GSTF Journal on Computing (JoC) Vol.2 No.4,January 2013

A Java-based Mobile Agent Framework for Distributed Network Applications

Salah Jowan and Tony Mullins
Department of Computer Science, Faculty of Sciences, University of Elmergib, Libya,
E-mail: salahjowan@hotmail.com
Computing Department, Griffith College Dublin (GCD), Ireland, E-mail: tony.mullins@gcd.ie

***Abstract*—Recently, a new paradigm has emerged for structuring and developing distributed network applications in open distributed and heterogeneous environments. Many application areas, such as electronic commerce, mobile computing, network management and information retrieval can benefit from the application of the Mobile Agent technology. The exploitation of Mobile Agents offers several peculiar advantages, such as reduction of network latency, asynchronous execution, robust and fault tolerant behavior. Java technology provides a platform-independent, portable software environment which makes it an excellent tool for mobile agent development. Mobile Agents are mainly intended to be used for applications distributed over large scale (slow) networks because they allow saving communication costs by moving computation to the host on which the target data resides. However, it has not become popular due to some problems such as security. In this paper, we present a distributed network architecture based on the Mobile Agent approach. A network of communicating servers each of which support multiple clients is our goal. We also propose a security approach for mobile agents, which protect critical data of mobile agents from malicious attacks, by using cryptographic techniques. We implement a bank service application to be tested on our mobile agent framework. The results suggest that for networks with high latency, Mobile Agents may provide improvements over more conventional client-server systems.**

***Keywords*—Java, Mobile Agent, Distributed Systems, Cryptography.**

I. Introduction

Large scale distributed systems typically involve a number of nodes, which may be distributed over a large geographical area. The Client-Server (CS) paradigm has been found useful for designing distributed systems. However, CS approach does not scale well when the number of servers increase [1] and build complex relations with one another. Also, the underlying network characteristics, such as link bandwidths and delays, may vary over a period of time. It is difficult and cumbersome to model such applications using only traditional architectures like client-server model. Agent paradigm is a promising choice for network-centric applications, especially for distributed applications and services, because it is intrinsically communication and cooperation oriented [2, 3]. Agent concepts and mobile software agents have become a part of the system and service architecture of the next generation networks. It is where agent's mobility offers important advantages because of the network load reduction, increased asynchrony between the communicating entities and higher concurrency. Global end-to-end interactions, typical for a client-server paradigm, are replaced by local interactions in a server, visited by a mobile agent. Consequently, the need for long reliable connections is reduced; bandwidth requirements are lower and repeated interactions less frequent.

In this paper we show that Mobile Agent paradigm can help in effective structuring of large-scale distributed systems. The gains are in terms of scalable and flexible architectures, and dynamically extensible applications. The advantages of this approach are more flexibility and its suitability for use in a client/server implementation model.

II. Background

Several approaches arose in attempts to improve upon performance and alleviate some of the problems and limitations, which were discovered. For example, the use of several remote procedure calls (RPC) to perform a client-server transaction may use more network bandwidth than sending a more complicated query to a server, performing necessary computation or accessing of databases locally, and returning the results to the client [4]. Initial attempts used the concept of process migration in an attempt to save bandwidth and increase performance. However, movement of an entire address space from one machine to another, as utilized by this technique, makes it difficult to return the results to the client without returning the entire process as well [5]. The concept of remote evaluation (REV) programming [6] improves on process migration by allowing a program to be sent within a request, having it executed on a remote server, and returning only the results to the client. However, lack of state information limits the usefulness of remote evaluation (REV) based systems. Mobile objects were subsequently developed, in which object-oriented programming techniques are used to encapsulate state as well as code.

In recent years, the Mobile Agents (MA) has emerged as a useful paradigm for overcoming the above limitations [7]. Mobile agents extend on the functionality of mobile objects by adding autonomous and asynchronous execution capabilities. This allows mobile agents to decide for themselves the most efficient means to obtain data, or route around network bottlenecks.

DOI: 10.5176/2251-3043_2.4.219

Several academic research projects (e.g., [8, 9]) explore the mobile agent paradigm, and several commercial systems (e.g., Aglets [10], Voyager [11], Concordia [12]) have been introduced recently. Most of these systems are based on Java for the programming of agents, but they largely differ in their migration and security models and most importantly in the support and services they provide for the agents. Some aspects of our own mobile agent framework will be presented further down in Section IV.

