ISSN 22773061

MOBILE AGENT APPLICATION DEVELOPMENT IN A SIMPLE JAVA-BASED MOBILE AGENT SYSTEM (SIMMAS)

O. Osunade
Dept of Computer Science University of Ibadan
Ibadan, Nigeria
seyiosunade@gmail.com
R. D. Oloritun
Masdar Institute of Science and Technology
Abu Dhabi,
United Arab Emirates
rahmanoloritun@gmail.com

ABSTRACT

As network information resources grew in size, it was most efficient to process queries and updates at the site where the data was located. The processing accomplished by a traditional client-server network interface constrained the client to the set of queries supported by the server, or required the server to send all data to the client for processing. The former was inflexible and the latter was inefficient. Mobile agents support the movement of the client computation to the location of the remote resource. It has the potential to be flexible and efficient. Mobile agents are capable of suspending their execution, transporting themselves to another host on a network, and resuming execution from the point at which they were suspended. Mobile agents consume fewer network resources and support systems that do not have permanent network connections, such as mobile computers and personal digital assistants. This work described a prototype java based mobile agent environment for the development and deployment of mobile agent applications. The prototype called SIMMAS allowed programmer-developed applications to inherit mobility.

General Terms

Java Mobile Agents Development Environment

Keywords

Network communications, distributed systems, mobile environment, mobile agents, Java

Council for Innovative Research
Peer Review Research Publishing System
Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY
Vol 7, No 2

editor@cirworld.com
www.cirworld.com, member.cirworld.com

coordinating application for synthesis. In other cases, a system of loosely knit applications work together to provide a single interface for an application or a user e.g. an end-user working from a web browser interface could perform data entry via a servlet that communicates with an application server, which in turn manages multiple databases residing on multiple database servers from several database vendors.

In the past, distributed application components were often implemented in traditional languages such as C or C++ as executable files or shared object-code libraries. Implementing application components as machine-dependent binary modules is problematic because it forces developers and system administrators to deal with a whole range of issues: portability of application components; interoperability of components developed with different programming languages; data communication across multiple machine architectures; and rogue modifications to binary modules that compromise security. Java supports programmer-defined content and protocol handlers. This framework for alternate communication protocols facilitates new communication models that were not envisioned when Java was created, as well as application specific communication protocols, both of which essentially plug into the existing URL-oriented framework. This flexibility facilitates network communication that is not bound by highly structured communication protocols such as remote method invocation (RMI), which is more suitable for traditional client-server designs.

With application components inter-operating from different network hosts, efficient task distribution and communication become very important. In lieu of distributed components communicating over the network, it is justified to transport an application component to a remote location, have it perform its work, and then either return or dispose of itself. These components could operate on behalf of a coordinating component at another site.

The mobile agent paradigm supports a powerful programming style. In many cases, the challenge for programmers is to resist viewing the "world" in terms of the traditional client-server paradigm. Instead, programmers should look for opportunities to empower Java classes with the capability of relocating themselves on various hosts and performing the requisite work.

Mobile Agents are programs, which move from machine to machine, such that they execute on a machine which is closest to the required resource [1,2]. They form an attractive paradigm for implementing objects that can migrate from one computer to another over the Internet. A mobile agent consists of program code, data and execution state that can be serialized into a message and sent across a network. The use of platform independent languages like Java to program mobile agents ensures that they can execute on remote computers. Features offered by mobile agents such as mobility, autonomy and state preservation, make them attractive for applications in distributed systems, network management and e-commerce applications.

There are several Application Programming Interfaces (APIs) that support component mobility and provide the appropriate frameworks and protocols for developing what are commonly called mobile agents. Mobile agent technologies support common solutions to implementing cooperating and distributed application components. Just as a graphical toolkit is useful for developing multiple graphical applications, a mobile agent framework facilitates the development of multiple distributed applications, as well as the ongoing enhancement of existing distributed applications.

The inherent unreliable and unpredictable nature of the distributed environments requires a setup that provides a layer of abstraction over the actual network and serves as a framework over which mobile agents can be developed, deployed and inter-operated without requiring catering for many of the inherent problems involved in distributed application development.

