migration service offers a quality of service feature, which allows the agent to decide how long to wait for the transport. If the time is over, the agent wakes up again at the starting point and can perform error handling. Because only the agent knows how to handle its situation in an application specific way. This feature can not be automated and implemented within the agent place.

Mobile Agent can experience migration delays at any lounge during their journey to the destination. The agent does not know the number of intermediate stops, so the latency of each drop is accumulated and the agent is activated after its predefined time slot is over. For a detailed illustration of agent routing see Chapter 4-G.

E. Service Discovery

Mobile Agents can only communicate locally and therefore have to travel to meet at the same agent place, called *lounge* in Okeanos. The agents have to find each other with the help of a service directory. A global agent directory to find a special agent would not scale as it would most likely be flooded with updates. Currently, Okeanos employs a distributed directory with two kinds of entries. The global entries map services to lounges and the local entries map services to agents. When a Mobile Agent looks for a certain service, it first requests the lounge where the service is offered and travels there. It then requests the id of an agent offering the service from the service directory and initiates local communication.

Any changes concerning the currently connected lounges or offered services have to be published within the system as soon as a change of a global entry occurs. This is accomplished by a special agent, the UpdaterAgent, by travelling to every connected lounge and communicating these changes.

F. Agent System Discovery

Within a mobile environment, a newly connected lounge has to find other lounges of a Mobile Agent System. We distinguish between Mobile-Lounges, Gateway-Lounges, and normal lounges. The first one runs on every participating mobile device, and is only temporarily connected. A Gateway-Lounge manages the connection to the Mobile-Lounges and to the other, fully connected lounges. The connection topology is depicted in Figure 1.

A Mobile-Lounge uses a TCP multicast to which a Gateway-Lounge within the network segment responds and sends its identifier. If there is more than one Gateway-Lounge, the Mobile-Lounge will choose one of them to connect to. After the handshake protocol, the Gateway-Lounge checks whether this Mobile-Lounge has already been connected before. In this case it checks whether its service directory has changed since and has to be updated on the Mobile Device. Otherwise, if it is a new connection, the service directory is replicated to the Mobile Device.

The Gateway-Lounge reads the service directory of the Mobile-Lounge and publishes it to the other participating lounges. The goal is to have a nearly up to date service directory on each lounge. Even if a Mobile-Lounge splits from the network, it has at least the last known state of the service directory.

While connected, the Mobile-Lounge checks regularly if the connection is still alive. If the connection is lost, the lookup process with multicasts will start over again.

G. Routing

Code mobility in a Mobile Agent Systems is based on agents that migrate over the network to accomplish a specific task, and return with the results to their origin. In a network that includes Mobile Devices, the routing of the agents has to cope with broken connections, delays and errors. As mentioned above, three types of agent lounges exist. Any Mobile-Lounge on a Mobile Device first registers at a Gateway-Lounge when setting up a connection to a fixed network. From that point, the agents coming from or migrating to a registered Mobile-Lounge are taken care of by the Gateway-Lounge, a kind of session is established. An UpdaterAgent publishes this information in the distributed service directory, and agent traffic for the Mobile-Lounge is routed automatically over the Gateway-Lounge.

In case the network connection of the Mobile Device breaks due to an error, power-saving or because the user decides to interrupt the connection for reducing network costs, the remaining network will not be affected by this change, even if agents still try to migrate to the disconnected Mobile-Lounge. The Gateway-Lounge provides a store and forward mechanism to deal with these agents, and acts as a representative of its registered lounges. As long as the Mobile Device remains disconnected, the agents willing to migrate there are put to sleep and managed in a datastructure. When it reconnects to the network, all agents with this specific destination will be put back to work.

Several problems arise in this solution: This approach may cause unlimited waiting times for agents if the Mobile-Lounge does not reconnect at all. To avoid this situation, a user can specify a maximum sleeping time for every agent, and be able to estimate the quality of service needed. After this period of time, the agent is awakened and can decide on its next steps. Assuming a completely autonomous agent, there is no way to avoid it getting stuck on a Mobile Device due to disconnection. A user may have to start another agent for solving his task if the maximum waiting time is exceeded.