III. WHAY MOBILE AGENTS?

The term 'Agent" is heard frequently today while it means a variety of things to a variety of people [13], commonly it is defined as an independent software program, which runs on behalf of a network user. An agent may run when the user is disconnected from the network, even if the user is disconnected involuntarily. Some agents run on specialized servers, others run on standard platforms. Many examples of agent systems exist, and they are receiving much attention on the World Wide Web. A Mobile Agent is specialized in that in addition to being an independent program executing on behalf of a network user, it can travel to multiple locations in the network. As it travels, it performs work on behalf of the user, such as collecting information or delivering requests. This mobility greatly enhances the productivity of each computing elements in the network and creates a uniquely powerful computing environment well suited for a number of tasks. A Mobile Agent is not bound to the system where it begins execution. The Mobile Agent is free to travel among the hosts in the network. Created in one execution environment, it can transport its state and code with it to another execution environment in the network, where it resumes execution. By transmitting executable programs between (possibly heterogeneous) machines, agent-based computing introduces an important new paradigm for the implementation of distributed applications in an open and dynamically changing environment. This paradigm can even be understood as an architectural concept for the realization of distributed systems. It is particularly well-suited if adaptability and flexibility are among the main application requirements. At the moment it is not yet clear whether mobile agent technology will establish itself as an independent computing paradigm for practical use in the long run. Chess et al. [4] made an attempt to estimate the benefit of mobile agent technology. They concluded that all the problems considered as good examples for the use of mobile agents can also be solved using traditional client/server solutions. However, mobile agents allow a general solution to all such problems. Instead of having to create different, well-tailored solutions to every problem, mobile agents provide a generic solution to all these problems. In [4, 5, 14] the use of mobile agents for distributed systems has several potential benefits such as reduction of network latency, asynchronous and autonomous execution, robust and fault tolerant behavior.

IV. SYSTEM MODEL

Our goal is to design and implement a distributed system in which we should be able to control a connection between possibly multiple Clients and a single server. The idea is that the Server stores data required by the Clients. When a Client requires data, it sends a request to the Server and waits for a reply. In the case of the data required by the Clients does not reside in the Server, the Server then consults other servers for such data and send it to the Clients. To design such system, we build two servers (see Fig. 1). The first one, is to communicate with Clients (Client/Server paradigm) and the other one, is to communicate with other servers (Mobile Agent paradigm).

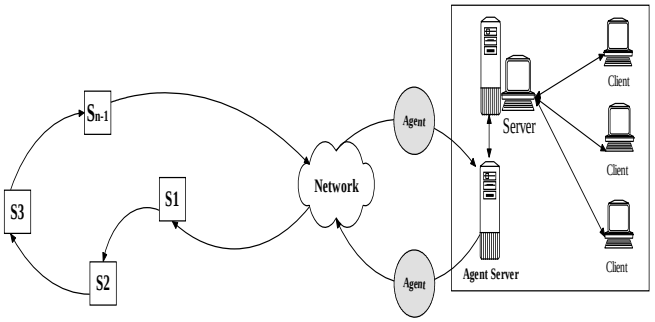

Figure 1. The overall architecture of the model.

A. *Client/Server Architecture*

Client-server paradigm describes the relationship between two computer programs in which one program, the Client, makes a service request to another program, the Server, which fulfills the request. In a network, the Client-server model provides a convenient way to interconnect programs that are distributed efficiently across different locations.

In Client-server model, one server is activated and awaits Client requests. There are two ways to build a server that can carry on conversations with multiple Clients at the same time. One solution is to write a non-blocking server; the other is to write a multithreaded server. The multithreaded alternative is almost always considered a better solution [15].

