Development of mobile agents by using J2ME and similar technologies

J2ME is Java the platform where you can use to develop applications on micro wireless devices with limited resource (memory, CPU power, etc) such as cell phones. This thesis is about to investigate how J2ME and similar technology can be used to develop mobileagent applications who can be used on cell phones.

The task is also about to take the network security issue into account and find a solution to make that feasible and secure to use mobile agent applications on micro wireless device.
Abstract

The intention of this thesis is to investigate how the J2ME (Java 2 Micro Edition) platform can be used to develop mobile agent application on micro wireless devices such as cell phones with limited resources (limited memory budget, small screen size, etc ...).

Several mobile agent framework had been tested. Several J2ME cell phones had also been investigated.

The network security issue had been investigated. The data encryption and authentication method (process) used on the network had been investigated. The HTTPS (Secure Hypertext Transfer Protocol) had been investigated. The TLS (Transport Layer Security) and SSL (Secure Socket Layer) protocol had been investigated. The Bouncy Castle Cryptographic Package had also been investigated.

The prototype had been implemented and tested on the J2ME Wireless Toolkit.
Preface

This Diploma thesis has been performed at the Department of Computer Science at the university of NTNU.

This diploma thesis is a part of the MOWAHS project, and I would like to thank my adviser Alf Inge Wang for giving me some good advice.

Trondheim, 16-juni-2003

Quoc Tuan Nguyen
Acknowledgements

I would like to thank professor Reidar Conradi to give me the possibility to write this thesis. And I also would like to ask for a apology to not inform about my thesis work. Professor Reidar Conradi didn’t known that I did this thesis until later April 2003. He was completely surprise when he find out that I did this thesis. My fault. I’m sorry.

I should have informed professor Reidar Conradi better, maybe I should send the mobile agent to professor Reida Conradi and inform him about my thesis …….
The Convention used in this thesis ................................................................. 15
The Readers Guide ....................................................................................... 16
1 Introduction ............................................................................................... 19
   1.1 Problem definition ........................................................................... 19
   1.2 Goal of the project ......................................................................... 19
2 Context ....................................................................................................... 20
3 The prestudy .............................................................................................. 22
4 The J2ME (Java 2 Micro Edition) technology ......................................... 23
   4.1 The Latest new : ............................................................................ 23
   4.2 The J2ME platform ....................................................................... 26
      4.2.1 The CLDC class libraries ......................................................... 29
      4.2.2 MIDP Class Libraries .............................................................. 31
      4.2.3 Parsing XML om J2ME ............................................................. 32
5 The Servlet Technology ........................................................................... 33
6 The Micro Devices .................................................................................... 34
7 The Mobile Agent framework ................................................................. 35
   7.1 JADE - LEAP ................................................................................. 37
   7.2 Bee-gent ......................................................................................... 38
8 The Network security in J2ME platform ............................................... 41
   8.1 The HTTPS Connection .................................................................. 41
   8.2 The Authentication in J2ME platform ............................................ 42
   8.3 The other methods for data encryption and authentication ............... 43
9 The Propose solution ............................................................................... 44
10 Requirement for the prototype .............................................................. 57
   10.1 Functional requirement ................................................................ 57
   10.2 Non Functional requirement .......................................................... 63
11 Design ..................................................................................................... 64
   11.1 Step 1, Identify roles .................................................................... 64
   11.2 Step2, Identify responsibilities for each roles ................................. 78
   11.3 Step3, Determine plans ................................................................. 79
   11.4 Step 4, Determine belief structure of the agents ............................. 81
12 Implementation ....................................................................................... 85
   12.1 Som small change from the design .................................................. 85
   12.2 The technical detail of the implementation process ........................ 86
   12.3 More description of the technical details of the implementation process ........................................................................ 94
13 Testing ..................................................................................................... 114
   13.1 Test of the functional requirement ................................................... 114
   13.2 Test of the network security ............................................................. 123
14 The Conclusion ........................................................................................ 132
   14.1 The conclusion .............................................................................. 132
14.2 The future work ............................................................................................................. 133
References .......................................................................................................................... 135
Term Explanation ............................................................................................................... 137
Appendix A .......................................................................................................................... 140
The Convention used in this thesis

The reference will be set like this [ a number ], the reference will be set at the end of the sentence, to state that, that statement is based on that reference.

The important issue will be written with the bold letter, for example like this **Bouncy Castle Package**. To emphasise the important meaning of this issue.
The Readers Guide

This rapport start with the problem definition and the goal of this thesis in the Introduction.

Chapter 1, the introduction.

Chapter 2 present the MOWAHS project.

Chapter 3 give the short presentation of the prestudy rapport.

The prestudy rapport include 5 chapter.

It start with chapter 4.

Chapter 4 describe the J2ME (Java 2 Micro Edition) Platform.

Chapter 5 describe the Servlet.

Chapter 6 present a couple of J2ME cell phones.

Chapter 7 present the mobile agent framework.

And chapter 8 present the network security in J2ME platform.

The propose solution for the tasks defined in this thesis will be present in chapter 9.

The requirement for the prototype will be present in chapter 10.

The design for the prototype will be present in chapter 11.

Chapter 12 will present the implementation specification for the prototype.

The test of the prototype will be present in chapter 13 and finally the conclusion for this thesis will be present in chapter 14.

The reader can find the references right after chapter 14. The Term explanation will be present after the references.

The term explanation will help the reader to understand the unknown term when the reader read this thesis. For example, if the reader meet the term obfuscator. If it is strange for the reader, the reader will find the explanation in the chapter about the term explanation.

The Java code for the prototype will be present in the Appendix A.
1 Introduction

1.1 Problem definition

In the last four years, the use of micro mobile devices such as cell phones and PDAs had increased extreme high, both in Europe and other continents, million of peoples around the world love to be able to do their tasks, both private and jobs by using their micro mobile devices, such as cell phones or PDAs instead of the PC or laptop. To be able to satisfy the peoples (users) demand, these micro mobile devices (cell phones, PDAs) must offer advance softwares which already exist on the PC or laptop, such as mobile agent application.

But the mobile agent applications requires large memory resources to run. It will not be a problem on the PC or laptop but on the micro mobile devices such as cell phones, it will be a challenge, because these devices have much more limited resources than PC or laptop.

This thesis is about to develop a secure solution to make it feasible for peoples to use mobile agent applications on micro mobile devices such as cell phones by using the J2ME technology and similars.

1.2 Goal of the project

During this thesis work, the technologies like J2ME, Java Servlet and several State of the art in mobile agent framework will be studied.

Then, finally, a secure solution to how to develop and use mobile agents on mobile devices such as cell phones will be proposed and a prototype will be designed and implemented.

The goal of the thesis will also be to take the network security issue into account. To make the solution secure to use mobile agent applications from J2ME cell phones.
2 Context

This diplom thesis is part of a larger project called MOWAHS, running at IDI, NTNU.

MOWAHS is a basic research project supported by the Norwegian Research Council in its IKT-2010 program. The project is carried out jointly by the IDI's groups for software engineering (prof. Reidar Conradi, coordinator) and database technology (prof. Mads Nygård).

The project will, respectively, have two parts: process support for mobile users using heterogeneous devices (PC, PDA, mobile phones) and support for cooperating transactions/workspaces holding work documents. The project will build upon CAGIS technology and an industrial cooperation is planned.
Budget: 5 MNOK in total over four years, two postdocs and two PhD students.

Introduction to the MOWAHS project

The Internet now has 350 million users. It is expected to be 3 billion mobile phones world-wide in 2004, and 1/10 of these will have Internet access and be attached to powerful computers (source: IDC). More and more companies are therefore working as virtual organizations, where people are distributed over several locations and time zones. That is, distributed, mobile and partly asynchronous technology enables sharing of documents and work plans. However, the infrastructure and tools for carrying out projects in virtual organizations are immature. We must deal with a heterogeneity of tools, equipments (laptops, PDAs, mobile phones) and work models. In addition, mobility of devices and partial lack of connectivity require regular synchronization of such devices against stationary servers and PCs.
A research challenge is therefore to provide an efficient and user-friendly environment for helping people in virtual organizations to perform and coordinate their work at their current location, time and device configurations.

The support technologies for the above is in an enormous flux of development. Thus, a research project with 5-6 researchers must pursue a limited set of achievable subtasks. By building on local research in the EPOS and CAGIS projects over the last 10 years and on recent middleware technologies (XML and mobile agents), we mean that we have a head start in establishing a functional, innovative, cheap and flexible technology platform, that quickly can lead to validatable and useful results.

Goals: The MOWAHS goals are threefold:

G1) Helping to understand and to continuously assess and improve workprocesses in virtual organizations.

G2) Providing a flexible, common work environment to execute and share real workprocesses and their artifacts, applicable on a variety of electronic devices (from big servers to small PDAs).

G3) Disseminating the results to colleagues, students, companies, and the community at large.

Partners: the software engineering and database research groups at IDI, NTNU. These have together 8 teachers, 12 PhD students and 25 MSc students. The groups will exploit and enhance their competence in process support, workspace management, agent
technology and cooperative transactions -- e.g., from the national EPOS and CAGIS projects, and from the ESPRIT projects PROMOTER and PROMOTER2.

**Approach:** to iteratively:
-- Define a flexible work environment for virtual organizations using heterogeneous devices, with support for processes and their artifacts and transactions.
-- Implement a testbed for process support for virtual organizations, using XML-based and mobile agents.
-- Use real scenarios to evaluate the environment, e.g., for software development and remote education. 25% of the work will be spent on formulating requirements and success criteria, and to perform empirical studies. In all this, own MSc students will be actively used for implementation and evaluation.

### 3 The prestudy

The prestudy rapport will consist of the following subjects:

Chapter 4 : The J2ME (Java 2 Micro Edition) technology.

Chapter 5 : The Servlet technology.

Chapter 6 : Description of some micro mobile devices (cell phones).

Chapter 7 : Some State - of - the - art of the mobile agents frameworks

Chapter 8 : The network security in J2ME platform

Chapter 9 : The propose solution.
4 The J2ME (Java 2 Micro Edition) technology

This chapter will give you a description of the J2ME platform.

First I will describe what is the latest new of the J2ME platform. Then I will give you a more detail description of the J2ME platform. If you are already familiar with the J2ME or had some knowledge of the J2ME platform before, you only have to read about the latest new in J2ME platform.

4.1 The Latest new:

The J2ME platform had been introduced in 2001. Since that day, the growth of this platform had been very fast. Today, in the year 2003, many more functionalities had been adopted to this platform.

Last year, in 2002, It was the MIDP (Mobile Information Device Profile) version 1 who had been released; that mean the MIDP 1.0 and the CLDC 1.0 (Connected Limited Device Configuration) but in this year, 2003, a new version of the MIDP and the CLDC had been released and I mean the MIDP 2.0 and the CLDC 1.1.

The CLDC 1.1 do not include any radically change to the old CLDC 1.0.

But the new MIDP 2.0 has now a lot more advance functionalities than the MIDP 1.0, for example [8].

1) Schedule
2) Secure networking
3) Multimedia
4) Form Enhancements
5) The Game API
6) RGB Images
7) Code Signing and Permission

But the most important point which will have great meaning to my thesis (to implement mobile agent) is the Network Security.

The MIDP 1.0 implementations did actually support HTTPS but you can not always rely on it availability. The new MIDP 2.0 provide a stable, consistent foundation for wireless applications that deal with money or sensitive information.

The MIDP 2.0 requires the HTTPS, which is basically the HTTP over Secure Socket Layer (SSL).

The MIDP 2.0 support the TLS protocol (Transport Layer Security) and SSLv3 (Secure Socket Layer).
This is very important. When you deal with or work with mobile agents applications. The important issue you always must think or care about is the security. Because mobile agents are mobile applications who are moving across the network. The mobile agents must find someway to communicate with their owners or uses. And how can the mobile agents know for sure that the messages they receive actually came from their owners or uses and not from someone else who pretend to be their owners? The best way is encryptions of the data and provide authentication. It will stop the hacker who eavesdrop the data to understand the information you send and receive from your mobileagents.

The TLS protocol is an update version of the SSLv3 protocol. Transport Layer Security (TLS) is a protocol that enables authentication and data encryption over insecure networks.
It is implemented as a layer between TCP/IP and higher-level network protocols like HTTP, SMTP, and NNTP.

The `javax.microedition.io.HttpsConnection` is one of many interfaces that support secure networking [8].

The `javax.microedition.io.SecurityInfo` support secure connection and the

`javax.microedition.pki.Certificate` support a cryptographic certificate [8].

HTTPS support is provided through the CLDC's Generic Connection Framework in the `javax.microedition.io` package [8].

**MIDP 2.0 have now four new packages [1]:**

1. `javax.microedition.lcdui.games`
2. `javax.microedition.pki`
3. `javax.microedition.media`
4. `javax.microedition.media.control`

In MID 2.0, a MID is a device with the following minimum characteristics [1]:

- A display with a screen size of 96x54 and depth of 1 bit (black and white)
- An input mechanism such as a one-handed keyboard, two-handed keyboard, or touch screen
- 256KB of memory beyond what's required for CLDC, 8 KB of non-volatile memory for persistent data created by applications, and 128KB of volatile memory (heap) for the Java runtime
- Connectivity to network: wireless with limited bandwidth
- The ability to play tones (sound capability)

All the MIDlets written for MIDP 1.0 will continue to run on MIDP 2.0 environments.
Overview of CLDC Generic Connection Framework and MIDP networking

From Last year:

![Diagram of the CLDC Generic Connection Framework](image)

Figure 1  CLDC Generic Connection Framework [3]
Now here are the new:

![Connection Diagram](image)

**Figure 2**  The new hierarchy of the Generic Connection Framework [2]

All yellow interfaces are part of CLDC 1.0, the HttpConnection interface was added by MIDP 1.0, the blue interfaces are added by MIDP 2.0.

**Summary of the news in J2ME platform:**

I will now short sum up the news on J2ME platform who are important for my project. The new MIDP 2.0 has now better possibility for authentication and data encryption than the old MIDP 1.0 from last year, because the new MIDP 2.0 must support the TLS (Transport Layer Security) and SSv3 (Secure Socket Layer) protocol. And this is useful and important. This will now give you better possibility to provide data authentication and encryption over the network.

Now, you had read about the latest new of the J2ME platform, I will now give you a description of the J2ME platform in general. If you are the reader who are already familiar with J2ME, you can skip the next general description of the J2ME.

### 4.2 The J2ME platform

The J2ME (Java 2 Micro Edition) platform is the latest platform of the Java 2 Edition platforms. The J2SE (Java 2 Standard Edition) contains the JVM (Java Virtual Machine
who can only run on computer desktop or laptop, because the Java Virtual Machine requires large amount of memory resources to run.

Today, peoples want to run the Java applications on micro device such as cell phones which had less memory resources than computer desktop. That is why the J2ME platform was created. The heart of the J2ME platform is the KVM (Kilo virtual Machine). The KVM is similar to the JVM (Java Virtual Machine). But the different is that the KVM requires less memory resources to run. The KVM is designed for micro wireless mobile devices with limited memory resources such as cell phones and PDAs.

The KVM had less functionalities than the ordinary JVM because the KVM can not adopt all of the functionalities available on the JVM, this is because of the limited memory resources on micro mobile devices such as cell phones or PDAs.

One of the most important function who is missing on the KVM is the custom class loader. Because of the security issue, the KVM didn’t allowed the custom class loader.

And this is the biggest challenges for you, if you want to run mobile applications (mobile agents) on cell phones. How can you transfer mobile applications (mobile agents) on micro mobile devices such as cell phones who are using J2ME platform without a custom class loader?

I will come back to this issue later in this thesis rapport. This is the important issue and challenges if you want to implement mobile agents on cell phones by using J2ME platform.

But now I will continue to describe the J2ME platform.

Figure 3 All the Java 2 Edition platforms [4]

Figure 3 show all the Java 2 platform, as you can see, the J2ME platform is the smallest and it is designed for micro devices.
What is the CLDC?

On figure 3 you see the CLDC, what is that? The CLDC is the connected limited devices configuration. It is the configuration for the J2ME platform.

The CLDC (Connected Limited Devices Configuration) inherited some of the classes from the J2SE (Java 2 Standard Edition), and it has their own classes too.

![CLDC Architecture](image)

Figure 4  CLDC architecture [4]

![CLDC Diagram](image)

Figure 5  The CLDC [4]

The CLDC (Connected Limited Devices Configuration) has some limitations compare to the CDC (Connected Device Configuration).

The limitations are as following:

1) CLDC does not support floating point number

2) CLDC does not include Object.finalize method

3) CLDC have limited error handling

4) CLDC does not implemented Java Native Interface.
5) CLDC does not support the user defined class loaders (Important Issue).
6) CLDC does not support RMI (Remote Method Invocation) or object serialization
7) CLDC does not support thread groups
8) CLDC does not support weak references

As mentioned earlier, the CLDC inherited some of the classes from J2SE. And CLDC had its own classes too. The classes specified to the CLDC is located in the javax.microedition package and its subpackages.