Wireless LAN in combination with DHCP can cause another difficulty in routing the agents to their proper destination. Imagine a user sending an agent into the network and interrupting the connection afterwards. When he reconnects again to pick up the results, the DHCP-Server might have given his Mobile Device a different IP-address. Since the Mobile-Lounges are identified by their IP, the Gateway-Lounge will not recognize that this is a lounge already registered in the network. We tried to solve this problem by storing the prior identifier of the Mobile-Lounge and redirect agents heading to this old IP-address to the new one. In practise this solution failed due to problems we faced with the migration protocol RMI. Although classes needed to transfer agents were exported again, the protocol still seemed to cache the old IP-address, and therefore tried to send any agent to this old and outdated address, which obviously failed.

Besides this problem the routing worked very well. To achieve a scalable network topology, the number of Gateway-Lounges in the network can be increased when facing many Mobile Device connecting to it.

5. Sample Application

Information retrieval in distributed systems and mobile environments can be a time consuming challenge. To illustrate the advantages of using a Mobile Agent System as middleware we implemented a distributed document searching and analyzing application. Think of a user searching for information on a specific keyword, maybe in the intranet of an office, using a Mobile Device to connect to it. This user does not want to download all promising files on his device and scan them personally, but rather get the information where to find files containing the specified keywords and decide on downloading afterwards.

In our application, this task is implemented using a couple of small interacting agents. It is assumed that an agent lounge is available at every participating node of the network.

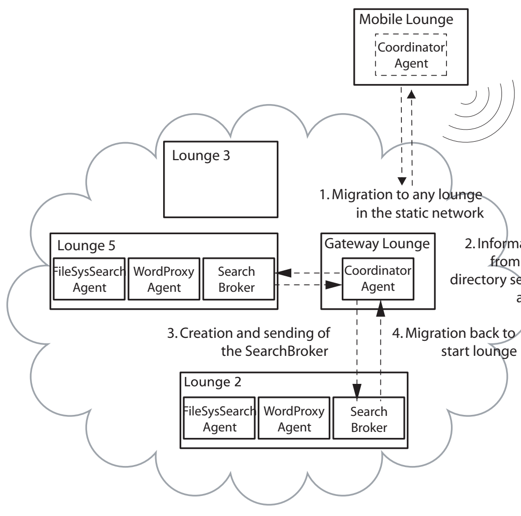

Fig. 2. The application workflow. After connecting to the network, the Coordinator agent migrates to the gateway lounge (1) and reads the local directory (2). It creates and disseminates the SearchBroker agents to the foreign lounges(3). The SearchBrokers migrate back to the Coordinator after receiving the results (4), and the Coordinator presents the results to the user at the Mobile Device.

There are four different kinds of agents involved in this scenario depicted in Figure 2. Two of them are stationary service agents, the File System Search agent (FSS agent) and the WordProxy agent to search for and analyze documents locally. The FSS agent encapsulates the file system and performs wildcard searches in directories and drives. The WordProxy agent instruments a Microsoft Word instance via its COM interface to search within a Word document. The other two agents are mobile, the Coordinator agent and the SearchBroker agent for the coordination of the different steps in the workflow. The user interface, where the search term can be specified, is provided by the Coordinator agent. As soon as the application is started, it will migrate to an other lounge in the network that is not on a mobile device. This reduces the risk of information loss over an intermittent connection. The Coordinator agent reads the service directory at its destination to find all lounges where full or partial searching of the underlying filesystem is allowed. For every lounge which offers these services, a SearchBroker agent is created by the Coordinator agent, and instructed with the search objectives before it migrates to its destination lounge. This way, the search can be paralleled and only local communication between agents is necessary. The SearchBroker sends the search parameters to the FSS agent and receives URLs of documents found in return. It hands any absolute path of a Word document to the WordProxy for further analysis. This workflow is outlined in Figure 3. The results are gathered and the SearchBroker migrates back to the Coordinator when all documents are dealt with. Once all SearchBroker agents returned, the Coordinator migrates to its originating lounge and communicates the results to the user.

