Intelligent Mobile Agents (Extended Version)

A Dissertation
submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE

in Networked Centred Computing

in the

Faculty of Science

The University of Reading

by

Marios Kalnis

18th September 2001

Prof. Dr. Jörg R. Mühlbacher

Abstract

Intelligent Mobile Agents is probably the most rapidly developing area in the electronic commerce nowadays. Everyday, more and more companies show great interest in getting involved in, or use this new area of technology. Many different areas can use agents, providing great, convenient, reliable and certain way to increase profits.

The aim of this project is to extend the already working Agent System, in order to expand it in the World Wide Web, by working with various other important tools of new technology. We will mainly focus on using Servlets and pure Java™ technology, in order to integrate and modify the already working system, and therefore to provide more flexibility to the system on the Internet, where the great interest is.

Finally, this report provides an overview of the work that took place and detailed information about agents, a well a description of the techniques used in order to overcome various difficulties during the implementation process.

Acknowledgments

I would like to give special thanks to my two colleagues and assistant supervisors, Mr. Michael Sonntag and Mrs. Susanne Reisinger for their continuous help and support on my project and on any of my problems concerning my stay in Linz, throughout my six months Erasmus placement in the University of Linz, Austria.

I would also to thank my superviror, Professor Jörg R. Mühlbacher for his continuous support and concern in each area, whether this is for project, to equipment support and to any other aspects, vital for my stay in Linz and the conclusion of the project.

Also, special thanks to Mr. Rudolf Hörmanseder (or Rudi for everyone at the office) for his concerns about my stay in Linz, his long discussions and coffee brakes during the night, as well as for his beautiful tours and dinners in the city and in the suburbs at the beautiful town of Linz of the county of Upper Austria (Oberösterreich).

Moreover, not to forget Mr. Franz Bauer for his precious help and hardware support throughout my stay in Linz (whatever I asked, I had it).

Finally, I would like to thank all other people of the FIM team, for their continuous support, help and friendly disposition, in helping me improve my poor German and making me feel that I was also part - even for just six months - of their team, the FIM team!

Contents

1.0 Introduction – Intelligent Mobile Agents (Extended Version)

1.1 Overview

The Intelligent Mobile Agents (Extended Version) project aims to give an extension on how to use the agent outside a Java window environment, by using mainly the World Wide Web, to have the same features as the already working Agent System. Moreover, to allow people with no computing experience - the everyday Internet users - to get familiar with this new arising technology, and to benefit from the advantages and features that agents can provide. Therefore, the aim of this project is to create the core model of a mobile (and an immobile) agent that would be extended afterwards, such as to be specifically used for various everyday need tasks.

1.2 Aims

- To modify the already working mobile Agent System, giving the ability to communicate with a web server using Servlets.
- To allow creating other agents through a Web-Interface.
- To create an agent, which can present its state as a web page, to retrieve information through forms, and ability to create its own mobile agent to fulfil tasks at remote computers. Handling of the agent will be done through the web page.
- To create a mobile agent, which will be a subagent of the previous agent, and will be able to move to another Agent System.

1.3 Objectives

- Learning Java and its extra features.
- Programming in Java to work in a web interface environment.
- Learning Servlets, their features and their use.
- Learning how to connect the client-side with the server-side of a system.
- Learning how to acquire information from a web server, how to create a web interface and how to create information forms, which will be able to store the information given by the user.
- Ability to create sockets and protocols in order to achieve the desired communication and exchange of data.

1.4 General Information

The use of Mobile Agents is probably the most interesting area of various businesses nowadays. They are used in diverse major divisions, allowing someone either to be able to acquire statistical results, or to gather information on potential and already existing customers, in order to increase the profits of a company.

On the other hand, since the interest in the Internet is growing rapidly day by day, there is a major need to give an Agent System the ability and flexibility to work either in a software environment, or through the Internet. Several factors have to be taken into consideration, especially in the security area, where transferring data must be done in a secure and reliable way, in order to avoid negative side effects, either by losing data or by exposing confidential information after a hack attack.

This project will focus on creating subclass of the existing Agent System, which will modify the already working Agent System, and they will be able to communicate and work through an

Internet environment, expanding the abilities of the existing system to result in working autonomously.

To start with, it would be good to mention what Mobile Agents are. It is widely used lately but it is not clear what it is. Probably a good fast answer would be that an *"agent is a software package that carries out tasks for others, autonomously without being controlled by its master once the tasks have been delegated. The others may be human users, business processes, workflows or applications"* [18].

In order to make it simpler, it could be a good idea to give a very simple example; let us assume that there is a chef who desperately needs a very good offer on potatoes. He wants to cook 5kgrs. of potatoes on a certain day (let us say Friday) and he wants to find the best price and quality of potatoes on the Internet which they will cost him not more than £10. In this case, the agent's job is to: Find the best price on the internet on potatoes, the quality of potatoes, the cost of delivery as well as to assure that the chef will have the potatoes on Friday and the total cost will not exceed the pre-defined available amount of £10. The agent therefore will have to start searching on the Internet and by combining all these factors, must find the best choice possible. After a successful search, the agent can create a personalised file for the user, of what products he prefers regularly, as well as keep information on products that have already been asked by the user, minimising therefore the time of searching next time and focusing on specific areas from the first time, in the case that the user will have very similar requirements in his/her next purchase, as the previous times.

Of course, in this case, we understand that the opportunities of an online store selling potatoes is not widely used yet, but it was probably a very good example to understand what the agent does. Moreover, in our case, in talking about mobile agents, we have to take into consideration as well the fact that the chef, due to his profession, has to move from town to town, not having therefore the ability to use his own computer for his online shopping, but he has the

chance to use the computer of the restaurant that he will be cooking for. Therefore, the mobile agent gives him the ability to use the system and acquire all the information in the same way, by using the personalised information that he had entered when he was in his home and the same way as if using his personal computer. In this case, as well as in the case mentioned before, the user (client) uses mobile agents. In the first choice it can be either mobile or immobile, where in the second choice that another computer is used, the user (client) must use mobile agent technology in order to acquire the desired information.

Finally, to conclude, from the simple example given above, as we all understand the intelligent mobile agents are probably the future of e-commerce, since they can virtually cover any area of electronic commerce.

1.5 Implementation

The implementation programming language that is used for this project is Java™. Everything is implemented in Java, using at the same time other features of the language, such as Servlets. Also, the implementation of the web interface of this project is done by Java™ Servlets, since they have the ability to create an HTML (Hyper Text Mark-up Language) code, without the use of any **.html* file. Finally, the software that is used for debugging and implementation, as well as for testing, is the Visual Café v.4.0 Expert Edition from Symantec.

1.6 General Information (Agent System)

As we mentioned before, mobile Agents have the ability to work autonomously in any environment, by taking all the information from a specific computer (unit) to any other unit anywhere, giving as a result the advantage to control the information gathered from any terminal securely. For various tasks that require solution, the agent can go around a wide area network, search, acquire and finally collect the desired information and display all of this in our system. In order to achieve this, for our specific case, various security levels are being used, all of them programmed in Java. The most important file of the Java code is probably the "*AgentSystem.java*", which works closely with the files "*AgentThread.java*" and "*AgentThreadStart.java*". The "*AgentSystem.java*" file is responsible for creating the agent, displaying the agent, and creating all other tasks related with the agent's behaviour, such as its unique identity key, its ability to create other agents inside the one pre-defined etc. This file is used as a base. It communicates with any other external platform, and with the combination of the files related with the Agent System, as well as with some modifications, it can produce virtually anything.

In this case we will talk more about the communication of the Agent System with Servlets, and its ability to interact with each other through sockets. The level of communication is based on sending information, such as retrieving general information about the server that we use, as well as the ability to create various number of agents, by sending all this data to the Agent System, and the ability to save, display, delete and move all this data (and therefore the agents). The system will be able to create these data without creating any external file; therefore it will be direct communication with the Agent System code, which has the ability to create the Agents autonomously.

1.7 Description (Tasks)

- *To modify the already working mobile Agent System, giving the ability to communicate with a web server using Servlets.*

In order to implement the above task, Java™ will be used. Also, since we need communication through a web server, Servlets will be used in order to achieve this task. The code using Servlets will be used to make a connection (to communicate) with the main code of the Agent System. Moreover, there will be the ability to view information such as URL details, port details, as well as the ability to create, view, delete, move or customize an agent is feasible, in each case respectively. The first part of the code (for this task) is used as a base code for the other tasks that they will follow.

This task will provide the ability to the user, through a web interface, to access data, to create new agents, to display the agents that already appear in the system, as well as the ability to delete or to view special information about the agent. For all these mentioned before, the user will be able to do them through a web page. Connection will be established between the web server and the Agent System, through the Servlet Agent code, which will send all the data to the Agent System, in order to save them or to send all the important information, when asked, by the user.

- *To create an immobile agent, which allows creating other agents through a Web-Interface.*

Immobile agents are characterized by their autonomous behaviour and by the fact they stay to a specific computer, and if destroyed, cannot be recreated on another remote computer. Contrary to immobile agents, mobile agents have the ability to move from a terminal to another one, and, in order to work successfully, they have to destroy all the files that they have in their

initial system and copy (move) them to their new system. Therefore, mobile agents can be also immobile, if they stop moving from one terminal to the other.

This task aims to create, by using the Agent System code, an immobile agent that will be stable, it will not move to another system, but it will work only from one computer. Therefore, it is the simple version of creating a mobile agent, which its special feature is to create other agents, from the specific computer that it will be used, and it will have the ability to create the new agents through a Web-Interface.

- *To create a mobile agent, which can present its state as a web page, to retrieve information through forms, and ability to fulfil tasks at remote computers. Handling of the agent will be done through the web page.*

In addition to the previous task, in this case the immobile agent will have the ability to move to various terminal systems. As we mentioned before, like in the previous task, it must destroy all the relevant information and files, before it will move to the new system, not leaving behind any files or any info which they involve its presence to the previous system.

1.8 Mobile Agents – Overview

There has been a great interest the past eight to ten years about Intelligent Agents, but what are they actually? A very informal term would be that mobile Agents are personalised, autonomous software, which purpose is, depending on the constraints and requirements of some human intervention, to produce results, after self-directed action, which will be for the human's (it's master) benefit.

To be more specific, Intelligent Agents can cover all the areas of the today's Internet use. They can be used widely in electronic commerce, for purchasing products, as well as in information retrieval, e.g. searching web pages and exchanging information from these, and also for special offers that appear on the Internet worldwide. Instead of that, Intelligent Agents can be used in more personalised software, such as proposing on sending an email or collecting news on your favourite areas, for example in technology.

Intelligent Agents can be used in all areas of transactions, in consumer-to-consumer (C2C), business-to-consumer (B2C) as well as in business-to-business (B2B).

There are two main areas of Intelligent Agents, the Mobile and Immobile Agents, both working on the Internet and on computer networks, by retrieving information, which of them to use depends on the requirements.

First of all, mobile Agents have the ability to search on the Internet, and the characteristic they have is that their thread of execution is terminated from the computer that they were initially created. They move to another networked computer around the world, searching and trying to find a solution for the problem they were given, and if this is the case, and if their user requested it initially, they may return finally to their user and display the result.

On the other hand, an immobile Agent is characterized by the fact that it stays always on the computer that it was initially created in. It collects information around the network and tries to find a solution through this information. The user can always refer to the immobile Agent. The

advantage in this case is, that the user can anytime request a result from the Agent and he/she can expect from it a response, where then this may be positive to the problem or negative. But the fact is that there is a response, in contradiction to the Mobile Agents, where the user will not get any response from the Agent, since the Agent will not exist anymore on the user's personal computer. Immobile Agents do not have the possibility to collect and solve their tasks on other computers, but they can retrieve information throughout the Internet. Of course, there may be the case that they create a mobile Agent by themselves, which will do this job for them, but we will come to that later on.

In order to assure that an Intelligent Agent is successful, some important key components have to be considered. It is certain that effective coordination requires cooperation, which in turn can be achieved through communication and good organization. But in order to succeed developing agent-based applications, it is vital to be able to answer some important questions that arise, such as concerning about the agent architectures and which one should be appropriate for problem solving, or in which way an agent should acquire its knowledge, and how this should be represented, or moreover, how a complex task can be decomposed and allocated to a number of agents, or by which way the agents should cooperate and communicate with each other, and finally, are we confident that an intelligent agent is trusted? The above questions can be seen in the following [18] *figure 1.8.1*.

figure 1.8.1 Multi-Agent Problem-Solving.

Furthermore, the Agents‘ flexibility to work semi (or completely) autonomously, provides them the ability to complete various tasks of electronic commerce, with an initial decision of the human, or, depending on the product, the required approval of the human on completing a proposed task, which could end on providing benefit to the user. There are several examples covering this area, providing for each case (for C2C, B2C and B2B respectively).

To begin with, in a Business-to-Business (B2B) area, it could be that an Agent would be used in order to provide services for buying everyday-use consuming products, such as printing paper. In this case, Agent's task would be, that depending on the daily office use of printing paper, and knowing beforehand the supplies that exist, then it could estimate and order automatically, when necessary (such as the previous quantity of paper would be low), new supplies without the interaction of the human. In this case, no further human external intervention is needed, since printing paper is used in everyday life, therefore the cost for a company is low but the need immediate. In some other cases, where again the interaction is B2B, probably the human's approval would be needed, if this would have to do with an exchange of great interest, such as stocks. Again, an Agent could work autonomously, but a further approval of a human would be needed, because it is on the person's opinion whether he/she would be interested to buy that kind of stocks or not. Also, this is a so-called "untrustworthy decision", since the choice of the Agent cannot be predicted (if it will buy stocks or not) and the agent will therefore not be trusted by its master. In contradiction to that, it would be of trusted, to buy specific stocks at specific prices; there is a great debate on whether a Agent can be characterized a trusted or not.

Furthermore, in the Business to Consumer area (B2C), presumably, the most common type of Agents is semi-autonomously. An Agent is used in order to provide the vital information to a consumer, before he/she finally decides to buy or not to. On the other hand, the Agent tracks the personal information of a user, and if it is the case that he/she will buy something, it will

automatically send this information relevant to the area of purchase, urged him/her to think about it. Also, in case that the consumer does not buy anything, an Agent can track his/her area of search and interest, and by retrieving more details about the consumer's personal record (by creating, for instance, a cookie automatically to the user's computer), it can send to him/her various information about products of interest, that the consumer searched for it in previous visit.

Finally, there is the Consumer-to-Consumer (C2C) area where two or more consumers decide to make exchanges in various sizes and terms. For instance, one user is selling something and another wants to buy, and puts some constraints to his/her personal Agent, such as a price limit and quality. Then the Agent collects all the information that is sent on a local or a wide area network (such as Internet) and stores the information in a database, where any user can view and select to make an offer to that particular user by buying the item of interest. It is something like an agent, who travels all around the net, gathers various types of (but most important, relevant) information to a single location, where a normal user can then logon and view this data, without the need of an extensive search.

On the other hand, as we mentioned before, although immobile or mobile Agents are possible to use in a system, probably the best solution would be to have a more complex combination, providing at the same time more stability, reliability and effectiveness to our system; that would be the case of using both an immobile and a mobile Agent.

To be more specific, an immobile agent would be used initially. The user would use this Agent as a base Agent, which would react in every request made. On the other hand, through this immobile Agent, a further mobile Agent could be created that would be responsible for moving to remote systems in order to retrieve information and to solve various tasks, as requested by its master. With this way, the system is very fast, efficient and robust, since all the requested tasks are solved on other (remote) systems, providing as well the possibility of an immediate, unexpected user's demand of response from the system, due to his/her question, even if the task

(through the mobile Agent) is not completed yet. The procedure described above is represented in *figure 1.8.2*.

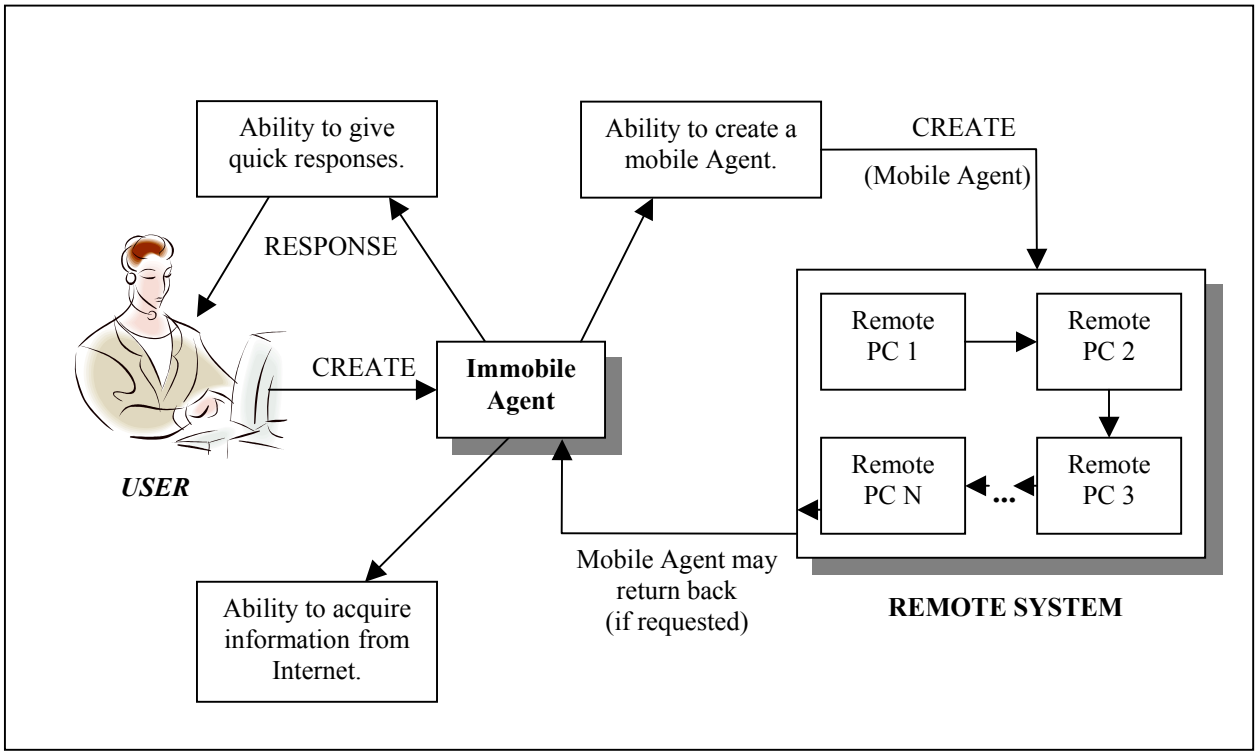


figure 1.8.2 Complex combination, using both immobile and mobile Agents.

Many companies showed from the initial [18] steps of this arising intelligent agent technology great interest in using it, such as IBM, AT&T, Apple, BT, HP, Microsoft and so forth. Many people, from various disciplines are involved in developing or applying this technology, since intelligent agents cover areas, such as robotics, entertainment, knowledge based systems, human-computer interaction, databases, distributed systems, communication networks, cognitive science and psychology. Moreover, several examples of commercial applications or software agents already exist. Probably, the first in using this technology was *Amazon.com*. Agents were used in order to gather information about various books.

The use of Intelligent Agents is virtually unlimited. It can be used mostly in a personalised way, where all the personal data is saved on the user's personal computer. Then the Agent,

depending on the user's interests or requirements, searches and collects everyday new information, which the user might be interested in, and then, he/she can grant the Agent the permission to either do some tasks automatically, and proceed to some other level with his/her final approval, or finally, to complete some tasks, where the agent appears to be the mediator of its master (for the user's decisions).

To conclude, it would be wise to notice for one more time that Agents can be used everywhere, minimizing our work and responsibilities, due to their autonomous way of working. Day by day, Intelligent Agents are being improved; agents have the potential to participate actively in accomplishing tasks, rather than serving passive tools, as do most of the today's applications [7].

2.0 Theory

The main part of theory that was used to design and implement this project, relies on pure Java programming, with specialization in Java Servlets, which is probably the most flexible, robust and reliable way to view all of the information that comes from Java, such as programming techniques and information on databases and security aspects of programming through a URL. Also, in order to achieve communication between the core part of a program, which is usually stored on a server and the client part of a program, which is initiated by every user's personal computer, Sockets were used. Both of these areas will be described in great depth further on.

2.1 Socket communications

As we mentioned before, in order to establish a communication between two different programs, socket technology was extensively used.

In scientific terms, *"socket is a software endpoint that establishes bi-directional communication between a server program and one or more client programs"* [14]. Generally, a socket is nothing more than a conventional abstraction. It represents a connection point into a TCP/IP network. It could be compared to the electrical sockets in each house, where they provide a connection point for the appliances. When two computers want to exchange information, each uses a socket [9]. One computer is termed as a server (it opens a socket and listens for connections) and the other is termed as the client (it calls the server socket to start the connection). Therefore, to establish a connection between the two computers, what is needed is a server's destination address and it's port number. In order to understand the difference between a server and a client program, let us specifically use the "Intelligent Mobile Agents" project.

To begin with, the "Intelligent Mobile Agents" project consists of two main parts of programs; the first part is what so called, server-side, and the other part is the client-side of the whole system.

The first one is the one, which stores all the data, information, unique identity keys as well as the security keys for each Agent respectively. It is the part of the code, which provides all the vital information and techniques, such as an Agent to work semi-or-completely autonomously in an external environment. This part of the code, which can be also used as a standalone application and which consists of several **.java* files, is the AgentSystem and it is the part which has to run all the time, in order to provide the services. We could characterize it as the server-side of a program, since, as a server has to work all the time in order the user to gain information from it, by the same way, the AgentSystem has to work all the time in order to provide the results to the user, whoever that may be.

On the other side, the client-side of the system, is that part of the program which runs on a web page, it only runs upon request and enter of a user, and depending on the user's demands, it provides results and vital other information, usually, through a neatly designed and easy to use HTML page.

Therefore, combining the above information, the socket works by linking the server program with a specific hardware port on the machine where it runs, so that any client program anywhere in the network with a socket associated with that same port, can communicate with the server program (*figure 2.1.1*).

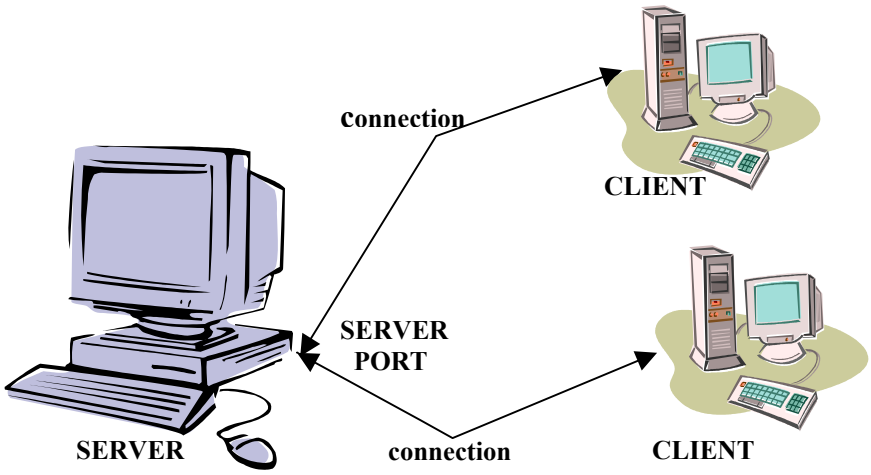


figure 2.1.1 Server-client socket communication (multithreaded in this case).

A server program provides the resources stored in it to a network of client programs. On the other hand, a client program requests this information, the server program responds to these requests, sends back the information (if applicable) and the client program then receives and uses this information.

Moreover, depending from our system's needs, it can serve more than one user at the same time. Of course, there are several ways to handle multiple requests of a system. One of them is to create a multithreaded server program, which will be able to handle more than one client programs.

A thread is a sequence of instructions that run independently of the program and of any other threads. Therefore, when a multithreaded server program exists, it creates one thread for each communication link that is established. That way, it can serve more than one client program simultaneously, since when one communication of a thread is established, it starts the thread for this communication and it continues to listen to requests from other clients, providing the same services to each one respectively.

To be more specific, each computer in a TCP/IP network has a unique IP address. Moreover, parts of that unique IP address are ports, where they are within that IP address and they represent individual connections. Although, in a local network, each computer has many ports, which

share the same address, on the other hand, the port number routs the data that each computer sends or receives within each computer. Therefore, when a socket is created, then it must be associated with a specific port number – this process is known as binding to a port.

2.2 Sockets and transmission modes

Sockets have the disadvantage that they can be used with two modes of operation: connection-oriented and connectionless modes. Connection-oriented sockets operate like a telephone line; they must establish a connection initially, they start transferring the data at the same order as it was send, and at the end, they hang up. On the other hand, connectionless sockets operate like the mail; delivery is not guaranteed, and the pieces contained in the envelope may not arrive in the same order as they were initially sent.

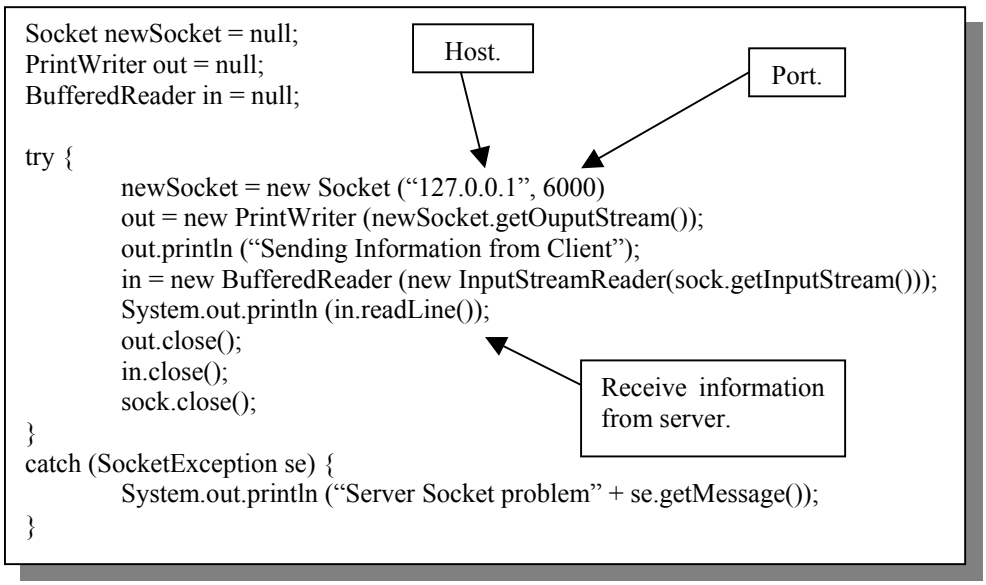
The main difference between these two modes is reliability and speed. In the first case, the data sent to a client or received by a server, is guaranteed to arrive, providing as a disadvantage the fact that there may be a long delay. On the other hand, when connectionless sockets are used, then the transmission is fast but it is not guaranteed that all the information will be received successfully (and vice versa). Usually, connectionless sockets are used to transfer data with fast way, and it can be viewed lately in the area of online voice communication, or in the area of media, online television viewing and radio listening. The quality is not excellent, but the delay is not big; to establish that kind of connection, User Datagram Protocols (UDP) is used.

Since in the “Intelligent Mobile Agents” project, only connection-oriented modes are used, therefore we will only concentrate on describing in more depth this area of sockets. May we also notice, that in order for a socket program to run successfully, the server-side of the program must start first and must keep running, and then the client-side may be connected to the whole system. Otherwise, the system will not be able to work, since on the client-side a request is being made

to a specific port; if this port does not exist, then it is impossible the request to be answered by the system.

2.3 Java™ Socket Programming

The structure of a connection-oriented mode is simple and it follows the same logic and procedure for both sides of a program, on client and on server side. Initially, the programmer has to open the sockets; on the client side, the address of the system must be declared, as well as the port that the system will listen to. On the other hand (on server-side), the listening port must be initiated, in order for the system to be ready to accept the requests from the client. A part of the code concerning the client-side of the program looks like the one in *source code 2.3.1*.



source code 2.3.1 Client-side socket declaration and initiation.