Profile of the J2ME platform is the MIDP (Mobile Information Device Profile). The CLDC does not defined APIs for Graphical User Interface, they leave use interface issue to profiles.

The applications for J2ME platform are called MIDlets.

![Figure 6 The MIDP architecture](image)

4.2.1 The CLDC class libraries

The CLDC class libraries include the following classes [10]:

- java.lang
- java.io
- java.util
- javax.microedition.io
java.lang


Interface:

Runnable

java.io

Object, InputStream, ByteArrayInputStream, DataInputStream, OutputStream, ByteArrayOutputStream, DataOutputStream, PrintStream, Reader, InputStreamReader, Throwable, Exception, IOException, EOFException, InterruptedException, UnsupportedEncodingException, UTFDataFormatException, Writer, OutputStreamWriter

Interface:

DataInput, DataOutput

java.util

Object, Calendar, Date, Hashtable, Random, Throwable, Exception, RuntimeException, EmptyStackException, NoSuchElementException, Timer, TimerTask, TimeZone, Vector, Stack

Interface:

Enumeration
### 4.2.2 MIDP Class Libraries

The MIDP Class libraries include the following classes [10]:

- `javafx.microedition.midlet`
- `javafx.microedition.lcdui`
- `javafx.microedition.rms`

#### `javafx.microedition.midlet`

Object, Midlet, Throwable, Exception, MidletStateChangeException

#### `javafx.microedition.lcdui`

Object, AlertType, Command, Display, Displayable, Canvas, Screen, Alert, Form, List, TextBox, Font, Graphics, Image, Item, ChoiceGroup, DateField, Gauge, ImageItem, StringItem, TextField, Ticker,

**Interface:**

Choice, CommandListener, ItemStateListener
javax.microedition.rms

Object, RecordStore, Throwable, Exception, RecordStoreException, InvalidRecordIDException, RecordStoreFullException, RecordStoreNotFoundException, RecordStoreNotOpenException

Interface:
RecordComparator, RecordEnumeration, RecordFilter, RecordListener

4.2.3 Parsing XML on J2ME

It is fully possible to use XML parsers on J2ME platform as on J2SE platform. But the
different is that you cannot use the same XML parses package for J2SE platform. You
must use the smaller XML package specific for J2ME platform. This is because limitation
of size and memory on J2ME platform.

There are three fundamental parser type [9]:

1) A Model Parser: Read an entire document and creates a representation of the
document in memory

2) A Push Parser: Read through an entire document

3) A Pull Parser: Read a little bit of document at once

Here are some of XML small parser appropriated for J2ME platform

- kXML1.2
- kXML2.0 alpha
- ASXMLP 020308
- MinML1.7
- NanoXML 1.6.4
- TinyXML1.7

I used the kXML parser when I parsed the data on the MIDlet, I used that in my
prototype.
5 The Servlet Technology

The big Picture

Java Servlet is a part of the J2EE (Java 2 Enterprise Edition Platform) platform.

Let take a look at big figure of the J2EE platform.

![Diagram of J2EE platform](image)

Figure 7 The big picture of the J2EE platform [5]

If you look at figure 7, you see that when you use Servlet technology, you can either use browser or application client. The application client mean J2SE applications or J2ME applications. This mean you can use J2ME applications program (Midlets) to connect to servlet.

This is a very useful way to do business. You will see later in this rapport that this Servlet technology will help you a lot when you want to send message to the mobile agents.

The Servlet will work as a bridge to connect the J2ME applications (Midlets) and the mobile agents applications (That mean if you want to use HTTP protocol to send message, otherwise, if you use Socket connection, you can go direct to the mobile agent server, you don’t need to go through Servlet)

I will describe that in detail when I present to you my propose solution.

Today, many micro devices in the market had been more powerful and a lot of them offer both the HTTP connection and the Socket connection and the other network connection.
But, the devices that offer more functionalities are also the most expensive one. The low-priced devices offer usually only the HTTP and HTTPS connection.

So, the solution who is low-priced for peoples to use, is the HTTP or HTTPS connection, you should use the HTTP or HTTPS connection (that mean using Servlet as a bridge to connect J2ME application to mobile agents applications). Because peoples can then use the low-priced cell phones, they don't need to buy the expensive cell phones.

Now, when we have the MIDP 2.0. We can use not only use the HTTP connection but also the HTTPS connection (The extra letter S stand for Secure). We will then achieve more security when we transfer our data or messages over the wireless network. The security is the most important Issue, peoples always demand security when they do business over the computer network.

The exist a lot of Servlet motor on the market, But for the research, you only need the Tomcat – Servlet motor. The Tomcat - Servlet motor is not perfectly for use in commercial business, but for research only, it is good enough.

I used the Tomcat – 4.0.2 motor in my project and it work very well.

6 The Micro Devices

I will now give you a short description of some micro mobile devices such as cell phones with the ability to run J2ME applications (Midlets).

But just keep in mind that the devices who offer most functionalities are also the most expensive one. The low-priced micro devices offer just some functionalities, so the challenges may be that we create a solution that make it feasible for peoples to use mobile applications on the low-priced micro devices (low-priced cell phones usually only offer the HTTP or HTTPS connection). There are only to important points I want to know when I need to know about the micro wireless devices and there are:

1) Can these micro devices run J2ME applications (Midlets)? and
2) How much memory storage capacity these micro devices do have?

Here are a couple micro mobile devices (cell phones).

1) **NOKIA 6310i**: 
• Java applications (space for up to 178 kB of Java applications); note: 128 kB of static memory and up to 50 kB of dynamic memory.

**Tri-band Operation:**

• EGSM 900, GSM 1800, and GSM 1900 networks in Europe, Africa, Asia, North, and South America; automatic switching between bands

2) **NOKIA 3650:**

• Java applications: 4 preset, max download size 64 kB per application

• World phone operation in GSM 900/1800/1900 networks

I will not describe the PDAs because most of the PDAs have more memory capacity than cell phones. There are more difficult to load J2ME applications on cell phones than PDAs because cell phones have much less memory capacity than PDAs.

### 7 The Mobile Agent framework
In this chapter, I will present some of the mobile agent framework. There exist today around the world many different mobile agent framework, written in many programming languages, such as Java, C++, and etc ….

Why mobile agent?

Why should we use mobile agent technology? What is the advantages the mobile agent technology will give us?

The mobile agent technology offer many advantages over the traditional client and server technology. Here are some of the advantages:

- It will reduce the network load.
- It don’t need to be continuous connected to the network.
- The tasks will be done 10 times faster than the traditional client-server technology.
- The mobile agent applications can make their own decision with or with out the user interactions. This will spare the user for a lot of times.

The mobile agents are the applications who are moving across the computer network, it move from computer to computer in the network and do their tasks, see figure 8.

Because of this characteristic (moving around). There must be a rule for how they will behave and communicate with each other. For example, when you travelling abroad, you must following the International rules and you must use some kind of language who is common with other peoples (English, French), so you can communicate with other peoples from other places (countries). And the rules (International law) are there to protect you too, make you feel safe.

The mobileagent applications need that to. How can they communicate with other agents from other places? How will they act or how must they behave to be able to avoid conflict?, and how can they be safe, how can they collaborate with other agents from other places.

The answer is that there must be an common environment so the mobileagent can run on it. In this common environment, there will exist rules, the rules are common for all the mobileagents who are running on this environment.

The rules will include some thing like: what kind of language the mobileagent must use to communicate with each other?, how will they share the memory resources in the computer they are running on, and how can they be safe or how will the environment protect them.

This enviroment are called the mobileagent framework. A mobile agent framework are the environment I mentioned above, a mobileagent framework define rules, all the mobileagents running on it must follow these rules. The mobileagent framework will also offer protection for the all the mobileagents who are running on it. All the mobileagents who are running on a specific framework will feel safe because the framework will protect them (protect them from computer virus or hacker). The framework will make it easier for the mobileagents to communicate and co-operate with each other, because they must speak the same language.
Figure 8  The mobile agent move from computer to computer across the network

There are a lot of mobile agent framework around the world today. They are written in many different programming language, Java, C++, Prolog, etc … .

I will only focus on the mobile agent framework who are written in Java language. Because I used the J2ME platform. I will present to you a couple of mobile agent framework next.

7.1  JADE - LEAP

In Europe. There was a project called LEAP (Lightweight Extensible Agent Platform) who goals was creating an intelligent agent platform running on micro wireless devices such as PDAs. (As of this writing, the LEAP project was over, finished). They had achieved running FIPA Agents on light weight PDAs under J2ME.

JADE-LEAP is a Java agent framework.
There is a tutorial about LEAP on the Web. But I am not sure that the LEAP had created mobile agents platform or only stationary intelligent agent platform. As of this writing, I can not find any information about LEAP which describe mobile agents running on micro wireless devices such as PDAs or cell phones. I can only find information about stationary agent running on micro wireless devices such as PDA when I read about LEAP.

There are many other mobile agents system around the world, but they are created to run only on computer desktop or laptop, not on micro wireless devices as cell phones.

During my project work, I tried to find a solution to make it feasible to run or use these mobile agent system (who was actually created for computer desktop only) on cell phones which support J2ME applications. And I had find a way to do that. I don’t had to create a mobile agent platform who only is specific for micro wireless devices, but I find a solution to make it feasible to use mobile agent on cell phones, and this mobile agent is a part of the mobile agent framework who run on computer desktop (I used the Bee-gent agent system).

I had tested many different mobile agents frameworks (who run on computer desktop) such as Aglet, Voyaer, Grasshopper, etc..... They all have advantages and drawbacks. But the mobile agent system who impress me most is the Bee-gent mobile agent system.

7.2 Bee-gent

Bee-gent (Bonding and Encapsulation Enhancement Agent).

The Bee-gent mobile agent system was created by TOSHIBA. This agent system is 100 percent pure agent. This is a Java agent system. It is very easy to use and have a lot of features.

As opposed to other systems which make only some use of agents, Bee-gent completely "Agentifies" the communication that takes place between software applications. The applications become agents, and all messages are carried by agents [7].
Figure 9  The Bee-gent mobile agent framework [7]

The Bee-gent architecture consist of to main types of agents .

1) The Mediation agents ( Who are the mobile agents ) can move from computer to computer to do their tasks .

2) The Agent Wrappers are used to agentify existing applications .

The mediation agents and the agent wrappers can communicate with each other by using ACL ( Agent communication language ) . And the Bee-gent framework support FIPA – Query Protocol .
Figure 10  The to type agents of Bee-gent framework, Agent Wrapper and Mediation agent [7]

When I create the prototype, I will use this Bee-gent framework as a demonstration to my proposed solution. The Bee-gent framework was created to run only on computer desktop or laptop. It is not created to run on micro wireless devices such as cell phones.

But I have found a way to use this mobile agent framework on cell phones which supporting J2ME.

And I will show you that in the prototype and in the chapter about the propose solution.

In Bee-gent framework, the mobileagent are the mediation agents, they move from place to place to do their tasks, and they communicate with each other by using ACL (Agent Communicating Language).

Figure 11  In Bee-gent framework, mediation agents are mobile agents [7].
8 The Network security in J2ME platform

This chapter will present the network security in J2ME platform. The security is a very important issue, peoples always demand security when they do business in the computer network. Because when you do E-commerce over the insecure network like Internet, the data exchange across the Internet often include very sensitive information, like credit card numbers and other personal data. That is why the network security is the very important issue. The J2ME platform must like the other platform as J2SE and J2EE platform, offer the network security, so peoples can feel safe when they use J2ME platform to do business. Because if peoples don’t feel safe, they don’t use it.

To offer the network security, the data transferring over the network must be encrypted, and some method must be use to authenticate the data.

A lot of work had been done to offer the network security in the J2ME platform. And as of this writing, the new MIDP 2.0 version had come and it offer the HTTPS connection. HTTPS (Secure Hypertext Transfer Protocol).

8.1 The HTTPS Connection

The network security in the old MIDP 1.0 is not fully, the MIDP 1.0 did actually support HTTPS but it is free for vendor to include the HTTPS, because the only mandatory protocol in the MIDP 1.0 is the HTTP. So as a user you cannot always rely on it availability.

A MIDP 2.0 implementation must support HTTPS connection.

The HTTPS is the HTTP carried over any of following protocols [11]:

- TLS 1.0
- SSLv3

The HTTPS Connection is the HTTP running on the TLS or SSL protocol
TLS (Transport Layer Security) is a protocol that enables authentication and data encryption over insecure networks. The TSL protocol is an updated version of the SSLv3 protocol (Secure Sockets Layer). The TSL protocol is implemented as a layer between TCP/IP and higher level network protocols like HTTP.

The TLS protocol begins with a handshake, in which the client and the server try to agree on a cipher suite (a group of cryptographic algorithms they will use for authentication and session encryption).

When the client and server agree on a cipher suite, they can authenticate each other and generate something called a master secret, which is used as the basic of the session key.

The session key is used to encrypt and decrypt all data sent between the server and the client.

The different between the TLS (Transport Layer Security) and the SSL (Secure Socket Layer) protocol are not much, the TSL protocol are an update version of the SSL protocol.

In the future, the TLS protocol will replace the SSL protocol.

In the MIDP 2.0, you can access the cipher suite name, by using the method `getCipherSuite()`.

For example [11]:

The following cipher suite name:

**TLS_RSA_WITH_RC4_128_SHA**

The name means that it is Transport Layer Security protocol, RSA is used for key exchange, an RC4 stream cipher with a 128-bit session key is used for data encryption, and SHA-1 is used as a message digest function.

The longer the session key, the more difficult it is to break the encryption code. So 128-bit session key is good for encryption. Today, most online commerce use 128-bit session key. The de facto standard for TLS cipher suites is 128 bits. You can also use longer session key than 128-bits and you must not use the session key that are shorter than 128-bits, because it will be much easier for the hacker to decrypt the short session key.

### 8.2 The Authentication in J2ME platform

There are two kinds of authentication [12]:

1) The Client authentication: When the client prove it identity to the server
2) The Server authentication: When the server prove it identity to the client

The MIDP 2.0 HTTPS implementation directly support the server authentication by using the **X.509 certificates**.
The de facto standard for certificates is X.509v3.

But the MIDP 2.0 still lack mechanics for client authentication, so you must find some other way for the client authentication process, for example, by using password, or message digest or certificate. I used password for the client authentication process in my prototype.

8.3 The other methods for data encryption and authentication

It is best to use the HTTPS connection for the encryption and authentication process, but before the MIDP 2.0 version came, back in 1999. There are other ways to encrypt data and provide authentication.

For example, the open source effort based in Australia, had produced the API for cryptographic algorithm in light weight J2ME. It is the Bouncy Castle Cryptographic Package.

If you want to use this package, you must first use the obfuscator to reduce the size of the package, and then using some methods to program the cryptographic process.

It is best to use the HTTPS connection. When you use the Bouncy Castle Package. You must self-design the encryption algorithms. And the drawback is that you can never know that the data encryption algorithms you create is good enough.

When you use the HTTPS connection on MIDP 2.0. The HTTPS connection provide the TLS protocol for you, and you can absolute be sure that the TLS is good enough for you.

The TLS is a well tested protocol. You can be sure that it is good. So, Even if you can provide data encryption without using HTTPS connection. I will recommend you to use the HTTPS connection as the first choice.
9 The Propose solution

This chapter will present the proposed solution to make it able to use mobileagent applications on micro wireless devices such as cell phones.

As I present to you on chapter 7, about the mobileagent framework. There are a lot of mobileagent framework on the world today, but they are created to run on computer desktop and laptop only. Now, if I want to use mobileagent applications on my cell phone, I must find some way to do that, to make that feasible.

What do I know about the cell phones and the mobileagent applications? By now, I know a lot about the cell phones and the mobileagent applications.

About the cell phones, they must use only the J2ME platform, that mean the cell phones can only run the applications who can run on J2ME platform. Because of the limited memory resource on cell phones. Today, as of this writing, no cell phones in the world can run J2SE applications.

And the J2ME platform have many limitation than the J2SE platform, as I have mentioned in chapter 4. One of the most important limitation on the J2ME platform is that the J2ME platform didn’t support the user defined class loaders. Because of the security concern, the KVM (Kilo Virtual Machine) in J2ME didn’t support the user defined class loaders.

So, it will not be possible to transfer mobileagent applications on cell phones. So, how can I use mobileagent on cellphones?

Now, I must go back and find out what do I known about the mobileagent applications. By now, I known a lot about the mobileagent applications.

The mobileagents are the system that are capable of independent action on behalf of its user or owner. They can move across the network. They have their own goals, they have their own beliefs, and they have their own knowledge.

So, what is that mean? That’s mean that I don’t had to transfer the mobileagent on the cell phones physically. But I only have to tell it what it shall do, and then the mobileagent will do it for me. That mean that the mobileagent can run on a remote computer, and I only had to send my mobileagent some messages to tell it what it shall do.

For example: If I want to sell a book, I can send my mobileagent to the e-market, my mobileagent will then try to find some other agents who are interested in buying my book.

And my mobileagent will negotiate with other buying agents and try to sell my book on my behalf.