Many agent toolkits do not implement true mobility; instead the movement of agents is done by cloning the software agents. The software agent is duplicated by cloning and the clone is transported to the destination host where it is executed and results are sent back to the origin or the agent is halted. The cloning of the agents wastes system resource like CPU cycles and memory. There are excessive computing activities at the origin because CPU cycles are needed to clone the agent and memory to temporarily store the clones before dispatch to the various target hosts. The agents are cloned as many times as there are target hosts. Hence, if the target hosts are very many then the system performance will be compromised.

Mobile agent application development in Java involves an in-depth knowledge of socket programming which is not very easy to program and can lead to mobile agent application with logical errors, since the programmer concentrates on both mobility and functionality but with the proposed application development environment called SIMMAS, the programmer need not program the mobility. The programmer can concentrate on the function the agent performs and leave mobility to SIMMAS. This work designed and implemented SIMMAS which provides the infrastructure for developing mobile agent applications and deploying them.

SIMMAS includes the Java class files for the Agent API (Application Programming Interface), extensive documentation, source code, and Daemon, an agent server/viewer. With a Daemon running on two different hosts, it's quite easy to design agents that travel between the local and remote hosts, as well as communicate with other agents across hosts. The API also includes classes for implementing the Daemon functionality within Java applications.

2. LITERATURE REVIEW

Maes [4] defines an electronic commerce agent as a piece of software that profiles users or buyers to provide personalized service. In the agents' developer community, experts define agents as: Entities that can communicate in an agent communication language [14]; Computer systems in a complex environment that realize a set of tasks and goals they were designed for: systems capable of autonomous purposeful action in the real world [9]. Standards bodies define agents as: computer programs that act autonomously on behalf of a person or organization [10] and computational processes implementing an application's autonomous communicating functionality.

All agents are not created equal. Some are more advanced than others. Basic software agents exhibit the common characteristics of autonomy (independence), persistence (long-livedness), monitoring of the environment; communication and collaboration with other agents and/or the user. More "intelligent" agents possess higher-level abilities, such as mobility, decision-making, and the ability to learn and may be characterized along three dimensions: agency, intelligence, and mobility [12].

Agency

Agency reflects the degree of independency of an agent. The agent should be able to collect and analyze information as well as perform tasks independently of the user using its knowledge about the user's profile.

Intelligence

Intelligence is the degree of learning by an agent. The agent should be able to perceive, understand and analyze the environment. Given that the environment constantly changes, the agent should be capable of learning and adapting to these changes. Further, the agent should draw conclusions from the information collected and perform actions on the user's behalf.

Mobility

Mobility is the last dimension to consider. Although agents may be static (e.g., an e-mail assistant that sorts incoming messages), more intelligent agents require some degree of mobility [5]. The agent should be able to travel from one machine to another, gathering information that it will process later. Furthermore, it may perform different tasks on a remote machine on behalf of the user. For example, a travel agent may visit a number of sites, gather information on prices and then process it on the user's machine or negotiate the details of the deal with other agents in cyberspace and even sign a binding agreement.

The most promising type of agents are mobile agents, which can themselves be intelligent or unintelligent. Unlike their static counterparts, which are content to execute in the cozy confines of a single machine or address space, mobile agents have wheels. They migrate about the network, executing tasks at each host, potentially interacting with other agents that cross their paths. An example of a mail delivery system can be used to contrast a mobile and static agent. A static mail agent is that in which a POP client communicates with an SMTP server and collects mail. Both players in the transaction stay on their respective machines, using the network only for the transfer of message content. A mobile agent design of the same transaction might define a mail carrier agent that travels about the network autonomously, handing messages to mail handler agents at each stop [6].

From 1977, research in agent technology concentrated mainly on deliberative-type agents with symbolic internal models, and most of the work focused on issues such as the interaction and communication between agents, the decomposition and distribution of tasks, coordination and cooperation, conflict resolution via negotiation, and so on. The main goal was to specify, analyze, design and integrate systems comprising of multiple collaborative agents, such as MACE [9], MICE [11], and MAS/DAI planning and game theories [5].

In the early 1990s, another distinct trend to the research and development work on intelligent agents emerged, with focus on the investigation of the diversification in the types of agents. A number of agent types were identified such as User Interface Agents, Information Agents, Multi Agent System, Mobile Agents, and so on. Mobile agents are perhaps the most exciting ones, which originally emerged from advances made in distributed systems research. In these years, many prominent research groups have been seduced by the research on Mobile Agents, which has resulted in a number of projects such as Grasshopper [8], AgentTCL [14], Aglet system [13], Bee-gent and Plangent [15].

