

# A Mobile Agent Architecture for Heterogeneous Devices

Alf Inge Wang\*

Carl-Fredrik Sørensen†

Eva Indal

Dept. of Computer and Information Science,  
Norwegian University of Science and Technology, N-7491 Trondheim, Norway.

## Abstract

This paper describes an approach for running mobile agents on various devices from mobile phones and Personal Digital Assistants (PDAs) to powerful PCs. Most mobile devices suffer from limited computation resources (memory and processors), limited network connection and bandwidth, and limited battery life. Mobile agents are a promising technology for minimizing the problems described above. However, most mobile agent systems today are very resource demanding both for the client and the server. In this paper we propose a simple mobile agent architecture that makes it possible to access a mobile agent system on various devices. This architecture proposes that clients will state their capabilities. Based on these capabilities, the client will either run the full mobile agent on the device or only run a light-weight version of the agent on the device. In our new approach, the mobile agents are basically the same on all clients, but for small mobile devices the code of the mobile agent is removed. This means that for mobile devices with minimal resources, only the data of the agent can be changed. The code of this agent is stored at the server. When the agent returns to the server, the two parts are joined and the agent is ready to be executed. The joined mobile agent can migrate to other agent servers and clients.

**Keywords:** *Mobile agents, Mobile Computing, Mobile Devices*

## 1 Introduction

Many people use mobile electronic devices such as mobile phones and PDAs both professionally and for pleasure. These devices suffer from limited battery time, limited memory and data storage, limited processing power, small screens, cumbersome input, and limited network bandwidth and network connections. Mobile agents have been suggested as a technology to deal with challenges such as the increased need for personalization, high latency, demand for large transfers and disconnected operation [9]. All these challenges are valid for mobile devices,

and therefore mobile agents can be ideal for mobile devices [12]. Sending a mobile agent to another computer to do the data collection and computation can save battery power [16]. This also means that heavy computations that will take long time on a mobile device can be executed at a more powerful computer with more memory, faster CPU and without any power limitation. In addition, problems with slow and error-prone network connections can be reduced by dispatching the agent to another machine, and get the agent with the result back when finished. Most mobile devices will consume a lot of battery power when transferring data over a network a long period of time. Mobile agents can reduce these problems.

Although mobile devices have become more powerful in the recent years, some of them still do not have the capacity to execute a mobile agent platform. For others more powerful mobile devices, this is not a problem. In this paper, we will propose an architecture that makes it possible to run the mobile agents on various kinds of mobile devices. Most agent platforms are implemented in Java and require a lot of memory and a powerful CPU to run efficient. This means that it can be very hard to run a full mobile agent system on e.g. a Java-enabled mobile phone. Our solution to this problem is the following:

- 1) The agent clients describe their execution context in terms on CPU speed, execution memory, storage space, and what Java virtual machines that can be run (edition and version).
- 2a) For clients with sufficient capabilities, the full-fledged agent client can be executed.
- 2b) For clients with limited memory, CPU or Java virtual machines, a thin agent client is executed that only can manipulate the data of the agent. The code of the mobile agent is stored at the agent server.

Our architecture is elegant and simple because the same mobile agents can be accessed by various devices, although on devices with minimal capabilities the agents can be accessed in a limited fashion. It is also easy to extent the architecture to give support for several types of agent clients tailored for a specific device. Tailoring for specific devices can be necessary to improve the usability of the agent clients based on the screen size, graphical capacities and