My mobileagent will only had to known about how much money I want for my book. Then my mobileagent can use that price as an issue for the negotiation process. For example, my mobileagent will not sell my book for the price below that price I tell it in advance.
If I tell my mobilagent that I want to sell my book for 160 dollar, then my mobileagent will not accept any offers from the other buying agent which are below the price 160 dollar.

I will now use the Bee-gent framework to present to you my solution to make you use mobileagent application on cell phones.

The Bee-gent mobileagent framework is the best mobileagent framework I know about. Because it is 100% present pure agent.

As opposed to other systems which make only some use of agents, Bee-gent completely "Agentifies" the communication that takes place between software applications. The applications become agents, and all messages are carried by agents.

This will make it very easier for you to use mobileagent application on cell phones.

Now, let get back to the big picture of the Bee-gent framework.

Figure 12 The Bee-gent framework [7]

Now, I wish that I can transfer the mediation agent (mobileagent) on my cell phone too, like this, please take a look at figure 13.
Figure 13  The Bee-gent framework was created to run only on computer desktop or laptop, not on cell phone.

But how can I use mobileagent on my cell phone?

I will show you how, please take a look at figure 14
Figure 14  The mobileagent run on a computer desktop, in this case Bee-gent framework, I don’t need to go to my computer to use my mobileagent, but I can use my java cell phone to connect to my computer, by using the HTTP connection.

The figure 14 show an example (this example will also be implemented in the prototype of this thesis), the Seller Agent and the mobileagent run on my computer desktop, waiting for my command. The Buyer agent belong to other peoples, and it run on the laptop, also belong to other peoples.

Now, instead of going to my computer desktop to start my mobileagent, I can use my java cell phone, to connect to my computer desktop, by using an HTTP connection to my computer desktop.

If you are confusing with the symbols I use in figure 14, please take a look at the figure 15, the figure 15 will explain what are the meaning of the different symbols I use in these figures in this chapter 9.
Figure 15  The meaning of the different symbols I use on several figures in this chapter
When I send the message to tell my mobile agent to go and sell my book for me, my mobile agent will then move to the agent who want to buy my book, and they will negotiate for a deal, please, take a look at figure 16

![Bee-gent framework diagram](image)

Figure 16 My mobile agent move to the laptop belong to other peoples who want to buy my book, The Buyer agent is the agent belong to other peoples

In figure 16, you see that my mobile agent mover to the laptop, belong to other peoples, my mobilagent will then negotiate with the Buyer agent, also belong to other peoples, to try to sell my book for me.

So, In this example, you see that I want to sell my book, some other peoples want to buy a book, I have a Seller Agent and a mobile agent running on my computer desktop. Some other peoples have a Buyer Agent running on their laptop.

I user my cell phone to send my mobile agent to the laptop where the Buyer Agent is running, I don’t had to go to my computer to do that, I can use my cell phone to send messages to my Seller Agent and my mobile agent who are running on my computer. And I use the HTTP connection to do that.
Figure 17  By now, my mobileagent is on the laptop belong to other peoples, it will negotiate with the Buyer Agent, also belong to other peoples, to try to sell my book for me.

Figure 17 show you that my mobileagent are now on the laptop belong to some one else than me.

The next figure 18, show you that my mobileagent is finished with the negotiation process and it move back to my computer desktop.
Figure 18  My mobileagent now move back to my computer desktop after the negotiation process. My mobileagent will carrying the result of the deal back to me (or to my computer). This result will be save on my computer, then my Seller Agent can send me a SMS message, tell me that I can get the result, saved in my computer, on a text file for example.
Figure 19  My mobileagent is now back to my computer desktop, the task is finished.

Now, in Figure 19, my mobileagent had finished his task, he is now back to my computer.

I can now get the result my mobileagent carrying back from a place in my computer, a database or a file, in this example, I used a text file to store the result my mobileagent carrying back.

**Now let summarize the example**

As you can see, I have show you how I can use mobilagent application on my cell phone. I don’t had to use my computer to start my mobileagent application. But I can use my J2ME cell phone to start my mobilagent, by sending to my mobileagent, who is running on my computer desktop the data. The data is the only thing my mobileagent need to known to be able to start doing his task.

Because, the mobileagent is the system who can take action without or with minimal of interaction from their user or owner.
In this example, I only had to send my mobileagent the price and the command to tell him to go and try to sell some books for me.

Simple, I used the Bee-gent mobileagent framework to show you as an demonstration. But it is not mean that you must or can only use the Bee-gent mobileagent framework only.

I’m now working to create my own mobileagent framework. It will take some times to do that, It will take some times to finished the work, that is why I used the Bee-gent framework to show you the way.

If you can or known any other mobileagent framework well, you can use them instead of the Bee-gent framework. But, it should be 100 percent Agentifies (It mean all the applications are intelligent Agents).

**But Important, There is a big weakness in this method you just see**

I tell you what is that, please take a look at figure 20.
Figure 20  Some hacker had eavesdropped the data I send and get from my mobileagent who is running on my computer.

The figure 20 show you the weakness of the method I just present to you, some hacker had eavesdropped the data I send and get from my computer desktop.
When you do E-commerce, the data you send often include very sensitive information, such as credit card or other personal data.

And on the insecure Internet, the data will very easily be eavesdropped by some hacker or criminal guys with computer science expertise.

In figure 20, you see that the hacker cannot eavesdrop the data from my mobileagent directly, because my mobilagent is protected by the Bee-gent framework.

But the hacker can eavesdrop the data when I use my cell phone to transfer the data from and to my computer desktop over the Internet.

But, fortunately, I have a way to stop that.

By using the HTTPS connection (Secure Hypertext Transfer Protocol).

I can encrypt the data I send and receive from my computer, so it will be very difficult for the hacker to understand the data, even if he can eavesdrop my data, he will not understand that.

Please take a look at the next figure 21.
Figure 21  Now, I use the HTTPS connection instead of the HTTP connection, the data will be encrypted
In figure 21, the data had been encrypted, by using the HTTPS connection, so even if the hacker can get my data, he will not be able to understand that. Then I will be safe.

10 Requirement for the prototype

This chapter will present the requirement for the prototype. The scenarios for the prototype will be like the example I present in chapter 9, the propose solution to use mobilagent on cell phones.

First, I will present the functional requirement for the prototype, then the non functional requirement

10.1 Functional requirement

The prototype include the following actors:

1) The user
2) The ServletAgent
3) The BookSeller Agent
4) The BookBuyer Agent
5) The mobileagent

1) The use case for the User
Figure 22  Use case for the User

2) Use case for the ServletAgent
Figure 23   Use case for the Servlet Agent

3) Use case for the BookSeller Agent
Figure 24 Use case for the BookSeller Agent
4) Use case for the mobileagent

Figure 25  Use case for the mobileagent
5) Use case for the BookBuyer Agent

Figure 26  Use case for the BookBuyer Agent
10.2 Non Functional requirement

To be able to run the prototype, there are some software and hardware you must include.

1) You must have the Window NT platform or Window XP Platform

2) You must have the J2SE 1.4.n version installed in you computer

3) You must have The J2ME Wireless Toolkit 1.4 or the J2ME Wireless Toolkit 2.0

4) You must have Bee-gent framework package installed in your computer

5) You must have the Servlet motor Tomcat - 4.0.2 or other Tomcat Servlet motor installed in your computer

6) You must configure your Servlet Tomcat to support the HTTPS connection (You must create a Certificate keystore and then import this Certificate keystore to your J2ME Wireless Toolkit emulator).
11 Design

This chapter presents the design specification for the prototype.

I choose to use the AAIL Methodology for the design.

11.1 Step 1, Identify roles

In the requirement, there are 5 different actors. So I can easily see that I need 5 roles:

1) User
2) Servlet Agent
3) BookSeller Agent
4) BookBuyer Agent
5) Mobileagent

That means I need 5 main classes. In addition, I will also need some helper classes. But first, the main classes.

The main classes are:

- For the User role: The MIDlet class, I name this class, the class mobileAgent
- For the Servlet Agent role: I name this class, the class ServletAgent
- For the BookSeller Agent: class BookSellerAgent,
- For the BookBuyer Agent: class BookBuyerAgent.
- For the Mobileagent: class MobileAgent.

Summary, I have now the following main classes:

1) mobileAgent class
2) ServletAgent class
3) BookBuyerAgent class
4) BookSellerAgent class
5) MobileAgent class

But I still need some helper classes, the ServletAgent need some helper class. The BookSeller Agent need some helper class. The BookBuyerAgent need some helper class.

The class hierarchy are as follow:
The big picture, relation of all main classes:

Figure 27  The relation of all main classes

Next figure, Figure 28 show the detail of the Midlet class, the mobileAgent class, the user use this MIDlet to connect to the MobileAgent class
Figure 28 The mobileAgent class, this is the MIDlet class

Next Figure 29 show the big picture of the ServletAgent class, the ServletAgent class have 2 help class, the getResult class and the Sender class. ServletAgent need these help classes to be able to send and get and respond message and encrypt and decrypt message.
Figure 29  The ServletAgent class need two help classes, getResult and Sender.

Figure 30 show the detail of the ServletAgent class

```
ServletAgent

in : ServletInputStream
din : DataInputStream
text : String
tok : StringTokenizer
seek : String
se : Sender
Re : getResult

doPost( request : HttpServletRequest , respond : HttpServletResponse ) : void
```

Figure 30  The ServletAgent class

Next figure 31 the Sender class
Figure 31  The Sender class, help the ServletAgent to encrypt and send message

Next figure 32 show you the getResult class

Figure 32  The getResult class
Next Figure 33, show you the big picture of the BookSellerAgent

Figure 33 The BookSellerAgent have two help classes,

Next, Figure 34 show the AwrIPState1 class

Figure 34 The AwrIPState1 of the BookSellerAgent
Next, Figure 35 show the AwrIPState2 class

```
AwrIPState2

xa : XmlAd
perf : String
content : String
fo : FileWriter
```

```
AwrIPState2()
action() : void
```

Figure 35 The AwrIPState2 of the ServletAgent

Next, Figure 36 show you the big picture of the MobileAgent class

```
MobileAgent
```

```
BeelIPState1  BeelIPState2  BeelIPState3  BeelIPState4  BeelIPState5  BeelIPState6
```

Figure 36 The big picture of the MobileAgent class

The MobileAgent class has 6 help classes.

Next Figure 37 show the MobileAgent class
Figure 37 The MobileAgent class

Next Figure 38 show the BeeIPState1 of the MobileAgent class

Figure 38 The first help class of the MobileAgent class

Next Figure 39 show the 2 help class of the MobileAgent class
BeeIPState2

xa : XmAdC

cnt : int

bid : Vector

perf : String

sender : String

BeeIPState2()
+ action () : void

Figure 39  The 2 help class of the MobileAgent class

Next Figure 40 , the 3 help class of the MobileAgent class

BeeIPState3

xa : XmAdC

bid : Vector

bidder : String

en : Enumeration

BeeIPState3()
+ action () : void

Figure 40  The 3 help class of the MobileAgent class

Next Figure 41 show the 4 help class of the MobileAgent class
Figure 41 The 4 help class of the MobileAgent class

Next Figure 42 show the 5 help class of the Mobileagent class

Figure 42 The 5 help class of The MobileAgent class

Next Figure 43 show the 6 help class of the Mobileagent class
Figure 43  The 6 help class of the MobileAgent class

Next Figure 44 show the big picture of the BookBuyerAgent

Figure 44  The BookBuyerAgent has 3 help classes

Next Figure 45 show the BookBuyerAgent class
**Figure 45** The BookBuyerAgent

Next Figure 46 show The first help class of the BookBuyerAgent

**Figure 46** The 1 help class of the BookBuyerAgent

Next Figure 47 show the 2 help class of the BookBuyerAgent
Figure 47  The 2 help class of the BookBuyerAgent

Next Finally, the last help class of the BookBuyerAgent

Figure 48  The 3 help class of the BookBuyerAgent
I forgot to write the `BookSellerAgent` class, here are the `BookSellerAgent` class

```java
public class BookSellerAgent {
    String s;
    BookBuyerAgent aw;
}

public BookSellerAgent(String s) {
    this.s = s;
    aw = new BookBuyerAgent();
}
```
11.2 Step2, Identify responsibilities for each roles

This step will output the goals of each roles or actor ( or agent and user ) .

1) The goals of the user are to sell book and to communicate with the MobileAgent.

2) The services which are required of the ServletAgent are as follow:

The ServletAgent must be able to get the message from the User, therefore goal number
1, get the message from the user.

The ServletAgent goal number 2 is to Identify the User (Check the password).

The ServletAgent goal number 3 is to authenticate it self for the BookSeller Agent (Send
password to the BookSeller Agent).

The ServletAgent goal number 4 is to send the message to the BookSeller Agent.

The ServletAgent goal number 5 is to get the data saved in the Result text file.

The ServletAgent goal number 6 is to send the result of the deal from the MobileAgent to
the User.

3) The goals of the BookSeller Agent are as follow:

The goal number 1 of the BookSeller Agent is to get the message from the Servlet Agent.

The goal number 2 of the BookSeller Agent is to identify the Servlet Agent (check the
password).

The goal number 3 of the BookSeller Agent is to send the message (the price) to the
MobileAgent.

The goal number 4 of the BookSeller Agent is to get the message (the result of the deal)
from the MobileAgent.

The goal number 5 of the BookSeller Agent is to save (the result of the deal) from the
MobileAgent to a text file.

4) The goals of the BookBuyer Agent are as follow:

The goal number 1 of the BookBuyer Agent is to get a message from the MobileAgent.

The goal number 2 of the BookBuyer Agent is to send a response to the MobileAgent.

The goal number 3 of the BookBuyer Agent is to offer a bid for the book to the
MobileAgent.
The goal number 4 of the BookBuyer Agent is to make a deal with the MobileAgent.

5) The goals of the MobileAgent are as follow:

The goal number 1 of the Mobile Agent is to get a message from the BookSeller Agent.

The goal number 2 of the Mobile Agent is to send a message (request, asking for a deal) to the BookBuyer Agent.

The goal number 3 of the Mobile Agent is to get a message from the BookBuyer Agent.

The goal number 4 of the Mobile Agent is to move to the BookBuyer Agent place (computer).

The goal number 5 of the Mobile Agent is to negotiate with the BookBuyer Agent.

The goal number 6 of the Mobile Agent is to make a good deal with the BookBuyer Agent.

The goal number 7 of the Mobile Agent is to move back to the BookSeller Agent place.

The goal number 8 of the Mobile Agent is to send the result of the deal to the BookSeller Agent.

11.3 Step3, Determine plans

In this step, for each goal, determine plan.

I start with the use.

1) The user main goal is to sell books.

The user plans are to authenticate to the ServletAgent and to send message to the MobileAgent through the ServletAgent,

Plan 1) the user must prove to the ServletAgent that the user is the real owner or user of the ServletAgent and not some hacker who pretend to be the user. Send the right password. The password will be encrypted and send to the Servlet Agent.

The user encrypt and send the message include the price of the book and send it to the MobileAgent. The user do that by asking the Servlet Agent to help to forward the message to the MobileAgent.

Plan 2) The user plan number two is to get the result of the deal from the MobileAgent, by the help of the ServletAgent. The user get and decrypt the message from the ServletAgent.
2) The Plans of the Servlet Agent

The Servlet Agent have 6 goals. Then he must determine plans for each goal.

Plan for goal number 1: ServletAgent listening, waiting to get the message from the user, the ServletAgent will decrypt the received message.

Plan for goal number 2: ServletAgent check the password from the user, if the password is valid, the ServletAgent will go to the next step, if not, the Servlet Agent send a respond back, telling the sender that the password is not valid. Servlet Agent will then continue to wait.

Plan for goal number 3: If the password from the user is valid, the ServletAgent will get the message from the user, if not, the ServletAgent will throw away the message.

If the password is correct, then the Servlet will authenticate itself to the BookSeller Agent, by sending a password to the BookSeller Agent, the password will be encrypted.

Plan for goal number 4: The ServletAgent will encrypted the message from the user and send that message to the BookSeller Agent.

Plan for goal number 5: Get the data in the result text file.

Plan for goal number 6: ServletAgent encrypt the data get from the result text file and send that to the user.

3) Plans of the BookSeller Agent

Plan for goal number 1: Get the message from the ServletAgent and decrypt the message.

Plan for goal number 2: Check to see if the password from the ServletAgent is valid.

Plan for goal number 3: Send the message from the ServletAgent to the MobileAgent. The message will be encrypted.

Plan for goal number 4: Get the message from the MobileAgent and decrypt the message.

Plan for goal number 5: Save the message from the MobileAgent to a text file.

4) Plans of the BookBuyer Agent

Plan for goal number 1: get and decrypt the message from the MobileAgent

Plan for goal number 2: encrypt and send the message to the MobileAgent

Plan for goal number 3: encrypt and send a bid to the MobileAgent

Plan for goal number 4: make a deal with the MobileAgent, offer a high bid for the book.
5) Plans of the Mobile Agent

Plan for goal number 1: get and decrypt the message from the BookSeller Agent

Plan for goal number 2: encrypt and send message to the BookBuyer Agent, asking for a deal.