In reality, software agents exist in a truly multi-dimensional space, even though some research papers try to collapse these multiple-dimensions to a single list. Agents can be classified in at least four dimensions: Quantitative, Mobility, Responsiveness, Behavioural as shown in Figure 1

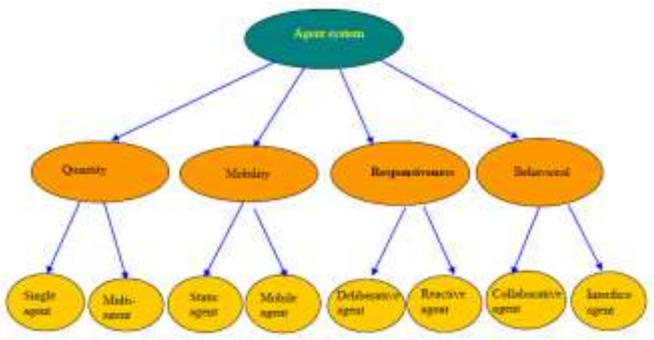


Figure 1: Multi-dimensional classification of software agent system

(i) Quantitative

Various agent technologies existing today can be classified as being either single-agent or multi-agent systems. In single-agent systems, an agent performs a task on behalf of the user or some process. While performing its task, the agent may communicate with the user as well as with local or remote system resources, but it will never communicate with other agents. In contrast, the agents in a multi-agent system (MAS) not only communicate with the user and system resources, but they also extensively communicate and work with each other, solving problems that are beyond the individual agent capabilities.

(ii) Mobility

Agents can also be classified by their mobility, i.e. by their ability to move around the network. This yields the classes of static or mobile agents. Static agents are constrained to a single computer; early agents were all static. The newcomers – mobile agents can move from one computer to another. They are bringing together telecommunications, software, and distributed system technologies to create new ways of building computing systems.

(iii) Responsiveness

The response of an agent to any request may be classified as either deliberative or reactive. Deliberative agents are from the deliberative thinking paradigm: the agents possess an internal symbolic, reasoning model and they engage in planning and negotiation in order to achieve coordination with other agents. Reactive agents do not have any internal, symbolic models of their environment, and they act using a stimulus/response type of behaviour by responding to the present state of the environment in which they are embedded.

(iv) Behavioural

Agents are further classified along three primary characteristics: autonomy, learning and cooperation.

Autonomy: autonomy refers to the principle that agents can operate on their own without the need for human guidance. A key element of their autonomy is their pro-activeness.

Learning: learning refers to an agent's ability to learn as they react, and/or interact with their external environment, and then modify its own behaviour.

Cooperation: cooperation among multiple agents denotes the ability to interact with other agents and possibly humans via some communication language, and coordinate to enhance their ability.

With regard to these three characteristics, an agent may have one or two of them. An agent with all three would be ideal.

Major problems of agent building

According to Maes [4], there are two main problems that confront the building of software agents. The first is one of competence: how does an agent acquire the knowledge it needs to decide when and how to help the user? The quality and competence of an agent will depend on the ability of its software developer to implement all desired features. The second is one of trust: how can we guarantee a user feels comfortable delegating tasks to an agent? This second problem is becoming less important nowadays when students and researchers develop their own agents (e.g., personal digital assistants or shopping agents) using agent toolkits. It is likely that user trust will be greater in those instances where people personally create agents, supplying them with knowledge and logic rules, compared to those instances where users utilize pre-canned, "off-the-shelf" agents. In addition, the level of trust a user has for an agent will depend on the user's experience with utilizing such toolkits [4].

3. PROPOSED SYSTEM

The analysis of mobile agent systems identified two (2) major components required [12]: Agents and Place (the agent execution environment).

The agents and the place form the basics of the system architecture. The agent is made up of library classes that can be extended to give mobility to applications while the place (the agent execution environment) will handle agent thread management and keep track of executing agents. The place also migrates the agent to its destinations, see Figure 2, therefore it must have a communication object.