As we can see in the above *source code 2.3.1*, in the beginning the system initiates the name of the socket that it will be using. Also, the read and write method of communication is initiated as well, such as the system to be able to either accept something that it is sent from the server (and therefore read it) or vice versa. After that, the declaration of the address and the port are given (in this case the local host address "127.0.0.1" is used and port 6000). Then, we declare the input and output streams of the system, and then we are ready to send or read a command from

the server. Also, everything is encompassed in a *try* statement, because we also have to consider that if anything may go wrong, to catch the exception (with the *catch* statement) and display the error to the user. In the end, we have to close everything that we initially opened (including the socket).

The server side of the system works in the same way, as we explained before. Again, both input and output streams must be initiated and declared plus the default value of the port (in this case, the port 6000). Then, the server waits to accept the call of the client, and if it is successful it sends back an answer, otherwise it produces an error (through the *catch* statement). The server does not need to declare an address, since it is the client's job to find out where the server is situated and therefore connect to. The server-side code is given in *source code 2.3.2*.

Finally, it would be wise to mention that a socket can be opened in any port that we wish to. The only problem is that it is not possible to use for our own purposes a port that it is by default registered. These ports are the ones that are below 1024, since they are reserved for other purposes of use. Some of the most popular and widely used ports are: Telnet (at port 23), smtp (Single Mail transfer protocol at port 25), http (at port 80), pop3 (at port 110), ftp (at port 21) and so forth.

```
import java.lang.*;
import java.io.*;
import java.net.*;

public class Server {
   public static final int PORT = 6000;

   public static void main(String[]args) {

      ServerSocket serSocket = null;
      Socket sock = null;

       try {
          serSocket = new ServerSocket(PORT);
          while(true) {
             try{
                  sock = serSocket.accept();
                  System.out.println("Server hit by client...");
                  BufferedReader br = new BufferedReader(new InputStreamReader(sock.getInputStream()));
                  PrintWriter out = new PrintWriter(sock.getOutputStream());
                  out.println("Server response");
                  String S = br.readLine();
                  out.close();
                   br.close();
                   sock.close();
                   serSocket.close();
             }
             catch(SocketException se) {
                System.out.println("Server Socket problem "+se.getMessage());
             }
       }//end of external try statement
       catch (Exception e) {
          System.out.println("Couldn't start "+e.getMessage());
       }
   }//end of main
}//end of Server class
```

source code 2.3.2 Server-side socket declaration, initiation and response.

2.4 Java™ Programming / Servlets®

In addition to the sockets that were used extensively in this project, Java Servlets were also used. Further on, we will introduce and describe this new feature of Java programming, which only appeared four years ago, when Java SDK (Software Development Kit) 2 was introduced.

To start with, since Java by itself is characterized by many people as the language of the Internet, and also the characterization that it is given by Sun Microsystems is that "The network is the computer", the philosophy of this new feature was to combine the features of the HTML, together with the new technology that is improved day by day.

It is evident nowadays that client access from the Internet or corporate intranets is a sure way to allow many users to access data and resources easily [5]. The way of connection relies on clients that use World Wide Web standards of access, such as Hypertext Mark-up Language (HTML) and Hypertext Transfer Protocol (HTTP). The Servlet API (Application Programming Interfaces) that will be described later on provides a solution by responding to the HTTP requests.

Of course, these are many traditional ways to provide access to a client, in order to let him/her make HTTP requests, such as updating a database or accessing information from a database through the World Wide Web. The user could fill in some text fields and then he/she could press the "submit" button, where the data that would be submitted, would give information to the server what to do with them, specifying at the same time and the location of the Common Gateway Interface (CGI) program that the server runs. Usually, the most common programming languages for CGI are Perl, Python, C, C++, or generally every standard language, that can read from standard input and write to a standard output. Then, the CGI is responsible for everything else; it will decide whether the information sent to the server was correct or not. If not, it will send back an HTML page to the user, explaining where was the error occurred. Otherwise, it will accept the information, and it will proceed to the next step of the program, which may either be

to provide the user with various information, or to save the data that were sent from the user to the database. Finally, it will produce an HTML page where it will inform the user that the request was successful.

Everything that was described before could be implemented without using Java programming. On the other hand, it could be ideal to use the technology that Java programming provides, probably for some very important reasons, such as: simplicity, flexibility and reluctivity.

Java Servlets technology provides a framework to the programmer that replaces CGI programming, and furthermore, they can provide an excellent solution for server-side programming, since they can display to the user in plain HTML the information, giving also at the same time the opportunity to the programmer to benefit from all of the Java's unique APIs' (except from the ones that produce GUIs', like Swing). Servlet is a generic extension; a Java class can be loaded dynamically to expand the functionality of a server. A Servlet runs inside a Java Virtual Machine (JVM) on the server, therefore it is safe and portable. Finally, Servlets operate solely within the domain of the server, where unlike applets, they do not require support for Java in the web browser.

The Java Servlets will be the future of server-side programming for the following important reasons: portability, power, efficiency, safety, elegance, integration and also, extensibility and flexibility.

To start with portability, Servlets have the advantage that they are highly portable with all operating systems and across server implementations, that's also why Servlets are also known as "write once, serve everywhere" program. Also, in terms of power, Servlets can use the entire core Java APIs': networking and URL access, multithreading, image manipulation, data compression, remote method invocation (RMI), and database connectivity, among others.

As for the efficiency, Servlets have the characteristic that once loaded, they stay in the server's memory as a single object instance. After that, if the server invokes the Servlet to handle a request, since the Servlet is stored in server's memory, it can reply to the request almost immediately. Moreover, Servlets can handle multiple requests at the same time (with the help of threads), making them extremely scalable.

To continue, in safety terms, Servlets can handle errors safely, with the help of Java's exception-handling mechanism. If the Servlet performs an illegal operation, then it throws an exception that can be safely caught and handled by the server, which can politely provide the error to the user and apologize him/her about that. On the other hand, if it was a C++ based server extension and an error was occurred, then it could simply crash the server.

Servlets are also clean, object oriented and amazingly simple. Even if the programmer is not familiar with Servlets, the simplicity of the Servlet API itself can provide him/her much help by including methods and classes, which can handle many of the routines chores of Servlet development. Also, they can be integrated in a simple way, since Servlets can cooperate in ways that CGI program cannot.

Finally, the Servlet API is designed to be easily extensible. For the moment, the API includes classes that are only optimised for HTTP Servlets. But this could be extended further on in order to support and other type of Servlets, either by Sun Microsystems or by some other third party.

Further on, we will give a basic HTTP Servlet, in order to understand the concept of creating Java Servlets.

2.5 What is HTTP, *Get()* and *Post()* requests

To begin, HTTP is a simple, stateless protocol. A client, usually using a web browser, makes a request. Then, the web server responds and displays to the client the result. Initially, when the client sends a request, it specifies an HTTP command, which tells the server the type of action that is requested. It is also specified the URL that this request is occurred, as well as the version that the HTTP protocol is using.

The most common method to request some information from the server is the *Get* method. The user asks from the server to send him some information, which may also be a URL page. For instance, when the user asks for a *mytry.html* page, then he/she sends to the server other additional information, in order to inform and help whether the information that the user requests can be supported by his/her browser (the content-type). If this is successful, then it sends the information to the user (as requested) making and a final check that the transfer was completed successfully.

On the other hand, when we talk about the *Post* request, then in this case this method is used to send information to the server, whether this may has to be to update a database. Contrary to the *Get* method, in this case it may be send a great amount of information, which may be megabytes of size. Therefore, *Post* method uses a socket in order to pass all its data, which can be of unlimited length, directly to the HTTP request body. The most important part is that the information sent to the server is invisible to the user, and the URL does not change at all, by giving the advantage to the *Post* method that it cannot be book marked, emailed, or (in some cases), even reloaded.

Finally, it is important to mention that there are several other methods that exist within the HTTP protocol, although they are not widely used, since we can gain full power of information with the two method (*Get* and *Post*) that we already mentioned. These methods are: *Head*, *Put*, *Delete*, *Trace* and *Options*.

2.6 Creating a simple HTTP Java™ Servlet®

After giving a brief introduction for the use of HTTP and the *Get* and *Post* methods, now it is time to move a step forward and talk specifically of the use of Servlets, by providing a very simple example.

In contrast to a regular Java program, a Servlet does not have a *main()* method. Instead of that, the server in the process of handling requests invokes specifically some certain methods that are related directly with the Servlet. Each time a server serves a request of the Servlet, it calls the Servlet's *service()* method. This method consists of several other subclasses, like the ones mentioned above (*doGet*, *doPost* etc.). A figure describing the process already mentioned [5] is given below (*figure 2.6.1*).

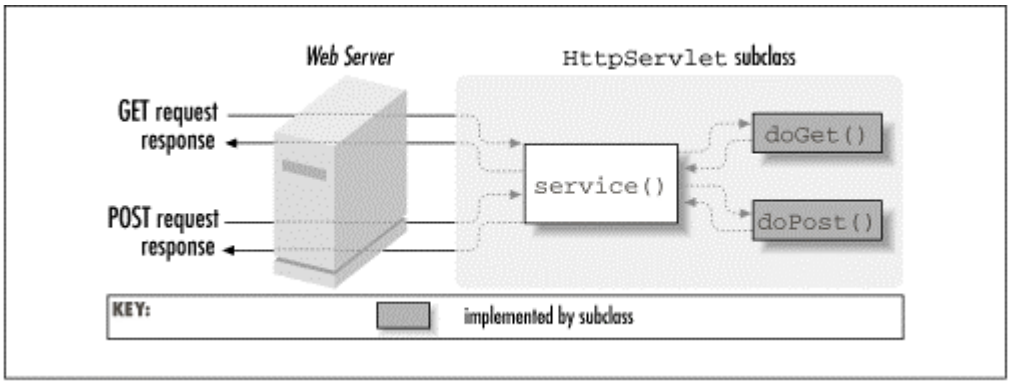


figure 2.6.1 An HTTP Servlet handling GET and POST requests.

Therefore, in order to use the Servlets, we have to import some of the libraries that they related directly with Servlets: javax.servlet and *javax.servlet.http*. Any other basic libraries are important to be declared initially, but also these two. The *javax* indicates that the Servlet API is a standard extension. Also, the two library packages mentioned above contain classes to support generic, protocol-independent Servlets.

To understand everything more clearly, it is better to show a simple example, called *HelloWorld.java*, which will help us to get more familiar with the way that Servlets are implemented and work. Please note that in this example the use of the *doPost()* method is not

visible for two main reasons: If we just need to request something from the server, then it is redundant to use this method, and furthermore, it is also possible to get out with it and use only the d*oGet()* method, if the data that we plan to send are not in terms of Mbytes, but only in Bytes or Kbytes.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException {

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<HTML>");
    out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
    out.println("<BODY>");
    out.println("<H1>Hello World</H1>");
    out.println("</BODY>");
    out.println("</HTML>");
  }
}
```

source code 2.6.1 Creating a Servlet that prints „Hello World“.

Looking in detail the above source code, initially all the relevant libraries have to be imported, the basic and the ones that they are important for use by the Servlets. Then, the main class of the system extends the HttpServlet and overrides the *doGet()* method that it came to it from the method mentioned before. Then, the *doGet()* method calls two objects, the “HttpServletRequest” and the “HttpServletResponse”, where they represent the client’s request and response respectively.

About the “HttpServletRequest”, this object gives access to a Servlet with this information about the client, as well as the parameters of the request. On the other hand, the

"HttpServletResponse" object is used in order to give the ability to the Servlet to send information to the server. Of course, since we need compatibility with the server's responses to display everything successfully to our browser, the *setContentType()* method is used, in order to provide compatibility with the HTML (content type response therefore to "text/html"). After that, the *getWriter()* method is used, in order to retrieve a PrintWriter and convert the Java's unicode to a locale-specific encoding. Finally, the Servlet from the PrintWriter's variable "out", uses it to create an HTML content which will represent it to the user's web browser the "Hello World" page.

Finally, it would be good to mention that in this case, since the Servlet will only display to us a "Hello World" message, whatever our action will be, since the requested text, when visiting the specific URL (where our Servlet will be stored on the server), it will only display this simple HTML page.

3.0 Primary Software Design

As we already mentioned in the *Introduction* section of this report, the main aim of this project was to create a user-friendly, web-browser type of communication between a client-side and a server-side program (the Agent System in this case).

Initially, the whole structure of the system was designed, in order to help us get familiar with what was required to implement.

The user views an HTML page with the initial fields that he/she will have access later on. In each of the cases the server has to be contacted, which has to verify the case requested, and then display the resulting page to the user. Every choice must be done in conjunction with the *ServletAgent.java* and the *AgentThread.java* code respectively, where the first has to create the URL with the choices and the second one has to give approval of the choices requested. A rough sketch, related to what described so far, is given below (*figure 3.1*).

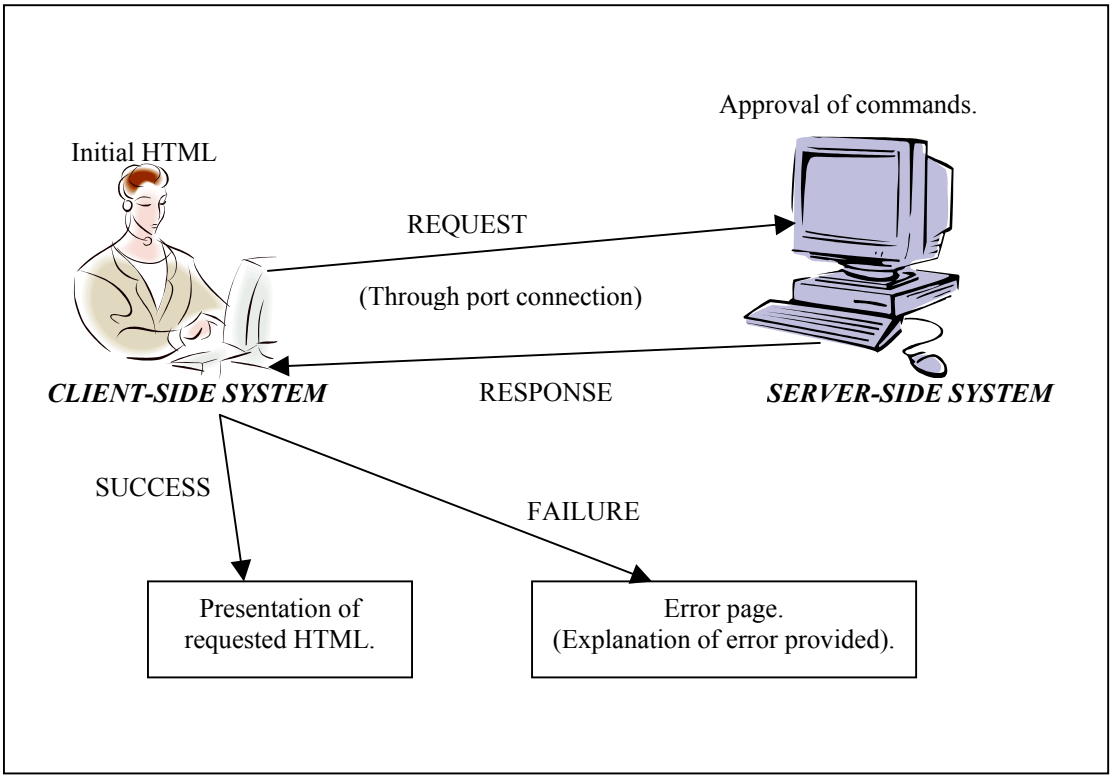


figure 3.1 Sketch representing how (in general terms) Agent System works.

On the other hand, since the above figure represents the final version of the layout, it would be better to describe how everything works, by giving a detailed diagram, which shows all the possible choices that the user has (*figure 3.2*). By the same way, as the client-side of the system works (*ServletAgent.java* code), the server-side follows the same procedure and steps, since it responds to the client's requests.

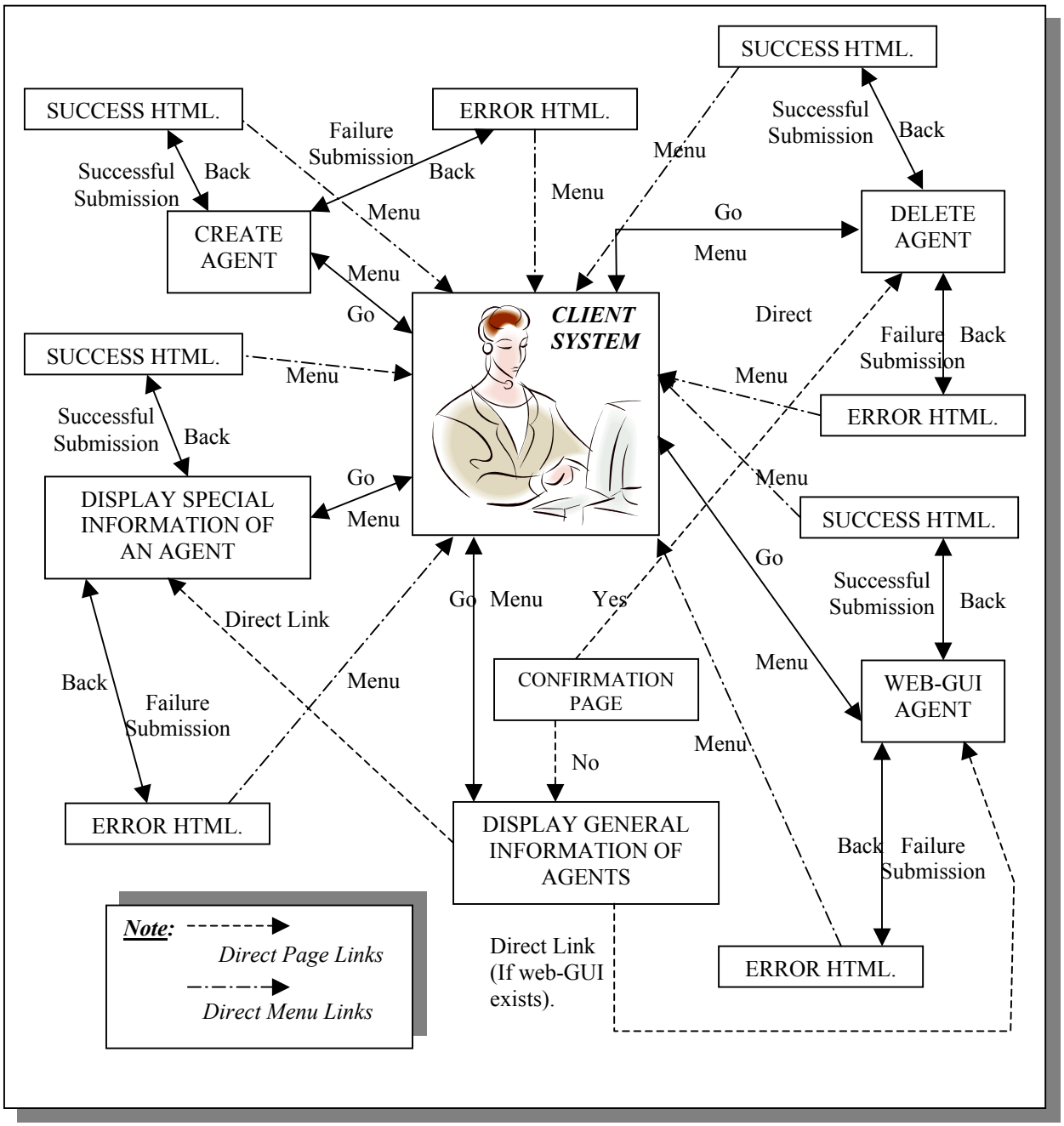


figure 3.2 Detailed diagram of the client-side of the Agent System.

To start with, the client-side of the program stores all the vital information, such as commands and direct communication with the server, providing with that way various HTML pages, each one depending on the status of the response of the server, since the server is the one that has to decide if a request is possible completed, or not.

Thereafter, the system has (by its own) to consider all the choices that must be made and collected. The program stores all those cases the user can choose. Depending on the ability to create for example an agent, the system has to communicate with the agent system (*AgentThread.java* code), in order to check and, if positive, to allow the creation of the agent requested. Therefore, the client-side is the one, which sends all the information that the user requests, and the server-side is the one that decides whether it is possible to fulfil the request, or not. If yes, then it sends back again the positive answer to the client-side, where this system is then responsible to display the successful response to the user, like, either a "Thank You" page, or (if the request is to display information), the fields requested. All the information is provided by the Agent System, therefore the communication between the two systems is direct; that is why the unique port is used, in order to send the information to the server and vice versa.

The user has interesting flexibilities on the system. Initially, the user can create an agent, only if all the fields (in this case, all of them are required) are filled in. The only case where the system will deny creating an agent is the case where some other agent with the same unique ID already exists on the system, if not all the fields are filled by the user, or the code could not be found. Again, in this case, the system will be responsible to inform the user where the error occurred (in this case, that there is a duplication of an ID), so that the user will be able to modify his/her request to create his/her unique agent.

Moreover, the system can then display in three different cases the agents: It is able to display all the agents already on the system, or, if the user requests, to display from each agent respectively, the detailed information, as well as giving the basic information of the agent to the

user, and to allow him/her to move the specific agent to a remote system, providing, of course, the destination URL. In order to make it more interesting in this case, the user can also send a comment, a picture and a title to the destination address. All of this information is displayed at the remote computer.

Since, the system is characterized mostly for its flexibility and simplicity, the user can also view the information of each agent through special links from the general display information page. The user can just click on these links, avoiding by that way the problem of remembering the unique ID, which may vary from 2 simple numbers to a whole line of different combination of characters and numbers. Therefore, avoiding all this hassle is very important for the user who wants to view the information of his/her agent immediately.

One other feature that was added to the extension of the Agent System is, that the system can distinguish whether the user created an agent with or without a web-interface. In this case, the Agent System is responsible to identify the kind of the agent by the interfaces implemented by the agent.

Moreover, the user (when having installed the Java Runtime) can use at the same time both the browser and the Java window version of the system. Therefore the system gives the ability to the user to create, either from the Java window or from the browser, an agent and thereafter displaying the information from any system. Also, it provides (in the case of moving the mobile agent to a remote system) the flexibility to move the mobile agent from the java window or the browser to the remote system, and to send it back to the system. In this case, the only possibility is from the remote system, through a java window. May we also notice, that the remote computer may only be a java window, but could be accessible through a web browser if a different web server was used.

Finally, each exception on the system is caught in order not to produce a run time error, which would freeze the system. Therefore, in any case, the user (and more important, the

developer) is able to view any error occurred that on the system, since the system is able to report the problems and display warning pages without the need of rebooting the system (avoiding the problem of null pointer exception and overflow). In any case the software will display something, while abandoning the procedure of the request.

4.0 Implementation Strategy

From the beginning of the project, Java™ programming language and all of its components was decided that would be used, since the initial version of Intelligent Mobile Agents was implemented in Java™.

Although it is evident that there are some other programming languages like TCL, where it is feasible to create and program an Intelligent Mobile Agent, on the other hand, Java™ is probably the most recent programming language, with enormous capabilities and flexibility in terms of compatibility and ability to be easily extended.

To be more specific, Java™ is portable. It is very accurately characterised as the programming language of the Internet, since it is compatible with all the areas that are involved on the Internet; since Internet is a WAN (a Wide Area Network, and therefore network of several computer terminals), the need of a language that combines flexibility, productivity, as well as compatibility with other software, in terms of expanding the capabilities, is immediate.

Java™ has the ability to provide all of the above features completed in one package, providing also simplicity in programming. But, let us now explain in great depth, why Servlets were used for this project, as well as why only the specific two methods of this new tool of the Java™ programming language (the *doGet()* and *doPost()* methods) were used.

As we described in the *Theory* section of this report, Servlets are server-side includes, which have the ability to communicate directly with a server, and to provide to a user (through the Internet), dynamically generated content, depending on the user's actions and requests. For example, if a user requests some information, then it is Servlet's responsibility to ask from the server if it is permitted to do so (or to check if this information exist on the server), and, depending on the server's answer, to allow the Servlet to retrieve information from the server and display it back to the user in neatly HTML. Therefore, the direct communication with a server, as well as the capability to generate an HTML page, with only optional intervention of an

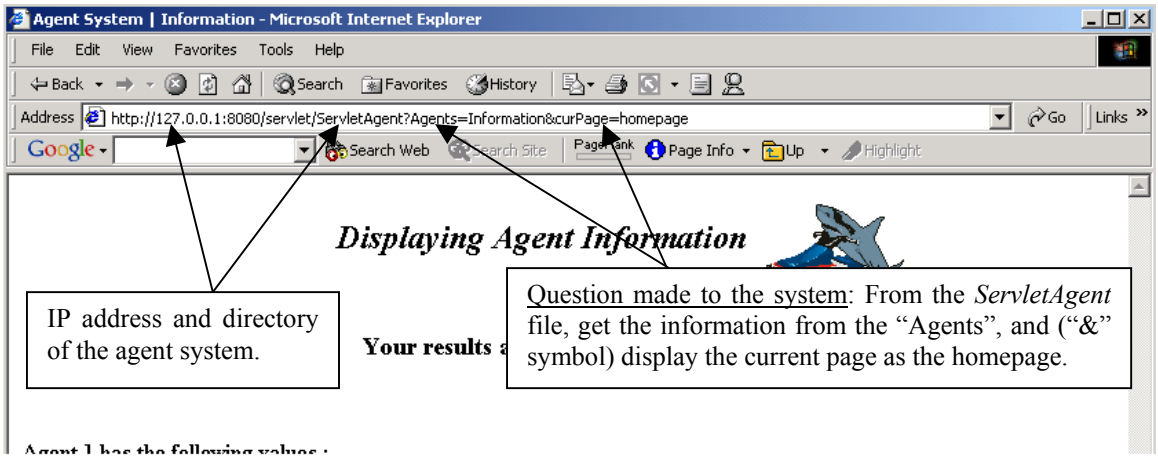
HTML code generated individually (as an **.htm* or **.html* file), is one of its main advantages. Moreover, Servlets can communicate and cooperate directly with most databases.

There are many methods that can be used within a Servlet program (depending on our needs), although the most common methods are the *doGet()* and *doPost()*, which are also the ones that have been also used in the "Intelligent Mobile Agents – Extended Version" project. Specifically, in this project, the *doGet()* method did the most important job instead of the *doPost()* and the reason will be explained later on.

As we described in the *Theory* section and specifically in the section about the Servlets (in this report), they have the characteristic to receive and send (usually, by using a form) data from and to a server. Moreover, depending on the size of the data sent, both *doGet()* and *doPost()* methods can be used. Usually, in terms of simplicity (it is usually simpler to work with one method, rather than with two), the *doGet()* method is used, since it is possible to both acquire data (make requests) or to submit data (send information to the server). Of course, especially for the second feature (to send data), it is feasible in terms of bytes, in contradiction to the *doPost()* method, which can be calculated in terms of megabytes or more.

On the other hand, this project's aim is to allow us to send typical information on a server-side program, in order to create a basic mobile agent. More personalised information is not included when submitting a form, therefore the size of data being sent cannot exceed, in any case, more than a few bytes.

The use of *doGet()* method allows us to make simple, straight requests to the server. A simple if-then-else statement is used in order to distinguish the various cases, where requests are made. Then, depending on the request, the browser (in the address area of the web browser) represents the request we made to the server; between the "?", "&", or "%" symbols that we usually come across; the constraints that the system took under consideration are also existent (*screenshot 4.1*).



screenshot 4.1 Explanation on how Servlets work, as displayed on the browser window.

Moreover, the *doGet()* method is responsible for informing the user about the status of the request. If it is a successful submission, then the user will receive a positive verification, otherwise an error will be displayed to him/her, informing also where the error occurred and why; therefore, it is even more user-friendly than a pure HTML code, since, its dynamically procedure, can allow the system to detect (or *catch*, as it is called in Java™) any errors and display them to the user in HTML format.

To conclude, the *doGet()* method (and generally, the Servlets) use HTML and text format. In this way, it provides the user with the ability to view its content in any browser, and therefore, in any type of browser output.

5.0 Detailed Software Design

The concept for the extension of the initial version of the Agent System was to extend the already working system to be able to accept access through a user-friendly Hyper Text Modelling Language (HTML) page.

The initial version of the Agent System can provide exactly the same information as the extension of it. The only problem is that in order to view the graphical user interface of the initial version, the user must include all the appropriate software in order to start the sofware. Also, the software is not compatible with a browser; therefore a user, who is not familiar with programming and the only way that he/she uses a computer, is for general purposes, could have great difficulties on how to use the system. Therefore, a browser version was decided to implement, such that users who regularly use the Internet, will be able to make full use of an immobile or mobile agent of their own. The concept was to create a web interface software, which will be easy to comprehend and easy to use, as if it was just a simple Internet page.

Initially, in order to design how the software would be implemented, the already working Agent System had to be examined. To start with, the Agent System consists of several different **.java* files, each one of which is interrelated with each other. The main code of the program is included in the *AgentSystem.java* file, where all the classes, imports and packages are used to give the final result. The *AgentSystem.java* is responsible to create a security key, to create hash tables to store the data, to create, delete and view general and special information of an Agent, to send an Agent to another remote computer (when creating initially a Mobile Agent), as well as to present all of this information to the user on a neatly designed java window, with the help of the *GUIAgentSystem.java* file. May we also notice, that there are also several other areas where this file can provide important help and information, but they are not related closely with this project (therefore, they are not mentioned).

On the other hand, the expansion of the initial version of the Agent System consists of several different **.java* files, where some of them are related directly with the *AgentSystem.java* file and the other are connected to the *AgentSystem.java* file through sockets.

To be more specific, the extension consists of the following files (as provided and viewed in *Appendices* section): *ServletAgent.java*, *AgentThread.java*, *AgentThreadStart.java*, *WebAgentInterface.java*, as well as contains some methods in the *MovingAgent.java* file. The first file works individually and retrieves information of the main *AgentSystem.java* file, where all the other are related to the *AgentSystem.java* file, but they are especially created to be used in conjunction with the *ServletAgent.java* and to represent its status autonomously, as separate programs. They are created in that way, that the *AgentSystem.java* will be used as the base code, where other part of code can be joined with it and implement various different tasks. In our case, the three parts of different code are used in order to help us communicate with the main system of the agent, retrieve different information and display them on a neatly designed HTML page on a specific Universal Resource Locator (URL).

But in order to understand clearly how everything works on the system, and how the system sends and retrieves information, as well as cooperates with its parts, we will initially analyse the part of the code that is connected directly to the *AgentSystem.java* code.

5.1 Detailed software description for *AgentThread.java*

To start with, one of the codes connected to the *AgentSystem.java* file, is the *AgentThread.java* file. This file's job is to create a new Thread, which will be used for extending the Agent System, so that it will work through a browser, allowing the user to have full access (at the same time) on the initial version of the Agent System. In order to specify the way that it will be connected with the other part of the code, it creates and opens a port at a specified

location (in our case, port 6000 is used) and listens for connections from the client side of the system. Not to forget that this part of the code is the so-called server-side of the program. Therefore, this part will be responsible for giving feedback whether a procedure was successful, or whether a request from the client-side of the system will be feasible to be completed.

Initially, the *AgentThread.java* creates a class called "*public class AgentThread extends Thread"*. This class is the only one that this file has. It extends the general Thread class, as it is found on the imports that are being made in that file. The *AgentThread.java* also creates a communication protocol between the server-side and the client-side. This protocol contains six different constants, all of them of the type integer. They are used in order to specify the name of the action of each case, providing also a unique number to each of them. Also, as a global variable of the system, the integer *port* is given, the specific port, which will be used generally throughout the system, in order to listen for connection from the client-side of the system. Since each protocol needs to use a specific port to complete its actions, therefore, the declaration of the port has to be global. The part of the code that it is related with the declaration of the six constants, as mentioned above, is shown in *figure 5.1.1*.