Plan for goal number 3: get and decrypt the message from the BookBuyer Agent

Plan for goal number 4: Move to the BookBuyer Agent

Plan for goal number 5: Negotiate with BookBuyer Agent for a good deal, accept no lower bid

Plan for goal number 6: Accept only the higher enough bid from the BookBuyer Agent

Plan for goal number 7: Move back to the BookSeller Agent place

Plan for goal number 8: encrypt and send the result of the deal to the BookSeller Agent.

11.4 Step 4, Determine belief structure of the agents

1) The User’s beliefs:

\[
goal \text{(The user, send the password to the ServletAgent)} \land\ goal \text{(The user, send the message to the MobileAgent by the help of the ServletAgent)} \land\ goal \text{(The user, get the result of the deal from the MobileAgent by the help of the ServletAgent)} \land \\
be\ (\text{The user, encrypt and send password to ServletAgent}) \land be\ (\text{The user, I-can(The ServletAgent, get and decrypt the password)}) \land be\ (\text{The user, encrypt and send the message to the MobileAgent by the help of the ServletAgent}) \land \\
be\ (\text{The user, I-can(The ServletAgent, get and decrypt the message)}) \land \\
be\ (\text{The user, encrypt and send the command to get the result of the deal from the MobileAgent by the help of the ServletAgent}) \land \\
be\ (\text{The user, I-can(The ServletAgent, get and decrypt the command and send back the result from the MobileAgent})})
\]

2) **ServletAgent’s beliefs:**

\[
goal \text{(ServletAgent, get the password from the user)} \land \\
goal \text{(ServletAgent, check the password)} \land \\
goal \text{(ServletAgent, send password to BookSeller Agent)} \land
\]
goal (ServletAgent, send message to BookSeller Agent) \land

green goal (ServletAgent, get the result of the deal from the MobileAgent) \land

green goal (ServletAgent, send the result to the user) \land

belief (ServletAgent, get password) \land belief (ServletAgent, I-can(The User, encrypt and send password)) \land belief (ServletAgent, decrypt password) \land

belief (ServletAgent, check password) \land

belief (ServletAgent, get message) \land belief (ServletAgent, I-can(The User, encrypt and send message)) \land

belief (ServletAgent, decrypt message) \land

belief (ServletAgent, encrypt password) \land belief (ServletAgent, send password) \land

belief (ServletAgent, I-can(BookSeller Agent, get and decrypt password)) \land

belief (ServletAgent, encrypt message) \land

belief (ServletAgent, send message) \land

belief (ServletAgent, I-can(BookSeller Agent, get and decrypt message)) \land

belief (ServletAgent, get the result of the deal) \land

belief (ServletAgent, I-can(BookSeller Agent, have the result of the deal)) \land

belief (ServletAgent, encrypt the result) \land belief (ServletAgent, send result) \land

belief (ServletAgent, I-can(The User, get and decrypt the result of the deal))

3) **BookSeller Agent’s beliefs**:

goal (BookSeller Agent, get the password from the Servlet Agent) \land

goal (BookSeller Agent, check the password) \land

goal (BookSeller Agent, get the message from the Servlet Agent) \land

goal (BookSeller Agent, send message to the Mobile Agent) \land

goal (BookSeller Agent, get the message from the MobileAgent) \land

goal (BookSeller Agent, save result of the deal to a text file) \land

belief (BookSeller Agent, get the password from the Servlet Agent) \land

belief (BookSeller Agent, I-can( ServletAgent, encrypt and send the password)) \land
bel (BookSeller Agent, decrypt the password) \&
bel (BookSeller Agent, check password) \&
bel (BookSeller Agent, get the message from the ServletAgent) \&
bel (BookSeller Agent, I-can(ServletAgent, encrypt and send the message)) \&
bel (BookSeller Agent, decrypt the message) \&
bel (BookSeller Agent, encrypt message) \&
bel (BookSeller Agent, send the message to the MobileAgent) \&
bel (BookSeller Agent, I-can(MobileAgent, get and decrypt the message)) \&
bel (BookSeller Agent, get and decrypt the result of the deal from the MobileAgent) \&
bel (BookSeller Agent, I-can(MobileAgent, encrypt and send the result of the deal)) \&
bel (BookSeller Agent, save the result of the deal to a text file)

3) **BookBuyer Agent’s beliefs**:

goal (BookBuyer Agent, get message from the MobileAgent) \& goal (BookBuyer Agent, send respond to the MobileAgent) \&
goal (BookBuyer Agent, offer a bid to the MobileAgent) \& goal (BookBuyer Agent, make a deal) \&
bel (BookBuyer Agent, get the message from the MobileAgent) \&
bel (BookBuyer Agent, I-can(MobileAgent, encrypt and send the message)) \&
bel (BookBuyer Agent, decrypt the message from the MobileAgent) \&
bel (BookBuyer Agent, encrypt message) \& bel (BookBuyer Agent, send message to the MobileAgent) \&
bel (BookBuyer Agent, offer a bid to the MobileAgent) \&
bel (BookBuyer Agent, I-can(MobileAgent, get the bid)) \&
bel (BookBuyer Agent, make a deal)

4) **Mobile Agent’s beliefs**:

goal (Mobile Agent, get message from BookSeller Agent) \& goal (Mobile Agent, send message to the BookBuyer Agent) \&
goal ( Mobile Agent , move to BookBuyer Agent place ) ∧ goal ( Mobile Agent , make a good deal with the BookBuyer Agent ) ∧ goal ( Mobile Agent , move back home ) ∧ goal ( send the result of the deal to the BookSeller Agent ) ∧
bel ( Mobile Agent , get the message from the BookSeller Agent ) ∧
bel ( Mobile Agent , I-can ( BookSeller Agent , encrypt and send the message ) ) ∧
bel ( Mobile Agent , decrypt the message ) ∧
bel ( Mobile Agent , encrypt message ) ∧
bel ( Mobile Agent , send message to the BookBuyer Agent ) ∧
bel ( Mobile Agent , I-can ( BookBuyer Agent , get and decrypt the message ) ) ∧
bel ( Mobile Agent , get the respond from the BookBuyer Agent ) ∧
bel ( Mobile Agent , I-can ( BookBuyer Agent , encrypt and send the respond ) ) ∧
bel ( Mobile Agent , move to the BookBuyer Agent place ) ∧
bel ( Mobile Agent , accept no lower bid from the bookBuyer Agent ) ∧
bel ( Mobile Agent , move back to the BookSeller Agent place ) ∧
bel ( Mobile Agent , encrypt and send the result of the deal to the BookSeller Agent ) ∧
bel ( Mobile Agent , I-can ( BookSeller Agent , get and decrypt the result of the deal ) )
12 Implementation

This chapter will describe the implementation process. The implementation will be according to the requirement and the design present in chapter 10 and 11.

The source Java codes for the prototype are in the Appendix A.

The class files and also the Java source files are in the CD-ROM of this thesis.

Some very small change had to be make in the implementation because of the small technical detail. I will describe that right next.

There will not be any change in the functional requirement described in chapter 10, all the functional requirement in chapter 10 will be implemented. The change I mean are the small technical detail. Very small technical detail.

In the Bee-gent agent framework, there are some small change from the design.

12.1 Som small change from the design

The Bee-gent mobileagent framework had it owner technical specification. When you use Bee-gent framework, you must follow these specification. And there are:

The class name of the Agent Wrapper must usually be AW, for eksempel AW1 and AW2 and AWn, etc, .... In Bee-gent framework. All class has it own name and password.

It mean you can name the Java class AW1.java but you use the method:

setName(String name) to set the name for the agent and the method:

setPassword(String password) to give the agent its own password.

For example:

For the class BookSellerAgent and the class BookBuyerAgent, The BookSellerAgent and the BookBuyerAgent are Agent Wrapper.

Then I had to set the Java class name be AW1.java for the BookSellerAgent, and then I use the method setName(String name) and the method setPassword(String password) to set the name and the password for the BookSellerAgent.

Like this: setName(“BookSeller”) and setPassword(“NTNU”)

So the java class name for the BookSellerAgent will be:
AW1.java and not BookSellerAgent.java.

If you name the Java file BookSellerAgent.java, you will be able to compile, but when you run the class, you will often get the trouble, and that is: the mobileagent will often fail to migrate to the Agent Wrapper place. The reason is the security issue. Bee-gent framework had their own security specification. If the class name for the Agent Wrapper is different than AW.n, you will often get the trouble when you run the class. And the migration process for the mobileagent will often fail.

And similar for the BookBuyerAgent class.

The name for the Java BookBuyerAgent class will be AW2.java and not BookBuyerAgent.java but the name for the agent will still be BookBuyer.

You can name the java class BookBuyerAgent.java but you will often get the trouble when you run the class. You will not get problem when you compile the class but trouble when you run it. There are because of the security concern of Bee-gent. You must use the standard AW name for Agent Wrapper classes.

The same is for the mobileagent class.

In the Design I call the mobileagent class for MobileAgent. Now I must use the standard name BEE1 for the mobileagent.

So the Java name for the mobileagent class is BEE1.java and not MobileAgent.java, but I can still set the name for the MobileAgent class as “mobileagent” by using the method CreateBee(String classname, String agentname).

Short Summary:

In the design I name the BookSellerAgent class: BookSellerAgent.java, here in the implementation, I must change that to AW1.java, but the name can still be BookSeller by using the method setName(String agentname).

Similar for the BookBuyerAgent class: In the design BookBuyerAgent.java, here change to AW2.java, and the name can still be BookBuyer.

Similar for the MobileAgent class: In the design (Chapter 11), MobileAgent.java, here, change to BEE1.java, but the name can still be mobileagent, by using the method createBee(String classname, String agentname).

12.2 The technical detail of the implementation process

The process you see in figure 21 in chapter 9 is the big picture. A lot of thing are going on below that big surface in figure 21.

The technical details are wide and extensive.

I will now start by describe the technical detail of the MIDlet and the Servlet

For the network security issue I mention in chapter 8. There are many way to encrypt and decrypt message and provide the authentication process. The best way is to use the
HTTPS connection (Secure HyperText Transfer Protocol). If you use the HTTPS connection, the communication between the J2ME cell phone (The User) and the ServletAgent will be done by using the TLS Protocol (Transport Layer Security Protocol). This will be show as the next figure 49.

![Diagram](image)

**J2ME Cell phone**

**ServletAgent**

---

**Send a list of cipher suite**

---

**Send back the choose cipher suite, and an X.509v3 certificate with public RSA key**

---

**Scrubalize the certificate, check that it's valid, then send the master secret, encrypted with the public key from ServletAgent**

---

**The sending data from now on will be encrypted with the session key**

---

**Decrypt the master secret by using the private key, generate the session key**

---

**The sending data from now on will be encrypted with the session key**

---

Figure 49  The Handshake between the J2ME cell phone and the ServletAgent

Figure 49 show the communication process between the J2ME cell phone and the ServletAgent if you choose to use the HTTPS connection.
you must configure the Servlet motor to support HTTPS connection, and then create a certificate keystore, here I used the Tomcat-4.0.2 Servlet motor. Then I import the certificate keystore to the J2ME Wireless Toolkit emulator. I used here the 1.4 version of the J2ME emulator, the 2.0 version didn’t work very well. I got often deadlock when I used the J2ME Wireless Toolkit 2.0. It work much better with the J2ME Wireless Toolkit 1.4 version.

If you don’t want to use the HTTPS connection, you can still provide encryption and decryption of sending data by using the Bouncy castle Package.

But the HTTPS connection is the best way to do. You should use the HTTPS connection. Because when you use the Bouncy Castle Package, you must have very large knowledge about encryption algorithms to be able to create the good solution. If you don’t have that, you will create the average quality encryption method and you will never be sure that you are secure enough.

The question you must always ask your self when you use the Bouncy castle Package is:

Is my method advance enough? Is that good enough? And you can never be sure about that.

When you use the HTTPS connection, you can absolute be sure that the HTTPS connection is good enough for you to protect your data.

The TLS protocol and the SSL protocol are the well tested and recognized protocols. You should always use these as the first choice.

Now, to the communication process between the ServletAgent and the Agent Wrapper BookSellerAgent. Here, the SSL protocol will be used to make the communication secure.

The handshake between the ServletAgent and the BookSellerAgent will be almost similar as the handshake between the J2ME cellphone (MIDlet) and the ServletAgent, see Figure 49.

The message sending from the J2ME MIDlet to the mobileagent can be similar to an SMS message. I had implemented in a way that make it easier the way the user authenticate to the ServletAgent. The password will be send with the message, as a form of an SMS message.

The Password here is set as the two words: sell-book or SELL-BOOK. The Servlet check this password. Then Servlet send it own password to the BookSellerAgent. The password of the Servlet is set as the word: price or PRICE.

The sending message the user can send to the mobilagent can be like this:

```
sell-book J2ME Programming API price 70 dollar or
SELL-BOOK J2ME Programming API PRICE 70 dollar
```

First the password, then the book name, then the price I want for the book. The mobileagent will get this message through the ServletAgent and again through the BookSellerAgent. Then the mobileagent will use the price I want as an issue for the
negotiation with the BookBuyerAgent. The mobileagent will never accept a bid lower than the price I set. Here in this example: 70 dollar.

The user must include the price, the word: price or PRICE will be used as a password for the ServletAgent to authenticate itself to the BookSellerAgent.

To get the result of the deal from the mobilagent agent, send the following command: RESULT or result. It will also be used as a password to authenticate to the ServletAgent.

You must start up the Tomcat-Servlet motor first.

I had implemented that so I can only sell one book at a time in this prototype.

I had implemented an easy to understand prototype to show you my solution.

The connection from the J2ME Cell phone to the mobileagent are as follow, please take a look at next figure.
Figure 50  The connection from the J2ME cellphone to mobilagent go through the ServletAgent, and through the BookSellerAgent.

The encryption and decryption and authentication of data between BookSellerAgent and MobileAgent and BookBuyerAgent are provide by the Bee-gent framework.

The Bee-gent framework provide these functionalities for you when you use Bee-gent framework.
Communication between the J2ME Cellphone and the ServletAgent is running on the TLS (Transport Layer Security) protocol or SSL (Secure Socket Layer) protocol. Between the ServletAgent and the BookSeller Agent. It run on the SSL (Secure Socket Layer) protocol even that it is best to use the HTTPS connection than to use the Bouncy Castle Package.

There are a lot of works you must done, to be able to run the prototype I created.

The most difficult thing are the configuration of the server. You must configure the server (both the Servlet and the other server) to support the HTTPS connection. It mean you must import and create the certificates and this process can be very difficult if you don’t known how to do thing in the right order.

I will describe in detail, step by step, what you must do to be able to configure the servers to support HTTPS connection.

You must first install the Tomcat, the J2SE and the J2ME Wireless Toolkit at the right place in you computer. Otherwise, it will be very difficult for you to run the prototype I created.

- The Tomcat servlet must be installed in C:\Tomcat
- The J2ME Wireless Toolkit must be installed in C:\WTK, I used the 1.4 version. It work better than the 2.0 version.
- The J2SE must be in C:\j2sdk 1.4.1 or C:\Program Files\j2sdk1.4.1
- The Bee-gent package must be in C:\BO

1) First the servlet Tomcat. You must configure the servlet Tomcat to support HTTPS connection. This process is not very difficult but there are some works to do.

You must configure the server.xml file to support HTTPS connection. Go to the directory C:\Tomcat\conf, you will find the server.xml file there. Open this file and find this text.

```xml
<Connector className="org.apache.catalina.connector.http.HttpConnector"
    port="8443" minProcessors="5" maxProcessors="75"
    enableLookups="true"
    acceptCount="10" debug="0" scheme="https" secure="true">
    <Factory className="org.apache.catalina.net.SSLServerSocketFactory"
        clientAuth="false" protocol="TLS" />
</Connector>
```

Remove the comments mark from this, then save the file.

Then you must use the keytool to create a certificate keystore.

The certificate will be used for the handshake process between the Cell phone (J2ME MIDlet) and the ServletAgent (Servlet Tomcat).
Use this command from the directory  

```bash
keytool -genkey -alias tomcat -keyalg RAS
```

and then enter the password and the other information. When you are finishing, the keytool will create the new file, .keystore in you home directory.

Then you must test if the Tomcat do work (support the HTTPS connection).

Open the ordinary Web browser, write:

```plaintext
https:\\127.0.0.1\ the name of some text file you make
```

The text file must be store in this directory:  

```plaintext
C:\tomcat\webapps\root
```

If the Tomcat work as it should, you shall see the text file you make. Otherwise, you must go back and start all over again.

**The Tomcat servlet motor must support the HTTPS connection. You must configure the Tomcat servlet motor to do that.**

2) Now the J2ME Wireless Toolkit. You must import the certificate you just create to the keystore of the J2ME Wireless Toolkit.

Go to  

```plaintext
C:\WTK1.4
```

and write this command:

```bash
java -jar bin\MEKeyTool.jar -import –alias tomcat –keystore “C:\documents and settings\your usename .keystore” –storepass changeit
```

Now, if all went as it should, you should be able to use the HTTPS connection from the MIDlets to Servlet. If not, you must go back and start it all over again from step 1) as I described.