---

\*Email: alfw@idi.ntnu.no, Phone: +47 73594485, Fax: +47 73594466

†Email: carlfrs@idi.ntnu.no

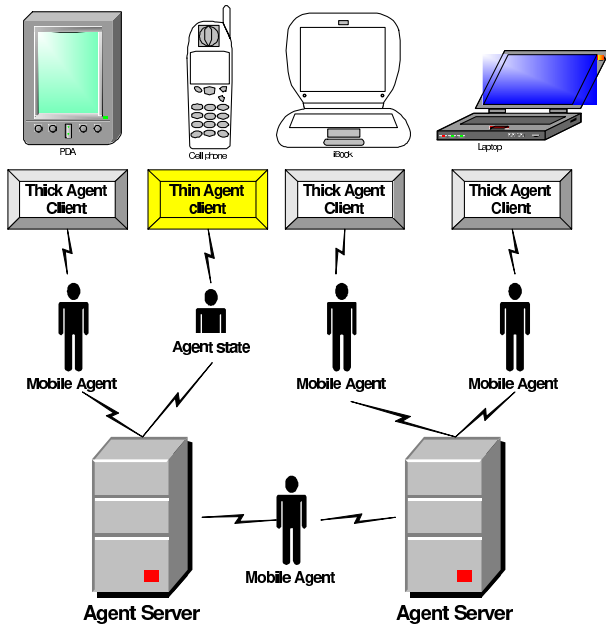


Figure 1. A heterogeneous mobile agent system

data entry capabilities of the devices. In addition, an agent client must be tailored depending on the network connectivity of the device. Further, our architecture makes it possible to update the client directly based on change of the client hardware and/or software and the agent software itself. Our approach for allowing different kinds of clients to access mobile agents is illustrated in figure 1.

To test our architecture we implemented a simple mobile agent system for electronic trading service of used items (e.g. electronic devices, games, bikes etc.). A typical scenario can be that the user initiates a mobile selling agent on a portable PC, before the agent is dispatched to various servers to look for possible buyers. The user may then leave his office and brings with him a powerful PDA with a Wireless LAN connection. The PDA has only wireless LAN network within the office building. The mobile sales agent will then return to the user on the PDA and ask for further instructions based on information about possible buyers gathered on various agent servers. The user will then instruct the mobile sales agent what to do, and dispatch it to the mobile agent servers. Because of an unforeseen event, the user has to leave the office building. This means that the mobile agent cannot be accessed using the PDA (no Wireless LAN network coverage). However, the user has a Java-enabled mobile phone that is capable of running the thin mobile agent client. This makes it possible for the agent system to bring back the results of the trade, by returning the state of the mobile sales agent.

This paper describes our high-level architecture for running mobile agents on various devices. We will also describe in more details how we implemented the thin client for limited devices. In addition, we will discuss possible solutions to

overcome the memory and CPU barrier, and technologies that can be used.

## 2 Related Work

The interest of implementing mobile agent systems for mobile devices has increased during the recent years. This section will give a brief overview of some of these projects.

Java Agent Development Framework (JADE) [17] is a software framework fully implemented in the Java language. The goal of JADE is to simplify implementation of multi-agent systems through a middleware that comply with the Foundation for Intelligent Physical Agents (FIPA) [5] specifications. The agent platform can be distributed across machines running different operative systems, and the configuration can be controlled via a remote GUI. It is also possible to change the configuration at run-time by migrating agents from one machine to another one. The minimal system requirement to run JADE is the version 1.2 of JAVA Standard Edition (J2SE) [11]. Although, it is possible to run JADE on all devices that support J2SE, there are currently only few mobile devices that are able to that. This means that JADE only partially solves the problem of running mobile agents on mobile devices, since only the most powerful ones are capable.

LEAP (Lightweight Extensible Agent Platform) is a FIPA platform that can be used to deploy multi-agent systems spread across a heterogeneous network of mobile and wired devices, ranging from mobile phones to enterprise servers [2]. LEAP has developed a new kernel for JADE that allows legacy JADE agents to run on small devices without any modifications, provided that the devices offers sufficient resources and processing power [1]. The initial version of LEAP had the following functional characteristics: It runs on desktop PCs, PDAs and Java-enabled mobile phones; it exploits wireless networks such as TCP/IP over GSM and IEEE 802.11 Wireless LAN [7]; and it runs on several operating systems. The current implementation of LEAP is possible to run on powerful PDAs, but not yet for Java-enabled mobile phones. Our approach makes it also possible for less powerful Java-enabled devices to access a mobile system.