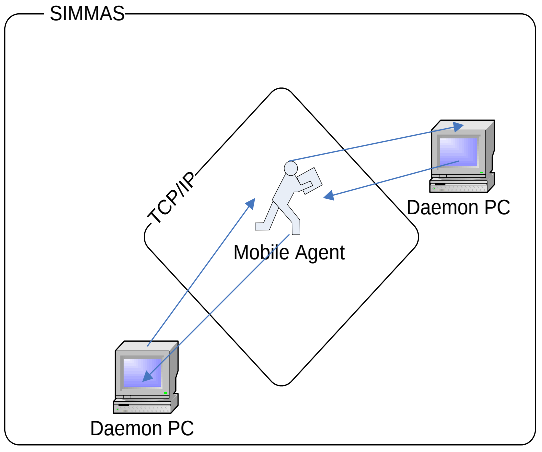


Figure 2: Mobile Agent Migration

The system has three basic layers based on the concept of agent and place: Agent API, Agent Execution Environment, and the Communication Layer as in Figure 3. Agent API will implement re-useable classes that can be extended by programmers to develop Mobile Agent Applications. Agent Execution Environment will implement an embeddable server called the Daemon in this work. The communication layer will be network classes that will implement socket programming to achieve connectivity between Daemons.

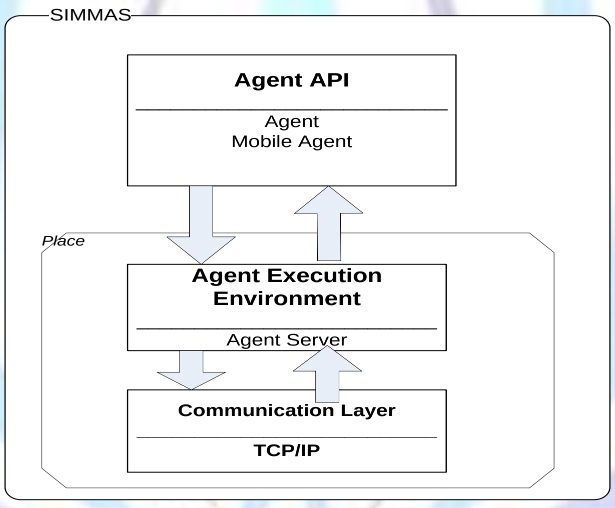


Figure 3: The basic architecture of the simple java based mobile agent system (SIMMAS)

3.1 (i) The Agent

The agent itself is run as a thread in the Java Virtual Machine (JVM) and can migrate to other machines. The heart of an Agent is the "executeAgent()" method. The Daemon will wrap the Agent in a Thread that will run the "executeAgent()" method. The agent can exist only within the daemon. The Agent can assume one state;

AGENT_START: This is the initial run state for agents. It has the value zero (0).

AGENT_CONTINUE: this is the default state after a migration. The value of this state is one (1).

AGENT_RETURN_TO_SENDER: The run state if an agent has been rejected by another Daemon. The value is two (2).

When the Agent Class is extended by programmers it should be noted that agents produced are static agents. The agent class must have methods for;

Displaying a message

Providing functionality for the Agent. Use executeAgent() for this to encourage readability

- Returning a description of the Agent
- Returning the unique ID of the Agent that was assigned at its spawn time
- Returning the number of the agent as it was created on its source host
- Returning the Daemon that started the Agent or received the agent from another host
- Returning the run state for the Agent
- Returning the IP of the host that the agent was spawned on
- Returning the name of the host that the agent was spawned on
- Returning the time the agent was spawned on the spawning host
- Making the Agent to halt execution

3.2 (ii) The Daemon

The Daemon stands for the place and is responsible for managing Agents and providing services to the running Agents. It is also responsible for transporting MobileAgents to Daemons running on other hosts. The daemon has a base class called the daemon-server which provides the daemon with the TCP/IP communication layer needed via java sockets. The daemon must have the ability to be started using command line with its parameters typed on the command line or kept in a configuration file in XML format. The daemon should allow programmers the ability to embed it in their various mobile agent applications.

3.1 (iii) The Mobile Agent

This is an extension to the Agent class that is transportable to other machines The Mobile Agent can call "migrateTo()" and specify a host and the Daemon will attempt to move the Mobile Agent to a Daemon running elsewhere.