3) Now, this step is the most difficult one.

Because I used the SSL connection to make the communication between the ServletAgent and the BookSeller Agent secure, you must have the certificate, it will be used for the handshake process between the ServletAgent and the BookSeller Agent.

I had create a certificate, the name of this certificate is **skey**. You must have this file in the same directory of the BookSeller Agent file (AW1.java, in the directory

```plaintext
C:\BO\AgentWrapper1
```

) to be able to start up the BookSeller Agent.

Then you must create the same trust certificate for the client side (it mean the SerletAgent side), I had done this, I had created the trust certificate for the ServletAgent (client side), the name of this trusted certificate is the file: **trustcertificate**.

You must also had to copy this file (**trustcertificate**) in the directory of the Servlet. In this directory:
Then you must also import this file (trustcertificate) into the security directory of the J2SE 1.4, do this:

Go to:  C:\j2sdk1.4.1\jre\lib\security

You will find a file name cacerts here. This is the trust store

Important, here you must not make any mistake, or you must reinstalled the J2sdk1.4.1 all over again.

The trick here is to move the default trust store (the name of this is: cacerts) of j2sdk1.4.1 to a another directory, don’t delete it.

Then, move the file: trustcertificate to the directory:

C:\j2sdk1.4.1\jre\lib\security

Then rename this file: trustcertificate TO cacerts.

Then the trust manager in j2sdk 1.4.1 will use this trust store (the trustcertificate you created, here, I created it).

If all went well, Then you will be able to use the secure connection between the ServletAgent and the BookSeller Agent.

If not, go back and start all again.

When you are finishing running the prototype, you must remove the cacerts (trustcertificate) file you created from the directory

C:\j2sdk1.4.1\jre\lib\security

And reinstalled the real cacerts file back to
The C:\j2sdk1.4.1\jre\lib\security directory

That is why you must not delete the real (original) cacerts file

4) Now the Bee-gent framework, Just go to the C:\BO\Agentwrapper1 and use the bat file to start up the BookSeller Agent. Then go to C:\BO\Agentwrapper2 and use the bat file to start up the BookBuyer Agent.

Then you are ready to run the prototype.
The `skey` file must be in the `C:\BO\AgentWrapper1` directory.

The `skey`, `trustcertificate` are in the CD-ROM of this thesis.

Here, I chose to use the certificate I created, because it is free. But you can also buy the Certificates from the Certificate Authorities (CAs).

12.3 More description of the technical details of the implementation process

In this section, I will give you much more detail description of the technical detail of the implementation process of my solution. As I said earlier in this chapter, the technical detail are wide and extensive.

The figure 21 in chapter 9 give you just the big picture, a lot of thing are happening under that big surface you saw in Figure 21 in chapter 9.

In the requirement in chapter 10 and in the design in chapter 11, I mentioned the ServletAgent. You may ask: Where is the ServletAgent? You cannot see the ServletAgent in Figure 21 in chapter 9. So where is the ServletAgent?

I will show you where the ServletAgent is, please, take a look at the next figure 51.
Figure 51  The Servlet Agent is running on the same computer with the BookSellerAgent.

Figure 51  show you that the ServletAgent is running on the same computer desktop as the BookSellerAgent. **But the ServletAgent is not a part of the Bee-gent framework, the ServletAgent is running outside the Bee-gent framework.**

That is why the communication line between the ServletAgent and the BookSellerAgent is vulnerable. Because the ServletAgent is not protected by the Bee-gent framework, because the ServletAgent is not a part of the Bee-gent framework.

The Bee-gent mobile agent framework use digital fingerprint authentication and secret key encryption for the security of the agents running on Bee-gent framework. Digital fingerprint is almost like Message Digest.

Digital fingerprint is presented as a long, random looking string of letters and numbers. (with punctuation marks for readability).

The ServletAgent task is to forward the message from the J2ME cellphone (the user) to the Mobile Agent. And is done as following, please take a look at next figure 52.
Figure 52  The ServletAgent task is to forward the message from the J2ME cellphone (the user) to the MobileAgent. And is done as you see in here in Figure 52, it go through BookSeller Agent, and BookSellerAgent forward the message to the MobileAgent.

The ServletAgent is running on the same computer as the BookSellerAgent but it run outside the Bee-gent framework. The BookSellerAgent, MobileAgent, And BookBuyerAgent all run on (inside) Bee-gent framework. They are protected by the Bee-gent framework. The communication lines between these agents (BookSellerAgent, MobileAgent, BookBuyerAgents) are not vulnerable. But the communication lines between the J2ME cell phone and the ServletAgent, and between the ServletAgent and the BookSellerAgent are vulnerable. Please take a look at the next figure 53.
Figure 53  The vulnerable communication lines

Here, in figure 53 you see that the hacker can attack the communication line between the J2ME cell phone and the ServletAgent (If I use only the HTTP connection), and the hacker can also attack the communication line between the ServletAgent and the BookSellerAgent (If I only use the Client Socket connection). The hacker can eavesdropper the data and understand that, please see next figure 54.
The hacker can eavesdrop and understand the data.

The Hacker cannot attack the communication lines between the BookSellerAgent, MobileAgent and BookBuyerAgent because they run inside Bee-gent framework and they are protected by the Bee-gent framework.

So, I must find some way to protect the communication lines between the user (J2ME cellphone) and the ServletAgent, and between the ServletAgent and the BookSellerAgent.
I do this by using the TLS (Transport Layer Security) Protocol and SSL (Secure Socket Layer) protocol. Please take the look at the next figure 55.

Figure 55 The communication lines are now safe. Because I now use the TLS and the SSL protocol.
The hacker will now have problem to understand the data he eavesdropper, because the data had been encrypted, and the authentication process will also be proceeding here.

The TLS (Transport Layer Security) protocol is an update version of the SSL (Secure Socket Layer) protocol version 3. The different of the two protocol is not much, the TLS protocol will replace the SSL protocol in the future.

And I can easily use the TLS and the SSL protocol by using the HTTPS connection and the SSLSocket connection. The HTTPS connection is the HTTP running on TLS or SSL protocol. The SSLSocket Client connection using SSL protocol.

Please take a look at the next figure 56
Figure 56 I can use the TLS and SSL protocol easily by using the HTTPS connection and the SSLSocket Client connection.
You may ask the question: why I did not draw the ServletAgent before in the Figures I present in the chapter 9?. The answer is that the Servlet Agent is optional. You don’t have to use the ServletAgent. Because, in Bee-gent framework, the Agent Wrapper can be invoked as the servlet, that mean the Agent Wrapper can be running up as the Servlet. And the BookSeller Agent is the Agent Wrapper in Bee-gent. Therefore the BookSellerAgent can be invoked as a servlet, please take a look at next figure 57

![Bee-gent framework](image)

**J2ME Cellphone**

**HTTPS**

**Servlet**

**BookSeller Agent**

**Mobile Agent**

---

This symbol mean that the Agent Wrapper BookSeller Agent can be invoked as a Servlet (Run up as a Servlet)

---

**Figure 57** The Agent Wrapper BookSeller Agent can be invoked instead as a servlet

In figure 57, you see that if the BookSeller Agent be invoked as the servlet, then I don’t have to use the Servlet Agent, that mean I don’t have to create the ServletAgent. But I can connect to my mobileagent through the BookSellerAgent directly from my java cell phone, I don’t have to go through the Servlet Agent. That mean I don’t have to create the ServletAgent.
But this method is more difficult to implement. It is more difficult to get connect to AgentWrapper BookSeller Agent if it will be invoked as a servlet. It is easier to get connect to the Agent Wrapper BookSeller if it be invoked as a Server socket.

The different between the Servlet and the Server socket are as follow:

![Server Socket Running](image)

Figure 58 The server socket is up running, you can see it

But if it is the Servlet, you will not see anything, the servlet will be invoked when the servlet motor is up running, for example Tomcat-4.0.2.

But it is easier to let the BookSeller Agent be invoked as the server socket, then I need the Servlet Agent. That is why I created the Servlet Agent. And that is why I didn’t draw the ServletAgent in the figures in chapter 9. Because the ServletAgent is optional.

Here in the prototype of this thesis, I choose to created the ServletAgent because it is much easier to implement the solution this way. I can use the HTTPS connection easily from the J2ME cellphone to the ServletAgent and then the SSL client Socket connection easily from the Servlet Agent to the BookSeller Agent.
Now, let me sum up the technical details in next figure 59

Figure 59 The communication between J2ME cell phone and the mobile agent

Figure 59 show you that I can use my J2ME cell phone to connect to my mobile agent, the communication go through the ServletAgent, and the BookSellerAgent.

The message I send from my cell phone will be encrypted, When the Servlet Agent get my message, he will decrypt that and check the password, if the password is correct, the ServletAgent will encrypt the message and forward that to the BookSeller Agent. The BookSellerAgent decrypt the message, if the password is correct the BookSellerAgent will again encrypt the message and send that to the Mobile Agent.
Finally, my Mobile Agent will get and decrypt the message. The message I send from my java cell phone will go indirect to my mobile agent as you see in next Figure 60.

Figure 60  The message from cell phone will be transfer encrypted all the way to the Mobile agent

When the Mobile Agent get the message, He will start to do his task, I don’t had to be continuous connected to the network or to my mobileagent. After the message had been transfer, I can disconnect and just wait for the result from my mobile agent or I can go and do some other work.
Figure 61  After that the message had been transfer, I can disconnect and go and do something else. My mobile agent will now start to go and do the task I want him to do.

By now I have show you in detail my solution to use mobile agent on java cell phone. And this solution are a secure solution. I can feel safe when I use this solution because the security issue had been included.

You may ask the question why there are so many agents?, ServletAgent, BookSeller Agent and Mobile Agent, etc, why can it just be one single Agent?, and that mean the Mobile Agent?, as in the next figure 62.
Figure 62 The alternative solution, only the Mobile Agent on each computer (desktop or laptop).

Figure 62 show you the alternative solution, only the Mobile Agent on each host. Of course, this will be technical feasible. But it is easier to have many different agents. It is easier to implement the solution in the way I just show in this section 12.3.

When you look at Figure 62, you think that it is much easier that way (it mean have only the Mobile Agent and not the other Agents) but actually it is much more difficult to implement the solution that way.

I can of course create just one single agent, and that is the MobileAgent as you see in Figure 62, but this will be more difficult to implement the solution that way.

You may ask why it is more difficult? I will show you why.

Because then the Mobile Agent will have more tasks to do, he must check the password, then he must migrate to other place, and negotiate, and then carrying back the result and then send back to the user, etc and etc ....... .

The Mobile Agent will then have to do everything by itself, He will not have any other Agents around to help him. The Mobile Agent will then have much more states.
It will be more difficult to create that kind of Mobile Agent. Because the Mobile Agent will be very large in size (measured in byte) because the Mobile Agent must have many different states, that mean many classes, and it will result in that the size of the Mobile Agent will be very large. The Mobile Agent will then be many MB large. It will make it more difficult for the Mobile Agent to migrate to other host.

The larger the size of the Mobile Agent are, the more difficult will it be for the Mobile Agent to migrate to other hosts.

In Bee-gent, the limitation of the Mobile Agent is set to 32Kbyte.

If the Mobile Agent is larger than 32Kbyte in size in Bee-gent framework, the Mobile Agent will not be able to migrate to other host.

That is why I create the Servlet Agent and the BookSeller Agent, so it will be easier for my Mobile Agent to get the message I send to it from my cell phone. The ServletAgent and the BookSeller Agent will help my mobile agent to get the messages I send and they will also identify me. My mobile agent didn't had to do that task. The ServletAgent and the BookSellerAgent will do that for him.

And the Mobile Agent will not be to large in size of byte, because the Mobile Agent will have less states, because the Mobile Agent will then have less tasks to do.

The less the tasks the Mobile Agent have to do, the less the states the Mobile Agent will have, that mean, the size of the Mobile Agent will be smaller (measured in byte), and the Mobile Agent will then migrate to other host more easier.

That is why I created so many agents, ServletAgent, BookSellerAgent and MobileAgent.
Figure 63  The dream solution , the Mobile Agent is running on all kind of devices .

It is easier to have many different agent to share the tasks . In my prototype , the Mobile Agent have the ServletAgent and the BookSellerAgent to help him . So he will have less tasks to do . He didn’t had to check the password from me , The ServletAgent do that for him . He didn’t had to cheat the password from the ServletAgent either , the BookSellerAgent do that for him . He only need to get the price and then go to do his tasks, that mean negotiate to try to make a good deal with the BookBuyerAgent .

But of course ,There will be absolutely feasible to have just one Mobile Agent and no other helping Agents . But it is more difficult to create that kind of solution .

It will be more difficult to implement that way .

By now I had show you my solution in detail to let you use mobile agent on cell phones.

My solution is easier and secure .

I choose to use the Bee-gent framework to show you my solution because the Bee-gent agent framework is 100 percent Agentifies . It mean all the applications are Agents .

When you use Mobile Agent from your cell phones or on your cell phones , you should use the Agent framework who is 100 percent Agentifies as the Bee-gent Agent framework ( It mean all applications are Agents ) , because it is much easier for you to communicate with your Mobile Agent from your cell phones if there are other intelligent Agents around to help you deliver the messages to your Mobile Agent .
Now let me short summarize:

The Bee-gent framework was created by TOSHIBA JAPAN, and not by me.

The Bee-gent mobile agent framework was created to run only on computer desktop or laptop and not on cell phones.

But as you just saw, I had show you that I can use my solution to make that feasible for you to use mobile agent on cell phones which support J2ME platform.

And I used the Bee-gent mobile agent framework as a help tool to present to you my solution. Because I didn’t had time to create my own mobile agent framework, it will take many years to create one mobile agent framework if you are alone. And I am alone. I’m working alone with my thesis.

I used the Bee-gent mobile agent framework because it is 100 percent Agentifies. All applications in Bee-gent framework are intelligent Agents.

And if you are using mobile agent on your cell phones (or I should say: from your cell phones), you should use the mobile agent framework who is 100 percent Agentifies (As Bee-gent framework). Because it will be much easier for you to communicate with your mobile agent from your cell phones, because the other intelligent Agents will be there to help you to deliver and get the messages (The messages you send to your mobile agent or get from your mobile agent).

And because the cell phone have much less memory resources than the computer desktop or laptop, you shall let the mobile agent run on the computer desktop or lap top and use the HTTPS connection to send messages from your cell phone to your mobile agents who is running on your computer desktop or laptop (As I had just show you).

And my solution is the secure solution, you can absolutely feel safe when you use my solution.

The security issue had been included in my solution I just show you in this section 12.3.

And this mean that the communication processes between your cell phones and your mobile agents must go through the authentication and encryption process. By using the TLS (Transport Layer Security) and
the SSL (Secure Socket Layer) protocol, (please take a look at figure 55 again).

And you should create many helping Agents who will help your mobile agent to do the tasks.

Because 1) It will be easier for your mobile agent to migrate to other hosts because the size of your mobile agent (measure in byte) will be much smaller.

Because if you create many other helping intelligent Agents who goals are to help your mobile agent to do his tasks, then your mobile agent will have much less tasks to do, it will result in that your mobile agent will have less states, and it mean, less classes, and the size of your mobile agent, measure in byte will be much smaller, then your mobile agent will be able to migrate to other hosts very easily.

If the size of your mobile agent is to large, it will be more difficult for your mobile agent to migrate to other hosts.

And I had just show you that, in Bee-gent mobile agent framework, the size of the mobile agent must not be larger than 32 Kbyte.

2) It will also be easier for you to implement the technical solution if you create more intelligent Agents, I just show you that it is much easier to implement the solution when I create the helping ServletAgent.

Because I can then easily use the TLS (Transport Layer Security) protocol and the SSL (Secure Socket Layer) protocol, by using the HTTPS connection and the SSLSocket client connection (Please take a look at figure 56 again).

If I don’t create the helping ServletAgent, it will be more technical difficult to implement the technical solution. It will still be technical feasible without the helping ServletAgent, but it will be more difficult to implement.

And last but not least: I didn’t mean that you can only use the Bee-gent mobile agent framework to be able to use my solution. You can use other mobile agent framework too But they should be 100 percent Agentifies (It mean all applications are intelligent Agents).

You can also create your own mobile agent framework if you have time. And it should be 100 percent Agentifies as Bee-gent framework.
You might ask the question: what about the security of the Bee-gent mobile agent framework?

The Bee-gent mobile agent framework use digital fingerprint and secret key for the authentication and encryption process, to protect the Agents.

You might ask a another question: What about the security again malicious agents or computer virus?.

In my opinion, it will be a little more complex to protect your mobile agent again computer virus or from malicious agents when you create a mobile agent framework but it will be able to protect your agents again virus or malicious agents.

It can be done like this: For example, you can create Agents who are police Agents. These police Agents will patrol the host they are running on, each host will have one or many police Agents. The policy Agents will check identity to all the Agents running in that host, if there exist any agents who can not identify itself, the police agents will kill that agents.