The aim of the project Mobile Information Agents (MIA) is to develop an intelligent information system, which puts information of local relevance from the WWW into the hands of mobile users [6]. The MIA project uses a client-server architecture where the user carries the client. The system is made for mobile use, requiring a wireless connection to the Internet (e.g. by using a capable mobile phone). The MIA system also needs to know where the user is. This can be done by using a GPS-receiver or the mobile phone can be tracked in the mobile network. The MIA system currently support three types of users [3]: 1) A mobile user equipped with a PDA, GPS and mobile phone; 2) A mobile

user equipped with a WAP enabled mobile phone and additionally a PDA; and 3) A stationary user with a PC and standard web browser.

In [4], the design of a mobile-agent system that provides a mobile user with a personalized information retrieval service is presented. The paper also describes the implementation of the infrastructure for such a system. The Personal Agent System was developed on a Hewlett Packard Jornada running Windows CE, and Wireless LAN was used for communication between the PDA and server. The personal Agent System gathers information from the Internet, and uses context-aware mechanisms to manage the information according to the needs of a mobile user. The schedule and location of the user are the context indicators in this system. The indicators are critical in ensuring that users only obtain the information they want, receive the information in a form that is best fit for viewing on their mobile device, and that the users are notified of new information in a minimal intrusive manner. The system incorporates a rule-based learning system to enhance the personalization achieved by the system.

### 3 High-level Architecture

A logical view [10] of the architecture of our heterogeneous mobile agent system is illustrated in figure 2. A central part of the architecture is the agent system repository where agent information and agents can be stored along with client information such as client device capabilities. We have defined a thin agent client that only deal with the agent state, where the code of the agent will be stored on the agent server. The thin agent client contains code for transmitting and receiving agent data and a simple GUI for manipulating this data. For devices with sufficient memory and CPU, mobile agents will run locally on the device.

Above the repository the general agent server services are located that controls the essentials agent services such as registration of agents, locating agents, management of agent lifetimes (initiate agents, clone agents, kill agents) etc. The rest of the architecture is split into two main parts; one for thick agent clients, and one for thin agent clients. The thick agent client is able to execute mobile agents locally, while the thin agent client only is able to manipulate the agent data (state). There is only one part that distinguishes the architecture of the thick and thin agent client; the Agent joiner/splitter. The *Agent joiner/splitter* makes it possible to split the data and code of the mobile agents before dispatching the agent to the thin client. This makes it possible to access mobile agents on less capable devices. A more detailed description of the Agent joiner/splitter is presented in section 3.2.

The *Agent initiator* is responsible for initiating the agent clients first time, or reconfiguring the agent client based on changes in the mobile agent software or hardware/software

changes in the client. The agent initiator is explained in more details in section 3.1. Although there are two Agent initiators drawn in the figure 2, only one such service is running (same for thick and thin clients).

The last component of the architecture is the *Mobile Agent Dispatcher* which makes it possible dispatch agents between clients and server. For thin clients the Agent joiner/splitter is used to split the agent before dispatching to a client, and to assemble the agent when received from the client.