Transmission is done using sockets and utilizes Java's serialization capabilities. When a Mobile Agent migrates to another Daemon, it will in essence be restarted. This means that the Mobile Agent itself must keep track of state. Only Mobile Agents are transported by the Daemon.

The Mobile Agent can assume three states;

AGENT_START: This is the initial run state for agents. It has the value zero (0).

AGENT_CONTINUE: this is the default state after a migration. The value of this state is one (1).

AGENT_RETURN_TO_SENDER: The run state if an agent has been rejected by another Daemon. The value is two (2).

These states are inherited from the Agent class but the agent class has no means of assuming all the states except the first state.

4. METHODOLOGY

SIMMAS is designed in three joint facets, the framework (an abstract implementation for the agents and agent mobility application programmer interface (API)); the implementation classes for the agent execution environment and transport layer, the utility classes that provided compression facilities and database connectivity, and the application agents written as sample mobile applications with a graphical user interface for testing the system.

The system has two main packages namely, **ng.edu.ui.csc.simma.agent** which contains abstract implementations and agent execution environment and **ng.edu.ui.csc.simma.util** which contains the implementations for data compression and database connectivity.

The mobility framework is designed using various abstract implementations i.e. “bare bones” implementations of Agent interfaces from which the programmer can quickly “flesh out” complete customized implementations.

5. RESULTS AND DISCUSSION

ISSN 22773061

A. *Starting the daemon*

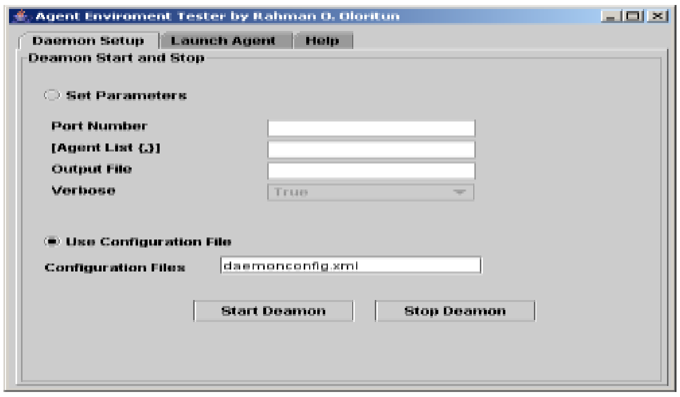


Figure 4: The Daemon Setup Screen

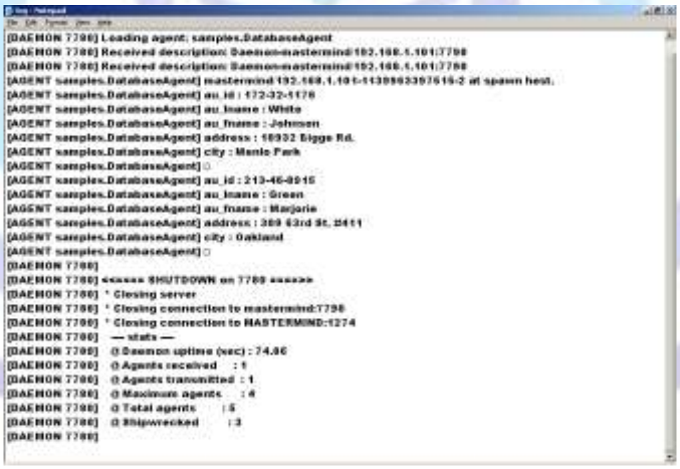
There are two options to providing parameters for the start up of the server. An option is the manual entry of parameters as shown in Figure 4. These parameters include:

- The port number of the port the server will listen to and use for migration of agents.
- The list of agents separated by commas to create at the start of the server
- The file in which to log events and errors

The verbosity, either true or false, when verbose is true, the system displays on screen what is written into the log file as it is being written. Figure 5 shows the log file of a daemon that launched the database agent to retrieve data from a remote host. The result of the retrieval operation is written against the agent name.

The second option is to specify a configuration file. The configuration file is written in xml format e.g. <daemon port="7790" log="Foo.log" remoteClassLoading="false" verbose="true"></daemon>.

When either of the two options are correctly specified, the start daemon button is clicked.