In the Aglet mobile agent framework, it have police Agents similar to I just describe above, but these Agents in Aglet framework have goals to distribute host memory resources only. Not about the security.

As of this writing, I cannot find any information about the security to protect mobile agents again virus or malicious agents in Bee-gent framework. But, I know that the next version of Bee-gent will have the mechanism call: Access control of mobile agents to platforms. It will be included to the next version of Bee-gent package when it be released.
13 Testing

This chapter will describe the testing process. The testing will include the test of all the functional requirement, and the test of the network security.

13.1 Test of the functional requirement

I had use many values to test, and the test had been processed as follow:

1) Test of the authentication process. Test to see if the authentication process is working.

2) Test of the functional requirement. Test to see if the functional requirement is working.

First test of the authentication process:

Here, The wrong and correct password will be send to see how the ServletAgent will react.

Here are the result of the test:

<table>
<thead>
<tr>
<th>The sending password</th>
<th>The respond</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bush</td>
<td>Bad Request</td>
</tr>
<tr>
<td>Saddam</td>
<td>Bad Request</td>
</tr>
<tr>
<td>Clinton</td>
<td>Bad Request</td>
</tr>
<tr>
<td>sell-book</td>
<td>Good ,</td>
</tr>
<tr>
<td>Sell-bok</td>
<td>Bad Request</td>
</tr>
<tr>
<td>Sell-bo</td>
<td>Bad Request</td>
</tr>
<tr>
<td>SELL-BOOK</td>
<td>Good</td>
</tr>
<tr>
<td>Byebye</td>
<td>Bad Request</td>
</tr>
<tr>
<td>Whitehouse</td>
<td>Bad Request</td>
</tr>
</tbody>
</table>

Now test of the functional requirement. To see if the reaction of the mobileagent is as expect.

As mentioned in the implementation chapter. The password must be send with the price, The message the user send to the mobiagent must be like an SMS sentence.

First the password, then the name of the book, then the price the user want for the book.

The user must include the price, the word: price or PRICE will be used as an password for the ServletAgent to authenticate it self to the BookSellerAgent.
Here are the test for the functional of the prototype:

The messages send are:

From the user

1) sell-book Java Programming Language price 160 dollar

   The result: Good, as expected

ServletAgent react as expected. ServletAgent accept the message, the password here is correct (sell-book). The BookSellerAgent accept the message from the ServletAgent because the password from the ServletAgent is correct (price), the mobileagent get the message.

The result of the deal here is 927165701, it is higher than the price set by the user, 160 dollar. That mean the mobileagent react as expected (he will not accept any bid lower than the price set by the user).

2) sell-book Java Programming Language Prise 80 dollar

   The result: Good, as expected

ServletAgent react as expected. ServletAgent accept the message, the password here is correct. The BookSellerAgent didn’t accept the message from the ServletAgent, the password is wrong, the password Prise is wrong (the word price is used as the password), the mobileagent didn’t get the message.

3) sell-book Java Programming Language price 20 dollar

   The result: Good, as expected

ServletAgent react as expected. ServletAgent accept the message, the password here is correct (sell-book). The BookSellerAgent accept the message from the ServletAgent because the password from the ServletAgent is correct (price), the mobileagent get the message.

The result of the deal here is 9895646 dollar, higher than the price set by the user, 20 dollar. That mean the mobileagent react as expected.

4) sell-book Java Programming Language price 5000 dollar

   The result: Good, as expected

ServletAgent react as expected. ServletAgent accept the message, the password here is correct (sell-book). The BookSellerAgent accept the message from the ServletAgent because the password from the ServletAgent is correct (price), the mobileagent get the message.
The result of the deal here is no one deal. The bid from the BookBuyerAgent is lower than the price set by the user, Bid = 456. The mobileagent didn’t accept the bid. That mean the mobileagent react as expected.

5) sell-book Java Programming Language price 50 dollar

The result: Good, as expected

ServletAgent react as expected. ServletAgent accept the message, the password here is correct (sell-book). The BookSellerAgent accept the message from the ServletAgent because the password from the ServletAgent is correct (price), the mobileagent get the message.

The result of the deal here is 6589 dollar, higher than the price set by the user, 50 dollar. That mean the mobileagent react as expected.

6) sell-book Java Programming Language price 300 dollar

The result: Good, as expected

ServletAgent react as expected. ServletAgent accept the message, the password here is correct (sell-book). The BookSellerAgent accept the message from the ServletAgent because the password from the ServletAgent is correct (price), the mobileagent get the message.

The result of the deal here is 4645666 dollar, higher than the price set by the user, 300 dollar. That mean the mobileagent react as expected.

7) sell-book Java Programming Language price 400 dollar

The result: Good, as expected

ServletAgent react as expected. ServletAgent accept the message, the password here is correct (sell-book). The BookSellerAgent accept the message from the ServletAgent because the password from the ServletAgent is correct (price), the mobileagent get the message.

The result of the deal here is 567569889 dollar, higher than the price set by the user, 400 dollar. That mean the mobileagent react as expected.

8) sell-book Java Programming Language price 100000 dollar

The result: Good, as expected

The result: Good, as expected

ServletAgent react as expected. ServletAgent accept the message, the password here is correct (sell-book). The BookSellerAgent accept the message from the ServletAgent
because the password from the ServletAgent is correct (price), the mobileagent get the message.

The result of the deal here is no one deal. The bid from the BookBuyerAgent is lower than the price set by the user, Bid = 45678. The mobileagent didn’t accept the bid. That mean the mobileagent react as expected.

9) SELL-BOOK  Java Programming Language PRICE 500000 dollar

The result : Good, as expected

The result : Good, as expected

ServletAgent react as expected. ServletAgent accept the message, the password here is correct (SELL-BOOK). The BookSellerAgent accept the message from the ServletAgent because the password from the ServletAgent is correct (PRICE), the mobileagent get the message.

The result of the deal here is no one deal. The bid from the BookBuyerAgent is lower than the price set by the user, Bid = 78945. The mobileagent didn’t accept the bid. That mean the mobileagent react as expected.

10) sell-book  Java Programming Language price 100000 dollar

The result : Good, as expected

The result : Good, as expected

ServletAgent react as expected. ServletAgent accept the message, the password here is correct (sell-book). The BookSellerAgent accept the message from the ServletAgent because the password from the ServletAgent is correct (price), the mobileagent get the message.

The result of the deal here is 134555666 dollar, higher than the price set by the user, 100000 dollar. That mean the mobileagent react as expected.

I will show you some screen shot of one of the test next
Figure 64  The BookSellerAgent and BookBuyerAgent  are up running , ready
Figure 65  The message sell-book J2ME programming price 160 dollar will be send from the J2ME cell phone to the mobileagent
Figure 66 The mobilagent get the message
Figure 67  The user send the command RESULT to get the result of the deal
Figure 68  The result of the deal is parsed back to the User from the ServletAgent
You must start up the Tomcat –4.0.2 Servlet motor first to be able to run the prototype.

Figure 69, The Tomcat - 4.0 Servlet motor must be started up first before the prototype can be run.

13.2 Test of the network security

Before testing the security of the prototype, I must ask the question

What is the threat? The threat is that some hacker can eavesdropper the data I transfer to my mobile agent, after a while he can find out what is the meaning of the message (what is the password I used, and other information, etc …).

Then the hacker can pretend to be me and get connect to my mobile agent, to get the message or to try to steal important information from me.

As you can see in the next figures.
Figure 70  The hacker can eavesdropper my data easily if I don’t do anything to protect the communication lines.
Figure 71  The hacker can then after a while, pretend to be me and get connect to my mobile agent to steal important personal information from me.

But I had use the solution to protect my data, as I had show you in detail in chapter 12. Now, for the test of the network security.
Here, I used the Telnet and other client program without the same trust certificate as a client and try to connect to the BookSeller Agent. The connection failed. The BookSellerAgent didn’t accept the data because it did not recognize the certificate. So the BookSeller Agent behave as it should.

Please see the next figures. These figures show you some screenshot of the test
Figure 72  I used first the Telnet as a client and try to connect to the BookSellerAgent

The connection will fail, because the BookSellerAgent use now the SSL protocol and don’t accept plaintext.

Please see next figure 73
The connection failed, because the SSLServerSocket where the BookSellerAgent is running on. Don’t accept plaintext message.
Now I use the SSLSocket Client who didn’t have the same certificate as the SSLServerSocket where the BookSellerAgent is running on and try to get connect to the BookSellerAgent.

![SSL Server Socket](image)

Figure 74  Now you see that the SSLServerSocket where the BookSellerAgent is running on didn’t accept the message because the certificate is unknown.

So the BookSellerAgent behave as it should. It will not accept any message if the authentication fail. Because the BookSellerAgent now run on the SSLServerSocket.

129
That mean the security is good, if some hacker pretend to be me and send message to my BookSellerAgent, they will not be able to get connect, because I now use the SSL protocol.

Now, to test the security of the ServletAgent.

To test the security of the Servlet, it is simple, I used the ordinary Web browser and try to get some text and HTML files by using the HTTPS connection. The connection went well, I had no problem to get theses file from Servlet by using HTTSPS connection. That mean the Servlet support the HTTPS connection.

![Hello there](https://i.imgur.com/123.png)

**Hello there**

*This Servlet support the HTTPS Connection, Don't Worry, Your data will be safe*

Figure 75 I start up the Tomcat Servlet motor and then I **use**

https://127.0.0.1/test.html on my Web browser and I can get the test.html file. It mean the Servlet now support the HTTPS connection.

The MobileAgent and BookBuyer Agent are protected by the Bee-gent framework. No test about security are need for these.
14 The Conclusion

This chapter present the conclusion of the work in this thesis.

14.1 The conclusion

The new MIDP 2.0 version give you more network security than the old MIDP 1.0. It will let you use the HTTPS connection (Secure HyperText Transfer Protocol).

The HTTPS connection provide data encryption and authentication. The security is always the most important issue, particular when you work with mobileagent.

When you create a technical solution on the Internet, you must always take into account the security issue, because peoples always demand security when they do business on the Internet. For example E-commerce. If peoples don’t feel safe when they use your solution, they don’t want to use it.

The solution I present in this thesis is the very easily way to make one use mobileagent on cellphone with J2ME, and the security issue had been included, the data sending from J2ME cellphone to mobileagent is encrypted and the communication process must go through the authentication process.

It is best to use HTTPS connection, you can trust the HTTPS connection. The HTTPS connection secure your data.

But not all the server are supporting the HTTPS connection. So if you don’t use the HTTPS connection. You can still achieve data encryption and authentication by using the Bouncy Castle Package. But you should always use the HTTPS connection as the first choice because you can be sure that the HTTPS connection provide the good network security for you.

The J2SE 1.4 version will also let you use SSL protocol very easily (Secure Sockets Layer). You don’t had to download extra package to be able to use SSL Protocol. I used the SSL protocol to make the communication between ServletAgent and BookSellerAgent secure.

There are still some technical drawback. When I used the J2ME Wireless Toolkit 2.0. I got often deadlock (The MIDP 2.0 is included in the J2ME Wireless Toolkit 2.0).

Maybe that because the J2ME Wireless Toolkit 2.0 didn’t accept the certificate from the Servlet Tomcat-4.0.2 I created. When I used the J2ME Wireless Toolkit 1.4, it work much better. But some times I got deadlock. Maybe because the Tomcat Servlet motor is not good enough to support the HTTPS connection. The Tomcat servlet motor is good only for research. Not for commerce.

So, except for some small technical drawback, the MIDP 2.0 is better than the old MIDP 1.0. because the MIDP 2.0 must include the HTTPS connection.

Of course, it still be able for the hacker with very high computer science expertise to decrypt your message even if you encrypt data when you transfer it over the Internet. But
, these hacker are very few, and the method to encrypt and authenticate data will continued to be more advanced in the future. So you can feel safe when you use the HTTPS connection.

One more issue may also be interesting. It is about the money. When you create a technical solution, you must also think about how much money it will cost peoples to use that solution. The HTTPS connection is mandatory on MIDP 2.0. All the cell phones with J2ME usually include the MIDP 2.0. The other connection such as Socket connection and other are optional.

Most of the J2ME cell phones on the market today usually only include the HTTPS connection. The cell phones which also include all the other network connection too are often the most expensive one. The most low-priced J2ME cell phones only include the HTTPS connection.

So, if you use the HTTPS Connection, you have the low-priced solution for peoples to use. Because peoples don’t had to buy the more expensive cell phones to use your solution.

When you use a mobileagent, the mobileagent must always run on it own framework. The framework protect the agent from virus and hacker too.

The Bee-gent framework is used as an example to demonstrate how I can use mobileagent on J2ME cell phones. It doesn’t mean you can only use the Bee-gent framework. You can use other mobileagent framework too. Or you can create your own mobileagent framework. But the mobile agent framework you use or create should be 100 percent Agentifies, it mean all applications are intelligent Agents, it will then be much easier for you to use your mobile agent from your cell phones if the mobile agent framework you are using is 100 percent Agentifies.

The idea is to let your mobileagent run on a remote computer, not on your cell phones. And then use HTTPS connection to get connect to your mobileagent. As I present in Chapter 9 and in chapter 12 (section 12.3).

Mobileagents are the applications who had their owner states, beliefs and goals and can act without or with minimal of the user or owner interaction. In the prototype, you see that the mobileagent only need one information to do the task, and it is the price. When he get the price, he can go and do the work all by his self (Of course, you must programme the mobileagent to be able to do that).

So, you only need to send your mobileagent a few data and then the task will be done for you.

The J2ME cell phones has much less memory resources than the computer desktop, that is why you better let the mobileagent run on the computer desktop. As I present in chapter 9.

14.2 The future work

I had only tested the prototype on a emulator and not on a real device. It should be interesting to do that. And I use the Bee-gent framework as an demonstration. Bee-gent framework was created by TOSHIBA.
When you use mobileagent, the mobileagent must run on a framework. The framework are there too protect the agent and make it easier for the agent to do their tasks.

In this thesis, I didn’t had time too create my own framework, it will take lot of times to create a mobileagent framework. So I use the Bee-gent framework as an demonstration to my solution.

I don’t think that IDI NTNU had any project where goal is to create a mobilagent framework. It should be interesting that IDI NTNU have some project where goal is to make a mobilagent framework who can run on all kinds of devices, such as computer desktop, laptop, cell phones, etc ….. .
References

[1] The future java technology for the wireless technology , February 2003 , Qusay Mahmud
http://wireless.java.sun.com/allarticles/#new

[2] J2ME Low-level Network Programming with MIDP 2.0 , Qusay Mahmud , April 2003 ,
http://wireless.java.sun.com/allarticles/#new

http://wireless.java.sun.com/midp/articles/socketRMI/

http://wireless.java.sun.com/midp/articles/api/

[5] Introduction to the J2EE Platform , By Monica Pawlan March 23, 2001,
http://developer.java.sun.com/developer/technicalArticles/J2EE/Intro/#thin


[8] What's New in MIDP 2.0 by Jonathan Knudsen ,

[9] Parsing XML on J2ME , XML in a MIDP Enviroment
Jonathan Knudsen , March 7 2002 ,
http://wireless.java.sun.com/midp/articles/parsingxml/


http://wireless.java.sun.com/allarticles
Term Explanation

**ACL**: Agent communication Language, The language the agent use to communicate with each other. FIPA had defined the standard for ACL, FIPA ACL.

**AW**: Agent Wrapper, An stationary intelligent agent in Bee-gent mobileagent framework. The Agent Wrapper are used to agentify existing applications. The Agent Wrapper can create an mobileagent. It can also welcome an mobileagent who come to visit its place.

**Bouncy Castle Package**: The APIs Package created by an open source base in Australia. A toolbox of cryptographic algorithms, If you don’t want to use HTTPS connection, you can use this Package instead to encrypt your data.

**Bee-gent**: Bounding and Encapsulation Enhancement Agent. A weak mobileagent framework created by TOSHIBA Japan.

**Certificate**: use for the authentication, either the Server authentication to the Client or the Client authentication to the Server. The de facto standard for certificates is X.509v3.

**Custom class loader**: A user own defined class loader.

**Cipher suites**: A group of encryption and decryption algorithms.

**CDC**: Connected Device Configuration, identified in J2ME, memory budgets between 2-16 MB.

**CLDC**: Connected Limited Device Configuration, a specification for J2ME platform. Memory budgets between 128 KB – 1 MB.

**Configuration**: Define class libraries or grouping of devices based on requirements for total memory budget and processing power.

**FIPA**: Foundation for Intelligent Physical Agents. Was formed in 1996 to produce software standards for heterogeneous and interacting agents and agent-based systems.

**FIPA ACL**: Specified in 1997. FIPA defined Language for describing agents actions, beliefs, intentions.

**HTTP**: Hypertext Transfer Protocol. It is a communication protocol designed to transfer hypertext documents between computers over the World Wide Web. It defines what actions Webs server and browsers should take in response to various command.

**HTTPS**: Secure Hypertext Transfer Protocol. The secure hypertext Transfer Protocol is a communication protocol designed to transfer encrypted information between computers over the World Wide Web.