```
private final int DisplayAgents = 0;
private final int CreateAgents = 1;
private final int DeleteAgents = 2;
private final int DisplaySpecialInfo = 3;
private final int DisplayAgentWebInterface = 4;
private final int SendAgentToRemoteComputer = 5;
private int state;
protected int port;
```

figure 5.1.1 Declaration of protocol variables (*AgentThread.java*).

Then, the thread constructor of the program must be created. This will help us to open a new thread for this system, to create a constructor that will listen to the specific port (port 6000 in our case) and it will complete our tasks. The protected method *servletThreadAgentSystem* that it is

used in the *AgentThread.java* file is closely related with the *AgentSystem.java* file. Initially, in the global declaration, it is initialised to a null value, but thereafter it is mainly used to provide access to the system, to either start an agent or terminate it (*figure 5.1.2*).

```
public AgentThread(AgentSystem sys, int port) {
        servletThreadAgentSystem = sys;
           System.out.println("Creating AgentThread Constructor");
        this.port=port;
}
```

figure 5.1.2 Creating the Agent Thread constructor for the system (*AgentThread.java*).

Further on, the socket has to be opened in order to accept the connection from the client-side of the system. Therefore, a socket connection is created, which will listen and wait for a client's connection. This socket is configured to accept (in this case) requests at port 6000, same as the client-side of the system (the *ServletAgent.java* file). This new socket provides the ability to the system to read from the socket and also print out the results through it (for sending information).

After that, a *while-loop* is used in order to consider all the choices defined in *figure 5.1.1*. In this loop, the port is opened in order to accept the connection. Moreover, all the choices are being considered in order to decide what to do in each case respectively. The socket is opened only once and it is closed, when all the different cases of the protocol's choices are being completed. The socket can be assumed that it is opened on that port for all the time, since, if it runs on a server, which does not shut down at all, it remains open and responds and serves to any of the client's (whoever that may be) requests. In that way, if a client revisits the server, then he/she can retrieve and view all the information of the Agent that he/she created. On the other hand, if the server crashes or shuts down, then all the information stored on the system is lost, since the data is only kept in the system. Opening of the port is shown in *figure 5.1.3*.

```
try {
        /**
         * The AgentThread() class (server-side) accepts the ServletAgent() class
         * request (client-side) to open the socket at PORT 6000.
         *
         * out The stream to write out
         * br The stream to read in
         */
        sock = serSocket.accept();
        System.out.println("Client connected to Server");
        BufferedReader br = new BufferedReader(new InputStreamReader(sock.getInputStream()));
        PrintWriter out = new PrintWriter(sock.getOutputStream());

        try {
                String S = br.readLine();
                out.println("AgentSystem responding");
                out.flush();
```

figure 5.1.3 Part of code, representing the declaration of the socket (*AgentThread.java*).

Moreover, in Java™, it must always be enclosed in a *try-catch* statement, since an exception may happen in our code, whereupon the relevant output to the user is created. This technique helps the user to understand what and where a problem occurred during the use of the program.

To continue, the code will be explained in hierarchical view and the points that have more interest will be displayed in figures.

To start with, this code follows the same procedure as the information displayed to the user. Therefore, the first choice is when the user chooses to display the agents that exist (if any) on the system. In this case, the most important of the server-side code is the way the system reads and checks the values, in order to display the relevant information to the user. In order to do that, the enumeration method of a hash table is used, which checks one-by-one the IPs' of the agents that shall be displayed to the user (*figure 5.1.4*). At the end, it uses an if-else statement in order to send the correct information to the client-side of the system (the *ServletAgent.java* file), whether the agent implements the web interface or not.

```
Enumeration enum=servletThreadAgentSystem.agents_by_id.keys();
while(enum.hasMoreElements())
{
   AgentIdentity id=(AgentIdentity)enum.nextElement();
   out.println(id.getAgentName());
   out.println(id.getID());
   out.println(id.getHomeAddress().toString());
   out.println(id.getOwnerName());
   boolean res=(((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).agent instanceof
                    WebAgentInterface);

        if (res == true)
          {
             out.println("TRUE with Agent");
```

figure 5.1.4 Part of code, displaying the information that is sent to client-side system (*AgentThread.java*).

On the other hand, if the user chooses to create an agent, then in this case this file's job is to distinguish whether all the fields are filled in by the user, reading all the values that correspond with the fields, and if everything is correct and the creation was successful, it then sends a *success* line. Otherwise, it sends an *error* line, where upon the relevant HTML page is activated and displayed to the user (*figure5.1.5*).

```
else if (state == CreateAgents) {
     try
       {
        String AgentName = br.readLine();
        String ID = br.readLine();
        String HomeAddress = br.readLine();
        String OwnerName = br.readLine();
        String className = br.readLine();
        AgentIdentity id = new AgentIdentity(AgentName,ID,new URL(HomeAddress),OwnerName);
        servletThreadAgentSystem.startAgent(className,null,false,id,null,null,true);
        out.println("SUCCESS of creating an agent!!!");
     }
     catch(Exception e)
     {
         out.println("ERROR: "+e.toString());
     }

}
```

figure 5.1.5 Actual code for the creation of an agent (*AgentThread.java*).

When the user chooses to delete an agent, the same procedure as the display of an agent is followed; the use must provide the unique ID to the system, in order for the system to allow him/her to delete the specific agent. Moreover, the server-side system has to check, whether more than one agents exist on the system, in order to manage to find the specific one and therefore to delete it. So, again the technique of the enumeration is used, for retrieving the agent of the unique ID and delete it from the system. In this case, the only extra feature that it is used, is the calling of the *stopAgent* method, which is implemented in the *AgentSystem.java* file, in order to destroy the specified agent. Of course, again, if the ID is wrong, then a problem occurs and an error page is displayed to the user with the help of the *error* line that it is sent to the client-side system.

After that, if the user chooses to display *special information of an agent*, exactly the same procedure is followed, since, again the system requires from the user to put the unique ID to the system in order to allow him/her to view the special information. In order for the system to display all the special information, a counting technique is used (since there may be the case that multiple lines of information are displayed to the user).

Furthermore, if the user wants to display the *web-GUI of an Agent*, then the system again prompts the user to enter the unique ID, by the same way as mentioned before. Also, in order to read all the information provided by the agent, the code counts the lines to pass to the Servlet, to assure that all the information will be saved and later on displayed to the user. Finally, again the same technique of detecting an error or a success to the submission is used in order to display (through the client-side code) the error or the success page to the user respectively.

5.2 Detailed software description for *AgentThreadStart.java*

The *AgentThreadStart.java* file is part of the *AgentThread.java* file, and it works closely with the *AgentSystem.java*. This file is responsible of calling all the related methods from the AgentSystem, in order to give the ability to the system, to work simultaneously, either in a web-browser environment, or in a Java window (as appears in the initial version of the Agent System). The constructors in this file (*figure 5.2.*1) include all the parameters of the Agent System, as well as the port, where the new system listens (in this case, port 6000).

```
public AgentThreadStart(int port, int PORT, String libraryDir, String agentDir, Boolean
                                    exitOnLast, String savedFile)
   {
      super(port, libraryDir, agentDir, exitOnLast, savedFile);
      myPort=PORT;
   }
```

```
protected AgentThreadStart(int port, int PORT, String libraryDir, String agentDir,
                                    boolean exitOnLast, String savedFile, boolean useSecurity)
   {
      super(port, libraryDir, agentDir, exitOnLast, savedFile, useSecurity);
      myPort=PORT;
   }
```

figure 5.2.1 Constructors of *AgentThreadStart*, subclass *AgentSystem* (*AgentThreadStart.java*).

Thereafter, the important declaration of the main method is used, which includes all the information such as to allow incoming requests and the ability to handle them. In this case, only requests for receiving an agent are allowed.

Concluding all the above, it is important to notice that this file contains similar (and sometimes exactly same information) as the *AgentSystem.java*. The main difference is that this file includes the new port of the system, as well as gives the ability to work in conjunction with the *AgentSystem.java* code, without changing the initial structure of the base code.

5.3 Detailed software description for *WebAgentInterface.java*

Taking under consideration the description of this file, the only use of it is to give the Agent System ability to distinguish whether an agent has the ability to present a web interface or not. Therefore, the *WebAgentInterface.java* file uses one only short method, in which a parameter of a hash table is used in such a way so that to be able to accept various parameters and therefore to display an HTML page in response to a form, including the various data that have been sent by the user. The actual (complete) code of this file is given in *figure 5.3.1*.

```
package PkgAgentSystem;

import java.util.Hashtable;

public interface WebAgentInterface
{
   public String getHTMLPage(Hashtable parameters);
}
```

figure 5.3.1 The actual code (*WebAgentInterface.java*).

5.4 Detailed software description for *MovingAgent.java*

The *MovingAgent.java* file is responsible to create initially the form, inside a mobile agent. Thereafter, if the user decides to fill in the form (by also providing a URL), then the system is able to send the specific mobile agent to a remote system, to the address entered.

Initially, this file did nothing more than creating a Java™ interface, in order to display the mobile agent that it has been created in the AgentSystem. Then, the only thing this file does, is that it displays its status (Name and ID of Agent is already decided in creation process), together with a button. This button, when clicked, produces a “beep” sound.

On the other hand, by adding two new methods in the *MovingAgent.java* file, the whole structure provides us new capabilities, which lead to use this type of agent (which is also called, Beeper Agent), as a mobile agent, which we will be able to send to another (remote) computer.

The *MovingAgent.java* file uses the *public String getHTMLPage(Hashtable parameters)* method. This method has the characteristic that it communicates with the *WebAgentInterface.java* file. This file creates an interface for this system, incorporating the information of the method mentioned above. In a normal web page creates a form on a web page, where the user has to fill in the information (for this agent).

Furthermore, in the *MovingAgent.java* file, the command "*Form*" is used in order to give the ability to the user to use submission and other vital buttons. Also, the declaration of the "*Action*" is given in there, in order to define the action of the agent that has to take place, providing with this way the unique ID of the agent. The system will then be able to understand which agent to send the information entered by the user to. Part of the code, showing how the (HTML format) form is created, is given in *figure 5.4.1*.

```
public String getHTMLPage(Hashtable parameters)
   {
      if(parameters==null)
      {
       String str= "<FORM METHOD=\"GET\"" +
       "ACTION=\"/Agent/"+getIdentity().getID()+"/\"><CENTER><H2><I>Welcome to the WebGUI
        Agent</I></H2></CENTER><BR>"+ "<H4>Please, fill in the form below</H4><BR>" +
        "<CENTER><TABLE BORDER=0 WIDTH=440>" +
        "<TR><TD ALIGN=right WIDTH=120>Title : </TD>" +
        "<TD ALIGN=right WIDTH=320>" +
        "<INPUT TYPE=text NAME=Title VALUE=\"\" SIZE=40 MAXLENGTH=40></TD></TR>" +
        "<TR><TD ALIGN=right WIDTH=120>AgentSystem's URL : </TD>" +
        "<TD ALIGN=right WIDTH=320>"+"<FONT COLOR=blue>amp://</FONT>"+
```

figure 5.4.1 Part of the code, for displaying an HTML form to the user to send an agent to a remote computer (*MovingAgent.java*).

After submitting the form, the system is responsible of verifying that the agent has been sent to the remote system successfully, in order to display thereafter the relevant information to the user. The user can then check whether the specific agent still exists on the system, or not (it should not, as it was moved to the other system).

Afterwards the system, in order to show the user that the agent has been sent (or not) to the remote system, presents either a successful or an error page to the user. These pages follow the same logic as the previous case (*figure 5.4.1*).

The methods that were used are: *goDestination()*, *onAfterMoving()* and *showMessage()*, which check the destination the user provided, the status of the system after moving to the remote system, as well as the system's duty to display the relevant message (images) to the user respectively. Part of the code related with the *showMessage()* method is given in *figure 5.4.2*.

```
private void showMessage()
   {
      // Representing the images of the system, depending on the user's choice.
      String imageName=null;
      if("GUIImageOne".equals(whichImage))
         imageName="BaseAgents/Beeper/agent.gif";
      else if("GUIImageTwo".equals(whichImage))
         imageName="BaseAgents/Beeper/Fimlogo.gif";
      else if("GUIImageThree".equals(whichImage))
         imageName="BaseAgents/Beeper/latest_eit_195.gif";
      else if("GUIImageFour".equals(whichImage))
         imageName="BaseAgents/Beeper/latest_eit_304.gif";
```

figure 5.4.2 Part of the *showMessage()* method (*MovingAgent.java*).

5.5 Detailed software description for *ServletAgent.java*

Probably, the most important file in our system is the *ServletAgent.java*, and the reason is because this one is responsible for creating the whole web interface for the Agent System. It is also responsible to view, create and delete agents, as well as to display the web interface of an agent. The *ServletAgent.java* works closely with the *AgentThread.java* file, and therefore with the *AgentSystem.java*.

At the beginning of the code, all the relevant imports of libraries and all the variables are declared, (which are global) the ones that are used throughout the code. Moreover, the declaration of the constant values of the port and URL address (which in this case, port 6000 and the local host address are used) is done globally, in order to give the chance to the programmer, in case of an upgrade (or creation of the system at some other URL) to easily change everything without looking through the whole code.

The main class that this file uses has the same name as the **.java* file, and it is called *ServletAgent*. This class consists of two main methods, the *doGet()* and the *doPost()* method respectively. The *doPost()* method is only used in order to call the *doGet()* method, where the *doGet()* is responsible for the normal operation of the whole Servlet system. In the *doGet()* method, the Servlet's requests and responses objects are included, to give the programmer the ability to acquire or send information to the server (and therefore the user). The declaration of this class, as well as for the two main methods mentioned above, is given below (*figure 5.5.1*).

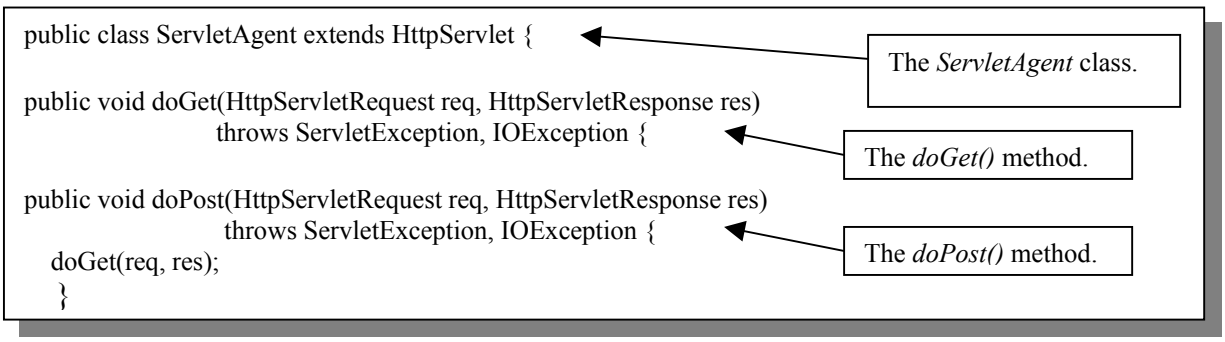


figure 5.5.1 The main class and the two methods used (*ServletAgent.java*).

All the variables used in the system are initialised and declared globally. Then, within the *doGet()* method, the six different choices are being considered; in each case, new ports are opened (always at the same port number) and closed afterwards, when the command requested is completed. There are several other important facts and innovations for each case which will be explained further on. May we also notice that in order to have successful communication between the server (*AgentThread.java*) and the client (*ServletAgent.java*) side, the same number of input/output calls (reading and writing or vice versa, respectively) must be done in order for the system to function correctly. Further on, we will describe the most important points appearing in this **.java* file, considering each part of the requests made by the user respectively.

To start with, we will consider the case where the user wants to display if any agent exists on the system. Here, the system takes under consideration the variables that will be used in order to display the general information of the agent. The variables are initialised, and after that, since in the first case the need of counting the number of agents that exist on the system is vital, the system starts counting the information that it receives (reads) from the server-side system. When the system is done with retrieving all the data, it produces an output to the user with all the information (if any) in an HTML format (*figure 5.5.2*).

```
size_of_Agents = Integer.parseInt(br.readLine());
info_of_AgentName = new String[size_of_Agents];
info_of_AgentID = new String[size_of_Agents];
info_of_AgentURL = new String[size_of_Agents];
info_of_OwnerName = new String[size_of_Agents];
info_of_AgentWebInterface = new String[size_of_Agents];

/**
 * Using a "for" loop to read each time the four strings that each agent
 * has respectively.
 */
for (int i=0; i<size_of_Agents; i++)
{
      info_of_AgentName[i] = br.readLine();
      info_of_AgentID[i] = br.readLine();
      info_of_AgentURL[i] = br.readLine();
      info_of_OwnerName[i] = br.readLine();
```

figure 5.5.2 Part of code, displaying the counting of the five different strings (*ServletAgent.java*).

On the other hand, if the user chooses to create an agent, then the system opens again a socket (at the same location, at port 6000), and sends the fields required to the Agent System (*figure 5.5.3*). Then the system posts them to the *AgentThread*, from where the system checks whether they are correct or not (i.e. if the ID is unique and not used before). In any case, it provides either a successful or an error page respectively, displaying as well the information in every case (*figure 5.5.4*).

```
else if ("Create".equals(req.getParameter("Agents")))
                {
                    out.println("<HTML>");
                    out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                    out.println("<HEAD><TITLE>Agent System | Creating Agent</TITLE></HEAD>");
                    out.println("<BODY>");
                    out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                                WIDTH=300><TR><TD>");
                    out.println("<TD ALIGN=right WIDTH=200><H2><I>Creating Agent</I></H2>");
                    out.println("</TD><TD ALIGN=left WIDTH=100>");
                    out.println("<A HREF=?curPage=getAboutPage>");
```

Link to „About“ page of the system.

```
                    out.println("Agent's Name :");
                    out.println("</TD><TD ALIGN=right>");
                    out.println("<INPUT TYPE=text NAME=value_for_AgentName VALUE=\"\" SIZE=40
                                MAXLENGTH=40>");
                    out.println("</TD></TR>");
```

figure 5.5.3 Part of the code, displaying the declaration fields provided to user (*ServletAgent.java*).

```
if(result_of_Submission.startsWith("SUCCESS"))
                {
                    out.println("<HTML>");
                    out.println("<HEAD><TITLE>Agent System | Creating Agent :
                                SAVED!</TITLE></HEAD>");
                    out.println("<BODY>");
                    out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                    out.println("<CENTER>");
                    out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3
                                CELLSPACING=3 WIDTH=300><TR><TD>");
                    out.println("<TD ALIGN=right WIDTH=200><H2><I>Agent Created!</I></H2>");
                    out.println("</TD><TD ALIGN=left WIDTH=100>");
                    out.println("<A HREF=?curPage=getAboutPage>");
                    out.println("<IMG SRC=images/agent_small.gif ALT=\"[About :
                                AgentSystem Extended Version]\" BORDER=0>");
                    out.println("</A></TD></TR></TABLE></CENTER>");
                    out.println("<BR><BR>");
                    out.println("<H4>Agent has been created! Please, press the button " +
                                "below to return to main menu</H4>");
```

figure 5.5.4 Part of code, displaying successful creation of agent (*ServletAgent.java*).

When the user's choice is to delete an agent, the system provides him/her with a new window, where he/she must enter the unique ID, in order for the system to identify the agent to delete. Again, the unique ID is retrieved and checked by the server-side system, and, depending on the result, provides to the user either a success or an error page (*figure 5.5.5*).

```
else if ("getDeleteParameters".equals(req.getParameter("curPage"))) {

                /**
                  * Opening Socket for the "Deleting an Agent" on Port Servlet_Port
                  */
                 try {
                    sock = new Socket(InetAddress.getByName(Servlet_Address), Servlet_Port);
                    pw = new PrintWriter(sock.getOutputStream());
                    pw.println("Sending Client's Information");
                    pw.flush();
                    br = new BufferedReader(new InputStreamReader(sock.getInputStream()));
                    System.out.println(br.readLine());
                    /**
                     * The command "2" = "Delete Agent" is sent to the AgentSystem via socket.
                     * Afterwards, buffer is flushed in order to send the command successfully.
                     */
                    pw.println("2");
                    pw.flush();
                    /**
                     * Sending the all the "value_for_AgentID" to the AgentSystem.
                     */
                    pw.println(req.getParameter("value_for_AgentID"));
                    pw.flush();
```

figure 5.5.5 Part of code, displaying the deletion process of an agent (*ServletAgent.java*).

By the same way, if the user wants to view *special information* of an agent, then the unique ID must be provided, which, depending on the case, might be correct or wrong, displaying with that way successful or error pages. The code for this part is very similar like the previous case.

Then, if the choice is to display the *Web-GUI Interface* of an agent, the system will make the same procedure again, as before (check the unique ID), with the only main addition that it will have to check whether the agent is a web-GUI agent or not. Otherwise, it will produce another extra error for an agent that does not implement Web-Agent Interface. Also, since in this case (when a successful ID is submitted), the system may send more than the typical information

(which may contain several lines of information), counting the lines is vital; in this way we assure that all the lines will be displayed to the user. Part of the code for checking this feature is given in *figure 5.5.6*.

```
/**
  * Reads the lines of what is being sent by the AgentThread() class.
  * It may be the case that there is more than one line.
  */
   String lines_for_HTMLAgentPage = br.readLine();
   int lines = 1;
    try
       {
           lines = Integer.parseInt(lines_for_HTMLAgentPage);
    }
    catch(NumberFormatException e)
    {
     }
     value_for_AgentWebInterface = br.readLine();
     for(int i=1; i<lines; i++)
           value_for_AgentWebInterface += "<br>\n" + br.readLine();
     }
```

figure 5.5.6 Part of code, displaying the counting of lines for a web-GUI agent (*ServletAgent.java*).

Finally, in the first choice of the system (when the user requests the general information of the agents), the system has some extra features, where it gives to the user the ability to delete or to display special information of an agent with the help of a link. The link makes the relevant requests to the server and then the system is responsible to send the commands and open the relevant page, without the user needing to re-enter the unique ID. This technique is made by using the “A HREF” command, as used when typing an HTML code. The relevant code for this case is given below (*figure 5.5.7*).