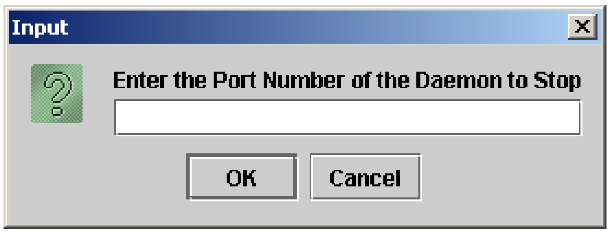


```
[DAEMON 7780] Loading agent: samples.DatabaseAgent
[DAEMON 7780] Received description: Daemon-mastermind/192.168.1.101:7798
[DAEMON 7780] Received description: Daemon-mastermind/192.168.1.101:7798
[AGENT samples.DatabaseAgent] mastermind/192.168.1.101-1139963397616-2 at spawn host.
[AGENT samples.DatabaseAgent] au_id : 172-32-1176
[AGENT samples.DatabaseAgent] au_lname : White
[AGENT samples.DatabaseAgent] au_fname : Johnson
[AGENT samples.DatabaseAgent] address : 10932 Bigge Rd.
[AGENT samples.DatabaseAgent] city : Menlo Park
[AGENT samples.DatabaseAgent] □
[AGENT samples.DatabaseAgent] au_id : 213-46-8915
[AGENT samples.DatabaseAgent] au_lname : Green
[AGENT samples.DatabaseAgent] au_fname : Marjorie
[AGENT samples.DatabaseAgent] address : 309 63rd St. #411
[AGENT samples.DatabaseAgent] city : Oakland
[AGENT samples.DatabaseAgent] □
[DAEMON 7780]
[DAEMON 7780] <<<<<< SHUTDOWN on 7780 >>>>>>
[DAEMON 7780]  * Closing server
[DAEMON 7780]  * Closing connection to mastermind:7798
[DAEMON 7780]  * Closing connection to MASTERMIND:1274
[DAEMON 7780]   --- stats ---
[DAEMON 7780]   @ Daemon uptime (sec) : 74.86
[DAEMON 7780]   @ Agents received     : 1
[DAEMON 7780]   @ Agents transmitted  : 1
[DAEMON 7780]   @ Maximum agents      : 4
[DAEMON 7780]   @ Total agents        : 5
[DAEMON 7780]   @ Shipwrecked         : 3
[DAEMON 7780]
```

Figure 5: A typical Daemon log file

B. *Stopping the daemon*

Click the stop daemon button and you see the screen in Figure 6. Enter the port number of the server you want to stop and click OK.



ISSN 22773061

Figure 6: Port Daemon to stop screen

C. *Running Agents*

Though the first option of specifying parameters manually to start the daemon allows for specifying agents that will start with the daemon but run agents after the daemon; the launch agent screen shown in Figure 7 can be used also.

This is done by entering the name of the class of the agent and clicking the go button. An input box comes up asking for the port number of the daemon to host the agent. For the sample agents, the name are samples.FileTransferAgent (for the file moving agent), samples.DatabaseAgent (the data retrieval Agent) and samples.FingerAgent agent (the agent that gets details of a computer on the network).

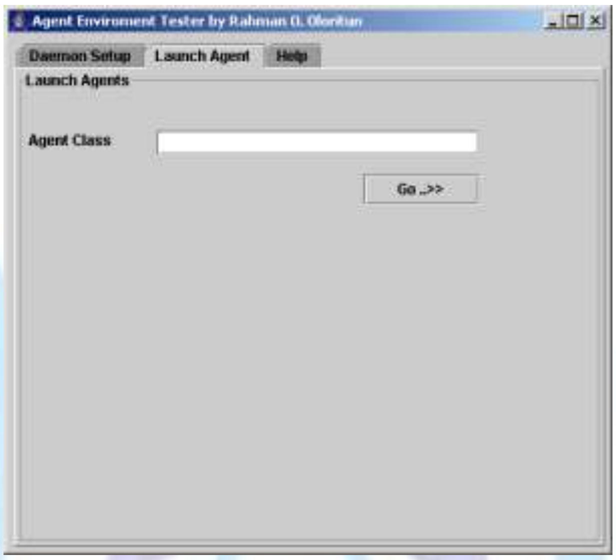


Figure 7: Launch Agent Screen

D. *Help screen*

The help screen gives a very brief description of the system, quick tips on how to use the testing window, and contact details of the developer of the system.