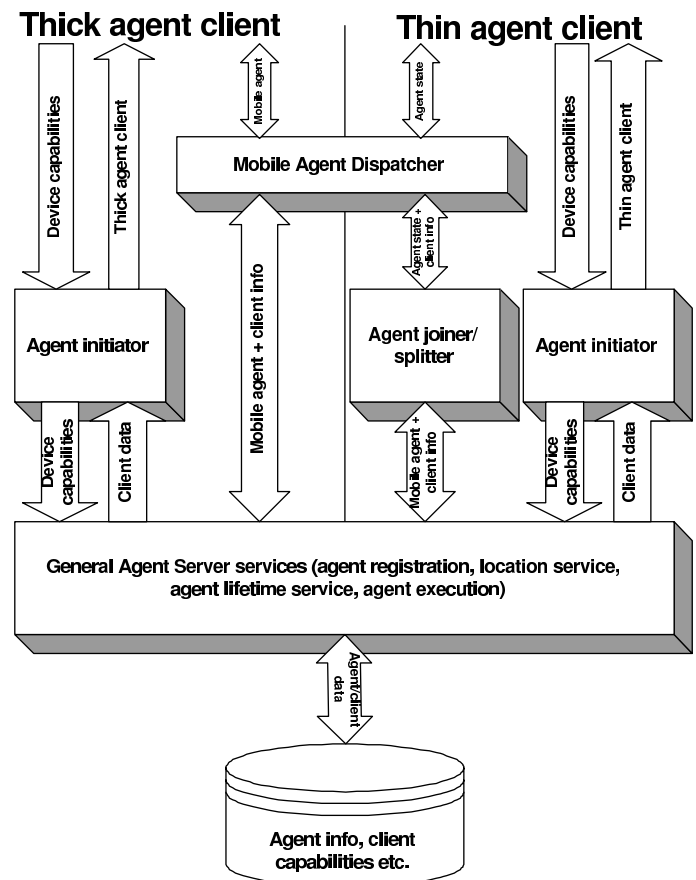


Figure 2. A logical view of the mobile agent architecture

#### 3.1 The Agent Initiator

When a client connects to the agent server for the first time, it must state the client device capabilities in terms of CPU speed, execution memory, storage, and Java Virtual Machine edition and version. Although a client device has enough memory and fast enough CPU, there could e.g. be no Java virtual machine available that support serialization for the device. This is the reason that Java Virtual machine is also a part of the client capabilities. The capabilities are described in XML. Figure 3 shows an example of a description for a client running on a Palm Tungsten T PDA. The

client capabilities described in XML consist of five main parts: *Device*, *CPU*, *Memory*, *Java* and *Download*. The *Device* part is used to describe the device in terms of name and operating system running. The next two parts *CPU* and *Memory* describe the device capabilities in terms of processing power and executing memory and storage space. The *Java* part describes the available Java environments on the device in terms of edition and versions. The last part, *Download*, makes it possible to specify how the agent client should be downloaded to the device. Our client capability description is a small subpart of terminal resource capabilities in Composite Capability/Preference Profiles (CC/PP) [18] tailored for mobile agent purposes. The terminal resource capabilities in CC/PP are tailored for use with Web-browsing.

```

01: <ClientCapabilities>
02:   <Device>
03:     <Name>Palm Tungsten T</Name>
04:     <OS>PalmOS 5.0</OS>
05:   </Device>
06:   <CPU>
07:     <Name>OMAP1510</Name>
08:     <Type>32bits</Type>
09:     <Speed>175MHz</Speed>
10:   </CPU>
11:   <Memory>
12:     <RAM>16MB</RAM>
13:     <Available>5MB</Available>
14:     <Secondary>64MB</Secondary>
15:   </Memory>
16:   <Java>
17:     <VirtualMachine>
18:       <Edition>MicroEdition</Edition>
19:       <Version>1.0</Version>
20:     </VirtualMachine>
21:     <VirtualMachine>
22:       <Edition>SuperWaba</Edition>
23:       <Version>1.3</Version>
24:     </VirtualMachine>
25:   </Java>
26:   <Download>
27:     <Type>Email</Type>
28:     <AddInfo>james@bond.007</AddInfo>
29:   </Download>
30: </ClientCapabilities>

```

Figure 3. Description of mobile client in XML

When a mobile agent client is initialized the first time, it follows the following process:

**1. Install agent initiator:** The agent initiator is a simple Java program that must be installed on the mobile device before installing the agent client. This program is used for extracting device capabilities on the client. The devices capabilities that cannot automatically be detected must be entered manually by the user. In addition, the agent initiator contains software for communicating with an agent server.