```
out.println("<A HREF=?curPage=getDeletionVerification&value_for_AgentID=" +
info_of_AgentID[i]+">Delete</A>");
                        out.println("</TD><TD>");
                        out.println("<A HREF=?curPage=getSpecialParameters&value_for_AgentID=" +
info_of_AgentID[i]+">Display</A>");
                        out.println("</TD><TD>");
                        if (info_of_AgentWebInterface[i].startsWith("TRUE")) {
                            out.println("<A HREF=?curPage=getWebInterface&value_for_AgentID=" +
info_of_AgentID[i]+">Display WebAgent</A>");
                        }
                        else {
                           out.println("");
                        }
```

Deletion link.

Special parameters link.

Link displaying agent with web-GUI (if any).

figure 5.5.7 Part of code, representing the links in the „display agents“ window (*ServletAgent.java*).

6.0 Conclusion

The Intelligent Mobile Agents (Extended Version) project was mostly completed in request to the initial proposal and tasks requested.

The most important files in this project are *ServletAgent.java* and *AgentThread.java*. These files are responsible in requesting information from a server by communicating directly with it, and representing this information on an HTML page to the user.

At this level, the extended version is able to create an Agent, delete an Agent, display all the information (general and more detailed) of an Agent, as well as the ability to send a mobile agent to a remote computer.

6.1 Testing

The system was extensively tested, in order to extinguish all the major and minor bugs, as well as to comply with the requirements defined, giving this way the ability to the user to use robust and reliable software.

Some problems were encountered at the initial stages, mostly for retrieving the vital information from the server. Thereafter, some additions were made, in order to avoid unexpected cases. For instance, the initial default choice of the radio button to "Create an Agent", was used for two main reasons: to give the flexibility and simplicity to the user to create initially an agent (since the most common is that an agent does not exist to the system), as well as to minimize the amount of code that had to be used; if there was not any choice and the user submitted his/her choice (blank in this case), it would produce a null exception error and a blank page, since the Servlet could not understand that (empty) choice.

Finally, the testing for this project was made with Visual Café v.4.0 Expert Edition, since it includes debuggers and a virtual Java runtime machine, where the code can be checked.

Therefore, before the last version, the system is double checked (and tested) in virtual environment, and finally in real situations, through the web browser.

6.2 Problems

Several problems were encountered, mostly during the implementation of the software. The areas that mostly covered in problems were in terms of support of the software, from simultaneously multiple users, the ability to move a mobile agent to a remote system, the ability to send information to the server (when creating an agent), as well as to present information, contained of multiple lines and various fields (by counting the lines that cover). These problems had the disadvantage to delay the project, since additional work and study needed to take place, as well as decision-making and valuable discussion with the supervisors, in order to come across to a possible solution.

Finally, various techniques used in Java were considered (enumeration, hash tables etc.) in order to overcome some problems and produce the final result, as it was initially scheduled and decided.

6.3 Additional work

As we already mentioned, the project was not completed as it was initially scheduled, therefore it does not cover all the pre-defined vital areas. On the other hand, the current level provides a neatly designed, working software, which has the ability to create, delete, display general and acquire and display special information of an agent, as well as the ability to send a mobile agent to a remote system (to a Java window) and return back (vice versa procedure). Of course, in order to achieve that (the vice-versa procedure), the security levels had to be retracted

(privileges were not allowed initially to remote systems); the code would have to be digitally assigned to avoid this.

An addition to the current working system would be to include a multi-threaded access, allowing multiple users to acquire information or ability to create agents. Although this is currently possible, on the other hand, if more than one user tries to create an agent with the same unique ID, although one is possible to be created, the system will create the one, without the second user (who was sure that the system did not have such an agent, with such an ID) to view an error message, since the first user would be able to create this agent, before the second one submits the form. Of course, again the second user would get the information of where the error occurred. Although it is rather rare to submit information to a server at exactly the same time, on the other hand, it would be better if the system could allow a user who tried to create an agent some time, in order to complete the request. This way, it would be possible to avoid viewing messages, without the understanding of how that happened. May we also notice, that the system, although it uses a thread, this thread is mostly to allow the user to make full use of both the Java window (the initial version of the project) and the web-browser version (the extended version of the project) of the system.

On the other hand, additional work could be needed on this project, and this has to do mostly with contacting a JDBC (Java Database Connectivity), where information could be retrieved from the database and displayed (the same way as initially) to the server. Also, the use of ebXML, which is a special area using XML (eXtensible Modelling Language) and descendant of SGML, could provide new technology features, such as dynamic programming and ability to communicate, acquire and send directly information to a database. Therefore, in general terms, a complex combination of programming in Java and in XML, together with the use of databases, such as to give us the ability to acquire information from a real storage system, as databases are.

7.0 References

[1] Bruce Eckel, *Thinking In Java™ 2nd Edition*, Prentice Hall, June 2000.
[2] Jim Farley, *Java™ Distributed Computing*, O' Reilly Press, January 1998.
[3] David Flanagan, *Java™ In A Nutshell*, O' Reilly Press, November 1999.
[4] David Flanagan, Jim Farley, William Crawford and Kris Magnusson, *Java™ Enterprise In A Nutshell*, O' Reilly Press, September 1999.
[5] Jason Hunter, William Crawford, *Java™ Servlet Programming*, O' Reilly Press, October 1998.
[6] Thomas W. Christofer, George K. Thiruvathukal, *High Performance Java™ Platform Computing (Multithreaded and Networked Programming)*, Sun Microsystems Press, 2000.
[7] David Canfield Smith, Allen Cypher and Jim Spohrer, *KIDSIM : Programming Agents Without a Programming Language***,** Communications of ACM, July 1994/Vol.37, No. 7, pages 55-67.

Addtitional information related to Intelligent Agents and Servlets can also be found in the following Internet links :

[8] http://java.sun.com.
[9] http://www.emu.edu.tr/edufacil/compcent/bookslib/Java,%20by%20Michael%20Morrison,%20Second%20Edition/ch26.htm.
[10] http://java.sun.com/docs/books/tutorial/servlets.
[11] http://java.sun.com/docs/books/tutorial/servlets/servletrunner/properties.html.
[12] http://novocode.de/doc/servlet-essentials/chapter2b.hml.
[13] http://java.sun.com/docs/books/tutorial/networking/sockets/index.html.
[14] http://developer.java.sun.com/developer/onlineTraining/Programming/BasicJava2/socket.html.
[15] http://www.javaworld.com/jw-12-1996/jw-12-sockets.html.
[16] http://www.javaworld.com/jw-09-1999/jw-09-servlet_p.html.
[17] http://www.servlets.com/jsp/examples.
[18] http://home1.gte.net/pfingar/agents_doc_rev4.htm.

Appendix A – Source code

AgentThread.java

```
/**
 * @(#)AgentThread.java
 *
 * Part of the AgentSystem "POND"
 * Developed at the Institute for Information Processing and Microprocessor Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 *
 */

package PkgAgentSystem;

import java.io.*;
import java.lang.*;
import java.net.*;
import java.util.*;
import java.util.Hashtable;

import FIM.Util.Threads.CancellableThread;
import FIM.Util.ClassLoaderObjectInputStream;
import FIM.Util.Crypto.NamedKeyPair;
import FIM.Util.Crypto.NamedKeyAndCertificate;
import PkgAgentSystem.Crypto.AgentIdentity;
import PkgAgentSystem.Crypto.PersonalSecurityStore;
import PkgAgentSystem.Crypto.PersonalSecurityStoreFactory;

/**
 * The AgentThread() class.
 * This class is used together with the main AgentSystem() class. It creates a new Thread,
 * it opens a new socket at PORT 6000, and by interacting together with the ServletAgent()
 * class, it compares all the requests of the ServletAgent() class. Then, when positive,
 * it produces responses to the AgentServlet() class, by displaying the result, with the use
 * of other importan classes of the main AgentSystem() class.
 *
 * @author  Marios Kalnis
 * @version 1.0, 1.7.2001
 */
 public class AgentThread extends Thread {

   /**
    * Creating a communication protocol, where the two sides, ServletAgent.java and
    * AgentSystem.java communicate and exchange information. There are six integers,
    * where in five of them there is a value assigned to them (from "0" to "5") and the
    * last one (the integer "state") is used to make the comparison and decision of which
    * choice will be followed each time, and which tasks will be achieved each time.
    *
    * DisplayAgents Method's internal integer, assigned for displaying information about agents,
    *              with value "0".
    * CreateAgents Method's internal integer, assigned for creating agents, with value "1".
    * DeleteAgents Method's internal integer, assigned for deleting agents, with value "2".
    * DisplaySpecialInfo Method's internal integer, assigned for displaying special information
    *                  about agents, with value "3".
    * DisplayAgentWebInterface Method that checks whether on the system a webGUI Agent was created, and
    *                        if so, it displays information about it. It is assigned the value of "4".
    * SendAgentToRemoteComputer Method that checks whether the information entered are correct, and if so
    *                         sends the Mobile Agent to a remote computer, at the defined URL, with value "5".
    * state An integer checking the state of each of the above integer cases.
    * servletThreadAgentSystem A protected method, that it is combined with the main class of the
    *                        AgentSystem(), such that to use the methods of startAgent() and stopAgent().
    */
   private final int DisplayAgents=0;
   private final int CreateAgents=1;
   private final int DeleteAgents=2;
   private final int DisplaySpecialInfo=3;
   private final int DisplayAgentWebInterface = 4;
   private final int SendAgentToRemoteComputer = 5;
   private int state;

   protected int port;
```

```
protected AgentSystem servletThreadAgentSystem=null;
/**
 * Creating the AgentThread constructor.
 */
public AgentThread(AgentSystem sys,int port) {
   servletThreadAgentSystem = sys;
      System.out.println("Creating AgentThread Constructor");
   this.port=port;
}

public void run() {

  /**
   * Initialising the value of the socket.
   */
  ServerSocket serSocket = null;
  Socket sock = null;

  try {
      /**
       * Opening ther socket to the PORT 6000, as defined above and sending back to the
       * system, a response that the request to open is accepted and it opens to the
       * predefined PORT.
       */
      serSocket = new ServerSocket(port);
      System.out.println("Starting server at port "+port+"...");
      while(true) {

            try {
               /**
                * The AgentThread() class (server-side) accepts the ServletAgent() class
                * request (client-side) to open the socket at PORT 6000.
                *
                * out The stream to write out
                                * br The stream to read in
                */
               sock = serSocket.accept();
               System.out.println("Client connected to Server (Port 6000)");
               BufferedReader br = new BufferedReader(new InputStreamReader(sock.getInputStream()));
               PrintWriter out = new PrintWriter(sock.getOutputStream());

               try {
                  String S = br.readLine();
                  out.println("AgentSystem responding");
                  out.flush();
                  S = br.readLine();
                  try
                  {
                  state=Integer.parseInt(S);
                  }
                  catch(NumberFormatException e)
                  {
                     System.err.println("System Error in "+e.toString());
                  }

                  /**
                   * IF-ELSE statements (user's choices).Following the correct
                   * path, depending on the choice.
                   */
                  if (state == DisplayAgents) {
                     /**
                      * Counting the size of the Agents (how many they are) and
                      * printing out the result
                      */
                     int numberOfAgents=servletThreadAgentSystem.agents_by_id.size();
                     out.println(""+numberOfAgents);
                     out.flush();

                     /**
                      * Printing out the values of each of the four Strings of
                      * the agent, such as "AgentName", "ID", "URL" and "OwnerName".
                      * Each of them is included in a while loop and they retrieve
                      * the values from the main AgentSystem's class by its relevant
                      * hashtable.
```

```
     */
    Enumeration enum=servletThreadAgentSystem.agents_by_id.keys();
    while(enum.hasMoreElements())
    {
       AgentIdentity id=(AgentIdentity)enum.nextElement();
       out.println(id.getAgentName());
       out.println(id.getID());
       out.println(id.getHomeAddress().toString());
       out.println(id.getOwnerName());
       boolean res=(((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).agent instanceof WebAgentInterface);

       if (res == true)
       {
          out.println("TRUE with Agent");
       }
       else
       {
          out.println("FALSE");
       }
    }
    out.flush();
}
else if (state == CreateAgents) {
                try
                {
                 /**
                  * When creating an Agent, initially all the values of the fields maentioned
                  * in the ServletAgent() class, are being read. Then the method startAgent()
                  * from the main class AgentSystem() class is called, in order to create the
                  * new Agents.
                  *
                  * AgentName The name of the Agent as it is entered in the fields, in the ServletAgent() class.
                  * ID The Agent's unique ID, provided by the user, in ServletAgent() class fields.
                  * HomeAddress The Agent's URL, as defined by the user.
                  * OwnerName The user who creates the agent must provide his/her name to the system.
                  * className The place where the new Agent will be saved.
                  */
                 String AgentName = br.readLine();
                 String ID = br.readLine();
                 String HomeAddress = br.readLine();
                 String OwnerName = br.readLine();
                 String className = br.readLine();
                 AgentIdentity id = new AgentIdentity(AgentName,ID,new URL(HomeAddress),OwnerName);
                 servletThreadAgentSystem.startAgent(className,null,false,id,null,null,true);
                 out.println("SUCCESS of creating an agent!!!");
                }
                catch(Exception e)
                {
                   out.println("ERROR: "+e.toString());
                }

}
 else if (state == DeleteAgents) {
                /**
                  * The only field that the "DeleteAgents" page needs to read is the
                  * string AgentID. From that one, it will recognise the values of the Agent
                  * created before and will procceed accordingly. Also, a flag set initially
                  * false is being used, in order to distinguish the choices whether there is
                  * a SUCCESS or an ERROR, considering the AgentID that the user will provide,
                  * and to procceed accordingly, either to a SUCCESS "deleted" page, diplaying
                  * all the relevant information, or to an ERRROR page, explaining to the user,
                  * where the error occured.
                  *
                  * AgentID Agent's unique identification.
                  * found_special A flag used to check when someting is true or false
                  *              and procceed accordingly.
                  */
                 String AgentID = br.readLine();
                 boolean found_delete = false;
                 Enumeration enum = servletThreadAgentSystem.agents_by_id.keys();
                 while(enum.hasMoreElements())
                 {
                      AgentIdentity id = (AgentIdentity)enum.nextElement();
                      if(AgentID.equals(id.getID()))
                      {
```

```
            try
            {
             servletThreadAgentSystem.stopAgent(((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).agent,true);
             out.println("SUCCESS of deleting an agent!!!");
            }
            catch(Exception e)
            {
               out.println("ERROR deleting agent : "+e.toString());
            }
            found_delete = true;
          }//end of if statement
        }//end of while statement
        if (!found_delete)
        {
          out.println("ERROR: No such AgentID exists");
        }
    }//end of "DeleteAgents" else-if statement

    /**
     * Else-if statement for displaying "Special Information Page" of Agent
     */
     else if (state == DisplaySpecialInfo) {
                    /**
                     * The only field that the "DisplaySpecialInfo" page needs to read is the
                     * string AgentID. From that one, it will recognise the values of the Agent
                     * created before and will procceed accordingly. Also, a flag set initially
                     * false is being used, in order to distinguish the choices whether there is
                     * a SUCCESS or an ERROR, considering the AgentID that the user will provide,
                     * and to procceed accordingly, either to a SUCCESS page, diplaying all the
                     * relevant information, or to an ERROR page, explaining to the user, where
                     * the error occured.
                     *
                     * AgentID Agent's unique identification.
                     * found_special A flag used to check when something is true or false
                     *               and procceed accordingly.
                     */
                    String AgentID = br.readLine();
                    System.out.println("agentID: "+ AgentID);
                    boolean found_special = false;

                    /**
                     * Considering again the Hashtable where all the Agents are saved.
                     */
                    Enumeration enum = servletThreadAgentSystem.agents_by_id.keys();
                    while(enum.hasMoreElements())
                    {
                     AgentIdentity id = (AgentIdentity)enum.nextElement();

                     /**
                      * Considering all the cases of Special Information, provided by the AgentSystem()
                      * class. Together with the ServletAgent() class, checking whether it is possible
                      * to display the results or not. The following if-else statements are only focused
                      * on providing information to the ServletAgent() class for all the values.
                      */
                     if(AgentID.equals(id.getID()))
                     {
                        try
                        {
                          /**
                           * If there is a SUCCESS, then procceed further on.
                           */
                  out.println("SUCCESS of displaying special information of an agent!!!");
                  out.flush();

                 //  System.out.println("after success message");
                            out.println(id.getAgentName());
                  out.println(id.getID());
                  out.println(id.getHomeAddress().toString());
                  out.println(id.getOwnerName());
                  out.println(id.getAgentPublicKeyName());
                  if(id.getOwnerCertificate() == null)
                  {
                     out.println(1);
                     out.println("No information available");
                  }
```

```
   else
   {
      String certStr=id.getOwnerCertificate().toString();
      // Count lines in certificate
      int lines = 1;
      int pos = certStr.indexOf('\n',0);
      while(pos >= 0)
      {
         lines = lines+1;
         if(pos+1 == certStr.length())
            pos = -1;
         else
            pos = certStr.indexOf('\n',pos+1);
      }
      out.println("" + lines);
      out.println(certStr);

   }
   if(id.isSigned() == true)
   {
      out.println("Signed");
   }
   else
   {
      out.println("Not signed");
   }
   if(id.initialStateDeployed() == true)
   {
      out.println("True");
   }
   else
   {
      out.println("False");
   }
   if(id.codeDeployed() == true)
   {
      out.println("True");
   }
   else
   {
      out.println("False");
   }
   out.println(((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).codeOrigin);
   if(((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).localCode == true)
   {
      out.println("True");
   }
   else
   {
      out.println("False");
   }
   out.println(""+((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).permissionGroup);
   out.println(""+((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).scale);
   out.println(((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).agent.getClass().getName());
            }
            catch(Exception e)
            {
               out.println("ERROR displaying special information about agent: "+e.toString());
               System.out.println("ERROR displaying Special Information");
            }
found_special = true;
         }
      }//end of while statement

      /**
       * Considering the initial boolean flag. If there is an error, then change the flag to false
       * and send to the ServletAgent() class an ERROR message, where it will help the ServletAgent()
       * class to display the relevant ERROR page to the user.
       */
      if(!found_special)
      {
out.println("ERROR: No such AgentID exists");
System.out.println("ERROR no such AgentID");

      }
```

```
            }//end of "DisplaySpecialInfo" else-if statement

        /**
     * Else-if statement for displaying "Displaying Agen Web Interface"
     */
else if (state == DisplayAgentWebInterface) {

          /**
            * The only field that the "DisplayAgentWebInterface" page needs to read is the
            * string AgentID. From that one, it will recognise the values of the Agent
            * created before and will procceed accordingly. Also, a flag set initially
            * false is being used, in order to distinguish the choices whether there is
            * a SUCCESS or an ERROR, considering the AgentID that the user will provide,
            * and to procceed accordingly, either to a SUCCESS page, diplaying all the
            * relevant information, or to an ERROR page, explaining to the user, where
            * the error occured.
            *
            * AgentID Agent's unique identification.
            * found_webInterface A flag used to check when something is true or false
            *                    and procceed accordingly.
            */
            String AgentID = br.readLine();
            System.out.println("agentID: "+ AgentID);
            boolean found_webInterface = false;

            /**
              * Considering again the Hashtable where all the Agents are saved.
              */
             Enumeration enum = servletThreadAgentSystem.agents_by_id.keys();
             while(enum.hasMoreElements())
               {
                AgentIdentity id = (AgentIdentity)enum.nextElement();

              /**
                * Considering all the cases of WebGUI of Agent, provided by the AgentSystem()
                * class. Together with the ServletAgent() class, checking whether it is possible
                * to display the results or not. The following if-else statements are only focused
                * on providing information to the ServletAgent() class for all the values.
                */
                if(AgentID.equals(id.getID()))
                {
                   /**
                    * hasWebInterface A flag used to check when something is true or false
                    *                 and procceed accordingly.
                    */
                boolean hasWebInterface = (((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).agent instanceof
                                                    WebAgentInterface);
                   try
                   {
                      if(hasWebInterface)
                      {
                        /**
                         * If there is a SUCCESS, then procceed further on.
                         */
                         out.println("SUCCESS of displaying special information of an agent!!!");
                         out.flush();
                         out.println(id.getAgentName());
                         out.println(id.getID());
                String webpage=((WebAgentInterface)
                                    ((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).agent).getHTMLPage(null);
                if(webpage == null || webpage.length()<1)
                {
                   out.println(1);
                   out.println("No information available");
                }
                else
                {
                   // Count lines in HTML for webGUI Agent
                   int lines = 1;
                   int position = webpage.indexOf('\n',0);
                   while(position >= 0)
                   {
                      lines = lines+1;
                      if(position+1 == webpage.length())
                         position = -1;
```

```
                    else
                        position = webpage.indexOf('\n',position+1);
                }
                out.println("" + lines);
                out.println(webpage);

            }
          }
          else
          {
             out.println("ERROR: Agent found, but agent is not able to display a Web Interface");
          }
        }
        catch(Exception e)
                {
                   out.println("ERROR displaying web interface about agent: "+e.toString());
                   System.out.println("ERROR displaying web interface of Agent");
                }
        found_webInterface = true;
      }
         }//end of while statement

     /**
      * The system checks whether a GUI Agent is created on the System. If not
      * then it displays an ERROR and explains why this error occured.
      */
     if(!found_webInterface)
             {
       out.println("ERROR: No such AgentID exists");
       System.out.println("ERROR no such AgentID");
             }
}//end of "DisplayAgentWebInterface" else-if statement


/**
  * Else-if statement for sending Agent to remote computer.
  */
     else if (state == SendAgentToRemoteComputer) {

       /**
         * The only field that the "SendAgentToRemoteComputer" page needs to read is the
         * string AgentID. From that one, it will recognise the values of the Agent
         * created before and will procceed accordingly. Also, a flag set initially
         * false is being used, in order to distinguish the choices whether there is
         * a SUCCESS or an ERROR, considering the AgentID that the user will provide,
         * and to procceed accordingly, either to a SUCCESS page, diplaying all the
         * relevant information, or to an ERROR page, explaining to the user, where
         * the error occured.
         *
         * AgentID Agent's unique identification.
         * found_webInterface A flag used to check when something is true or false
         *                    and procceed accordingly.
         */
        String AgentID = br.readLine();
        Hashtable table=new Hashtable();
        int lines=Integer.parseInt(br.readLine());
        for(int i=0;i<lines;i++)
        {
            String paramName=br.readLine();
            String paramValue=br.readLine();
            table.put(paramName,paramValue);
        }
             System.out.println("agentID: "+ AgentID);
        boolean found_webInterface = false;

          /**
            * Considering again the Hashtable where all the Agents are saved.
            */
           Enumeration enum = servletThreadAgentSystem.agents_by_id.keys();
           while(enum.hasMoreElements())
             {
              AgentIdentity id = (AgentIdentity)enum.nextElement();

             /**
              * Considering all the cases of WebGUI of Agent, provided by the AgentSystem()
```

```
                    * class. Together with the ServletAgent() class, checking whether it is possible
                    * to display the results or not. The following if-else statements are only focused
                    * on providing information to the ServletAgent() class for all the values.
                    */
                    if(AgentID.equals(id.getID()))
                    {
                     /**
                        * hasWebInterface A flag used to check when something is true or false
                        *               and procceed accordingly.
                        */
                    boolean hasWebInterface = (((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).agent instanceof
                                                WebAgentInterface);
                       try
                       {
                          if(hasWebInterface)
                          {
                              /**
                               * If there is a SUCCESS, then procceed further on.
                               */
                               out.println("SUCCESS of displaying special information of an agent!!!");
                               out.flush();
                               out.println(id.getAgentName());
                               out.println(id.getID());
                               String webpage= ((WebAgentInterface)
                                           ((AgentData)servletThreadAgentSystem.agents_by_id.get(id)).agent).getHTMLPage(table);
                                if(webpage == null || webpage.length()<1)
                                {
                                        out.println(1);
                                        out.println("No information available");
                                }
                                else
                                {
                                        // Count lines in HTML for webGUI Agent
                                        lines = 1;
                                        int position = webpage.indexOf('\n',0);
                                        while(position >= 0)
                                        {
                                                lines = lines+1;
                                                if(position+1 == webpage.length())
                                                position = -1;
                                                else
                                                position = webpage.indexOf('\n',position+1);
                                        }
                                        out.println("" + lines);
                                        out.println(webpage);

                                }
                           }
                           else
                           {
                              out.println("ERROR: Agent found, but agent is not able to be sent the specified URL");
                           }
                          }
                          catch(Exception e)
                          {
                                out.println("ERROR sending agent to the specified URL address: "+e.toString());
                                System.out.println("ERROR sending agent to the specified URL address");
                          }
                          found_webInterface = true;
                        }
                      }//end of while statement

                    /**
                     * The system checks whether a GUI Agent is created on the System. If not
                     * then it displays an ERROR and explains why this error occured.
                     */
                    if(!found_webInterface)
                                {
                       out.println("ERROR: No such AgentID exists");
                       System.out.println("ERROR no such AgentID");
                                }
                }//end of "SendAgentToRemoteComputer" else-if statement
```

```
                }
                catch (Exception e) { e.printStackTrace();
                }

                /**
                 * Closing the output and the socket of the AgentThread() class.
                 */
                out.close();
                sock.close();
            }

            /**
             * Catching the socket exception error (in case something is going wrong).
             */
            catch(SocketException se) {
                System.out.println("Server Socket Problem" + se.getMessage());
            }
        }//end of while
    }
    catch(Exception e) {
        System.out.println("Couldn't start " + e.getMessage());
    }

    System.exit(0);
  }
}
```

AgentThreadStart.java

```
/**
 * @(#)AgentThreadStart.java
 *
 * Main class
 * Part of the AgentSystem "POND"
 * Developed at the Institute for Information Processing and Microprocessor Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 *
 */

package PkgAgentSystem;

import java.io.*;
import java.lang.*;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.net.*;
import java.util.*;
import java.util.Hashtable;
import java.security.KeyStoreException;
import java.security.Policy;
import java.security.Security;
import java.io.InputStream;
import java.util.Hashtable;
import java.util.Vector;
import java.util.Enumeration;
import java.util.GregorianCalendar;

import FIM.Util.Threads.CancellableThread;
import FIM.Util.ClassLoaderObjectInputStream;
import FIM.Util.Crypto.NamedKeyPair;
import FIM.Util.Crypto.NamedKeyAndCertificate;
import PkgAgentSystem.Crypto.AgentIdentity;
import PkgAgentSystem.Crypto.PersonalSecurityStore;
import PkgAgentSystem.Crypto.PersonalSecurityStoreFactory;
import PkgAgentSystem.GUI.GUIAgentSystem;

/**
 * The main AgentThreadStart() class.
 * This class is used together with the main GUIAgentSystem() class and the AgentThread()
 * class. It uses the same constructors that the GUIAgentSystem() class has, as well as
 * it uses and opens the same socket PORT at 6000, as AgentThread() class does. The
 * AgentThreadStart() class, is used as the main class (used as a server) from where the
 * main project runs. All the other classes are used to help this method to create the
 * security keys and to initialize the system, such as to be used successfully with the
 * use of the ServletAgent() class.
 *
 * @author  Marios Kalnis
 * @version 1.0, 1.7.2001
 */
public class AgentThreadStart extends GUIAgentSystem
{

   /** The second standard port of the agent, where communicates with the Servlet */
   private static final int PORT=6000;

   /** The port of this system */
   private int myPort=PORT;


   /**
   Creates a new agent system with the default value for the port and the users home directory
   as directory from where the agents are loaded. The system will not terminate after the last agent is
   destroyed and no state will be loaded from a saved file. The security is turned on.
   @param libraryDir the directory where the common libraries are stored. Must be provided.
   */
/*
   public AgentThreadStart(String libraryDir)
   {
      super(libraryDir);
   }
```

```
*/

  /**
  Creates a new agent system with the security turned on.
  @param port the port the system will listen on (must be >1024 or -1)
  @param libraryDir the directory, where the common libraries are from; All agents will load classes
        first from there and only after not finding them there from their local path/package. Must be provided.
  @param agentDir the directory, where the AGENTS are from, NOT where the JDK or the agent system is located
  @param exitOnLast if true the system will terminate after the last agent is destroyed (may start empty however)
  @param savedFile if not null, the agentsystem will load the state from this file
  */
  public AgentThreadStart(int port,int PORT,String libraryDir,String agentDir,boolean exitOnLast,String savedFile)
  {
     super(port,libraryDir,agentDir,exitOnLast,savedFile);
     myPort=PORT;
  }

  /**
  Creates a new agent system.
  @param port the port the system will listen on (must be >1024 or -1)
  @param libraryDir the directory, where the common libraries are from; All agents will load classes
        first from there and only after not finding them there from their local path/package. Must be provided.
  @param agentDir the directory, where the AGENTS are from, NOT where the JDK or the agent system is located
  @param exitOnLast if true the system will terminate after the last agent is destroyed (may start empty however)
  @param savedFile if not null, the agentsystem will load the state from this file
  @param useSecurity false if no security system is wanted
  */
  protected AgentThreadStart(int port,int PORT,String libraryDir,String agentDir,boolean exitOnLast,String savedFile,boolean useSecurity)
  {
     super(port,libraryDir,agentDir,exitOnLast,savedFile,useSecurity);
     myPort=PORT;
  }

  /**
   * Main method of the system: Wait for incoming requests and handle them.
   * Currently only requests for receiving an agent are allowed.
   * @see PkgAgentSystem.ServingThread
   * @see PkgAgentSystem.AgentSystem
   */
  public void run()
  {
     /** Start thread. Create new agentThread for connection */
     AgentThread agentThread = new AgentThread(this,myPort);
     /** Start serving this request */
     agentThread.start();
     /** Calling the super method "run" of the AgentSystem */
     super.run();
  }


  public static void main(String[] args)
  {
     String agentDir=System.getProperty("user.home");
     String libraryDir=System.getProperty("user.dir");
     int customPort=-1;
     String savedFile=null;
     if(args.length<2)
        System.err.println("Usage: java PkgAgentSystem.GUI.GUIAgentSystem agentDir libraryDir [Port [savedFile]]\n");
     if(args.length<1)
        System.err.println("Location of agents not specified: Using the users home directory: "+agentDir);
     else
        agentDir=args[0];
     if(args.length<2)
        System.err.println("Location of shared libraries not specified: Using  current dir: "+libraryDir);
     else
        libraryDir=args[1];
     if(args.length>2)
     {
        try
        {
           customPort=Integer.valueOf(args[2]).intValue();
        }
        catch(NumberFormatException e)
        {
           System.err.println("Port number is not a number. Using default port.");
```

```
            }
        }
        if(args.length>3)
            savedFile=args[3];
        AgentThreadStart as=new AgentThreadStart(customPort,PORT,libraryDir,agentDir,false,savedFile,false);
        as.run();