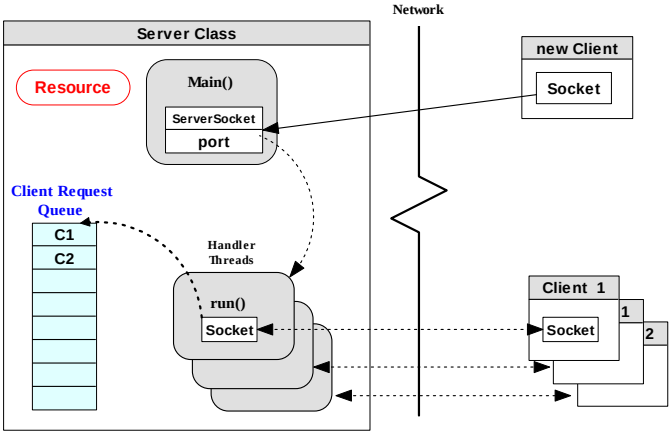

Figure 2. Multithreaded Server dealing with Clients by launching a handler thread for each connection.

In our architecture, the Client sends a request to the Server for some information; the Server accepts the Client connection and

then launches a handler to deal with the Client's request. The main function of the handler is to take the Client's request and look for the required information in the Server resources. If the data required by the Client in the Server database, the handler then will send such data to the Client, otherwise, the handler put the Client's request in a queue called Client request queue as it is illustrated in the previous diagram. The Server uses this queue in order to retain the Clients which the Server could not satisfy their requests from its resources. The Server also, uses this mechanism (the queue) so that it can be able to communicate with other servers to satisfy its Client's requests one by one.

B. *AgentServer Architecture*

In our architecture, the Agent server is the mechanism of conversation between servers. When the main server cannot satisfy a Client request for information, the Server should be able to contact other servers in order to bring such information to the Client. In this case, the main server keeps those requests in a queue and contacts the Agent server to handle such requests one by one. The main server has a thread, called queue handler, which is responsible of managing the queue as well as dealing with the Agent server. The Agent server is in a loop listening to the requests coming from the main server. Once the Agent server gets a request, it creates an Agent for that request and sends it off through the network. The Agent, which has the Client's request, can travel from server to another over the network in search of such information for the Client. The Agent also has the capability to return to its original point whenever it gets the Client's data and once the Agent returns to its original Agent server, then the Agent server takes the data from the Agent and send it to the Client (see Fig. 3).

Mobile Agents are transferred between computing locations, which are Agent servers/hosts. Agent servers are also, responsible for providing the resources Agents need to work. In addition, they are responsible for handling procedures of packaging an Agent and moving its resource and data pieces from one host to another.

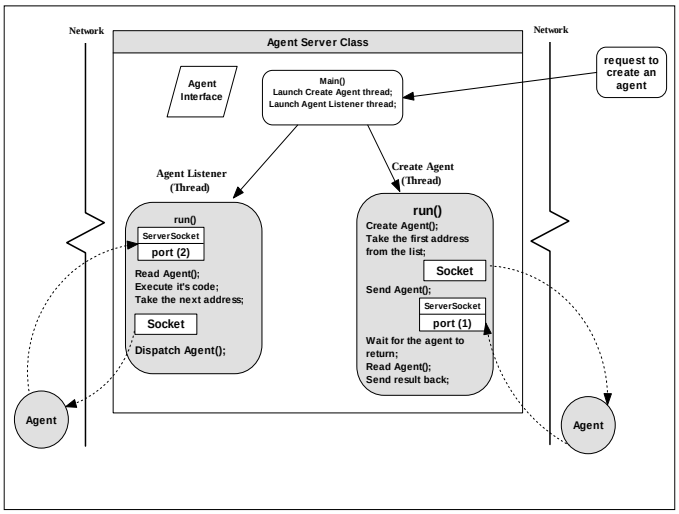

Figure 3. Agent Server Architecture.

In the diagram above, it is obvious that the architecture has three main components, which are the Agent Creator, the Agent Listener and the Agent object.

1) Agent: The Agent is a serialized object that represents a data structure and carries some information for its user. The Agents use specialized servers (Agent server) that receive them, allow them to interact with the given environment and, when finished, dispatch them to their next destination. Each Agent has its own migration mechanism to determine its own path through the network. Our Agent object consists of two fundamental parts: state and code. State describes attributes of Agent and code is the path of control of the Agent thread.