**2. Configure the agent initiator:** The user should look through the device capabilities detected, and add missing,

or change faulty information.

**3. Initiate the agent initiator:** The agent initiator sends the device capability information to the agent server.

**4. Install agent client:** Based on the device capabilities, a suited agent client will be sent to the device and then installed. The agent client can be sent directly using the agent initiator, or indirectly using transport channels such as email or HTTP-download.

If any characteristics of the client are changed (e.g. a new virtual machine for Java has been released), a new initializing process is initiated. Also if there are major changes of the agent system itself, the agent server may initiate a client update process (similar to the process shown above). The reason for this is that the new version of the mobile agent system could be more CPU and/or memory efficient than the previous one, making it possible to run the full mobile agent system on clients that previously were not powerful enough.

### 3.2 The Agent Joiner/Splitter

For clients that cannot run the mobile agents locally (because of client capabilities), we have designed an agent joiner/splitter for allowing these clients to access the agents anyway. This is done by separating the state (or data) of the agent with the code of the agent. This means that the thin agent client only receives the data part of the mobile agent while the code part resides on agent server. In addition, the client on the mobile device has a graphical user interface (GUI) making it possible to instruct the agents. When the user decides to send an agent from the mobile device, the state of the agent is transmitted to an agent server. This means that both the code and the state of the agent are joined on the agent server. Figure 4 illustrates the process view of a migration of an agent (client application) from the mobile device to the agent server. When an agent migrates to another agent server or thick agent client, both the code and the state are transmitted (as for normal mobile agent systems).

The agent's lifecycle is illustrated in figure 5 and can be split into respectively client and server states. The client states are:

**1. Initialize:** The state of a new agent is populated by options given by the client (see section 4).

**2. Migrate (C/S):** The state of the agent migrates from the client to the server.

**8. Receive:** The state of the agent is transmitted from the server to the client.

The server states are:

**3. Initialize:** The agent performs one-time setup activities by building the initial data structures of the agent. In this state the code and data of the agent are joined.

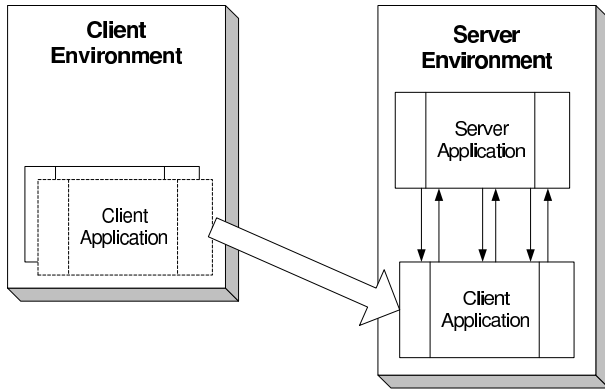


Figure 4. A process view of the mobile agent architecture

4. **Start:** The agent is running (and performs its tasks).
5. **Stop:** The agent stops the execution and saves the state.
6. **Migrate (S/S):** The agent migrates from one server to another (both code and state).
7. **Complete:** The agent is prepared for call-back by the client.
9. **Terminate:** The agent is removed from the agent server.

The states shown in Figure 5 do not include acknowledgement of successful transmission between the client and the server. Note also in the case of migration to thick agent clients, the clients will act as an agent server and server states (as described above).

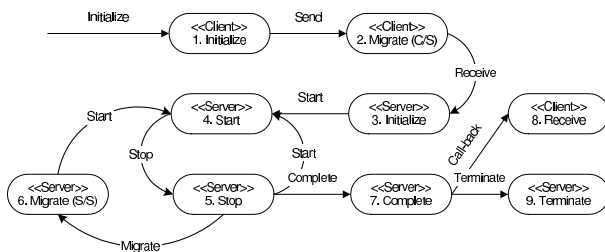


Figure 5. State view of the mobile agent architecture

## 4 The Mobile Agent Prototype

To demonstrate a useful mobile agent system, we wanted to implement a prototype for buying and selling second-hand products using mobile devices. This means that the user configures a mobile buying agent or selling agent on the mobile device. The mobile agent will dispatch to an electronic market, where selling and buying agents try to get to an agreement. This prototype focused on the implementation of the agent joiner/splitter and its interaction with the agent server.