**J2SE**: Java 2 Standard Edition. A Java Platform for computer desktop or laptop applications with require large memory resources.
**J2ME** : Java 2 Micro Edition. A Java platform for applications running on micro devices with limited memory resources.


**JVM** : Java Virtual Machine. The heart of the J2SE. Manager memories resource, and execute bytes code.


**Keytool** : A tool you can use to create a self issued certificate for your server.

**Message Digest** : A message digest smushes a large piece of data into a small piece of data. A Message digest value is sometimes called a digital fingerprint.

**Mediation agent** : It is the mobileagent in Bee-gent mobileagent framework.

**MIDlet** : The MIDlet is the application on Java 2 Micro Edition Platform.

**MIDP** : Mobile Information Device Profile. Provide device-specific functionality. This functionalities includes on devices application management, low level and high level graphical user interfaces, persistent storage, and extended network capability.

**Obfuscator** : By default, compiled bytecode still contains a lot of debugging information: source file names, line numbers, field names, method names, argument names, variable names, etc. This information makes it straightforward to decompile the bytecode and reverse-engineer entire programs. Sometimes, this is not desirable. Obfuscators such as **ProGuard** can remove the debugging information and replace all names by meaningless character sequences, making it much harder to reverse-engineer the code. It further compacts the code as a bonus. The program remains functionally equivalent, except for the class names, method names, and line numbers given in exception stack traces.

**ProGuard** : **ProGuard** is a free Java class file shrinker and obfuscator. It can detect and remove unused classes, fields, methods, and attributes. It can then rename the remaining classes, fields, and methods using short meaningless names. The resulting jars are smaller and harder to reverse-engineer.

**PDA** : Personal Digital Assistant. A micro device where the user can store appointments, for example. This device can run J2ME applications too.

**Profile** : defines additional sets of APIs and features for a particular vertical market, device category or industry.

**RSA** : The RSA Algorithm was named after Ronald Rivest, Adir Shamir, and Leonard Adelman. Who first published the algorithm in April 1977. The RSA algorithm is only one implementation of the more general concept of public key cryptography, which permits two parties who have never met and who can only communicate on an insecure channel to nonetheless send secure and verifiable messages to each other. With public key technique, each user has two different keys, one made available to the public and the other kept secret. One of the key is used to encrypt the message, and the other is used to decrypt the message.
**SSL**: Secure Socket Layer protocol. The SSL security protocol provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection.

**Session Key**: The key used to encrypt and decrypt all data sent between client and server.

**TLS**: Transport Layer Security. Is a protocol that enables data authentication and encryption over the insecure network such as the Internet. It is an update version of the SSLv3 protocol.

**XML**: Extensible Markup Language. Provides a system-independent standard format for specifying the information exchanged over networks and between applications.
Appendix  A

The source java code for the MIDlet, mobileAgent

/**
 * 
 * mobileAgent.java
 * 
 */

import javax.microedition.rms.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.pki.*;
import java.io.*;
import java.util.Vector;
import java.util.*;
import org.kxml.*;
import org.kxml.parser.*;

/**
 * The class mobileAgent is a MIDlet who can enable the user to
 * invoke a request as an SMS message and send it to the mobile
 * agent who is running on a specific host
 * by using HTTPS connection through ServletAgent
 * 
 * @author Quoc Nguyen
 * @version 1.1 15.05.03
 * 
 */

public class mobileAgent extends MIDlet implements CommandListener {

    Display display = null;
    List menu = null;
    TextBox input = null;
    String user = null;

    // url is the address of the ServletAgent
    String url = "https://localhost/servlet/ServletAgent";

    // define the buttons for the UI API
    static final Command backCommand = new Command("Back", Command.BACK, 0);
    static final Command submitCommand = new Command("Send", Command.OK, 2);
    static final Command exitCommand = new Command("Exit", Command.STOP, 3);
    static final Command resultCommand = new Command("Result", Command.STOP, 3);
String currentMenu = null;

public mobileAgent() {
}

// this method invoke to start the MIDlet
public void startApp() throws MIDletStateChangeException {
    display = Display.getDisplay(this);
    menu = new List("Menu", Choice.IMPLICIT);
    menu.append("Sell command to agent", null);
    menu.addCommand(exitCommand);
    menu.addCommand(resultCommand);
    menu.setCommandListener(this);
    mainMenu();
}

// close to MIDlet to enable to receive the call
public void pauseApp() {
}

// close down the MIDlet
public void destroyApp(boolean unconditional) {
    notifyDestroyed();
}

void mainMenu() {
    displaysetCurrent(menu);
}

/*
 * This method is call to display the UI where the user
 * can be able to type a request as an SMS message and send it
 * to the mobileagent
 *
 */
public void writeMessage1() {
    input = new TextBox("Enter a message:", "", 500, TextField.ANY);
    input.addCommand(submitCommand);
    input.addCommand(backCommand);
    input.setCommandListener(this);
    input.setString(""");
    display.setCurrent(input);
}

/*
 * This method is call to display the UI where the user
 * can be able to type a request for the result as an command message
 * and send it to the ServletAgent
 *
 */
public void writeTheCommand() {
    input = new TextBox("Enter a message:", "", 500, TextField.ANY);
    input.addCommand(submitCommand);
    input.addCommand(backCommand);
input.setCommandListener(this);
input.setString("RESULT");
display.setCurrent(input);
}

/**
 * This method invokeServlet en-able the user to send an message almost like an
 * SMS message and enable the MIDlet to
 * display the respond from Java servlet by using kXML parser
 * *
 */
void invokeServlet(String url) throws IOException {
    HttpConnection c = null;
    DataInputStream is = null;
    DataOutputStream os = null;
    StringBuffer b = new StringBuffer();
    TextBox t = null;
    String resultO =null;

    try {
        c = (HttpConnection)Connector.open(url);
        c.setRequestMethod(HttpConnection.POST);/* use the POST method */
        c.setRequestProperty("IF-Modified-Since", "10 mai 2003 12:00:16 GMT");
        c.setRequestProperty("User-Agent", "Profile/MIDP-1.0 Configuration/CLDC-1.0");
        c.setRequestProperty("Content-Language", "en");
        c.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");

        os = c.openDataOutputStream();
        os.write(""+user).getBytes());/* prepare to send the data */
        os.flush();

        is = c.openDataInputStream();

        int ch;
        while ((ch = is.read()) != -1) {
            b.append((char) ch);
            System.out.print((char)ch);
        }

        resultO = b.toString();
        String[] items;
        items = parseUsingkXML(resultO);
        display.setCurrent( new ItemList( items ) );/* display the result */
    }

    finally {
        if(is!= null) {
            is.close();
        }
        if(os != null) {
            os.close();
        }
        if(c != null) {
            c.close();
        }
    }
}
// exit the MIDlet
public void exitMIDlet(){
    notifyDestroyed();
}

/*
 * This method is call to parse the XML as kXML on the
 * MIDlet ,
 * 
 * Parses a document using kXML, looking for "item"
 * nodes and returning their content as an
 * array of strings.
 *
 * private String[] parseUsingkXML( String xml ){

    try {
        ByteArrayInputStream bin = new ByteArrayInputStream( xml.getBytes() );
        InputStreamReader in = new InputStreamReader( bin );

        XmlParser parser = new XmlParser( in );

        Vector items = new Vector();
        parseXMLItems( parser, items );
        String[] tmp = new String[ items.size() ];
        items.copyInto( tmp );
        return tmp;
    } 

    catch( IOException e ) {
        return new String[]{ e.toString() };
    }
}
*/

private void parseXMLItems( XmlParser parser, Vector items )
    throws IOException {
        boolean inItem = false;

        while( true ){
           ParseEvent event = parser.read();
            switch( event.getType() ) {
                case Xml.START_TAG: 
                    if( event.getName().equals( "item" ) ){
                        inItem = true;
                    }
                    break;
                case Xml.END_TAG:
                    if( event.getName().equals( "item" ) ){

                }
            }
        }
    }
inItem = false;
}
break;
case Xml.TEXT:
if (inItem ){
    items.addElement( event.getText() );
}
break;
case Xml.END_DOCUMENT:
return;
}
}

// Simple List UI component for displaying the list of
// items parsed from the XML document.
class ItemList extends List implements CommandListener {

ItemList( String[] list ){
    super( "Items", IMPLICIT, list, null );
    addCommand( exitCommand );
    setCommandListener( this );
}

public void commandAction( Command c, Displayable d ){
    if (c == exitCommand ){
        exitMIDlet();
    }
}

// This method is called when the user presses the buttons on the
// UI of the MIDlet
public void commandAction(Command c, Displayable d) {
    String label = c.getLabel();
    if (label.equals("Exit")) {/* when the user press button Exit */
        destroyApp(true);
        } else if (label.equals("Back")) {/* when the user press button Back */
        mainMenu();
        } else if (label.equals("Send")) {/* when the user press button Send */
        user = input.getString();
        try {
            invokeServlet(url);
        } catch (IOException e) {}  
        } else if (label.equals("Result")) {/* when the user press button Result */
        writeTheCommand();
        } else {
            writeMessage1();
        }
    }
The source for the Servlet Agent

```java
import java.io.*;
import java.net.*;
import java.util.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * The ServletAgent is a servlet run on Jakarta-tomcat-4.0.2 Web server
 * @author Quoc Nguyen
 * @version 1.0.05.03
 * @since J2SE 1.4
 *
 */

class ServletAgent extends HttpServlet {

    /** The doPost() method handles all the POST requests received by the ServletAgent */
    public void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws IOException, ServletException
    {
        // send the response back to client
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // get the request send from the client
        ServletInputStream in = request.getInputStream();
        DataInputStream din = new DataInputStream(in);
        String text = din.readLine();
        StringTokenizer tok = new StringTokenizer(text);
        String seek = tok.nextToken();

        // Check if a valid password
        if(seek.equals("SELL-BOOK") || seek.equals("sell-book") ) {
            Sender se = new Sender();
            se.send(text);
        } else if(seek.equals("RESULT") || seek.equals("result")) {
            getResult Re = new getResult();
        }
    }
```
// parser the result to the client in XML
out.println("<list>");
out.println("<item> Here is the result : </item>");
out.println("<item> " + Re.readResult() + "</item>");
out.println("</list>");

// Respond to client that the password is a bad request
else {
    out.println("<list>");
    out.println("<item> HTTP1/1 400 Bad request : </item>");
    out.println("</list>");
}

} // doPost


The help class for ServletAgent

import java.util.Date;
import java.io.*;
import java.net.*;
import javax.net.*;
import javax.net.ssl.*;
import java.security.*;
import java.util.StringTokenizer;

/**
 * The help class Sender create the socket connection
 * to BookSellerAgent
 *
 * @version 1.1 15.05.03
 * @author Quoc Nguyen
 */

public class Sender {
    public Sender() { }
    
    OutputStream os = null;
    
    /**
     * The send method forward the message from client MIDlet
     * to BookSellerAgent
     */
    
    public void send(String msg) {

}
try {
    SSLSocketFactory factory = (SSLSocketFactory)
        SSLSocketFactory.getDefault();

    Socket client = (SSLSocket)
        factory.createSocket("localhost", 4400);

    os = new DataOutputStream(client.getOutputStream());
    os.writeBytes(msg); // message body
    os.flush();
    os.close();

} catch (Exception e) {
    System.err.println("Error1");
}

} // send()

} // Sender

The help class getResult

import java.io.*;

/**
 * The class getResult is the help class that
 * get the result from text fil result2.txt stored by the BookSellerAgent
 *
 * @author Quoc Nguyen
 * @version %I%, %G%
 * @since J2SE 1.4
 *
 */

public class getResult {

    File f;
    FileReader in = null;

    String readResult() {

        String line = null;

        try {

            f = new File("c:\", "result2.txt");
            in = new FileReader(f);
            int size = (int) f.length();

        } catch (IOException e) {
            System.err.println("Error2");
        }

        BufferedReader br = new BufferedReader(in);
        line = br.readLine();

        System.out.println(line);

        br.close();

    }

} // getResult

147
```java
char[] data = new char[size];
int chars_read = 0;
BufferedReader bufread = new BufferedReader(in);
line = bufread.readLine();

} catch (Exception e) { System.out.println("error"); }

return line;

} // readResult()

} // getResult

The BookSellerAgent class

/*
 * AW1.java
 *
 * Copyright (c) 1995-2002 TOSHIBA Corp. All Rights Reserved.
 *
 * This software is the confidential and proprietary information of
 * TOSHIBA Corp. ("Confidential Information"). You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with TOSHIBA.
 *
 * TOSHIBA MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE
 * SUITABILITY OF THE
 * SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
 * TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR
 * PURPOSE, OR NON-INFRINGEMENT. TOSHIBA SHALL NOT BE LIABLE FOR
 * ANY DAMAGES
 * SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
 * DISTRIBUTING
 * THIS SOFTWARE OR ITS DERIVATIVES.
 */

import java.io.*;
import java.util.*;
import com.toshiba.beegent.*;
import com.toshiba.beegent.xml.*;
import com.toshiba.beegent.util.*;
import java.net.*;
import javax.net.*;
import java.security.*;
import java.util.StringTokenizer;
```
/**
 * AW1 is BookSellerAgent .
 *
 * This Agentwrapper AW1 (BookSellerAgent) class receive and forward the valid
 * request message
 * (parameter) from the user to the mobileagent (BEE1), and AW1
 * also stores the result of the deal into the result2.txt,
 *
 * To be able to do the different actions, this AW1 class have
 * two AwrlIPStates, 1 and 2
 *
 * @version 1.1 15.05.03
 * @author Quoc Nguyen
 *
 */
public class AW1 extends AgentWrapper {

    public static void main(String[] argv) throws Exception {

        AW1 aw = new AW1();
        aw.setName("BookSeller");
        aw.printLog(true);
        aw.fileLog(true);
        aw.setPassword("IDI");
        aw.addIPStates(new AwrlIPState1());
        aw.addIPStates(new AwrlIPState2());
        aw.startIP(); /*

    } // main

} // AW1


/*
 *
 * this AwrlIPState1 have Precondtion as INIT state and
 * PostCondition as RESULT state
 * the INIT state is to wait for the valid request
 * from the user, the RESULT state is to wait
 * for result of the deal the mobile agent BEE1 brought back
 *
 */

class AwrlIPState1 extends AwrlIPState {

    AwrlIPState1() { setPrecond("INIT"); }

    public void action() {
        String line;
        DataInputStream is = null;
        String text = null;
        String Seek;
        String price = null;
    }
// Check the certificate
String keystore = "skey";

// Here, the password must be the same as when you created the certificate
char keystorepass[] = "hellothere".toCharArray();
char keypassword[] = "hellothere".toCharArray();

final int HTTPS_PORT = 4400;

/*
* Set the server up on port 4400, this enable AW1 (BookSellerAgent) to receive
* the request from the user (through ServletAgent)
*/

try {
    KeyStore ks = KeyStore.getInstance("JKS");
    ks.load(new FileInputStream(keystore), keystorepass);
    KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
    kmf.init(ks, keypassword);
    SSLContext sslcontext = SSLContext.getInstance("SSLv3");
    sslcontext.init(kmf.getKeyManagers(), null, null);
    ServerSocketFactory ssf = sslcontext.getServerSocketFactory();
    SSLServerSocket serversocket = (SSLServerSocket) ssf.createServerSocket(HTTPS_PORT);
    SSLSocket client = (SSLSocket) serversocket.accept();

    System.out.println("Get a client");
    is = new DataInputStream(client.getInputStream());
    text = is.readLine();
    System.out.println("text is "+text);
    StringTokenizer tok = new StringTokenizer(text);

    while (tok.hasMoreTokens()) { /* check if the request had more word */
        Seek = tok.nextToken();

        // Check if the request is a valid request
        if(Seek.equals("PRICE") || Seek.equals("price")) {
            price = tok.nextToken(); /* get the price from the user */

        // create BEE1
        try {
            createBee("BEE1","mobileagent");
        } catch(Exception e) {

            Debug.printLog(getMyname(),"failed to create Bee.");
            setPostcond("INIT"); /* go back onto INIT state and wait for new request */
        } /* end try */
    } /* end while */
} /* end try */
}
return;

}

} // if

} // while

} catch (Exception e) {
    System.out.println("Register exception: "+ e.getMessage());
    e.printStackTrace();
}

// Send XML/ACL
XmlAcl xa = new XmlAcl();
xa.setTag2Value("performative","request");
xa.setTag2Value("sender",getMname());
xa.setTag2Value("receiver","mobileagent");
xa.setTag2Value("action",text);
xa.setTag2Value("args",price);

if(!sendXML(xa)){
    Debug.printLog(getMname(),"failed to send XML/ACL.");
    setPostcond("INIT");
    return;
}

setPostcond("RESULT"); /* goes onto the RESULT state */

} // action

} // AwrIPState1

/*
 *
* this AwrIPState2 have Precondtion as RESULT state and
* PostCondition as INIT state,
* the RESULT state is to wait for the result from the mobileagent BE1
* , the RESULT state is to wait for the new valid request from the user
*
*/
class AwrIPState2 extends AwrIPState{