This application makes use of many benefits of the mobile agent paradigm. It works inherently parallel by distributing the task among different agents and offers a flexible, fault tolerant environment. The user does not have to be connected while the search is executed, and he can pick up the results whenever needed due to the store and forward mechanisms implemented. Since there is a possibility to specify the quality of service needed, the maximum time of waiting for the result delivery may be determined.

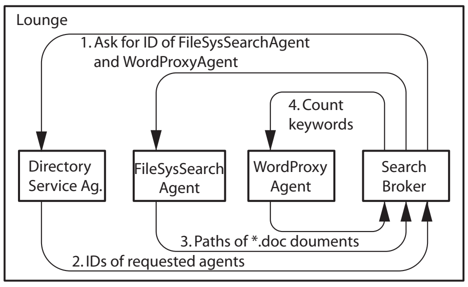
Lounge
1. Ask for ID of FileSysSearchAgent and WordProxyAgent
4. Count keywords
Directory Service Ag.
FileSysSearch Agent
WordProxy Agent
Search Broker
3. Paths of *.doc douments
2. IDs of requested agents

Fig. 3. Local interaction to query the Word documents at a lounge.

6. Related Work

Wang, Sørensen, and Indal [4] describe a Mobile Agent System for very limited Mobile Devices. If the Mobile Agents migrate to such a device, they are split up into code and data, and just the data is transferred. The only thing a device has to accomplish, is to run a Java Virtual Machine which supports network connections and display the data in a fixed GUI.

In case the user moves physically with his device to an other network, the code is not available any more. The data looses its semantics. Vasiu and Mahmoud with the MobiAgent[5] system use the same approach.

Our current implementation does not support such limited devices. Mobile Agents should be treated as entities which fulfill the object-oriented paradigm of encapsulating data. Furthermore, the split of an agent into code and data turns the Mobile Agent into a C/S-System.[6]

Other Mobile Agent Systems like JADE require J2SE version 1.4.[7] This conflicts with interoperability when agents targeted to this platform use packages which are not available with J2ME.

7. Summary

In this work we presented our approach of a middleware based on Mobile Agents for Mobile Devices. We examined the hardware and connection constraints Mobile Devices place on the development of distributed applications. Among these are expensive, intermittent, low bandwidth connections and limited devices in respect to memory, performance, and power supply. Applications for Mobile Devices have to deal with these challenges. To avoid application specific development with every new piece of software the management of the mobility aspect should be transferred to a middleware. The logical mobility designed into a Mobile Agent System is capable to support the physical mobility of Mobile Devices. Our sample application proved the required middleware qualities of Mobile Agents.

References

[1] *Java 2 Micro Edition*, Sun, http://java.sun.com/j2me/.
[2] "Super waba," http://www.superwaba.org.
[3] R. Schimkat, W. Blochinger, C. Sinz, M. Friedrich, and W. Küchlin, "A service-based agent framework for distributed symbolic computation," in *Proceedings of the 8th International Conference on High Performance Computing and Networking Europe (HPCN'00)*, ser. LNCS, vol. 1823. Amsterdam, Netherlands: Springer-Verlag, Berlin, May 2000, pp. 644–656.
[4] A. Wang, C.-F. Sørensen, and E. Indal, "A mobile agent architecture for heterogeneous devices," in *In Proc. of the Third IASTED International Conference on Wireless and Optical Communications (WOC 2003)*, Banff, Canada, 2003.
[5] L. Vasiu and Q. H. Mahmoud, "Mobile agents in mobile devices," *IEEE Computer*, vol. 37, pp. 104–105, 2004.
[6] A. Fugetta, G. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 342–361, May 1998.
[7] "Jade," TILAB, http://sharon.cselt.it/projects/jade/description-index.htm.