Here is a short description of running the prototype. Af-

ter starting the agent application users can choose between initiating a new agent and recalling a previous agent. If the user chooses to create a new agent, the screenshot shown to the left in figure 6 will be displayed.

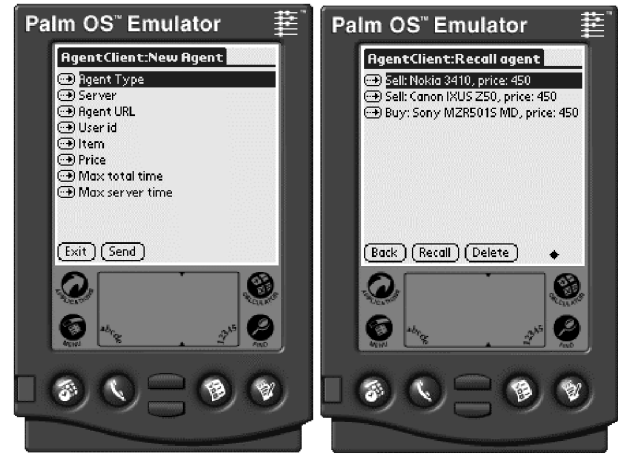


Figure 6. Screenshots from running the agent system on a PDA

The screenshot to the left in figure 6 shows how an agent can be configured. The four first options are used to configure the agent system by entering the agent type, agent server, agent URL, and User ID. The option 1, **Agent type**, may either be a *Sales agent* or a *Buyer agent*. The option 2 is the **Agent server** that is configured to a default server as the agent system starts. Here it is also possible to select another agent server. The third option is the **Agent URL** that specifies the location of the agent's jar-files. These files are remote, because of the limited resources on the mobile device. The agent URL option has also its default values that can be overridden if wanted. Note that the *Agent server* and *Agent URL* options should not be available for the user if we wanted to implement a real end-user system. Since the users of our prototype were system experts, these options did not cause any problem. The option 4, **User id**, is where a user writes in the email address. Email addresses are used for the user identification. This is needed when the agent returns for example with a buyer for an item, in order to make the user able to contact the buyer through an email. It is also possible to use a phone number for user identification.

The option 5, **Item**, is where the user can choose from a predefined list of items. It is also possible to add new items to this selection list.

The option 6, **Price**, is the next option users can set. If the agent has chosen a buyer agent this will be the maximum price the agent can bid, and if the user has chosen a sales agent this will be the minimum price the agent will accept. The two last options (7 and 8) concern time. The first option 7 sets a **maximum total time** the agent can be away before being recalled. The option 8 sets a **maximum**

**server time** the agent will spend on each server before migrating.

When the user wants to recall an agent, this can be done by choosing the recall agent button as shown in the screenshot to the right in figure 6. A list of the agents already sent out will then be shown up. The user can also delete (kill) agent from this screen.

## 5 Issues and Discussion

This section first describes the technology we used, and then it will comment on some implementation issues, and last discuss some limitations with our prototype.

### 5.1 Technology used

We used Java 2 Micro Edition (J2ME) to implement the thin agent client in our prototype. J2ME is a definition of technologies and specifications that are designed for various types of devices. The modular design of J2ME is ensured through configuration and profiles [15]. A J2ME configuration defines a minimum of the Java platform for a family of devices [13] like a set of Java programming language features, a set of Java virtual machine features, and a set of supported Java libraries and application programming interfaces (APIs). For our prototype we chose the Connected, Limited Device Configuration (CLDC) that supports personal, mobile devices. CLDC devices usually have the following properties [13]: 160 to 512 KB total memory available for the Java platform, 16-bit or 32-bit processor, Low power consumption, often battery powered, and Intermittent network connectivity (often wireless) with potentially limited bandwidth.