        System.out.println("Agent system terminated.");
        System.exit(0);
    }

}
```

WebAgentInterface.java

```
/**
 * @(#)WebAgentInterface.java
 *
 * Part of the AgentSystem "POND"
 * Developed at the Institute for Information Processing and Microprocessor Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 *
 */

package PkgAgentSystem;

import java.util.Hashtable;

/**
 * The WebAgentInterface method.
 * This method is related closely with the AgentThread, and therefore, the AgentSystem.
 * It is responsible to distinguish whether an agent has a web interface or not, and
 * retrieving the webpage (while passing in parameters, e.g. when filling in a form).
 *
 * @author  Marios Kalnis
 * @version 1.0, 1.7.2001
 */
public interface WebAgentInterface
{
   public String getHTMLPage(Hashtable parameters);
}
```

MovingAgent.java

```
/**
 * @(#)MovingAgent.java
 *
 * Part of the AgentSystem "POND"
 * Developed at the Institute for Information Processing and Microprocessor Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 *
 */

package BaseAgents.Beeper;

import java.util.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;

import PkgAgentSystem.*;
import PkgAgentSystem.GUI.*;
import PkgAgentSystem.Messaging.Message;
import PkgAgentSystem.Messaging.SignedMessage;
import PkgAgentSystem.Crypto.AgentIdentityExtension;

/**
 * The MovingAgent() class.
 * This class is used in combination with the AgentThread, and therefore, with the AgentSystem.
 * This class is used in two major cases: When the user create an Agent, then (instead of the
 * web interface system) the new Agent appears on the Java window of the initial version of the
 * "Intelligent Mobile Agents" project (the user has in access in this window as well). Therefore,
 * it is responsible on creating the interface of this new agent, appearing to the Java window as a
 * button. Thereafter, it is used when the user selects to move the mobile agent to a remote system.
 * In this case, this code is responsible to provide the HTML format form to the user, as well as to
 * show a successful or a failure page (depending on the case) and verify that the agent was successfully
 * sent to the remote system, with all the information the user provided.
 *
 * Modification of the initial version BeeperAgent.java
 *
 * @author  Marios Kalnis
 * @author  Michael Sonntag
 * @version 1.0, 1.7.2001
 */
public class MovingAgent extends GUIAgentBase implements ActionListener, WebAgentInterface
{
   protected transient JButton beep=new JButton("Beep!");

   /**
    * Protected methods, providing all the fields that being sent by the
    * user to the remote system.
    */
   protected URL whereToGo=null;
   protected String title;
   protected String text;
   protected String whichImage;

   public MovingAgent()
   {
      registerConversation(new BeepConversation(this));
   }

   public String getHTMLPage(Hashtable parameters)
   {
      if(parameters==null)
      {
       String str= "<FORM METHOD=\"GET\"" + " ACTION=\"/Agent/"+getIdentity().getID()+"/\"><CENTER><H2><I>Welcome to the
                    WebGUI Agent</I></H2></CENTER><BR>"+
                "<H4>Please, fill in the form below</H4><BR>" +
                "<CENTER><TABLE BORDER=0 WIDTH=440><TR><TD ALIGN=right WIDTH=120>Title : </TD>" +
                "<TD ALIGN=right WIDTH=320><INPUT TYPE=text NAME=Title VALUE=\"\" SIZE=40 MAXLENGTH=40></TD></TR>"
                  +  "<TR><TD ALIGN=right WIDTH=120>AgentSystem's URL : </TD>" +
                   "<TD ALIGN=right WIDTH=320>"+"<FONT COLOR=blue>amp://</FONT><INPUT TYPE=text NAME=GUIAgentURL
```

```
                VALUE=\"\" SIZE=40 MAXLENGTH=40></TD></TR>"+
            "<TR><TD ALIGN=right WIDTH=120>Comments : </TD>" +
            "<TD ALIGN=right WIDTH=320><TEXTAREA NAME=text ROWS=5 COLS=31.5></TEXTAREA>" +
            "</TD></TR></TABLE></CENTER><BR><BR>" +
            "<CENTER><TABLE BORDER=0 WIDTH=250><TR><TD WIDTH=125 ALIGN=left><INPUT TYPE=radio
                    NAME=GUIAgents" + " VALUE=\"GUIImageOne\">" +
            "<I> Image One</I></TD>" +
            "<TD WIDTH=125 ALIGN=left><INPUT TYPE=radio NAME=GUIAgents" + " VALUE=\"GUIImageTwo\">" +
            "<I> Image Two</I></TD></TR>" +
            "<TR><TD WIDTH=125 ALIGN=left><INPUT TYPE=radio NAME=GUIAgents" + " VALUE=\"GUIImageThree\">" +
            "<I> Image Three</I></TD>" +
            "<TD WIDTH=125 ALIGN=left><INPUT TYPE=radio NAME=GUIAgents" + " VALUE=\"GUIImageFour\">" +
            "<I> Image Four</I></TD></TR></TABLE></CENTER>" +
            "<BR><BR>" +
            "<CENTER><TABLE WIDTH=200><TR><TD WIDTH=50 ALIGN=center>" +
            "<INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)></TD>" +
            "<TD WIDTH=50 ALIGN=center>" +
            "<INPUT TYPE=SUBMIT VALUE=Go>" +
            "<TD WIDTH=50 ALIGN=center>" +
            "<INPUT TYPE=\"reset\"" + " VALUE=\"Reset\">" +
            "<TD WIDTH=50 ALIGN=center>" +
            "<INPUT TYPE=hidden name=\"action\" value=\"WebAgent\">"+
            "<INPUT TYPE=hidden name=\"AgentID\" value=\""+getIdentity().getID()+"\">"+
            "<INPUT TYPE=BUTTON VALUE=Menu onClick=parent.location='ServletAgent'>" +
            "</TD></TR></TABLE></CENTER>" +
                        "</CENTER></FORM>";
     return str;
    }
    else
    {   // The user has submitted the form printed above
        title=(String)parameters.get("Title");
        text=(String)parameters.get("text");
        whichImage=(String)parameters.get("GUIAgents");
        if((String)parameters.get("GUIAgentURL")==null || ((String)parameters.get("GUIAgentURL")).length()<1)
            return printErrorpage("No address for the destination agent system provided");
        try
        {
            whereToGo=new URL("amp://"+(String)parameters.get("GUIAgentURL"));
        }
        catch(MalformedURLException e)
        {
            return printErrorpage(e.toString());
        }
        if(title==null || title.length()<1)
            return printErrorpage("No title provided");
        if(text==null || text.length()<1)
            return printErrorpage("No text provided");

        //The system prints the success page (the agent moved to remote system) to the user
        String Success_Page = "<BR><FORM METHOD=\"GET\"" + " ACTION=\"\"><CENTER><H3>" +
                    "Thank you for your submission. The specified Agent was " +
                //      " sent to the URL you provided (" + GUIAgentURL + "). Please, press the " +
                    "<I>Menu</I> button below to return back to main page.</H3></CENTER>" +
                    "<BR><BR><CENTER>" +
                    "<INPUT TYPE=BUTTON VALUE=Menu onClick=parent.location='ServletAgent'>" +
                    "</CENTER></FORM>";

/*
        String str="";
        Enumeration enum=parameters.keys();
        while(enum.hasMoreElements())
        {
            String key=(String)enum.nextElement();
            String value=(String)parameters.get(key);
            str+=key+" = "+value+"<br>";
        }
        return str;
*/

    final MovingAgent agent=this;
    (new Thread()
    {
        public void run()
        {
            try
```

```
            {
               Thread.sleep(1000);
            }
            catch(InterruptedException e)
            {
            }
            agent.goDestination();
         }
      }).start();
         return Success_Page;
      }
   }

   //Method for moving to the Destination of the mobile agent.
   private void goDestination()
   {
      logMessage("Moving to "+whereToGo.toString()+"...");
      try
      {
         move(whereToGo);
      }
      catch(ReceiveAgentDeniedException e)
      {
         logMessage("Destination denied receiving agent: "+e.getMessage());
      }
      catch(CannotSendException e)
      {
         logMessage("Error transferring agent: "+e.getMessage());
      }
   }

   //Displaying message after arrival.
   protected void onAfterMoving()
   {
      final MovingAgent agent=this;
      (new Thread()
      {
         public void run()
         {
            try
            {
               Thread.sleep(1000);
            }
            catch(InterruptedException e)
            {
            }
            agent.showMessage();
         }
      }).start();
   }

   /**
    * Method showing all the information that has been sent to the remote
    * system, in a Java window.
    */
   private void showMessage()
   {
      // Representing the images of the system, depending on the user's choice.
      String imageName=null;
      if("GUIImageOne".equals(whichImage))
         imageName="BaseAgents/Beeper/agent.gif";
      else if("GUIImageTwo".equals(whichImage))
         imageName="BaseAgents/Beeper/Fimlogo.gif";
      else if("GUIImageThree".equals(whichImage))
         imageName="BaseAgents/Beeper/latest_eit_195.gif";
      else if("GUIImageFour".equals(whichImage))
         imageName="BaseAgents/Beeper/latest_eit_304.gif";
      URL url=getAgentSystem().getResource(this,imageName);
      ByteArrayOutputStream bos=new ByteArrayOutputStream();
      try
      {
         if(url!=null)
         {
            InputStream in=url.openStream();
            byte[] buf=new byte[1024];
```

```
            int len=1024;
            len=in.read(buf,0,len);
            while(len>0)
            {
               bos.write(buf,0,len);
               len=1024;
               len=in.read(buf,0,len);
            }
            bos.close();
            Icon icon=new ImageIcon(bos.toByteArray());
            JOptionPane.showMessageDialog(null,text,title,JOptionPane.PLAIN_MESSAGE,icon);
         }
         else
            JOptionPane.showMessageDialog(null,text,title,JOptionPane.PLAIN_MESSAGE);
      }
      catch(IOException e)
      {
         logMessage("Could not load image!");
         JOptionPane.showMessageDialog(null,text,title,JOptionPane.PLAIN_MESSAGE);
      }
//       stopAgent();
   }

   /**
    * Method that shows an Error page to the user, if something goes wrong during
    * the submission process.
    */
   protected String printErrorpage(String text)
   {
      String Error_Page = "<FORM METHOD=\"GET\"" + " ACTION=\"\"><BR><CENTER><H3>" +
                          "Failure to send the agent to the destination address provided. " +
                          "The error occured because : <FONT COLOR=red>" + text + "</FONT>. " +
                          "Please go back and try again.</H3></CENTER>" +
                          "<BR><BR><CENTER>" +
                          "<INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>" +
                          "</CENTER></FORM>";
      return Error_Page;
   }

            protected void showDialog()
            {
               doBeep();
            }

            protected void doBeep()
            {
               Toolkit.getDefaultToolkit().beep();
               logMessage("Beep!");
            }

   protected javax.swing.JPanel createVisualization()
   {
      javax.swing.JPanel visualization=new javax.swing.JPanel();
      visualization.setLayout(new BorderLayout());
      visualization.setBorder(new javax.swing.border.EtchedBorder());
      javax.swing.JPanel pane=new javax.swing.JPanel();
      pane.setLayout(new GridBagLayout());
      pane.setBackground(Color.white);
      pane.setOpaque(true);
      javax.swing.JLabel label=new javax.swing.JLabel("New Beeper ("+getIdentity().getAgentName()+")");
      GridBagConstraints cons=new GridBagConstraints();
      cons.insets=new Insets(10,10,10,10);
      cons.fill=GridBagConstraints.HORIZONTAL;
      cons.anchor=GridBagConstraints.CENTER;
      cons.weightx=1.0;
      cons.weighty=0.0;
      cons.gridx=0;
      cons.gridy=0;
      pane.add(label,cons);
      cons.insets=new Insets(10,10,10,10);
      cons.fill=GridBagConstraints.NONE;
      cons.anchor=GridBagConstraints.CENTER;
      cons.ipadx=10;
      cons.ipady=10;
      cons.weightx=1.0;
```

```
      cons.weighty=0.0;
      cons.gridx=0;
      cons.gridy=1;
      if(beep==null)
      beep=new JButton("Beep!");
      beep.setHorizontalAlignment(SwingConstants.CENTER);
      beep.addActionListener(this);
      pane.add(beep,cons);
      visualization.add(pane);
      return visualization;
   }

   public void actionPerformed(ActionEvent e)
   {
      showDialog();
   }

}

class BeeperTestSystem extends GUIAgentSystem
{
   public BeeperTestSystem(int port,String library_dir,String homedir,boolean exitOnLast,String savedFile,boolean useSecurity)
   {
      super(port,library_dir,homedir,exitOnLast,savedFile,useSecurity);
   }

   public static void main(String[] args)
   {
      String home=System.getProperty("user.home");
      String librarydir=System.getProperty("user.dir");
      boolean security=true;
      if(args.length<2)
         System.err.println("Usage: java BaseAgents.CommandReturn.BeeperAgent agentDir libraryDir [NOSEC]\n");
      if(args.length<1)
         System.err.println("Location of agents not specified: Using  users home dir: "+home);
      else
         home=args[0];
      if(args.length<2)
         System.err.println("Location of libraries not specified: Using  current dir: "+librarydir);
      else
         librarydir=args[1];
      if(args.length>2)
      {
         if(args[2].equalsIgnoreCase("NOSEC"))
            security=false;
      }
      BeeperTestSystem as=new BeeperTestSystem(-1,librarydir,home,false,null,security);
      as.start();
      AgentIdentityExtension identity=new AgentIdentityExtension("Beeper","0",as.getLocation(),"Michael Sonntag");
      try
      {
         identity.addInformation(AgentIdentityExtension.FIRST_NAME,"Michael");
         identity.addInformation(AgentIdentityExtension.LAST_NAME,"Sonntag");
         identity.addInformation(AgentIdentityExtension.TITLE,"Dipl.-Ing.");
         identity.addInformation(AgentIdentityExtension.EMAIL,"sonntag@fim.uni-linz.ac.at");
         identity.addInformation("Magic word","James POND");
         as.startAgent("BaseAgents.Beeper.BeeperAgent","BaseAgents.jar",false,identity,null,null,true);
      }
      catch(Exception e)
      {
         System.err.println(e.toString());
      }
      try
      {
         as.join();
      }
      catch(InterruptedException e)
      {   // Do nothing, just end (Shouldn't happen anyway)
      }
      System.exit(0);
   }
}
```

ServletAgent.java

```
/**
 * @(#)ServletAgent.java
 *
 * Combined with the AgentSystem "POND"
 * Developed at the Institute for Information Processing and Microprocessor Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 *
 */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Enumeration;
import java.util.Hashtable;
import java.text.*;
import java.lang.*;
import java.net.*;
import java.net.InetAddress;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.lang.*;



/**
 * The ServletAgent() class.
 * This class is used as a communication between the ServletAgent and the AgentSystem.
 * When connected to the AgentSystem via socket, it can create, delete, display general
 * and special agent information, and depending on the AgentSystem's response, it can
 * generate various results. The ServletAgent is mainly used for retrieving all the
 * information of the AgentSystem, interacting and displaying them through a friendly HTML.
 *
 * @author  Marios Kalnis
 * @version 1.0, 1.7.2001
 */
public class ServletAgent extends HttpServlet {

        //The standard port of the Servlet Agent.
        final static int Servlet_Port = 6000;

        //Declaration of the Servlet's URL.
        final static String Servlet_Address = "127.0.0.1";

        /**
    * Called by the web server when a HTTP GET request is made.
    * @param req The GET request information.
    * @param res The HTTP response object.
    * @exception IOException Used when inputing form data or writing
    *           the form data back to the browser.
    */
        public void doGet(HttpServletRequest req, HttpServletResponse res)
                                                        throws ServletException, IOException {

          /**
           * Initialisation of all the Strings used to display all the requested values
           * of the ServletAgent in conjuction with the AgentSystem. These values are
           * used to create, display, delete, display special information, as well as to
           * check whether there is a success of the user's request in order to continue
           * in creating, displaying, deleting or displaying special information of an
           * agent respectively (used by String result_of_Submission). Only the first five
           * strings realted with the agent are provided by the user. All the other are
           * automatically generated with the help of the AgentSystem.
           *
           * result_of_Submission The string that checks in combination with the AgentSystem,
           *         whether the submission is correct, such as to proceed to the next stage.
           * value_for_AgentName The string submitting and displaying the value of the name of the agent.
           * value_for_AgentID The string submitting and displaying the value of the unique ID of the agent.
           * value_for_URL The string submitting and displaying the value of the agent's URL.
           * value_for_OwnerName The string submitting and displaying the value of the owner's name.
           * value_for_ClassName The string submitting the value of the agent's class name.
```

```
* value_for_PublicKeyName The string displaying the value of the agent's public key name.
* value_for_Certificate The string displaying the value of the agent's certificate.
* value_for_isSigned The string displaying the value of whether the agent is signed or not.
* value_for_InitialDeployed The string displaying whether the value of the agent initial deployed or not.
* value_for_CodeDeployed The string displaying the value of the agent's deployed code.
* value_for_CodeOrigin The string displaying the value of the agent's code origin.
* value_for_LocalCode The string displaying the whether agent's local code is true or false.
* value_for_PermissionGroup The displaying giving the level of the agent's permission group.
* value_for_Scale The string displaying the level of the agent's scale.
* value_for_AgentClassName The string displaying the value of the agent's class name.
* value_for_AgentWebInterface The string that displays the information sent by the WebGUI Agent.
*/
String result_of_Submission = "";
String value_for_AgentName = "";
String value_for_AgentID = "";
String value_for_URL = "";
String value_for_OwnerName = "";
String value_for_ClassName = "";
String value_for_PublicKeyName = "";
String value_for_Certificate = "";
String value_for_isSigned = "";
String value_for_InitialDeployed = "";
String value_for_CodeDeployed = "";
String value_for_CodeOrigin = "";
String value_for_LocalCode = "";
String value_for_PermissionGroup = "";
String value_for_Scale = "";
String value_for_AgentClassName = "";
String value_for_AgentWebInterface = "";


if("WebAgent".equals(req.getParameter("action"))) {

    //Open socket connection
    try {
                    Socket sock = new Socket(InetAddress.getByName(Servlet_Address), Servlet_Port);
                     PrintWriter out= new PrintWriter(sock.getOutputStream());
                     out.println("Sending Client's Information");
                     out.flush();
                     BufferedReader br = new BufferedReader(new InputStreamReader(sock.getInputStream()));
                     System.out.println(br.readLine());
                    /**
                     * The command "5" = "SendAgentToRemoteComputer" is sent to the AgentSystem via socket.
                     * Afterwards, buffer is flushed in order to send the command successfully.
                     */
                       out.println("5");
                       out.println(req.getParameter("AgentID"));

                       int lines_counted = 0;
                       Enumeration paramNames = req.getParameterNames();
                       while (paramNames.hasMoreElements())
                       {
                          lines_counted++;
                          paramNames.nextElement();
                       }//end of "while" statement

                       out.println("" + lines_counted);
                       paramNames = req.getParameterNames();
                       while (paramNames.hasMoreElements())
                       {
                         String paramName = (String) paramNames.nextElement();
                         out.println(paramName);
                         String[] paramValues = req.getParameterValues(paramName);
                         if (paramValues.length == 0) {
                            out.println("No value");
                         }
                         else {
                               String str="";
                    for (int n=0; n<paramValues.length; n++) {
                        if(n>0)
                          str+=" / ";
                          str=str+paramValues[n];
                        }
                        int pos=str.indexOf('\r');
                        while(pos>=0)
```

```
                        {
                           str=str.substring(0,pos)+str.substring(pos+1);
                           pos=str.indexOf('\r');
                        }
                        pos=str.indexOf('\n');
                        while(pos>=0)
                        {
                           str=str.substring(0,pos)+" / "+str.substring(pos+1);
                           pos=str.indexOf('\n');
                        }
                out.println(str);
                     }
                   }//end of "while" statement
                   out.flush();
                   res.setContentType("text/html");

/**
 * Reads the relevant line from the AgentSystem, and depending the react of the AgentSystem
 * to the System output, it displays either the "SUCCESS" or the "ERROR" HTML page.
 */
result_of_Submission=br.readLine();
System.out.println("result_of_Submission " + result_of_Submission);
/**
 * Taking under consideration the chance that there was successfull communication
 * between the ServletAgent and the AgentSystem, then the values of all the strings
 * of the AgentSystem are being read, in order to be displayed on an HTML page later on.
 * This technique catches the exception, where the system checks beforehand, whether it
 * can display the information or not, distinguishing whether to display an error or a
 * successfull HTML page.
 * The values that they are read from the AgentSystem, are all these defined initially
 * in the doGet method.
 */
if (result_of_Submission.startsWith("SUCCESS"))
{
  System.out.println("in success");
  value_for_AgentName = br.readLine();
  value_for_AgentID = br.readLine();
  /**
   * Reads the lines of what is being sent by the AgentThread() class.
   * It may be the case that there is more than one line.
   */
   String lines_for_HTMLAgentPage = br.readLine();
   int lines = 1;
   try
   {
     lines = Integer.parseInt(lines_for_HTMLAgentPage);
   }
   catch(NumberFormatException e)
   {
   }
   value_for_AgentWebInterface = br.readLine();
   for(int i=1; i<lines; i++)
     value_for_AgentWebInterface += "\n<br>" + br.readLine();
}//end of if-statement
                out = res.getWriter();

/**
 * If the output of the AgentSystem gives a message which start with the word SUCCESS,
 * then the "Display GUI Agent" HTML page of the requested agent are displayed.
 *
 * result_of_Submission The output of the AgentSystem.
 */
 if (result_of_Submission.startsWith("SUCCESS"))
    {
                        out.println("<HTML>");
                        out.println("<HEAD><TITLE>Agent System | Displaying Web GUI of an Agent</TITLE></HEAD>");
                        out.println("<BODY>");
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=0 CELLSPACING=2 WIDTH=450><TR>");
                        out.println("<TD ALIGN=center><H2><I>Displaying Web GUI of an Agent</I></H2></TD>");
                        out.println("<TD ALIGN=right ALIGN=center WIDTH=100>");
                        out.println("<A HREF=?curPage=getAboutPage>");
                        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem Extended
                                Version]\" BORDER=0>");
                        out.println("</A></TD></TR><TD>");
```

```
                        out.println("<TABLE BORDER=1 CELLPADDING=1 CELLSPACING=1 WIDTH=450>");
                        out.println("<TR><TD ALIGN=center WIDTH=175><B>Agent Name</B></TD>");
                        out.println("<TD ALIGN=center WIDTH=175><B>Agent ID</B></TD></TR>");
                        out.println("<TR><TD ALIGN=center WIDTH=175>" + value_for_AgentName + "</TD>");
                        out.println("<TD ALIGN=center WIDTH=175><FONT COLOR=red>" + value_for_AgentID +
                                    "</FONT></TD></TR></TABLE></TD>");
                        out.println("<TD ALIGN=center WIDTH=100>");
                        out.println("<CENTER>");
                        out.println("<A HREF=?curPage=getSpecialParameters&value_for_AgentID=" + value_for_AgentID+">Display
                                     Special Information</A>");
                        out.println("</CENTER>");
                        out.println("</TD></TR></TABLE>");
                        out.println("<HR>");
                        out.println("<BR>");
                        out.println(value_for_AgentWebInterface);
                        out.println("<BR>");
                        out.println("</FORM>");
                        out.println("</BODY>");
                        out.println("</HTML>");
                    }

else {

   /**
    * Else, if such a GUI Agent does not exist, then display an "Error" HTML. In this case, the
    * "Error" HTML takes under consideration the case of an undefined error, where there is
    * such an agent created in the system, but due to internal error cannot be displayed.
    *
    * result_of_Submission The output of the AgentSystem.
    */
    if(result_of_Submission.startsWith("ERROR displaying"))
      {
                        out.println("<HTML>");
                        out.println("<HEAD><TITLE>Agent System | Displaying Web GUI of an Agent : UNDEFINED
                                     ERROR!</TITLE></HEAD>");
                        out.println("<BODY>");
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                                     WIDTH=600><TR><TD>");
                        out.println("<TD ALIGN=right WIDTH=500><H2><I>Undefined Error displaying WebGUI of an
                                     Agent!</I></H2>");
                        out.println("</TD><TD ALIGN=left WIDTH=100>");
                        out.println("<A HREF=?curPage=getAboutPage>");
                        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                                     Extended Version]\" BORDER=0>");
                        out.println("</A></TD></TR></TABLE></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><H4>There was an undefined error displaying the web graphical user interface of the
                                     <FONT COLOR=blue>" + value_for_AgentID +
                                     " </FONT>Agent ID : " + result_of_Submission + "Please, go back and try submitting another
                                      Agent ID again!</H4></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)></CENTER>");
                        out.println("</FORM>");
                        out.println("</BODY>");
                        out.println("</HTML>");
   }

                       /**
                        * "Error" HTML taking under consideration the case where there was not such
                        *  an agent created by the AgentSystem, therefore could not be displayed!
                        */
                       else {
                            out.println("<HTML>");
                            out.println("<HEAD><TITLE>Agent System | Displaying Web GUI of an Agent :
                                         ERROR!</TITLE></HEAD>");
                            out.println("<BODY>");
                            out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                            out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                                         WIDTH=600><TR><TD>");
                            out.println("<TD ALIGN=right WIDTH=500><H2><I>Error displaying Web GUI of an Agent!</I></H2>");
                            out.println("</TD><TD ALIGN=left WIDTH=100>");
                            out.println("<A HREF=?curPage=getAboutPage>");
                            out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                                         Extended Version]\" BORDER=0>");
```

```
                out.println("</A></TD></TR></TABLE></CENTER>");
                out.println("<BR><BR>");
                out.println("<CENTER><H4>There was an error displaying the graphical user interface of the <FONT COLOR=blue>" +
                            value_for_AgentID +
                            " </FONT>Agent ID : " + "<FONT COLOR=red>" + result_of_Submission +
                            "</FONT>. Please, go back and try submitting another Agent ID again!</H4></CENTER>");
                out.println("<BR><BR>");
                out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)></CENTER>");
                out.println("</FORM>");
                out.println("</BODY>");
                out.println("</HTML>");
               }
     }
     /**
      * Closing the socket on the client-side.
      */
     sock.close();
}//end of socket "try" statement
catch (SocketException se) {
      System.out.println("Server Socket Problem" + se.getMessage());
}

                    }//end of "action"'s if-statement

                    else {

                    /**
                     * Initialising the output of the HTML page. Defining the output content type.
                     *
                     * res The HTTP response object
                     * out The stream to write out
                     * in The stream to read in
                     */
                    res.setContentType("text/html");
                    PrintWriter out = res.getWriter();
                    BufferedReader in = req.getReader();

                    /**
                     * Initialising the socket of the ServletAgent (Client-side) system.
                     *
                     * sock The URL address and the port where the socket will open later on
                     * br The stream to read in of the Socket
                     * pw The stream to write out of the Socket
                     */
                     Socket sock = null;
                    BufferedReader br = null;
                    PrintWriter pw = null;

try {
        /**
         * This if-statement compares the values between the value "homepage" and the parameter
         * "curPage". If these values are equal, then it procceeds to other choices, otherwise,
         * it returns a null value and then it displays the intial HTML page of the ServletAgent.
         *
         * homepage The value of the initial HTML (introductory) page, defined as a hidden field.
         * curPage The name of the initial HTML (introductory) page, defined as a hidden field.
         */
        if ("homepage".equals(req.getParameter("curPage"))) {

          /**
           * If the command "Information" is equal to the pre-defined Parameter "Agents", then
           * display the HTML page of displaying an agent. Provide the number of Agents currently
           * created, as well as the basic details of each agent separately.
           *
           * Information The value of the variable of the initial HTML page of the ServletAgent.
           * Agents The name of the variable of the initial HTML page of the ServletAgent.
           */
           if ("Information".equals(req.getParameter("Agents"))) {

             /**
              * At the beginning, the values of each of the strings are initialised as well as
              * an integer, in order to display the number of agents that are currently created.
              *
              * size_of_Agents An integer retrieving and displaying the number of Agents (their size).
              * info_of_AgentName The string submitting and displaying the value of the name of the agent.
```