2) Creat an Agent: The main role of the Agent creator is to create an Agent whenever is required by the Agent server. When the Agent server receives a Client's request from the main server, it launches a thread to create an Agent, which will carry out actions to fulfill the Client's request. The Agent creator thread creates the Agent with a unique id in order to identify and locate the Agent during its lifetime. Also, the Agent when first created is provided by the Client's request and the migration mechanism (the path) in order for the Agent to be able to migrate between different hosts in the network. To create an Agent the Agent creator instantiates the Agent object with initialization arguments. Only when the initialization has been completed can the Agent assume that it has been fully and correctly installed in the Agent creator. After being fully installed in the Agent creator, the Agent is now capable of executing independently of other Agents in the same place. The Agent creator now opens a Socket to send the Agent to the first destination in the Agent's path. The gent now can start migrating under its own control from one host to another over the network and it can move to the place where data is stored and select information the user wants. After the Agent is sent, the Agent creator opens a server socket and listens on a specific port for the Agent to come back (see Fig.3). In the Agent's path, the last destination is always the address of the Agent creator, which created the Agent. So it is absolutely guaranteed that the Agent will return to its original place after the completion of its journey. When the Agent has returned from its journey, the Agent creator extracts the information and sends it directly to the Client who is waiting for the results.

3) Agent Treansfer: As we mentioned that Agents need an execution environment (place) in which Agents operate and perform their tasks. The most common view of a place is that it is a context in which an Agent can execute (see Fig. 4). We can regard it as an entry point for a visiting Agent that wishes to execute. The place provides a uniform set of services that the Agent can rely on irrespective of its specific location.

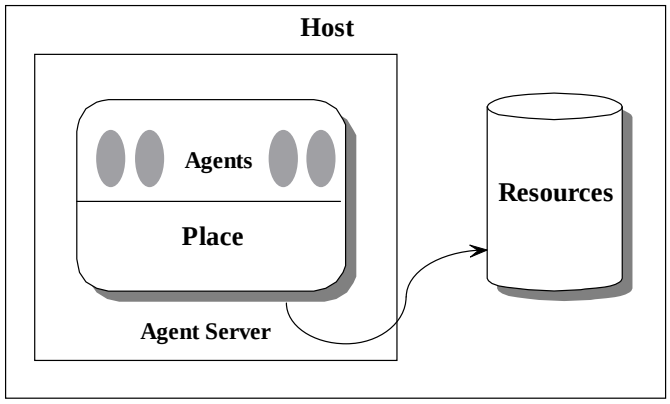

Figure 4. Execution Environment of Agents

In our Agent server model, the Agent listener is the execution environment (place) for Agents. The Agent listener is responsible for receiving Agents, executing their code and dispatching them to their next destination according to their path. The Agent listener is in a loop listening on a specific port for incoming Agents. When it receives an Agent, it will receive the Agent as a serialized object. The Agent listener then deserializes (unpack) it and sends it to a queue so the Agent can wait for its turn for executing its code in the local system. The queue mechanism is built in the Agent listener so that it can receive multiple Agents at the same time. After executing the Agent code, the Agent listener extracts the next destination address from the Agent's path, which is in the Agent object data structure.

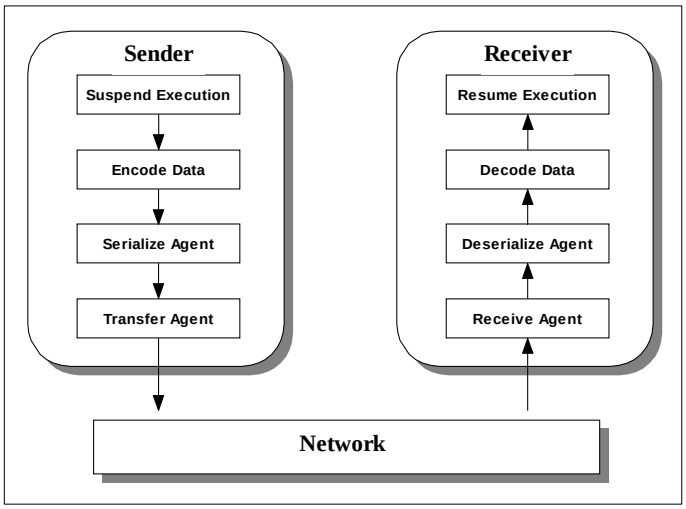

Figure 5. Agent Transfer.