The CLDC configuration can run on most PDAs and also on Java-enabled mobile phones. However, because of the limited memory and processing power, several features from the standard Java Edition is not supported, such as floating point calculations, object serialization, object finalization, reflection, user-defined class loaders, etc. These limitations provide challenges when implementing a mobile agent system on mobile devices.

### 5.2 Implementation Issues

As described in the last section, we decided to use J2ME to implement our thin mobile agent client for mobile devices. By using J2ME, there were several limitations we had to overcome. Here is our solution to these challenges:

- **Agent separation:** Since the mobile devices still have minimal resources to run mobile agents, it was never an option to run an entire agent on a mobile device.

To overcome this problem, the agent application was separated in two parts (code and data), with only the data residing on the mobile device along with the user interface classes.

- **Persistence:** The CLDC for J2ME we used in the implementation does neither support object serialization nor reflection. This means that there is no built-in mechanism to persist objects into a byte stream. This is a very important part of a mobile agent system since it allows the agent to carry the state to a new server. We had to make our own persistence mechanism by using the `DataOutputStream` and `DataInputStream` classes from the `java.io` package. This made it possible to persist the agent data, send it to the next server, and then resurrect it.
- **Writing to the memory:** Since an agent can be configured to operate for a long period of time, it was necessary to save the agent ids in the non-volatile memory of the mobile device. This makes it possible to exit the application, and recall the same agent the next time the application is started. J2ME's Record Management System (RMS) was used to store the agent id.

### 5.3 Limitations of the prototype

As described in section 3.2, our solution for solving the problem of running mobile agents with limited processing power and limited memory was to split the mobile agent into two parts: The data of the agent on the mobile device and the code at the server. By doing this separation, the mobile agent cannot be executed at the mobile device. By choosing this approach, we have also implicitly split the graphical user interface (GUI) code from the agent itself. This means that the agent on the mobile device only have data items that can be filled with values using GUI code at the device. For our application of the mobile agents, this worked rather well and efficient. However, if advanced interactions with the user or computations are required at the mobile client, this approach will not work. Our approach will also limit the look-and-feel of the GUI, because the GUI code must be able to handle any kind of data and is not tailored for specific data input. But since J2ME anyhow has a rather limited GUI library, this is not a big issue.

### 5.4 Use of the Architecture

In this paper, our presentation of the architecture has focused on two different types of clients: Thin and thick client. However, it is possible to use our architecture to allow more than the two kinds of agent clients as previously described in this paper. One possibility is to allow more

than one kind of “thick client”. A simple light-weight version could contain only the most necessary services for retrieving, sending and executing an agent. A more resource demanding version of the client could e.g. have transaction support or other more advanced agent services that demands more resources. It is also possible to use our architecture as a starting point to allow tailoring of the agent clients. This means that user can specify his needs of services for the mobile client along with the client device capabilities. The agent server will then deliver the client that is tailored to the users needs and is able to run on the device.

## 6 Conclusion

Mobile devices have many limitations compared to stationary PCs. Not all of these limitations such as the screen size, network bandwidth, battery capacity, and input devices, have much influence on a mobile agent application. There are other more noticeable constraints that limit an agent application. However, these constraints are possible to overcome. The most noticeable limitations when making a mobile agent application for mobile devices where:

- Limited connection time – users have to call back the agent themselves.
- Memory - if the device does not have enough memory, applications will not run. This was however not observable for our simple prototype.
- No object serialization – this was solved using the `DataOutputStream` and `DataInputStream` classes from the `java.io` package.

Our architecture as presented in this paper aims at making a flexible mobile agent architecture that enables various mobile devices to access the same mobile agent system. This is possible by letting the clients state their device capabilities before installing the agent client. Also the agent splitter/joiner makes it possible to access the mobile agents on devices with very limited resources such as Java-enabled mobile phones.