```
 * info_of_AgentID The string submitting and displaying the value of the unique ID of the agent.
 * info_of_AgentURL The string submitting and displaying the value of the agent's URL.
 * info_of_OwnerName The string submitting and displaying the value of the owner's name.
 * info_of_AgentWebInterface The string for checking whether to display the link of an Agent
 *                          which has GUI, or not.
 */
int size_of_Agents = 0;
String[] info_of_AgentName = null;
String[] info_of_AgentID = null;
String[] info_of_AgentURL = null;
String[] info_of_OwnerName = null;
String[] info_of_AgentWebInterface = null;

        /**
         * Opening Socket for the "Display Agent Information" page on Port Servlet_Port
         */
           try {
   sock = new Socket(InetAddress.getByName(Servlet_Address), Servlet_Port);
   pw = new PrintWriter(sock.getOutputStream());
   pw.println("Sending Client's Information");
   pw.flush();
   br = new BufferedReader(new InputStreamReader(sock.getInputStream()));
   System.out.println(br.readLine());
   /**
    * The command "0" = "Display Agent Information" is sent to the AgentSystem via socket.
    * Afterwards, buffer is flushed in order to send the command successfully.
    */
   pw.println("0");
   pw.flush();
   size_of_Agents = Integer.parseInt(br.readLine());
   info_of_AgentName = new String[size_of_Agents];
   info_of_AgentID = new String[size_of_Agents];
   info_of_AgentURL = new String[size_of_Agents];
   info_of_OwnerName = new String[size_of_Agents];
   info_of_AgentWebInterface = new String[size_of_Agents];

   /**
    * Using a "for" loop to read each time the four strings that each agent
    * has respectively.
    */
   for (int i=0; i<size_of_Agents; i++)
   {
         info_of_AgentName[i] = br.readLine();
         info_of_AgentID[i] = br.readLine();
         info_of_AgentURL[i] = br.readLine();
         info_of_OwnerName[i] = br.readLine();
         info_of_AgentWebInterface[i] = br.readLine();
   }
   sock.close();
}//end of socket "try" statement

catch (SocketException se) {
   System.out.println("Server Socket Problem" + se.getMessage());
}

out.println("<HTML>");
out.println("<HEAD><TITLE>Agent System | Information</TITLE></HEAD>");
out.println("<BODY>");
out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3 WIDTH=500><TR><TD>");
out.println("<TD ALIGN=right WIDTH=400><H2><I>Displaying Agent Information</I></H2>");
out.println("</TD><TD ALIGN=left WIDTH=100>");
out.println("<A HREF=?curPage=getAboutPage>");
out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem Extended Version]\"
                  BORDER=0>");
out.println("</A></TD></TR></TABLE></CENTER>");
out.println("<BR>");
out.println("<CENTER><H3>Your results are as follows : " + size_of_Agents + " Agent(s)</H3></CENTER>");
out.println("<BR><BR>");

/**
 * Using a "for" loop to display all the agents, each one with all of each value respectively.
 *
 * i The variable counting the number of agents currently available.
 */
```

```
    for(int i=0; i<size_of_Agents; i++)
    {
       out.println("<B>");
       out.println("Agent "+ (i+1) + " has the following values : ");
       out.println("<BR>");
       out.println("</B>");
       out.println("<TABLE><TR><TD>");
       out.println("<A HREF=?curPage=getDeletionVerification&value_for_AgentID=" + info_of_AgentID[i]+">Delete</A>");
       out.println("</TD><TD>");
       out.println("<A HREF=?curPage=getSpecialParameters&value_for_AgentID=" + info_of_AgentID[i]+">Display</A>");
       out.println("</TD><TD>");
       if (info_of_AgentWebInterface[i].startsWith("TRUE")) {
          out.println("<A HREF=?curPage=getWebInterface&value_for_AgentID=" + info_of_AgentID[i]+">Display
                       WebAgent</A>");
       }
       else {
          out.println("");
       }
       out.println("</TD></TR></TABLE>");
       out.println("<TABLE BORDER=0><TR VALIGN=left><TD ALIGN=left>");
       out.println("<I>Agent's Name </I>: </TD>" + "<TD>" + info_of_AgentName[i] + "</TD>");
       out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
       out.println("<I><FONT COLOR=blue>Agent's ID  </FONT></I> : </TD>" + "<TD>" + info_of_AgentID[i] + "</TD>");
       out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
       out.println("<I>Agent's URL  </I>: </TD>" + "<TD>" + info_of_AgentURL[i] + "</TD>");
       out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
       out.println("<I>Owner's Name </I>: </TD>" + "<TD>" + info_of_OwnerName[i] + "</TD>");
       out.println("</TD></TR></TABLE>");
       out.println("<BR>");
    }//end of "for" loop
    out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Menu onClick=parent.location='ServletAgent'></CENTER>");
    out.println("</FORM>");
    out.println("</BODY>");
                    out.println("</HTML>");
}//end of "DisplayInformation" if-statement

/**
 * If the command "Create" is equal to the pre-defined Parameter "Agents", then
 * display the initial HTML page of creating an agent. Also, provide the fields
 * for the values needed to filled in by the user, in order to display them later on.
 *
 * Create The value of the variable of the initial HTML page of the ServletAgent.
 * Agents The name of the variable of the initial HTML page of the ServletAgent.
 * value_for_AgentName The name of the specified field that its value will be saved.
 * value_for_AgentID The name of the specified field that its value will be saved.
 * value_for_URL The name of the specified field that its value will be saved.
 * value_for_OwnerName The name of the specified field that its value will be saved.
 * value_for_ClassName The name of the specified field that its value will be saved.
 */
else if ("Create".equals(req.getParameter("Agents")))
{
    out.println("<HTML>");
    out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
    out.println("<HEAD><TITLE>Agent System | Creating Agent</TITLE></HEAD>");
    out.println("<BODY>");
    out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3 WIDTH=300><TR><TD>");
    out.println("<TD ALIGN=right WIDTH=200><H2><I>Creating Agent</I></H2>");
    out.println("</TD><TD ALIGN=left WIDTH=100>");
    out.println("<A HREF=?curPage=getAboutPage>");
    out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem Extended Version]\"
                 BORDER=0>");
    out.println("</A></TD></TR></TABLE></CENTER>");
    out.println("<BR>");
    out.println("<CENTER><H4>Please, fill in the fields below :</H4></CENTER>");
    out.println("<BR>");
    out.println("<CENTER>");
    out.println("<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=3 WIDTH=428>");
    out.println("<TR ALIGN=center><TD ALIGN=right>");
    out.println("Agent's Name :");
    out.println("</TD><TD ALIGN=right>");
    out.println("<INPUT TYPE=text NAME=value_for_AgentName VALUE=\"\" SIZE=40 MAXLENGTH=40>");
    out.println("</TD></TR>");
    out.println("<TR ALIGN=center><TD ALIGN=right>");
    out.println("Agent's ID :");
    out.println("</TD><TD ALIGN=right>");
```

```
        out.println("<INPUT TYPE=text NAME=value_for_AgentID VALUE=\"\" SIZE=40 MAXLENGTH=40>");
        out.println("</TD></TR>");
        out.println("<TR ALIGN=center><TD ALIGN=right>");
        out.println("Agent's URL :");
        out.println("</TD><TD ALIGN=right>");
        out.println("<FONT COLOR=blue>amp://</FONT><INPUT TYPE=text NAME=value_for_URL SIZE=40
                        MAXLENGTH=60>");
        out.println("</TD></TR>");
        out.println("<TR ALIGN=center><TD ALIGN=right>");
        out.println("Owner's Name :");
        out.println("</TD><TD ALIGN=right>");
        out.println("<INPUT TYPE=text NAME=value_for_OwnerName VALUE=\"\" SIZE=40 MAXLENGTH=40>");
        out.println("</TD></TR>");
        out.println("<TR ALIGN=center><TD ALIGN=right>");
        out.println("Class Name :");
        out.println("</TD><TD ALIGN=right>");
        out.println("<INPUT TYPE=text NAME=value_for_ClassName VALUE=\"\" SIZE=40 MAXLENGTH=80>");
        out.println("</TD></TR>");
        out.println("</TABLE>");
        out.println("</CENTER>");
        out.println("<BR><BR>");
        out.println("<CENTER><TABLE><TR><TD>");
        out.println("<INPUT TYPE=hidden NAME=curPage VALUE=getCreateParameters>");
        out.println("<INPUT TYPE=submit VALUE=Create>");
        out.println("<INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>");
        out.println("<INPUT TYPE=\"reset\"" + " VALUE=\"Reset\">");
        out.println("</TD></TR></TABLE></CENTER>");
        out.println("<BR><BR><CENTER><U>Note</U> : All fields are required!</CENTER>");
        out.println("</BODY>");
        out.println("</FORM>");
        out.println("</HTML>");
    }

    /**
     * If the command "Delete" is equal to the pre-defined Parameter "Agents", then
     * display the initial HTML page of deleting an agent. Ask from the user to provide
     * the unique ID of the created agent, in order to procceed to deletion.
     *
     * Delete The value of the variable of the initial HTML page of the ServletAgent.
     * Agents The name of the variable of the initial HTML page of the ServletAgent.
     * value_for_AgentID The name of the specified field that its value has to be given.
     */
    else if ("Delete".equals(req.getParameter("Agents")))
      {
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Agent System | Deleting Agent</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3 WIDTH=300><TR><TD>");
        out.println("<TD ALIGN=right WIDTH=200><H2><I>Deleting Agent</I></H2>");
        out.println("</TD><TD ALIGN=left WIDTH=100>");
        out.println("<A HREF=?curPage=getAboutPage>");
        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem Extended Version]\"
                        BORDER=0>");
        out.println("</A></TD></TR></TABLE></CENTER>");
        out.println("<BR>");
        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
        out.println("<CENTER><H4>Please, fill in the field below and press the <I>Delete</I> button.</H4></CENTER>");
        out.println("<BR>");
        out.println("<CENTER>");
        out.println("Agent's ID : <INPUT TYPE=text NAME=value_for_AgentID VALUE=\"\" SIZE=40 MAXLENGTH=40>");
        out.println("</CENTER>");
        out.println("<BR><BR>");
        out.println("<CENTER>");
        out.println("<TABLE><TR><TD>");
        out.println("<INPUT TYPE=submit VALUE=\"Delete\">");
        out.println("</TD><TD>");
        out.println("<INPUT TYPE=hidden NAME=curPage VALUE=getDeleteParameters>");
        out.println("<INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>");
        out.println("</TD></TR></TABLE>");
        out.println("</CENTER>");
        out.println("</FORM>");
        out.println("</BODY>");
         out.println("</HTML>");
    }
```

```
/**
  * If the command "Special" is equal to the pre-defined Parameter "Agents", then
  * display the initial HTML page of displaying special information of an agent. Ask
  * from the user to provide the unique ID of the created agent, in order to procceed.
  *
  * Special The value of the variable of the initial HTML page of the ServletAgent.
  * Agents The name of the variable of the initial HTML page of the ServletAgent.
  * value_for_AgentID The name of the specified field that its value has to be given.
  */
  else if ("Special".equals(req.getParameter("Agents"))) {

     out.println("<HTML>");
     out.println("<HEAD><TITLE>Agent System | Displaying Special Information</TITLE></HEAD>");
     out.println("<BODY>");
     out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3 WIDTH=600><TR><TD>");
     out.println("<TD ALIGN=right WIDTH=500><H2><I>Displaying Special Agent Information</I></H2>");
     out.println("</TD><TD ALIGN=left WIDTH=100>");
     out.println("<A HREF=?curPage=getAboutPage>");
     out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem Extended Version]\"
                  BORDER=0>");
     out.println("</A></TD></TR></TABLE></CENTER>");
     out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
     out.println("<BR><BR>");
     out.println("<CENTER><H4>Please, fill in the field below and then press the <I>Display</I>
                  button.</H4></CENTER><BR>");
     out.println("<CENTER>");
     out.println("Agent's ID : <INPUT TYPE=text NAME=value_for_AgentID VALUE=\"\" SIZE=40 MAXLENGTH=40>");
     out.println("</CENTER>");
     out.println("<BR>");
     out.println("<BR>");
     out.println("<CENTER><TABLE><TR><TD>");
     out.println("<INPUT TYPE=hidden NAME=curPage VALUE=getSpecialParameters>");
     out.println("<CENTER><INPUT TYPE=submit VALUE=\"Display\">");
     out.println("</TD><TD>");
     out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)></CENTER>");
     out.println("</TD></TR></TABLE></CENTER>");
     out.println("</FORM>");
     out.println("</BODY>");
     out.println("</HTML>");
  }

  /**
   * If the command "WebInterface" is equal to the pre-defined Parameter "Agents", then
   * display the initial HTML page of displaying information of an agent which is with
   * a Graphical User Interface (GUI). Ask from the user to provide the unique ID of the
   * created agent, in order to procceed.
   *
   * WebInterface The value of the variable of the initial HTML page of the ServletAgent.
   * Agents The name of the variable of the initial HTML page of the ServletAgent.
   * value_for_AgentID The name of the specified field that its value has to be given.
   */
 else if("WebInterface".equals(req.getParameter("Agents"))) {

     out.println("<HTML>");
     out.println("<HEAD><TITLE>Agent System | Display Agent Information with GU Interface</TITLE></HEAD>");
     out.println("<BODY>");
     out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3 WIDTH=700><TR><TD>");
     out.println("<TD ALIGN=right WIDTH=600><H2><I>Display Agent Information with GU Interface</I></H2>");
     out.println("</TD><TD ALIGN=left WIDTH=100>");
     out.println("<A HREF=?curPage=getAboutPage>");
     out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem Extended Version]\"
                  BORDER=0>");
     out.println("</A></TD></TR></TABLE></CENTER>");
     out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
     out.println("<BR><BR>");
     out.println("<CENTER><H4>Please, fill in the field below and then press the <I>Display</I>
                  button.</H4></CENTER><BR>");
     out.println("<CENTER>");
     out.println("Agent's ID : <INPUT TYPE=text NAME=value_for_AgentID VALUE=\"\" SIZE=40 MAXLENGTH=40>");
     out.println("</CENTER>");
     out.println("<BR>");
     out.println("<BR>");
     out.println("<CENTER><TABLE><TR><TD>");
     out.println("<INPUT TYPE=hidden NAME=curPage VALUE=getWebInterface>");
     out.println("<CENTER><INPUT TYPE=submit VALUE=\"Display\">");
```

```
        out.println("</TD><TD>");
        out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)></CENTER>");
        out.println("</TD></TR></TABLE></CENTER>");
        out.println("</FORM>");
        out.println("</BODY>");
                        out.println("</HTML>");
                    }

                }//end of ("homepage".equals(req.getParameter("curPage"))) if-statement

                  /**
                 * Considering the choice of "Creating an Agent", opening a socket and
                 * procceeding to user's creation choice of an agent.
                 *
                 * getCreateParameters The value of the "Creating an Agent" HTML page,
                 *                     defined as a hidden field.
                 * curPage The name of the initial HTML (introductory) page, defined
                 *         as a hidden field.
                   */
else if ("getCreateParameters".equals(req.getParameter("curPage")))
{
                    int NUMBER_OF_PARAMETERS = 5;

                    /**
                     * Opening Socket for the "Creating an Agent" on Port Servlet_Port
                     */
                    try {
    sock = new Socket(InetAddress.getByName(Servlet_Address), Servlet_Port);
    pw = new PrintWriter(sock.getOutputStream());
    pw.println("Sending Client's Information");
    pw.flush();
    br = new BufferedReader(new InputStreamReader(sock.getInputStream()));
    System.out.println(br.readLine());
    /**
     * The command "1" = "Create Agent" is sent to the AgentSystem via socket.
     * Afterwards, buffer is flushed in order to send the command successfully.
     */
    pw.println("1");
    pw.flush();
    /**
     * Sending all the values of the parameters to the AgentSystem.
     */
    pw.println(req.getParameter("value_for_AgentName"));
    pw.println(req.getParameter("value_for_AgentID"));
    pw.println("amp://" + req.getParameter("value_for_URL"));
    pw.println(req.getParameter("value_for_OwnerName"));
    pw.println(req.getParameter("value_for_ClassName"));
    pw.flush();
   /**
    * ServletAgent reads whether the submission of the parameters where successfull
    * to the AgentSystem or not.
    */
    result_of_Submission = br.readLine();
    System.out.println(result_of_Submission);
    sock.close();
  }//end of socket "try" statement

  /**
   * Catching the exception of the socket, in case there is an error.
   */
  catch (SocketException se) {
        System.out.println("Server Socket Problem" + se.getMessage());
  }

  /**
                    * If the submission is successfull, then display a "Thank You" HTML.
                    *
                    * result_of_Submission The output of the AgentSystem.
                    */
                   if(result_of_Submission.startsWith("SUCCESS"))
                       {
                        out.println("<HTML>");
                        out.println("<HEAD><TITLE>Agent System | Creating Agent : SAVED!</TITLE></HEAD>");
                        out.println("<BODY>");
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
```

```
    out.println("<CENTER>");
    out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                WIDTH=300><TR><TD>");
    out.println("<TD ALIGN=right WIDTH=200><H2><I>Agent Created!</I></H2>");
    out.println("</TD><TD ALIGN=left WIDTH=100>");
    out.println("<A HREF=?curPage=getAboutPage>");
    out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                Extended Version]\" BORDER=0>");
    out.println("</A></TD></TR></TABLE></CENTER>");
    out.println("<BR><BR>");
    out.println("<H4>Agent has been created! Please, press the button " +
                "below to return to main menu</H4>");
    out.println("<BR><BR>");
    out.println("<CENTER><INPUT TYPE=button VALUE=Menu
                onClick=parent.location='ServletAgent'></CENTER>");
    out.println("</CENTER>");
    out.println("</FORM>");
    out.println("</BODY>");
    out.println("</HTML>");
  }

  /**
   * Else, if submission is wrong, then display an "Error" HTML.
   *
   * result_of_Submission The output of the AgentSystem.
   */
  else {
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Agent System | Creating Agent : ERROR!</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
        out.println("<CENTER>");
        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                    WIDTH=400><TR><TD>");
        out.println("<TD ALIGN=right WIDTH=300><H2><I>Error Creating Agent!</I></H2>");
        out.println("</TD><TD ALIGN=left WIDTH=100>");
        out.println("<A HREF=?curPage=getAboutPage>");
        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                    Extended Version]\" BORDER=0>");
        out.println("</A></TD></TR></TABLE></CENTER>");
        out.println("<BR><BR>");
        out.println("<H4>There was an error creating the Agent : <FONT COLOR=red>" + result_of_Submission
                    +".  " + "</FONT>Please, go back and try creating the Agent again!</H4>");
        out.println("<BR><BR>");
        out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>");
        out.println("<INPUT TYPE=button VALUE=Menu
                    onClick=parent.location='ServletAgent'></CENTER>");
        out.println("</CENTER>");
        out.println("</FORM>");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}//end of "getCreateParameters" else-if statement

/**
 * Considering the choice of "Deleting an Agent" this code checks whether the user
 * wants to delete the specified agent. It is combined with the link given to the
 * "Display Information" page. Depending on the user's choice (to delete or not), it
 * will either procceed to "getDeleteParameters" code (as shown below), or it will
 * go back respectively.
 *
 * getDeletionVerification The value of the verification to procceeding to the "Deleting
 *                         an Agent" HTML page or not, defined as a hidden field in the
 *                         "Display Agent Information" HTML page.
 * curPage The name of the initial HTML (introductory) page, defined as
 *         a hidden field.
 */
else if("getDeletionVerification".equals(req.getParameter("curPage"))) {

  out.println("<HTML>");
  out.println("<HEAD><TITLE>Verification | Are you sure you want to Delete the Agent?</TITLE></HEAD>");
  out.println("<BODY>");
  out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
  out.println("<CENTER>");
  out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
```

```
                        WIDTH=300><TR><TD>");
                    out.println("<TD ALIGN=right WIDTH=200><H2><I>Delete Agent?</I></H2>");
                    out.println("</TD><TD ALIGN=left WIDTH=100>");
                    out.println("<A HREF=?curPage=getAboutPage>");
                    out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                        Extended Version]\" BORDER=0>");
                    out.println("</A></TD></TR></TABLE></CENTER>");
                    out.println("<BR><BR>");
                    out.println("<H4>Are you sure you want to delete the specific agent? " +
                            "If so, please press <I>YES</I> button below.</H4>");
                    out.println("<BR><BR>");
                    out.println("<CENTER><TABLE><TR><TD>");
                    out.println("<INPUT TYPE=hidden Name=curPage VALUE=getDeleteParameters>");
                    out.println("<INPUT TYPE=hidden Name=value_for_AgentID
                        VALUE="+req.getParameter("value_for_AgentID")+">");
                    out.println("<INPUT TYPE=submit VALUE=Yes>");
                    out.println("</TD><TD>");
                    out.println("<INPUT TYPE=BUTTON VALUE=No onClick=history.go(-1)>");
                    out.println("</TD></TR></TABLE></CENTER>");
                    out.println("</FORM>");
                    out.println("</BODY>");
                    out.println("</HTML>");
                  }

                  /**
                 * Considering the choice of "Deleting an Agent", opening a socket and
                 * procceeding to user's deletion choice of an agent.
                 *
                 * getDeleteParameters The value of the "Deleting an Agent" HTML page,
                 *                     defined as a hidden field.
                 * curPage The name of the initial HTML (introductory) page, defined as
                 *         a hidden field.
                 */
                   else if ("getDeleteParameters".equals(req.getParameter("curPage"))) {

                     /**
                      * Opening Socket for the "Deleting an Agent" on Port Servlet_Port
 */
try {
   sock = new Socket(InetAddress.getByName(Servlet_Address), Servlet_Port);
   pw = new PrintWriter(sock.getOutputStream());
   pw.println("Sending Client's Information");
   pw.flush();
   br = new BufferedReader(new InputStreamReader(sock.getInputStream()));
   System.out.println(br.readLine());
   /**
    * The command "2" = "Delete Agent" is sent to the AgentSystem via socket.
    * Afterwards, buffer is flushed in order to send the command successfully.
    */
   pw.println("2");
   pw.flush();
   /**
    * Sending the all the "value_for_AgentID" to the AgentSystem.
    */
   pw.println(req.getParameter("value_for_AgentID"));
   pw.flush();
   result_of_Submission = br.readLine();
   /**
    * Closing the socket on the client-side.
    */
   sock.close();
}//end of socket "try" statement
catch (SocketException se) {
      System.out.println("Server Socket Problem" + se.getMessage());
}

                  /**
                    * If the deletion is successfull, then display a "Deleted!" HTML.
                    *
                    * result_of_Submission The output of the AgentSystem.
                    */
                      if (result_of_Submission.startsWith("SUCCESS"))
                       {
                         out.println("<HTML>");
                         out.println("<HEAD><TITLE>Agent System | Deleting Agent : DELETED!</TITLE></HEAD>");
```

```
                        out.println("<BODY>");
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                        out.println("<CENTER>");
                        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                                WIDTH=300><TR><TD>");
                        out.println("<TD ALIGN=right WIDTH=200><H2><I>Agent Deleted!</I></H2>");
                        out.println("</TD><TD ALIGN=left WIDTH=100>");
                        out.println("<A HREF=?curPage=getAboutPage>");
                        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                                Extended Version]\" BORDER=0>");
                        out.println("</A></TD></TR></TABLE></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<H4>Agent has been deleted! " +
                                "Please, press the button below to go to main menu.</H4>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><INPUT TYPE=button VALUE=Menu
                                onClick=parent.location='ServletAgent'></CENTER>");
                        out.println("</CENTER>");
                        out.println("</FORM>");
                        out.println("</BODY>");
                        out.println("</HTML>");
                    }

/**
 * Else, if deletion is wrong, then display an "Error" HTML.
 *
 * result_of_Submission The output of the AgentSystem.
 */
else {
   /**
    * "Error" HTML taking under consideration the case where there was such an agent that
    *  could be deleted, but there was an undefined internal error of the AgentSystem.
    */
   if(result_of_Submission.startsWith("ERROR deleting"))
   {
                        out.println("<HTML>");
                        out.println("<HEAD><TITLE>Agent System | Deleting Agent : UNDEFINED
                                ERROR!</TITLE></HEAD>");
                        out.println("<BODY>");
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                        out.println("<CENTER>");
                        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                                WIDTH=500><TR><TD>");
                        out.println("<TD ALIGN=right WIDTH=400><H2><I>Undefined Error deleting Agent!</I></H2>");
                        out.println("</TD><TD ALIGN=left WIDTH=100>");
                        out.println("<A HREF=?curPage=getAboutPage>");
                        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                                Extended Version]\" BORDER=0>");
                        out.println("</A></TD></TR></TABLE></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<H4>There was an undefined error deleting the specified Agent : <FONT COLOR=red>"
                                + result_of_Submission + "</FONT>. " + "Please, go back and try deleting again!</H4>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>");
                        out.println("<INPUT TYPE=button VALUE=Menu
                                onClick=parent.location='ServletAgent'></CENTER>");
                        out.println("</CENTER>");
                        out.println("</FORM>");
                        out.println("</BODY>");
                        out.println("</HTML>");
   }

   /**
    * "Error" HTML taking under consideration the case where there was not such
    *  an agent created by the AgentSystem, therefore could not be deleted!
    */
                    else
                    {
                        out.println("<HTML>");
                        out.println("<HEAD><TITLE>Agent System | Deleting Agent : ERROR!</TITLE></HEAD>");
                        out.println("<BODY>");
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                        out.println("<CENTER>");
                        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                                WIDTH=400><TR><TD>");
```

```
                                out.println("<TD ALIGN=right WIDTH=300><H2><I>Error deleting Agent!</I></H2>");
                                out.println("</TD><TD ALIGN=left WIDTH=100>");
                                out.println("<A HREF=?curPage=getAboutPage>");
                                out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                                        Extended Version]\" BORDER=0>");
                                out.println("</A></TD></TR></TABLE></CENTER>");
                                out.println("<BR><BR>");
                                out.println("<H4>There was an error deleting the specified Agent : <FONT COLOR=red>"
                                        + result_of_Submission + "</FONT>. " + "Please, go back and try deleting again!</H4>");
                                out.println("<BR><BR>");
                                out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>");
                                out.println("<INPUT TYPE=button VALUE=Menu
                                        onClick=parent.location='ServletAgent'></CENTER>");
                                out.println("</CENTER>");
                                out.println("</FORM>");
                                out.println("</BODY>");
                                out.println("</HTML>");
                            }
                        }
}//end of else-if "getDeleteParameters" statement

/**
                    * Considering the choice of "Displaying Special Information of an Agent",
                    * opening a socket and procceeding to user's display choice of an agent.
                    *
                    * getSpecialParameters The value of the "Displaying Special Information of
                    *                      an Agent" HTML page, defined as a hidden field.
                    * curPage The name of the initial HTML (introductory) page, defined as
 *        a hidden field.
                    */
                      else if ("getSpecialParameters".equals(req.getParameter("curPage"))) {

  //Opening Socket for the "Display Special Agent Information" on Port Servlet_Port
  try {
     sock = new Socket(InetAddress.getByName(Servlet_Address), Servlet_Port);
     pw = new PrintWriter(sock.getOutputStream());
     pw.println("Sending Client's Information");
     pw.flush();
     System.out.println("Display Special Info");
     /**
      * The command "3" = "Display Special Information" is sent to the AgentSystem via socket.
      * Afterwards, buffer is flushed in order to send the command successfully.
      */
     pw.println("3");
     pw.flush();
     pw.println(req.getParameter("value_for_AgentID"));
     pw.flush();
     br = new BufferedReader(new InputStreamReader(sock.getInputStream()));
     System.out.println(br.readLine());
     /**
      * Reads the relevant line from the AgentSystem, and depending the react of the AgentSystem
      * to the System output, it displays either the "SUCCESS" or the "ERROR" HTML page.
      */
     result_of_Submission=br.readLine();
     System.out.println("result_of_Submission" + result_of_Submission);
     /**
      * Taking under consideration the chance that there was successfull communication
      * between the ServletAgent and the AgentSystem, then the values of all the strings
      * of the AgentSystem are being read, in order to be displayed on an HTML page later on.
      * This technique catches the exception, where the system checks beforehand, whether it
      * can display the information or not, distinguishing whether to display an error or a
      * successfull HTML page.
      * The values that they are read from the AgentSystem, are all these defined initially
      * in the doGet method.
      */
     if (result_of_Submission.startsWith("SUCCESS"))
     {
       System.out.println("in success");
       value_for_AgentName = br.readLine();
       value_for_AgentID = br.readLine();
       value_for_URL = br.readLine();
       value_for_OwnerName = br.readLine();
       value_for_PublicKeyName = br.readLine();
       String lines_for_certificate = br.readLine();
       int lines = 1;
```