AwrIPState2() { setPrecond("RESULT"); }

public void action(){

    // Receive XML/ACL
    if(waitXML(0)) {
        XmlAcl xa = getXML();

    }
}
String perf = xa.getTag2Value("performative");
String content = xa.getTag2Value("content");/*/ get the result from BEE1 */

if(perf.equals("inform"))
    Debug.printLog(getMyname(),"Result: " + content);

// stores the result to the text file result2.txt
try {
    FileWriter fo = new FileWriter("c:\\result2.txt");
    fo.write(content);
    fo.flush();
    fo.close();
}

} catch(Exception e) {
    System.out.println("Error");
}

} // if

setPostcond("INIT");//* goes onto the INIT state */

} // action

} // AwrIPSate2

The BookBuyer Agent class

/*
 *
 */

* Copyright (c) 1995 - 2002 TOSHIBA Corp. All Rights Reserved.
*
* This software is the confidential and proprietary information of
* TOSHIBA Corp. ("Confidential Information"). You shall not
* disclose such Confidential Information and shall use it only in
* accordance with the terms of the license agreement you entered into
* with TOSHIBA.
*
* TOSHIBA MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
* THE
* SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
* IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
* PURPOSE, OR NON-INFRINGEMENT. TOSHIBA SHALL NOT BE LIABLE FOR ANY DAMAGES
* SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING
* THIS SOFTWARE OR ITS DERIVATIVES.
*/
import java.io.*;
import java.net.*;
import java.util.*;
import com.toshiba.beegent.*;
import com.toshiba.beegent.xml.*;
import com.toshiba.beegent.util.*;

/**
 * AW2 is an AgentWrapper , This AW2 class is the BookBuyerAgent , BookBuyerAgent have three states
 */
public class AW2 extends AgentWrapper {

    AW2(String s) {
        super(s);
    }

    public static void main(String[] argv) throws Exception {

        AW2 aw = new AW2("BookBuyer");
        aw.printLog(true);
        aw.fileLog(true);
        aw.setPassword("IDI");

        /**
         * define the states INIT, WAIT, NEG by using the addPublicIPStates(),
         *
         */
        aw.addPublicIPStates();
        aw.startIP();

    }

}

The help classes of the BookBuyerAgent

/*
 * AwrIPStateS1.java
 *
 * Copyright (c) 1997, 1998 TOSHIBA Corp. All Rights Reserved.
 *
 * This software is the confidential and proprietary information of
 * TOSHIBA Corp. ("Confidential Information"). You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with TOSHIBA.
 * 
 * TOSHIBA MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
 * THE
 * SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
 * PURPOSE, OR NON-INFRINGEMENT. TOSHIBA SHALL NOT BE LIABLE FOR ANY DAMAGES
 * SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING
 * THIS SOFTWARE OR ITS DERIVATIVES.
 */

import java.io.*;
import java.net.*;
import java.util.*;
import com.toshiba.beegent.*;
import com.toshiba.beegent.xml.*;
import com.toshiba.beegent.util.*;
public class AwrIPStateS1 extends AwrIPState{

    public AwrIPStateS1(){ super("INIT"); }

    public void action(){

        XmlAcl xa = null;
        String perf = new String();
        String sender = new String();
        String action = new String();
        String args = new String();

        // Receive XML/ACL
        while(waitXML(0)){
            xa = getXML();
            perf = xa.getTag2Value("performative");
            sender = xa.getTag2Value("sender");
            action = xa.getTag2Value("action");
            args = xa.getTag2Value("args");

            if(perf.equals("cfp"))
                break;
        } //while

        // Send XML/ACL

        int rand = new Random(System.currentTimeMillis()).nextInt();
        int dice = Math.abs(rand) % 2;
        switch(dice){

            case 1:
                xa = new XmlAcl();
                xa.setTag2Value("performative","propose");
                xa.setTag2Value("sender" ,getMyname());
                xa.setTag2Value("receiver" ,sender);
                xa.setTag2Value("action" ,action);
                xa.setTag2Value("actor" ,getMyname());
                xa.setTag2Value("args" ,"I'm interested.");

                setPostcond("WAIT");
                break;

            default:
                setPostcond("END");
                return;
        }

        if(!sendXML(xa))
            Debug.printLog(getMyname(),"failed to send XML/ACL.");
    }

    /*
    * AwrIPStateS2.java
    *
    * Copyright (c) 1997, 1998 TOSHIBA Corp. All Rights Reserved.
    */
* This software is the confidential and proprietary information of
* TOSHIBA Corp. ("Confidential Information"). You shall not
* disclose such Confidential Information and shall use it only in
* accordance with the terms of the license agreement you entered into
* with TOSHIBA.
*
* TOSHIBA MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE
* SUITABILITY OF THE
* SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE
* IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR
* PURPOSE, OR NON-INFRINGEMENT. TOSHIBA SHALL NOT BE LIABLE FOR
* ANY DAMAGES
* SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING
* THIS SOFTWARE OR ITS DERIVATIVES.
*/

import java.io.*;
import java.net.*;
import java.util.*;
import com.toshiba.beegent.*;
import com.toshiba.beegent.xml.*;
import com.toshiba.beegent.util.*;

//The second states of the BookBuyerAgent
public class AwrlIPStateS2 extends AwrlIPState{

    public AwrlIPStateS2(){ super("WAIT"); }

    public void action(){

        XmlAcl xa = null;
        String perf  = new String();
        String sender = new String();
        String action = new String();

        // Receive XML/ACL
        while(waitXML(0)){
            xa     = getXML();
            perf   = xa.getTag2Value("performative");
            sender = xa.getTag2Value("sender");
            action = xa.getTag2Value("action");

            if(perf.equals("reject-proposal")){
                setPostcond("INIT");
                return;
            }
            else if(perf.equals("accept-proposal")){
                setPostcond("NEGOT");
                return;
            }

        }
    }
}
/*
* AwrIPStateS3.java is the 3 state of the BookBuyerAgent
*
* Copyright (c) 1997, 1998 TOSHIBA Corp. All Rights Reserved.
*
* This software is the confidential and proprietary information of
* TOSHIBA Corp. ("Confidential Information"). You shall not
* disclose such Confidential Information and shall use it only in
* accordance with the terms of the license agreement you entered into
* with TOSHIBA.
*
* TOSHIBA MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE
*SUITABILITY OF THE
* SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
*TO THE
* IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
*PARTicular
* PURPOSE, OR NON-INFRINGEMENT. TOSHIBA SHALL NOT BE LIABLE FOR
*ANY DAMAGES
* SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
*Distributing
* THIS SOFTWARE OR ITS DERIVATIVES.
*/

import java.io.*;
import java.net.*;
import java.util.*;
import com.toshiba.beegent.*;
import com.toshiba.beegent.xml.*;
import com.toshiba.beegent.util.*;

// The 3 State of the BookBuyerAgent
public class AwrIPStateS3 extends AwrIPState{

  public AwrIPStateS3(){ super("NEGO"); }

  public void action(){

    XmlAcl xa = null;
    String perf   = new String();
    String sender = new String();
    String action = new String();
    String args   = new String();

    // Receive XML/ACL
    while(waitXML(0)){

      xa = getXML();
    }
  }
}
perf = xa.getTag2Value("performative");
sender = xa.getTag2Value("sender");
action = xa.getTag2Value("action");
args = xa.getTag2Value("args");

if(perf.equals("request") || (action.indexOf("sell") != -1))
    break;

// Send XML/ACL
int cost = new Random(System.currentTimeMillis()).nextInt();

if( cost <0 ) { cost = 1500; }

int cost1 = 6;

if(cost1 > 0){
    xa = new XmlAcl();
    xa.setTag2Value("performative","inform");
    xa.setTag2Value("sender",getMyName());
    xa.setTag2Value("receiver",sender);
    xa.setTag2Value("content","" + new Integer(cost).toString());
}
else {
    xa = new XmlAcl();
    xa.setTag2Value("performative","failure");
    xa.setTag2Value("sender",getMyName());
    xa.setTag2Value("receiver",sender);
    xa.setTag2Value("action",action);
    xa.setTag2Value("args",args);
    xa.setTag2Value("reason","malfonctioned!");
}

if(!sendXML(xa))
    Debug.println(getMyName(),"failed to send XML/ACL.");

setPostcond("INIT");

}


The MobileAgent Class

/*
 * BEE1.java, The BEE1 Class is the MobileAgent Class, It has 6 different states
 * Copyright (c) 1997, 1998 TOSHIBA Corp. All Rights Reserved.
 * This software is the confidential and proprietary information of
 * TOSHIBA Corp. ("Confidential Information"). You shall not
* disclose such Confidential Information and shall use it only in
* accordance with the terms of the license agreement you entered into
* with TOSHIBA.
*
* TOSHIBA MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE
* SUITABILITY OF THE
* SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE
* IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR
* PURPOSE, OR NON-INFRINGEMENT. TOSHIBA SHALL NOT BE LIABLE FOR
* ANY DAMAGES
* SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING
* THIS SOFTWARE OR ITS DERIVATIVES.
*/

import java.io.*;
import java.net.*;
import java.util.*;
import com.toshiba.beegent.);
import com.toshiba.beegent.xml.);
import com.toshiba.beegent.util.);

/**<p>
 * Class definition of the body of the mediation agent "mobileagent",
 * Definition of the "INIT" state,
 * Definition of the "CFP" state,
 * Definition of the "SELECT" state,
 * Definition of the "MGRT" state,
 * Definition of the "NEGO" state,
 * Definition of the "REPORT" state,
 * 
 * <p>
 * @see BEE1
 * 
 * @version 1.1 15.05.03
 * @author Quoc Nguyen
 * 
 * 
 */

public class BEE1 extends Bee implements I_Bee{

    public void init() throws InstantiationException{

        addIPStates(new BeeIPState1());
        addIPStates(new BeeIPState2());
        addIPStates(new BeeIPState3());
        addIPStates(new BeeIPState4());
        addIPStates(new BeeIPState5());
        addIPStates(new BeeIPState6());
    }
}
*/
* In the "INIT" state, messages from other Agents are received.
*/

// The first state of the MobileAgent
class BeeIPState1 extends BeeIPState implements I_BeeIPState{

BeeIPState1() { super("INIT","CFP"); }

public void action()
{
// Receive XML/ACL
while(waitXML(0)) {
    XmlAcl xa = getXML();
    String perf = xa.getTag2Value("performative");
    String sender = xa.getTag2Value("sender");
    String action = xa.getTag2Value("action");
    String args = xa.getTag2Value("args");
    if (perf.equals("request")) {
        putBaggage("BookSeller",sender);
        putBaggage("deal",action);
        putBaggage("spec",args);
        break;
    }
}
}

} // end of class BeeIPState1

} // end of class MobileAgent

/*
* In "CFP", the mobile agent sends the message
* and waits for the reply
*/

// The second state of the MobileAgent
class BeeIPState2 extends BeeIPState implements I_BeeIPState{

BeeIPState2() { super("CFP"); }

public void action()
{
// Multicast XML/ACL
    XmlAcl xa = new XmlAcl();
    xa.setTag2Value("performative","cfp");
    xa.setTag2Value("sender",getMyname());
    xa.setTag2Value("receiver","BookBuyer");
    xa.setTag2Value("action",(String)getBaggage("deal");
    xa.setTag2Value("args",(String)getBaggage("args");
    sendXML(xa);
}
// Receive XML/ACL
int cnt = 0;
Vector bid = new Vector();
while(waitXML(0)) {
    xa = getXML();
    String perf = xa.getTag2Value("performative");
    String sender = xa.getTag2Value("sender");

    if(perf.equals("propose")) {
        bid.addElement(sender);
        if(++cnt > 0) break;
    }
    else if(perf.equals("not-understood") || perf.equals("refuse")) {
        if(++cnt > 0) break;
    }
}

if(bid.isEmpty()) {
    xa = new XmlAcl();
    xa.setTag2Value("performative","failure");
    xa.setTag2Value("sender",getMyName());
    xa.setTag2Value("receiver",getBaggage("BookSeller"));
    xa.setTag2Value("action","cfp");
    xa.setTag2Value("args",getBaggage("deal"));
    xa.setTag2Value("reason","no one deal");
    sendXML(xa);

    setPostcond("END");
}
else {
    putBaggage("bid",bid);
    setPostcond("SELECT");
}

// The 3 state of the MobileAgent
class BeeIPState3 extends BeeIPState implements I_BeeIPState {
    BeeIPState3() { super("SELECT","MGRT"); }
}
public void action()
{

    // Select one
    Vector bid = null;
    String bidder = null;
    try {
        bid = (Vector) getBaggage("bid");
        bidder = (String) bid.firstElement();
        bid.removeElement(bidder);
        } catch (Exception ex) {}

    // Send XML/ACL
    Enumeration en = bid.elements();
    while (en.hasMoreElements()) {

        XmlAcl xa = new XmlAcl();
        xa.setTag2Value("performative", "reject-proposal");
        xa.setTag2Value("sender", getMyName());
        xa.setTag2Value("receiver", (String) en.nextElement());
        xa.setTag2Value("action", (String) getBaggage("deal"));
        sendXML(xa);
    }

    // Migrate to BookBuyerAgent
    migrateBee(bidder);

    }

    }

    // The 4 state of the MobileAgent
    class BeeIPSState4 extends BeeIPSState implements I_BeeIPSState {

        BeeIPSState4() { super("MGRT"); }

        public void action()
        {

            // Receive XML/ACL
            String perf = new String();
            String sender = new String();
            if (waitXML(0)) {
                XmlAcl xa = getXML();
                perf = xa.getTag2Value("performative");
                sender = xa.getTag2Value("sender");
            }

            if (perf.equals("failure")) {
                Debug.printLog(getMyName(), "failed to migrate Bee.");
                setPostEvent("END");
            }
            else if (sender.equals((String) getBaggage("BookSeller"))) {


        }
setPostcond("REPORT");
}
else{
    setPostcond("NEGO");
}
}

// The 5 state of the MobileAgent
class BeeIPState5 extends BeeIPState implements I_BeeIPState{
    BeeIPState5() { super("NEGO","MGRT"); }

    public void action(){
        XmlAcl xa;
        String perf = new String();
        String result = new String();

        /* find the determined price from the user, use it for
         * the negotiation process
         */
        String com = (String)getBaggage("spec");

        int lel = Integer.parseInt(com); /* convert the determined price to int */

        // Send XML/ACL
        xa = new XmlAcl();
        xa.setTag2Value("performative","accept-proposal");
        xa.setTag2Value("sender",getMname());
        xa.setTag2Value("receiver","");
        xa.setTag2Value("action", (String)getBaggage("deal"));
        if(!sendXML(xa))
            Debug.printLog(getMname(),"failed to send XML/ACL.");

        // Send XML/ACL
        xa = new XmlAcl();
        xa.setTag2Value("performative","request");
        xa.setTag2Value("sender",getMname());
        xa.setTag2Value("receiver","");
        xa.setTag2Value("action", (String)getBaggage("deal"));
        xa.setTag2Value("args",(String)getBaggage("spec"));
        if(!sendXML(xa))
            Debug.printLog(getMname(),"failed to send XML/ACL.");

        // Receive XML/ACL
        if(waitXML(0)){
            perf = ((String)xh.getAttribute("performative","");
    
            if(perf.equals("accept-proposal"))
                Debug.printLog(getMname(), "Accepted Proposal");
            else
                Debug.printLog(getMname(), "Proposal Not Accepted");
        }else{
            Debug.printLog(getMname(), "Failed to receive XML/ACL");
        }
    }
}
xa = getXML();
perf = xa.getTag2Value("performative");

if(perf.equals("inform")){
    result = xa.getTag2Value("content"); /* get the price from the BookBuyerAgent */
    int le2 = Integer.parseInt(result); /* convert to int */
    /*
     * compare the bid from the BookBuyerAgent with the
     * determined price from the user, accept
     * the bid if it larger than the determined price
    */
    if( le2 > le1 ) {
        putBaggage("result",result); /* accept and put into Bagge the bid */
    } else if( le2 < le1 ) {
        result = "Sorry, no one deal "; /* put the message no one deal */
        putBaggage("result",result); /* to Bagge */
    }
} // if
else if(perf.equals("failure")) {
    result = xa.getTag2Value("reason");
    putBaggage("result",result);
}
} // if

// Migrate to BookSellerAgent
migrateBee((String)getBaggage("BookSeller"));

// The 6 state of the MobileAgent
class BeeIPState6 extends BeeIPState implements I_BeeIPState{
    BeeIPState6() { super("REPORT","END"); }

    public void action(){
        // Send XML/ACL
        XmlAcl xa = new XmlAcl();
        xa.setTag2Value("performative","inform");
        xa.setTag2Value("sender",getName());
    }
}
xa.setTag2Value("receiver", "BookSeller");
xa.setTag2Value("content", (String) getBaggage("result"));

if (!sendXML(xa))
    Debug.printLog(getMyname(), "failed to send XML/ACL.");