When a Mobile Agent is preparing for a trip, it must be able to identify its destination. Once the location of the destination is established, the Agent system (Agent listener) should do the following (see Fig. 5):

- *Suspend the Agent*: The Agent is notified about the transfer and allowed to prepare for departure (complete its current task). When that is done its execution thread is halted.
- *Encode the Agent's data:* The Agent system encodes the data the Agent contains for security purposes. Special cryptography techniques are used to implement encryption and decryption algorithms.
- *Serialize the Agent:* The Agent (its state and class) is serialized by the Agent system. *Serialization* is the process of creating a persistent representation of the Agent object that can be transported over a network. Serialization of the Agent may include the execution state.
- *Transfer the Agent*: The Agent system establishes a network connection to the specified destination host and transfers the encoded serialized Agent.

Before an Agent system receives an Agent, the Agent system must determine whether it can accept an Agent from the sending host. Only after the sender has successfully authenticated itself to the receiving Agent system will the actual data transfer take place (see Fig. 5):

- *Receive the Agent:* When the destination Agent system agrees to the transfer, the encoded Agent is received.
- *Deserialize the Agent:* The persistent representation of the Agent is deserialized. The Agent class is instantiated, and the transferred Agent state is restored.
- *Decode the Agent's data:* The Agent system decodes the data the Agent contains.
- *Resume Agent execution:* The re-created Agent is notified of its arrival at the destination place. It can now prepare to resume its execution and is given a new thread of execution.

V. Design Assumptions

Designing and implementing distributed applications over a distributed network is considerably different from those applications that run on one machine. In this section we will discuss some important design issues that must be considered and overcome for the development of our project.

1) Security: Data is vulnerable at many points in any computer system, and many security techniques can be employed to protect it. Clients who are connected to our system should be confident that their request and information would be handled in privacy and confidentiality. When an Agent travels over a distributed network, it introduces potential security threats [16, 17]. As Mobile Agent can carry out actions autonomously, this requires that Agent knows the information about its users (Clients). It must be robust enough to prevent from revealing this information to third parties (malicious hosts) [17]. Otherwise, a host may insert its own tasks into an Agent or modify Agent's state, which can lead to theft of Agent's resources. As we mention previously, Mobile Agents are composed of code, data, and state. Agents migrate from one host to another taking the code, data, and state with them. Our approach to prevent attacks to Mobile Agents from

malicious hosts is to employ cryptographic algorithms in order to encrypt the Agents' data including their state information in a way that makes undesired hosts unable to reveal the Agent's information.

2) Scalability: In our application the Server must be scalable and must be able to contain a large number of Clients without causing any decrease in the efficiency of overall system. This requires an architecture in which the Server does not process one connection and wait another, but where it always remains available and accepts several connections at a time. Threads are employed to achieve this goal. The Server can receive a connection from a Client and can pass it to the proper thread to take care of it. The Server then is available again for next connection.

3) Transparency: The aim of transparency is to make certain aspects of the distribution invisible to the user of the system. In our design, the user of the system should not be worried about how the request will be transported between Servers and Agent Servers. Also, how the Agent will move between different hosts is invisible. Even the failure of networks and the processes is invisible and can be presented to the user in the form of exceptions.

4) Concurrency: At any given moment, there might be many Clients connected to the Server. When more than one Client will try to access the resources that should be consistent, and should be accessed and modified by one user at a time. If multiple Clients access such resources simultaneously, it can lead the Server to be inconsistent. This inconsistency of the Server can lead to unexpected and unwanted results. By using proper Synchronization techniques, we overcome this problem and the Server is stable and consistent.

5) System Reliability & Fault Tolerance: Computer systems sometimes fail. Communication failure between Client and Server can happen. The system must be able to respond in any such situation. Supposing that the Agent Server, for example, is waiting for an Agent to return, but the Agent is dead in one of the hosts, what would happen. The Agent system should be able to find out such problem, notify the user about the failure, and go back to work. In our system, in order to build such a reliable and fault tolerant system, we use the power of exceptions provided by Java to handle all possible problems.

VI. System Implementation

To implement the Client/Server architecture, we simulate a bank service system. The bank service acts as a Server to its customers (users) who are looking for some information and transactions from their accounts. The customers' accounts represent the Server's database. The ATM machine in the bank system acts as a Client in which the client communicates with the server on behalf of the user. Fig. 6 illustrates the interaction between the Client and The server within the bank system.