6. CONCLUSIONS

This study presents the analysis, design and implementation of a simple java based mobile agent environment for the development and deployment of mobile agent applications (SIMMAS). In the context of this work, several existing mobile agent systems were reviewed and a thorough analysis of an ideal mobile system was done and sample mobile agents doing the common but difficult to implement activities like file transfer, database access and remote host identification. SIMMAS provides the programmer with a library for writing mobile agent applications and an environment in which to execute the agents.

The simple java based mobile agent environment for the development and deployment mobile agent applications (SIMMAS) is an infrastructure for building mobile agents. It is entirely based on java sockets programming. The prototype mobile agent system implementation has successfully demonstrated the basic features of a mobile agent system. Agents can suspend their execution and transfer themselves across a network to another host, and then resume execution where they left off. Future work include incorporating a mechanism for inter-agent communication into our transportable agent system; adding an agent reproduction mechanism that would allow agents to spawn child agents to perform subtasks; implementing various security techniques such as credential verification, file access restriction, permits; and, improving the error handling mechanisms.

7. REFERENCES

[1]D.B. Lange and M. Oshima, "Programming and Deploying Java Mobile Agents with Aglets",Addison-Wesley, Reading, MA, USA,1998.

[2]M. Donszelmann "Mobile Agent Systems", Cern School of Computing http://webcast.cern.ch/Projects/CSC99/lectures. [accessed 2012]

[3]Kozierok, R. and P. Maes, "Learning Interface Agent for Scheduling Meetings", Proceedings of ACM SIGCHI International Workshop on Intelligent Interface, ACM Press, N.Y., 81-88, 1993

[4]Maes, P., "Agents that reduce work and information overload", Communication of the ACM, 37(7), 31-40 1994.

[5]M. Wooldridge and N. Jennings, Intelligent Agents: "Theory and Practice", Knowledge Engineering Review, 10(2), 1995. Available at http://www.elec.qmw.ac.uk/dai/pubs/KER95/[accessed 2012]

ISSN 22773061

[6]J. Yang, P. Pai, V. Honavar, and L. Miller. “Mobile Intelligent Agents for Document Classification and Retrieval: A Machine Learning Approach” in Proceedings of the European Symposium on Cybernetics and Systems Research, 1998. Available at http://www.cs.cmu.edu/afs/cs.cmu.edu/user/honavar/www/Papers/emcsr98.ps. [accessed 2012]

[7]James E. White. Telescript technology: The foundation for the electronic marketplace. General Magic White Paper, General Magic, Inc., Sunnyvale, California, 1994.

[8]M. Breugst, I. Busse, S. Covaci, and T. Magedanz, “Grasshopper – A Mobile Agent Platform for IN Based Service Environments” in Proceedings of IEEE IN Workshop 1998, pages 279–290, Bordeaux, France, May 1998. Available at http://www.ikv.de/download/grasshopper/IEEEINWS98.pdf [accessed 2012]

[9]D.B. Lange and M. Oshima, “Seven Good Reasons for Mobile Agents”, Communications of the ACM, 42(3):88–89, March 1999.

[10] D. Chess, C. Harrison, and A. Kershenbaum, “Mobile Agents: are they a good idea?”, IBM research Report, IBM T. J. Watson Research Center, Yourktown Heights, N.Y. RC 1987, December 1994.

[11] Gensereth, M., R. and S. P. Ketchpel, ‘Software Agents’, Communication of the ACM, 37(7), p 48-53, 1994.

[12] Y. Aridor, and D.B. Lange, “Agent Design Patterns: Elements of Agent Application Design”, in Proceedings of Second International Conference on Autonomous Agents (Agents ’98), ACM Press, pp. 108-115. 1998.

[13] http://www.aglets.sourceforge.net [accessed 2012]

[14] R.S. Gray, “Agent Tcl: A Flexible and Secure Mobile-Agent System,” doctoral dissertation, CS Dept., Dartmouth College, Hanover, N.H., 1997.

[15] Beegent, http://www.toshiba.co.jp