```
        try
        {
            lines = Integer.parseInt(lines_for_certificate);
        }
        catch(NumberFormatException e)
        {
        }
        value_for_Certificate = br.readLine();
        for(int i=1; i<lines; i++)
        value_for_Certificate += "\n<br>" + br.readLine();
        value_for_isSigned = br.readLine();
        value_for_InitialDeployed = br.readLine();
        value_for_CodeDeployed = br.readLine();
        value_for_CodeOrigin = br.readLine();
        value_for_LocalCode = br.readLine();
        value_for_PermissionGroup = br.readLine();
        value_for_Scale = br.readLine();
        value_for_AgentClassName = br.readLine();
      }
      /**
       * Closing the socket on the client-side.
       */
      sock.close();
}//end of socket "try" statement
catch (SocketException se) {
       System.out.println("Server Socket Problem" + se.getMessage());
}

/**
 * If the output of the AgentSystem gives a message which start with the word SUCCESS,
 * then the "Special Information" HTML page of the requested agent are displayed.
 *
 * result_of_Submission The output of the AgentSystem.
 */
if (result_of_Submission.startsWith("SUCCESS"))
  {
                        out.println("<HTML>");
                        out.println("<HEAD><TITLE>Agent System | Displaying Special Agent Information</TITLE></HEAD>");
                        out.println("<BODY>");
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                                WIDTH=600><TR><TD>");
                        out.println("<TD ALIGN=right WIDTH=500><H2><I>Displaying Special Agent Information</I></H2>");
                        out.println("</TD><TD ALIGN=left WIDTH=100>");
                        out.println("<A HREF=?curPage=getAboutPage>");
                        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                                Extended Version]\" BORDER=0>");
                        out.println("</A></TD></TR></TABLE></CENTER>");
                        out.println("<CENTER>");
                        out.println("<BR><BR>");
                        out.println("<H4>Your request for Agent <I>" + value_for_AgentName + "</I> is as follows :</H4>");
                        out.println("<BR>");
                        out.println("<TABLE BORDER=1 CELLPADDING=1 CELLSPACING=1><TR VALIGN=left><TD
                                ALIGN=left>");
                        out.println("<I>Agent's Name </I>: </TD>" + "<TD>" + value_for_AgentName + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
                        out.println("<I><FONT COLOR=blue>Agent's ID  </FONT></I> : </TD>" + "<TD>" + value_for_AgentID
                                + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
                        out.println("<I>Agent's URL  </I>: </TD>" + "<TD>" + value_for_URL + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
                        out.println("<I>Owner's Name </I>: </TD>" + "<TD>" + value_for_OwnerName + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left WIDTH=300>");
                        out.println("<I>Public Key Name </I>: </TD>" + "<TD>" + value_for_PublicKeyName + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
                        out.println("<I>Owner's Certificate </I>: </TD>" + "<TD>" + value_for_Certificate + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
                        out.println("<I>Signed </I>: </TD>" + "<TD WIDTH=600>" + value_for_isSigned + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
                        out.println("<I>Initial Deployed </I>: </TD>" + "<TD>" + value_for_InitialDeployed + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
                        out.println("<I>Code Deployed </I>: </TD>" + "<TD>" + value_for_CodeDeployed + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
                        out.println("<I>Code's Origin </I>: </TD>" + "<TD>" + value_for_CodeOrigin + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
```

```
                        out.println("<I>Local Code </I>: </TD>" + "<TD>" + value_for_LocalCode + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
                        out.println("<I>Permission Group </I>: </TD>" + "<TD>" + value_for_PermissionGroup + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
                        out.println("<I>Scale </I>: </TD>" + "<TD>" + value_for_Scale + "</TD>");
                        out.println("</TD></TR><TR VALIGN=left><TD ALIGN=left>");
                        out.println("<I>Agent's Class Name </I>: </TD>" + "<TD>" + value_for_AgentClassName + "</TD>");
                        out.println("</TD></TR></TABLE>");
                        out.println("</CENTER>");
                        out.println("<BR>");
                        out.println("<CENTER><TABLE><TR><TD>");
                        out.println("<INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>");
                        out.println("</TD><TD>");
                        out.println("<CENTER><INPUT TYPE=button VALUE=Menu
                                onClick=parent.location='ServletAgent'></CENTER>");
                        out.println("</TD></TR></TABLE>");
                        out.println("</CENTER>");
                        out.println("</FORM>");
                        out.println("</BODY>");
                        out.println("</HTML>");
                    }

else {

/**
 * Else, if deletion is wrong, then display an "Error" HTML. In this case, the "Error" HTML
 * takes under consideration the case of an undefined error, where there is such an agent
 * created in the system, but due to internal error cannot be displayed.
 *
 * result_of_Submission The output of the AgentSystem.
 */
  if(result_of_Submission.startsWith("ERROR displaying")) {
                        out.println("<HTML>");
                        out.println("<HEAD><TITLE>Agent System | Displaying Special Agent Information : UNDEFINED
                            ERROR!</TITLE></HEAD>");
                        out.println("<BODY>");
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                            WIDTH=600><TR><TD>");
                        out.println("<TD ALIGN=right WIDTH=500><H2><I>Undefined Error displaying Special Information of
                            Agent!</I></H2>");
                        out.println("</TD><TD ALIGN=left WIDTH=100>");
                        out.println("<A HREF=?curPage=getAboutPage>");
                        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                            Extended Version]\" BORDER=0>");
                        out.println("</A></TD></TR></TABLE></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><H4>There was an undefined error displaying the special information of the
                            <FONT COLOR=blue>" + value_for_AgentID +
                          " </FONT>Agent ID : " + result_of_Submission + "Please, go back and try submitting another
                            Agent ID again!</H4></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>");
                        out.println("<INPUT TYPE=button VALUE=Menu
                            onClick=parent.location='ServletAgent'></CENTER>");
                        out.println("</FORM>");
                        out.println("</BODY>");
                        out.println("</HTML>");
                    }

  /**
    * "Error" HTML taking under consideration the case where there was not such
    *  an agent created by the AgentSystem, therefore could not be displayed!
    */
    else {
                        out.println("<HTML>");
                        out.println("<HEAD><TITLE>Agent System | Displaying Special Agent Information :
                            ERROR!</TITLE></HEAD>");
                        out.println("<BODY>");
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                            WIDTH=600><TR><TD>");
                        out.println("<TD ALIGN=right WIDTH=500><H2><I>Error displaying Web GUI of an
                            Agent!</I></H2>");
                        out.println("</TD><TD ALIGN=left WIDTH=100>");
```

```
                        out.println("<A HREF=?curPage=getAboutPage>");
                        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                                Extended Version]\" BORDER=0>");
                        out.println("</A></TD></TR></TABLE></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><H4>There was an error displaying special information of the <FONT
                                COLOR=blue>" + value_for_AgentID +
                               " </FONT>Agent ID : " + "<FONT COLOR=red>" + result_of_Submission +
                               "</FONT>. Please, go back and try submitting another Agent ID again!</H4></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>");
                        out.println("<INPUT TYPE=button VALUE=Menu
                                onClick=parent.location='ServletAgent'></CENTER>");
                        out.println("</FORM>");
                        out.println("</BODY>");
                        out.println("</HTML>");
                    }
                  }
                }

                  /**
                * Considering the choice of "Displaying a Web GU Interface of an Agent",
                * opening a socket and procceeding to user's display choice of an agent.
                *
                * getWebInterface The value of the "Displaying a Web GU Interface of
                *               an Agent" HTML page, defined as a hidden field.
                * curPage The name of the initial HTML (introductory) page, defined as
*       a hidden field.
                */
                  else if ("getWebInterface".equals(req.getParameter("curPage"))) {

 //Opening Socket for the "Display web Interface of a GUI Agent" on Port Servlet_Port
 try {
    sock = new Socket(InetAddress.getByName(Servlet_Address), Servlet_Port);
    pw = new PrintWriter(sock.getOutputStream());
    pw.println("Sending Client's Information");
    pw.flush();
    System.out.println("Display Web GUI Agent");
    /**
     * The command "4" = "Display GUI Agent" is sent to the AgentSystem via socket.
     * Afterwards, buffer is flushed in order to send the command successfully.
     */
    pw.println("4");
    pw.flush();
    pw.println(req.getParameter("value_for_AgentID"));
    pw.flush();
    br = new BufferedReader(new InputStreamReader(sock.getInputStream()));
    System.out.println(br.readLine());
   /**
     * Reads the relevant line from the AgentSystem, and depending the react of the AgentSystem
     * to the System output, it displays either the "SUCCESS" or the "ERROR" HTML page.
     */
    result_of_Submission=br.readLine();
    System.out.println("result_of_Submission" + result_of_Submission);
    /**
     * Taking under consideration the chance that there was successfull communication
     * between the ServletAgent and the AgentSystem, then the values of all the strings
     * of the AgentSystem are being read, in order to be displayed on an HTML page later on.
     * This technique catches the exception, where the system checks beforehand, whether it
     * can display the information or not, distinguishing whether to display an error or a
     * successfull HTML page.
     * The values that they are read from the AgentSystem, are all these defined initially
     * in the doGet method.
     */
    if (result_of_Submission.startsWith("SUCCESS"))
    {
      System.out.println("in success");
      value_for_AgentName = br.readLine();
      value_for_AgentID = br.readLine();
      /**
       * Reads the lines of what is being sent by the AgentThread() class.
       * It may be the case that there is more than one line.
       */
      String lines_for_HTMLAgentPage = br.readLine();
      int lines = 1;
```

```
      try
      {
          lines = Integer.parseInt(lines_for_HTMLAgentPage);
      }
      catch(NumberFormatException e)
      {
      }
      value_for_AgentWebInterface = br.readLine();
      for(int i=1; i<lines; i++)
      value_for_AgentWebInterface += "\n<br>" + br.readLine();
    }
    /**
     * Closing the socket on the client-side.
     */
    sock.close();
}//end of socket "try" statement
catch (SocketException se) {
      System.out.println("Server Socket Problem" + se.getMessage());
}

/**
  * If the output of the AgentSystem gives a message which start with the word SUCCESS,
  * then the "Display GUI Agent" HTML page of the requested agent are displayed.
  *
  * result_of_Submission The output of the AgentSystem.
  */
if (result_of_Submission.startsWith("SUCCESS"))
  {
                        out.println("<HTML>");
                        out.println("<HEAD><TITLE>Agent System | Displaying Web GUI of an Agent</TITLE></HEAD>");
                        out.println("<BODY>");
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=0 CELLSPACING=2
                                WIDTH=450><TR>");
                        out.println("<TD ALIGN=center><H2><I>Displaying Web GUI of an Agent</I></H2></TD>");
                        out.println("<TD ALIGN=right ALIGN=center WIDTH=100>");
                        out.println("<A HREF=?curPage=getAboutPage>");
                        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                                Extended Version]\" BORDER=0>");
                        out.println("</A></TD></TR><TD>");
                        out.println("<TABLE BORDER=1 CELLPADDING=1 CELLSPACING=1 WIDTH=450>");
                        out.println("<TR><TD ALIGN=center WIDTH=175><B>Agent Name</B></TD>");
                        out.println("<TD ALIGN=center WIDTH=175><B>Agent ID</B></TD></TR>");
                        out.println("<TR><TD ALIGN=center WIDTH=175>" + value_for_AgentName + "</TD>");
                        out.println("<TD ALIGN=center WIDTH=175><FONT COLOR=red>" + value_for_AgentID +
                                "</FONT></TD></TR></TABLE></TD>");
                        out.println("<TD ALIGN=center WIDTH=100>");
                        out.println("<CENTER>");
                        out.println("<A HREF=?curPage=getSpecialParameters&value_for_AgentID=" +
                                value_for_AgentID+">Display Special Information</A>");
                        out.println("</CENTER>");
                        out.println("</TD></TR></TABLE>");
                        out.println("<HR>");
                        out.println("<BR>");
                        out.println(value_for_AgentWebInterface);
                        out.println("<BR>");
                        out.println("</FORM>");
                        out.println("</BODY>");
                        out.println("</HTML>");
                    }

else {

/**
  * Else, if such a GUI Agent does not exist, then display an "Error" HTML. In this case, the
  * "Error" HTML takes under consideration the case of an undefined error, where there is
  * such an agent created in the system, but due to internal error cannot be displayed.
  *
  * result_of_Submission The output of the AgentSystem.
  */
   if(result_of_Submission.startsWith("ERROR displaying")) {
                            out.println("<HTML>");
                            out.println("<HEAD><TITLE>Agent System | Displaying Web GUI of an Agent : UNDEFINED
                                    ERROR!</TITLE></HEAD>");
                            out.println("<BODY>");
```

```
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                                WIDTH=600><TR><TD>");
                        out.println("<TD ALIGN=right WIDTH=500><H2><I>Undefined Error displaying WebGUI of an
                                Agent!</I></H2>");
                        out.println("</TD><TD ALIGN=left WIDTH=100>");
                        out.println("<A HREF=?curPage=getAboutPage>");
                        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                                Extended Version]\" BORDER=0>");
                        out.println("</A></TD></TR></TABLE></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><H4>There was an undefined error displaying the web graphical user interface of
                                the <FONT COLOR=blue>" + value_for_AgentID +
                              " </FONT>Agent ID : " + result_of_Submission + "Please, go back and try submitting another
                              Agent ID again!</H4></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>");
                        out.println("<INPUT TYPE=button VALUE=Menu
                                onClick=parent.location='ServletAgent'></CENTER>");
                        out.println("</FORM>");
                        out.println("</BODY>");
                        out.println("</HTML>");
                    }

    /**
       * "Error" HTML taking under consideration the case where there was not such
       *  an agent created by the AgentSystem, therefore could not be displayed!
       */
     else {
                        out.println("<HTML>");
                        out.println("<HEAD><TITLE>Agent System | Displaying Web GUI of an Agent :
                                ERROR!</TITLE></HEAD>");
                        out.println("<BODY>");
                        out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                        out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                                WIDTH=600><TR><TD>");
                        out.println("<TD ALIGN=right WIDTH=500><H2><I>Error displaying Web GUI of an
                                Agent!</I></H2>");
                        out.println("</TD><TD ALIGN=left WIDTH=100>");
                        out.println("<A HREF=?curPage=getAboutPage>");
                        out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem
                                Extended Version]\" BORDER=0>");
                        out.println("</A></TD></TR></TABLE></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><H4>There was an error displaying the graphical user interface of the <FONT
                                COLOR=blue>" + value_for_AgentID +
                              " </FONT>Agent ID : " + "<FONT COLOR=red>" + result_of_Submission +
                              "</FONT>. Please, go back and try submitting another Agent ID again!</H4></CENTER>");
                        out.println("<BR><BR>");
                        out.println("<CENTER><INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>");
                        out.println("<INPUT TYPE=button VALUE=Menu
                                onClick=parent.location='ServletAgent'></CENTER>");
                        out.println("</FORM>");
                        out.println("</BODY>");
                        out.println("</HTML>");
                    }
                }

}//end of else-if "getWebInterface" statement

                /**
                 * Represents the "About" page of the system, when the image logo is clicked.
                 *
                 * getAboutPage The value of displaying an "About" HTML page for the system.
                 * curPage The name of the initial HTML (introductory) page, defined as
                 *         a hidden field.
                 */
                else if ("getAboutPage".equals(req.getParameter("curPage"))) {

                    out.println("<HTML>");
                    out.println("<HEAD><TITLE>About | Agent System Extended Version</TITLE></HEAD>");
                    out.println("<BODY>");
                    out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                    out.println("<CENTER><TABLE BORDER=0 CELLPADDING=0 CELLSPACING=2
                            WIDTH=600><TR>");
```

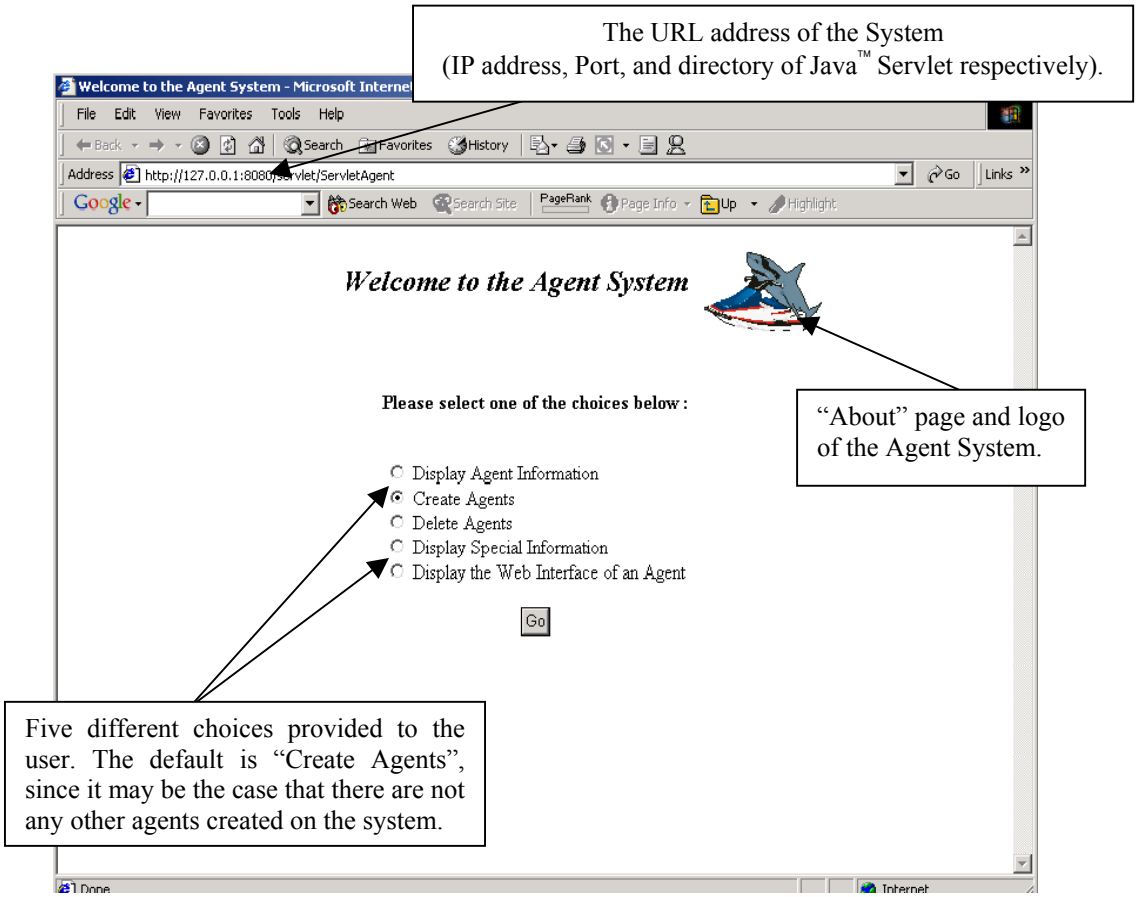
```
                                out.println("<TD ALIGN=center WIDTH=500><H2><I>About Agent System Extended
                                    Version</I></H2></TD>");
                                out.println("<TD ALIGN=right ALIGN=center WIDTH=100>");
                                out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[AgentSystem POND :
                                    Extended Version]\" BORDER=0>");
                                out.println("</TD></TR></TABLE>");
                                out.println("<BR>");
                                out.println("<H3><FONT COLOR=green>Agent System POND | Extended Version</FONT></H3>");
                                out.println("<TABLE BORDER=0 CELLPADDING=1 CELLSPACING=1 WIDTH=800>");
                                out.println("<TR VALIGN=top><TD VALIGN=top ALIGN=right WIDTH=200>");
                                out.println("Version : </TD><TD VALIGN=top ALIGN=left WIDTH=600>");
                                out.println("<B>1.0</B></TD></TR><TR VALIGN=top><TD VALIGN=top ALIGN=right
                                    WIDTH=200>");
                                out.println("Implementation period : </TD><TD VALIGN=top ALIGN=left WIDTH=600>");
                                out.println("<B>March 2001 - August 2001</B></TD></TR><TR VALIGN=top><TD VALIGN=top
                                    ALIGN=right WIDTH=200>");
                                out.println("Connection Port : </TD><TD VALIGN=top ALIGN=left WIDTH=600>");
                                out.println("<B>" + Servlet_Port + "</B></TD></TR><TR VALIGN=top><TD VALIGN=top ALIGN=right
                                    WIDTH=200>");
                                out.println("URL of System : </TD><TD VALIGN=top ALIGN=left WIDTH=600>");
                                out.println("<B>" + Servlet_Address + "</B></TD></TR><TR VALIGN=top><TD VALIGN=top
                                    ALIGN=right WIDTH=200>");
                                out.println("Copyright : </TD><TD VALIGN=top ALIGN=left WIDTH=600>");
                                out.println("<B>(c) 2001, Marios Kalnis for <I><A HREF=http://www.fim.uni-
                                    linz.ac.at>FIM</A></I></B></TD></TR></TABLE");
                                out.println("<BR><BR>");
                                out.println("<CENTER><TABLE><TR><TD>");
                                out.println("<INPUT TYPE=BUTTON VALUE=Back onClick=history.go(-1)>");
                                out.println("</TD><TD>");
                                out.println("<CENTER><INPUT TYPE=button VALUE=Menu
                                    onClick=parent.location='ServletAgent'></CENTER>");
                                out.println("</TD></TR></TABLE>");
                                out.println("</CENTER>");
                                out.println("</FORM>");
                                out.println("</BODY>");
                                out.println("</HTML>");
                }

                /**
                 * The initial HTML page. This HTML appears when it is the first time that the
                 * page is loaded, and each time there isnnot any other choice/request to be done.
                 */
                else{
                                out.println("<HTML>");
                                out.println("<FORM METHOD=\"GET\"" + " ACTION=\"\">");
                                out.println("<HEAD><TITLE>Welcome to the Agent System</TITLE></HEAD>");
                                out.println("<BODY>");
                                out.println("<CENTER><TABLE BORDER=0 CELLPADDING=3 CELLSPACING=3
                                    WIDTH=500><TR><TD>");
                                out.println("<TD ALIGN=right WIDTH=400><B><H2><I>Welcome to the Agent System</I></H2></B>");
                                out.println("</TD><TD ALIGN=left WIDTH=100>");
                                out.println("<A HREF=?curPage=getAboutPage>");
                                out.println("<IMG SRC=c:/Agents/AgentSystem/images/agent_small.gif ALT=\"[About : AgentSystem Extended
                                    Version]\" BORDER=0>");
                                out.println("</A></TD></TR></TABLE></CENTER>");
                                out.println("<BR>");
                                out.println("<BR>");
                                out.println("<CENTER><H4>Please select one of the choices below :</H4></CENTER><BR>");
                                out.println("<CENTER>");
                                out.println("<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>");
                                out.println("<TR><TD>");
                                out.println("<INPUT TYPE=radio NAME=Agents" + " VALUE=\"Information\">");
                                out.println(" Display Agent Information");
                                out.println("</TD></TR><TR><TD>");
                                out.println("<INPUT TYPE=radio + checked + NAME=Agents " + " VALUE=\"Create\">");
                                out.println(" Create Agents");
                                out.println("</TD></TR><TR><TD>");
                                out.println("<INPUT TYPE=radio NAME=Agents" + " VALUE=\"Delete\">");
                                out.println(" Delete Agents");
                                out.println("</TD></TR><TR><TD>");
                                out.println("<INPUT TYPE=radio NAME=Agents" + " VALUE=\"Special\">");
                                out.println(" Display Special Information");
                                out.println("</TD></TR><TR><TD>");
                                out.println("<INPUT TYPE=radio NAME=Agents" + " VALUE=\"WebInterface\">");
                                out.println(" Display the Web Interface of an Agent");out.println("</TD></TR>");
```

```
                                out.println("</TABLE>");
                                out.println("</CENTER>");
                                out.println("<BR>");
                                out.println("<CENTER>");
                                out.println("<INPUT TYPE=submit VALUE=Go>");
                                out.println("</CENTER>");
                                out.println("</BODY>");
                                out.println("<INPUT TYPE=hidden NAME=curPage VALUE=homepage>");
                                out.println("</FORM>");
                                out.println("</HTML>");

                    }
                    out.close();
                }//end of "try" statement
                catch(Exception e) {e.printStackTrace();
                }
            }//end of "ACTION" else-stament

    }//end of "doGet" method


    /**
     * Called by the web server when a HTTP POST request is made
     * @param req The POST request information
     * @param res The HTTP response object
     */
    public void doPost(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException {
    doGet(req, res);
    }