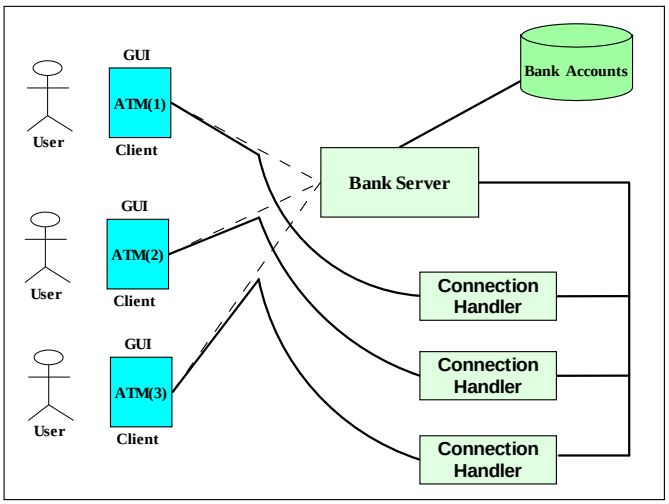

Figure 6. Bank System Simulation.

A. Client Application

The Client application in our system represents the ATM machine action. As we mentioned previously that the Client makes a request for some information to the Server and waits for a reply. As the Client gets a reply from the Server, the Client displays the replay as a result to its user. To implement the ATM Client, we divide this task into two parts. The first part, we implement the user interface in order to enable the user to interact with the Bank Server by allowing the user to enter a request as well as to see the result (see Fig. 7). The second part is to implement the connection required to have the Client and the Server talking to each other.

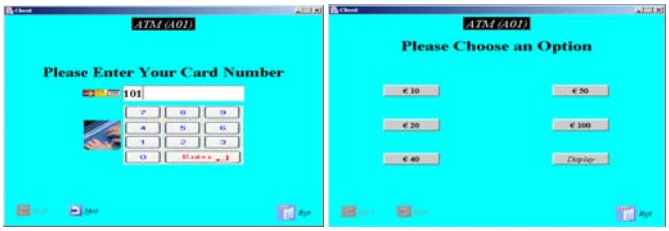

Figure 7. Screenshot of the ATM client.

B. Server Application

The basic procedure for implementing a Server is to open a server socket on a particular local port number, and then to wait for connections. Clients will connect to this port and, a connection will be established. The Agent Server is responsible for handling the communication with other servers in the network. It takes a Client request from local Server and creates a Mobile Agent for the Client and then sends it off over a distributed network in order to fulfill the Client request from other servers. In addition, the Agent Server is responsible for receiving Mobile gents from other servers and executing their codes as well as providing them with access to the local Server resources.

To implement the targeted Agent Server, we divide the Agent Server entity into two parts (see Fig. 8). The first part is concerned with creating Agents and sending them off. The other part is the place where Agents are received and executed.

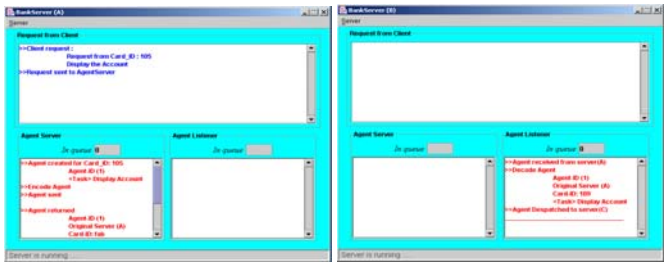

Figure 8. Screenshots from running the agent system.

VII. CONCLUSION

In this paper, a model of distributed network architecture was presented. This architecture is based on the client-server approach in cooperation with the mobile agent approach. At one end, the client-server paradigm was used to form the communication mechanism between clients and the server. At the other end, the mobile agent technology was used to facilitate the communications between servers over the network. Also, a number of techniques were used which assist in supporting security, concurrency, scalability and fault tolerance. We showed that the advantages of mobile agent approach are more flexibility and its suitability for use in a client/server implementation model.