Future work will explore a more general architecture that allows a variety of agent clients tailored for different kinds of clients and by the user.

### Acknowledgement

This paper is a result of work in the project MOBILE Work Across Heterogeneous Systems (MOWAHS) [14]. The MOWAHS project is sponsored by the Norwegian Research Council's IKT-2010 program. We would give special thanks to Letizia Jaccheri for giving useful comments on software architecture and the paper.

## References

- [1] Adorni, Giovanni, Bergenti, Federico, Poggi, Agostino, Rimassa, and Giovanni. Enabling FIPA Agents on Small Devices. In [8], pages 248–257, 2001.
- [2] Bergenti, Federico, Poggi, and Agostino. LEAP: A FIPA Platform for Handheld and Mobile Devices). <http://leap.crm-paris.com/public/docs/ATAL2001.pdf>, 2001.
- [3] Gerd Beuster, Bernd Thomas, and Christian Wolff. MIA - A ubiquitous multi-agent web information system. In *Proceedings of International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000)*, 2000.
- [4] Debbie O. Chyi. An Infrastructure for a Mobile-Agent System that Provides Personalized Services to Mobile Devices. Technical Report TR2000-370, Dartmouth College, Computer Science, Hanover, NH, May 2000.
- [5] FIPA. The Foundation for Intelligent Physical Agents. <http://www.fipa.org/>, Accessed October 30th 2002.
- [6] MIA Research Group. MIA - Mobile Information Agents for the WWW. [http://www.unikoblenz.de/~bthomas/MIA\\_HTML/](http://www.unikoblenz.de/~bthomas/MIA_HTML/), 2002.
- [7] IEEE. IEEE 802.11 Wireless Local Area Networks. <http://grouper.ieee.org/groups/802/11/>, 2002.
- [8] Matthias Klusch and Franco Zambonelli, editors. *Cooperative Information Agents V, 5th International Workshop, CIA 2001, Modena, Italy, September 6-8, 2001, Proceedings*, volume 2182 of *Lecture Notes in Computer Science*. Springer, 2001.
- [9] David Kotz and Robert Gray. Mobile Code: The Future of the Internet. In *Workshop Mobile Agents in the Context of Competition and Cooperation at Autonomous Agents'99*, pages 6–12, May 1999.
- [10] Philippe Kruchten. Architectural Blueprints – The '4+1' View Model of Software Architecture. *IEEE Software*, 12(6):42–50, November 1995.
- [11] Sun Microsystems. Java 2 Platform, Standard Edition (J2SE). Web: <http://java.sun.com/j2se/>, 2002.
- [12] Dejan Milojicic, Frederick Douglass, and Richard Wheeler, editors. *Mobility; Processes, Computers and Agents*, chapter Chapter 13 Mobility on the Internet, pages 451–456. ACM Press/Addison-Wesley, 1999. ISBN 0-201-37928-7.
- [13] Vartan Piroumian. *Wireless J2ME Platform Programming*. California: Sun Microsystems Press, 2002.
- [14] MOWAHS project. MOBILE Work Across Heterogeneous Systems. Web: <http://www.mowahs.com>, 2001.
- [15] Roger Riggs, Antero Taivalsaari, and Mark VandenBrink. *Programming Wireless Devices with the Java TM 2 Platform, Micro Edition*". Addison-Wesley, 2001. ISBN 0-201-74627-1.
- [16] Umar Saif and David Greaves. Communication primitives for ubiquitous systems or rpc considered harmful.
- [17] TILAB. JADE (Java Agent DEvelopment Framework). <http://sharon.csel.it/projects/jade>, 2002.
- [18] W3C. Composite Capability/Preference Profiles (CC/PP) Structure and Vocabularies. Web: <http://www.w3.org/TR/CCPP-struct-vocab/>, 2003.