}
```

Appendix B – User Manual

This section will deal with how a novice user can make full use of the extension of the Mobile Agent System into the World Wide Web. May we also notice, that since a user manual is usually referred to people with no knowledge of programming or a very low level, we will assume that the server-side of the Agent System is installed on a server, as well as the Servlet itself, therefore giving information to the user, when he/she visits the specific URL, where the system is installed.

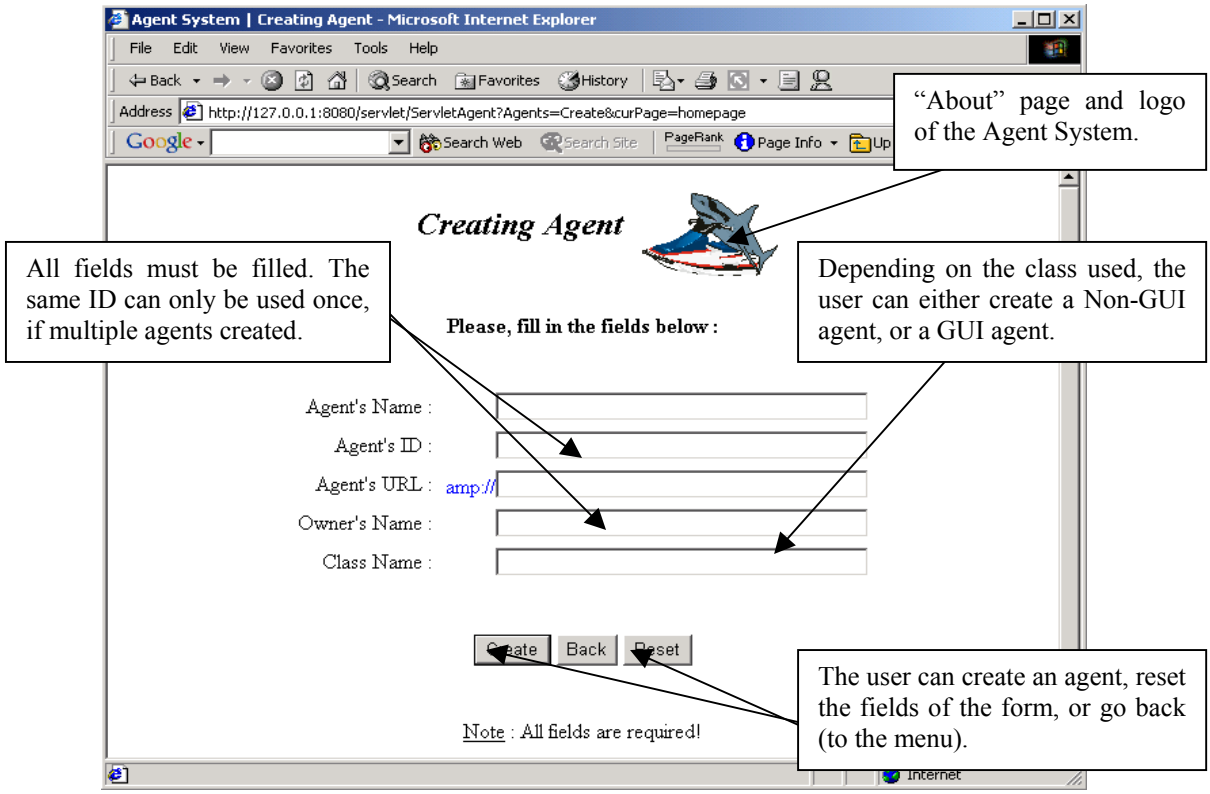
To start with, the user knows by default the address that the Servlet is installed on the server. This may be in terms of a normal URL address (such as *www.name.location*) with an extra extension of a Servlet directory. In our case, let us assume that the address is *www.name.location/servlet/ServletAgent*.



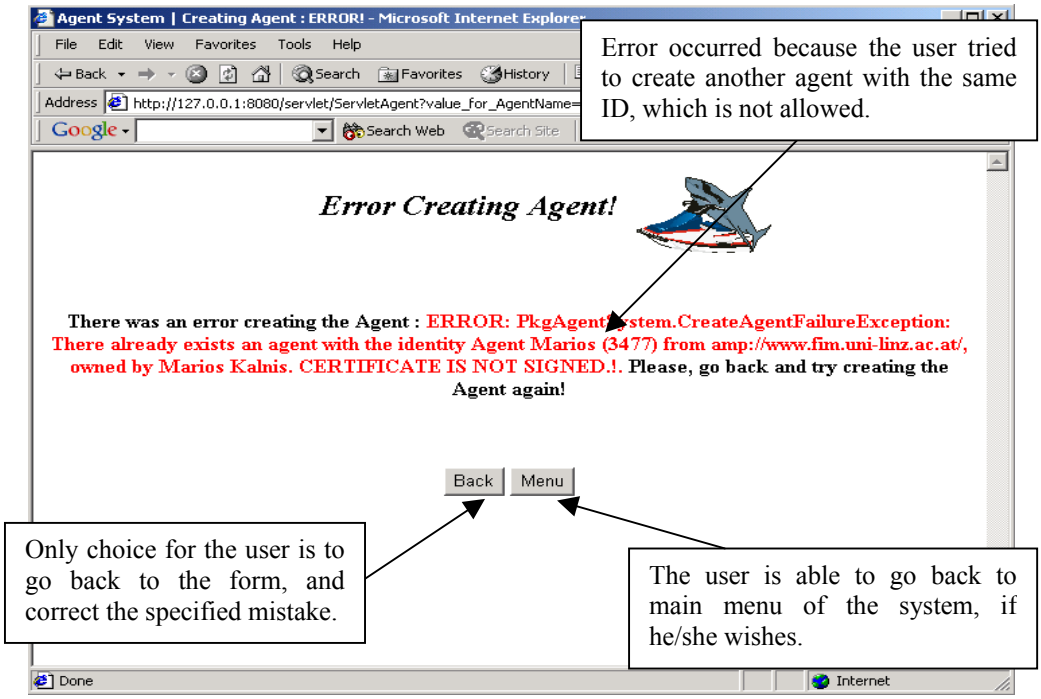
screenshot 1 Initial page of the Agent System.

The user logs on to the system and appears initially to him/her the main menu page of the system (*screenshot 1*). Since there may be the case that there is not any agent on the system created, then the system gives by default the choice on the radio buttons appearing to the user, on creating an agent. In this case, the user has four different choices, such as: *display agent information*, *create agents*, *delete agents*, *display special information* (of an agent), and *display a web interface of an agent* (we will come to this later on). Also, the user can view the "*About*" page of the system, by clicking on the Intelligent Mobile Agent System logo, on the top right position of the browser. Finally, the user in this stage has only one button to press (the *Go* button), since this is the initial page, and no other choices can be made.

If the user initially selects to *create an agent* by clicking on the relevant radio button and on the *Go* button afterwards, a new window appears (*screenshot 2*). The user is required then to fill in the form that appeared on this window, without leaving any blank fields. In case that the user does not provide a correct class to the system, an error is displayed, explaining the reason. Moreover, if the user tries to create more than one agent with the same ID, then the system does not allow to do such an action, since the ID is unique within an Agent System and it is not possible for a second one to send with the same value (*screenshot 3*). In all other cases, if one of the fields is not filled, then the system prompts him/her to fill in the blank field, by showing an error message. Also, the system informs the user (when the submit button is pressed) if the submission was successful or not. If yes, then it provides the user with a thank you page, providing as well the button to return to main menu. If not, it provides information to the user, in order to understand and correct his/her error, by going back. Finally, the user can again view the "*About*" page of the agent system, as well as, it is possible in this case to either go back or go to the main menu, with the relevant buttons that they are provided at the end of the form, as well as a submit button, in case the user wishes to create a new agent.

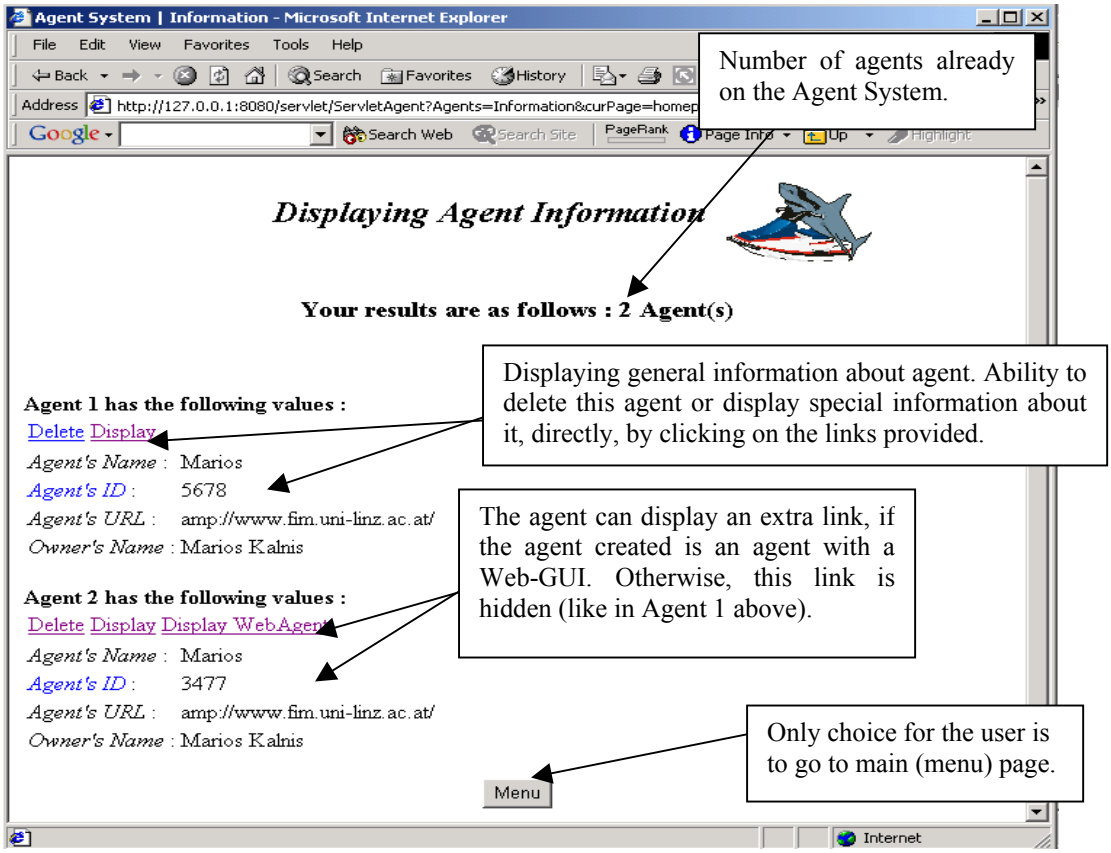


screenshot 2 *"*Creating Agent" page of the Agent System.



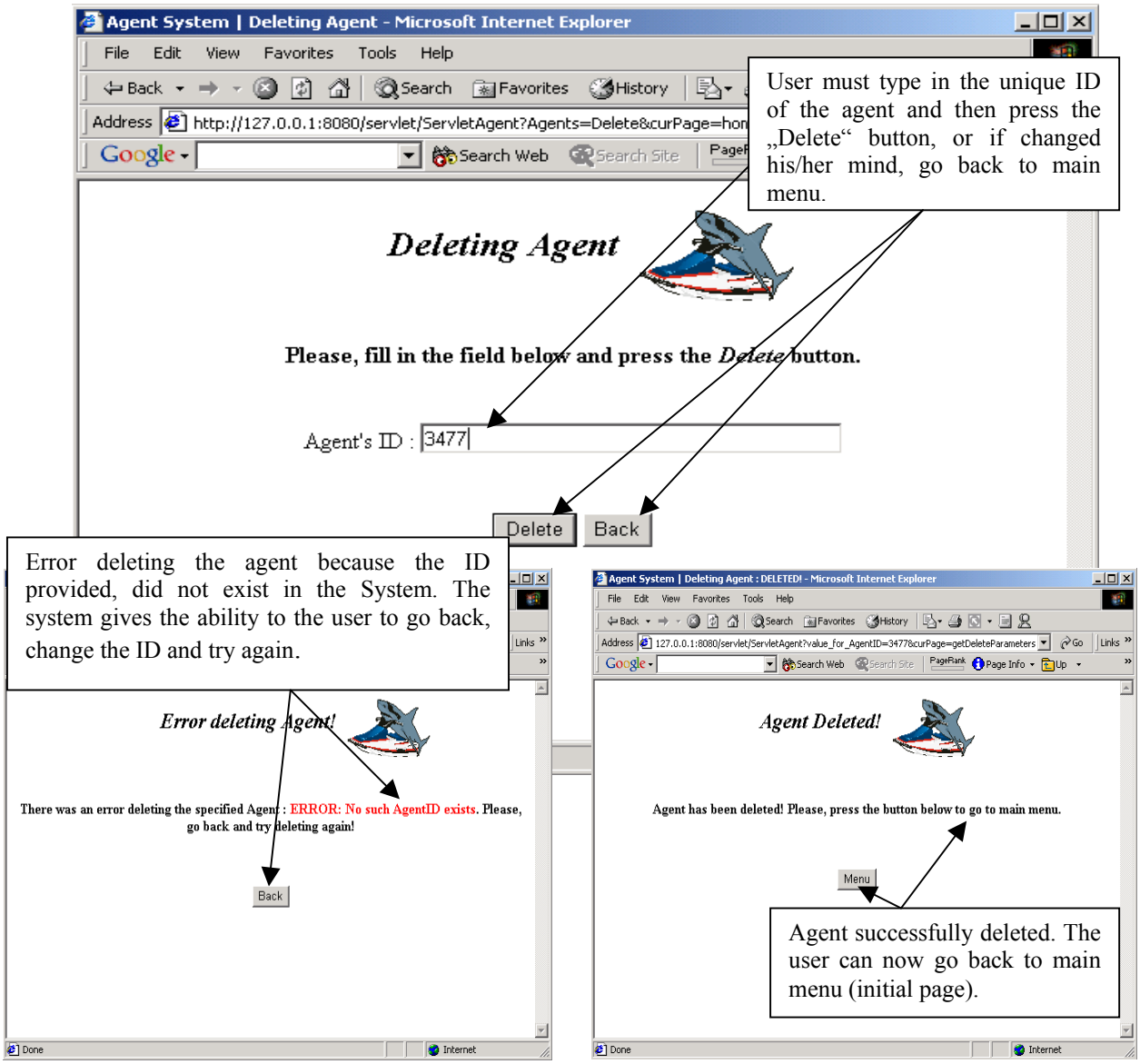
screenshot 3 Error in creating an agent.

If the user has already created an agent, or an agent already existed on the system, and the user chooses the first radio button from the main menu (the *display information about agents*), then a new window opens (*screenshot 4*). In this case, the window shows on the top the number of agents that exist on the system. Then, it provides general information for each agent individually, as well as direct links for *special information of the agent* and *display web-GUI of an agent* window. May we also notice, that these two links can be viewed by the user from the main menu, but in this case, this page provides this information directly (by using only the link) to the user. Also, depending on the agent that was initially created, the link to display the web-GUI agent may appear (in the case that the agent created has this ability or not). Finally, again the user has the opportunity to go back or to the main menu with the buttons provided at the bottom of the page, as well as to view the "*About*" page of the system, by clicking on the agent logo.



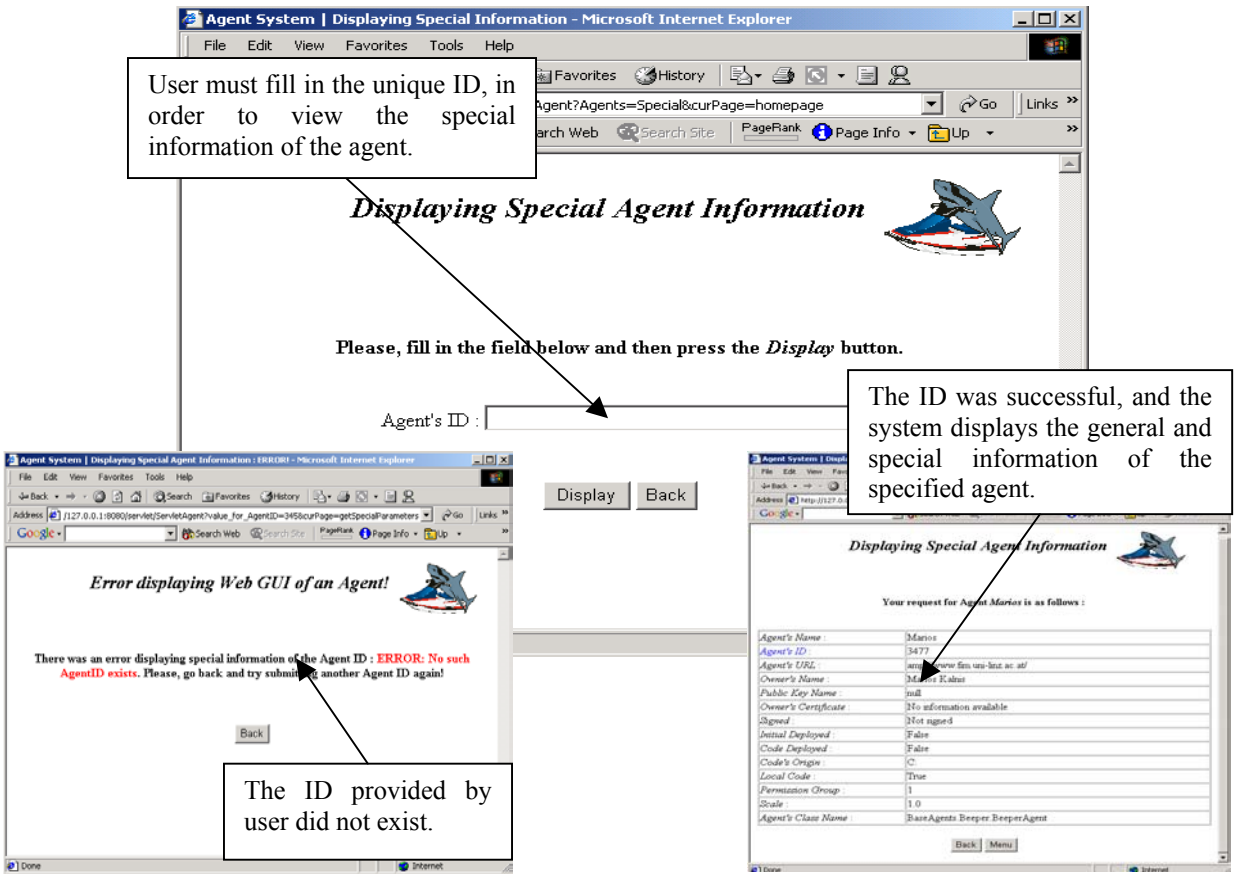
screenshot 4 Displaying general information of agents.

Then, if the user wishes to *delete an agent*, he/she can either choose that to do this, either from the main menu, or from the *display agent information* page (as mentioned before). In the latter case, the user has the advantage that he/she does not need to remember the unique ID (which maybe too long). If an invalid ID is given, then the system displays the error to the user and prompts him/her, to go back and correct it (*screenshot 5*), otherwise, the system verifies and thanks the user for deleting the agent and gives a button link to return to main menu (initial page of the system).



screenshot 5 Deleting an agent, together with successful and unsuccessful cases.

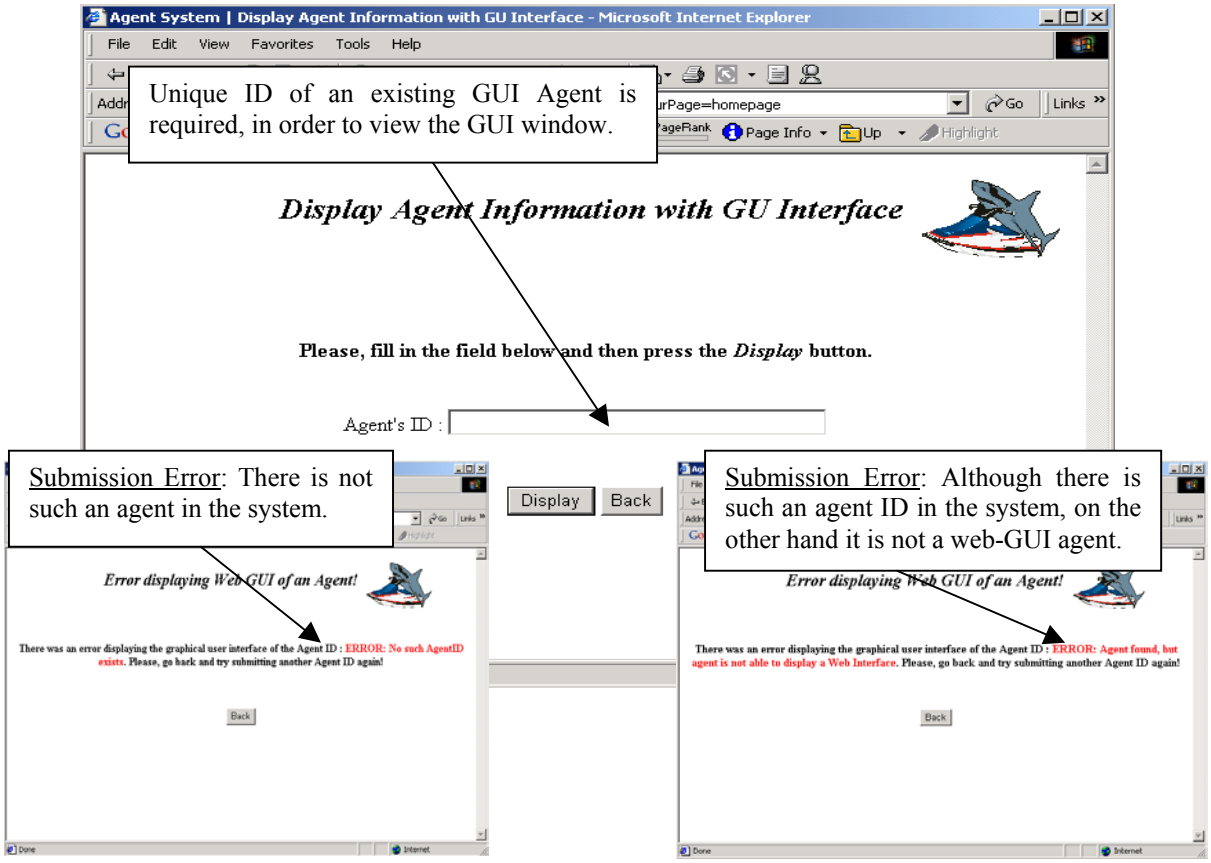
After that, if the user chooses from the main menu the *display special information of an agent*, then a new window opens (*screenshot 6*). In this case, the user has to enter the unique ID of the agent that he/she had initially created. If the ID is not correct, then the system provides an error page; otherwise, it opens a new window, which provides information about the specific agent that it was requested. The information provided are analytically, and they include the general information, as the ones mentioned at the *display agents* page, as well as more special information, like the security level and from where the agent was loaded, and so on. The user can only view the information of the agent requested. The only thing that the user can view again is the "*About*" page, as described above.



screenshot 6 Displaying special information of an agent, together with successful and unsuccessful cases.

Furthermore, if the user selects from the main menu to view the *web-GUI agent information* page, then again a new window opens, where it prompts the user to enter the unique ID of the

web-GUI agent that initially created. If the user provides a non-existing ID or an invalid ID (invalid can be also the case where a non-GUI agent ID is used), then the system will provide to the user an error page, distinguishing the type of the error, whether it is for a non-existing or an existing non-web-GUI agent (*screenshot 7*).

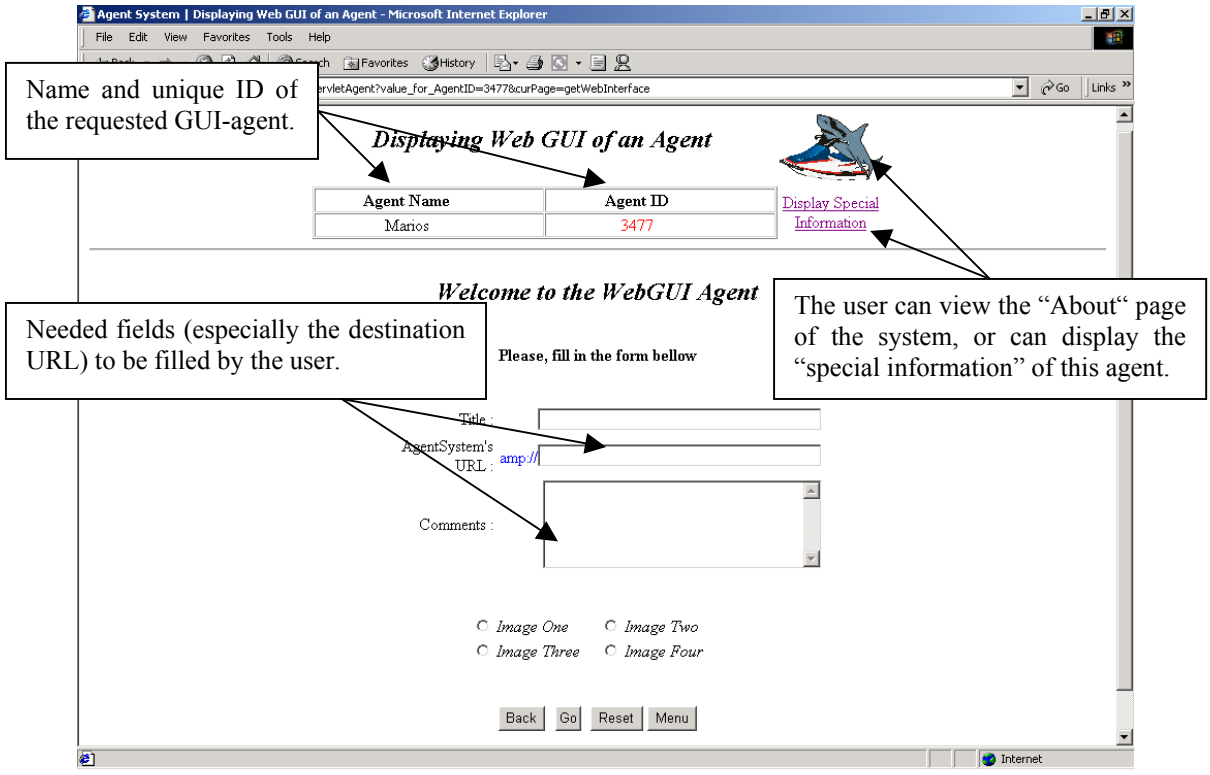


screenshot 7 Error displaying web-GUI window of an agent.

On the other hand, if the submission is successful, then the system provides to the user a new window (*screenshot 8*). In this case, the general information of the agent is provided (such as its name and its unique ID) at the top of the page, together with the link of the logo to the "*About*" page.

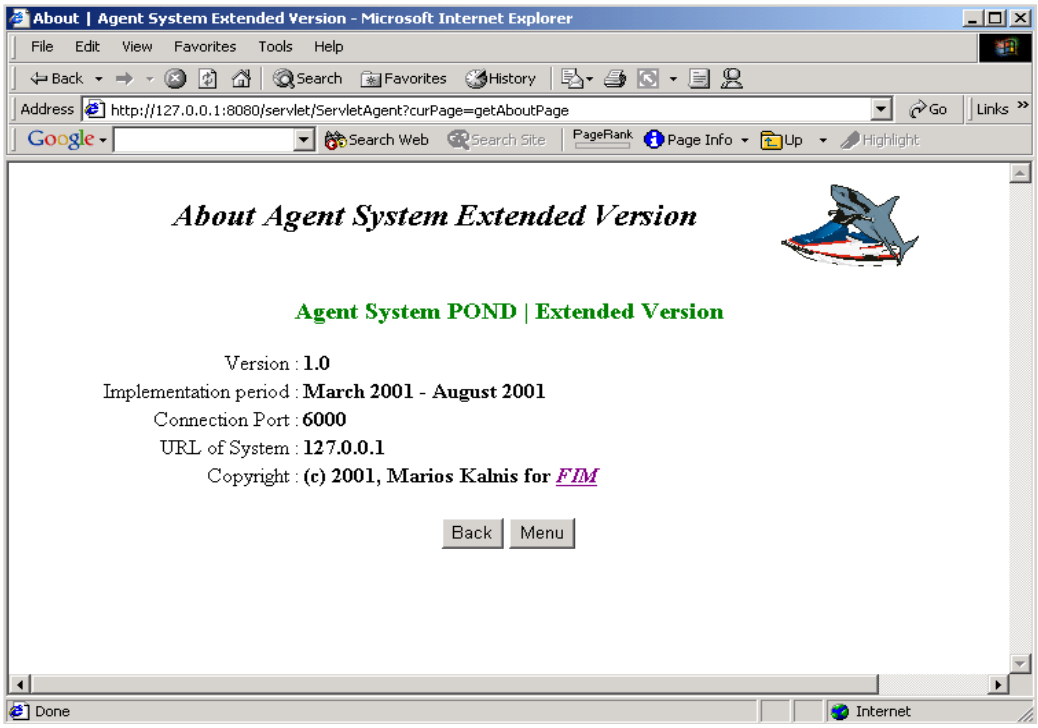
With this Example Agent form, the user can create a mobile agent, where by specifying the destination URL, can then send the agent to the remote computer (by providing its unique URL). In order to achieve this, the user has to fill in all the fields provided. Then, by pressing the

submit button, all the information is sent, the mobile agent is destroyed on the user's system, and it is recreated at the specified remote system. In order the system to show that the mobile agent was successfully sent to the remote computer, a new window opens, which shows the user the text with the selected image. In case that the submission is not successful, or an error occurs, the system provides to the user an error page, together with the message (where the error occurred). Finally, the user can view again the "About" page of the system, or can select to view the web-GUI page from the *display information of agents* page, by clicking on the direct link, provided only in the case of an existing GUI agent.



screenshot 8 The web-GUI agent window, together with the form, to send a mobile agent to a remote computer (Example Agent).

Finally, if the user clicks on the logo and therefore the "About" page of the system, then he/she can view information related with the Agent System, the implementation period, as well as other important information, related with this project (*screenshot 9*).



screenshot 9 The „About“ page of the Agent System (Extended Version).

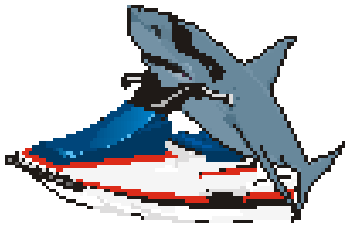
Supervisor
Prof. Dr. Jörg R. Mühlbacher
Assistant supervisors
Dipl.-Ing. Susanne Reisinger
Dipl.-Ing. Michael Sonntag

Student
B.Eng. Marios Kalnis

FIM – Intelligent Mobile Agents (Extended Version) / Time Plan (Gantt diagram)

MONTHS	MARCH				APRIL				MAY				JUNE					JULY				AUGUST				
WEEKS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
TASKS																										
Learning Java™	**																									
Learning Java™ Servlets																										
Communication with webserver using Servlets.																										
Mid-term report.																	V									
Create immobile agent (ability to create agents using a Web Interface).																										
Create immobile agent, work through Internet. Handling of agent through URL.																										
Create a mobile agent to move to a remote system.																										
Final Testing.																										
Final Report.																										

** ***Week 1*** starting from 5/3/2001



Intelligent Mobile Agents (Extended Version)

Implementation period : March 2001 – August 2001

Location : Institute for Information Processing and Microprocessor Technology (FIM)
Johannes Kepler University Linz
Altenbergerstr. 69, A-4040 Linz, Austria

Developed by : Marios Kalnis © (marios@kalnis.com)

Official URL (containing information) : http://www.kalnis.com/marios/AgentProject