Although initial results using the current architecture have been encouraging, there is still some future work that could be done to improve upon the agent architecture described in this work. We tested our architecture by implementing a Bank Service application. There are many other similar applications can be implemented on our agent architecture. It is better to write code for any of those applications and test it on our architecture.

REFERENCES

[1] J. White. Mobile Agents, in Software Agents, J. Bradshaw (ed.), AAAI Press / The MIT Press, 1996.

[2] Y.Aridov and D.Lang, “Agent design patterns: element of agent application design”, Proc. Autonomous Agents ‘98, pp108-115. ACM, ’98.

[3] Antonio Carzaniga, Gian Pietro Picco and Giaovanni Vigna, “Designing Distributed Applications with Mobile Code Paradigms”, Proc. 19th International Conference on Software Engineering (ICSE '97) , pp.22-32, ACM , ‘97.

[4] C. G. Harrison, D. M. Chess, A. Kershenbaum, “Mobile Agents: Are They a Good Idea?”, Technical report, IBM T.J. Watson Research Center, Mar. 1995.

[5] D. Wong, N. Paciorek, D. Moore, “Java-based Mobile Agents”, Communications of the ACM, Vol. 42, No. 3, Mar. 1999.

[6] J. W. Stamos, D. K. Gifford, “Remote evaluation”, ACM Transactions on Programming Languages and Systems, Vol. 12, No. 4, pp 537-565, Oct. 1990.

[7] F. C. Knabe, “An overview of mobile agent programming”, Proceedings of the 5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages, Stockholm, Sweden, Jun. 1996.

[8] Gray R.S., Agent Tcl: “A Flexible and Secure Mobile-Agent System”, Proc. 4th Annual Tcl/Tk Workshop, Monterey, CA, 1996, pp 9-23.

[9] Peine H., Stolpmann T., “The Architecture of the Ara Platform for Mobile Agents”, in Rothermel K., Popescu-Zeletin R. (eds), Mobile Agents (Proc. 1st Int.Workshop), Springer-Verlag, LNCS 1219, 1997, pp 50-61.

[10] Lange D., Chang D.T., IBM Aglets Workbench – “Programming Mobile Agents in Java”, white paper, IBM Corporation, Japan, August 1996.

[11] Voyager: http://www.objectspace.com/voyager/

[12] Wong D., Paciorek N.,Walsh T,Concordia: “An Infrastructure for Collaborating Mobile Agents”, in Rothermel K., Popescu-Zeletin R. (eds), Mobile Agents (Proc. 1st Int.Workshop), Springer- Verlag, LNCS 1219, 1997, pp 86-97.

[13] S. Franklin, A. Graesser, “Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents”, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Institute for Intelligent Systems, University of Memphis, 1996.

[14] D.B. Lange and M. Oshima. “Seven good reasons for mobile agents”. Communications of the ACM, 45(3):88–89, March 1999.

[15] Hughes, Shoffner, Hamner, “Java Network Programming”, Second Edition, Manning Publications Co., 1999.

[16] Hyungjick Lee, J. Alves-Foss, S. Harrison, “The use of Encrypted Functions for Mobile Agent Security”, System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on, 5-8 Jan, 2004, Pages: 297-306.

[17] T. Sander, C. Tschudin, “Towards Mobile Cryptography”, In Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, 1998. IEEE Computer Society Press.

Salah Jowan has BSc in Computer Science from University of Tripoli in Libya. He also obtained an MSc in Computing from Griffith College Dublin (GCD) in 2004 and then joined Elmergib University in Libya as a lecturer. Mr. Jowan has been demonstrating/lecturing in computer science for over seven years and is currently the course director for Computer Science undergraduate programmes. His areas of interest include concurrent programming, distributed computing and mobile agents.

Tony Mullins graduated with an MA in Philosophy from University College Dublin (UCD) in 1977 and a HDipEd from Trinity College in 1979. He subsequently went on to study Mathematics in the Dublin Institute of Technology (DIT) and Computer Science at UCD where he obtained an MSc (Qual) in 1986. He worked as a researcher and project manager for Decision Support Systems on a number of Esprit projects. He joined the Faculty of Computing in 1992. Mr. Mullins is Course Director of the MSc degree in Computing and specialises in formal methods, concurrent programming, real-time systems, and programming languages. In 2000 he published his book *"A First Course in Programming with Java"*.