2009 First International Conference on Advances in Future Internet

Full and Autonomic Mobility Management for Mobile Agents

Leila Ismail and Boumediene Belkkhouche

College of IT
United Arab Emirates University
P.O.Box 17555, Al-Ain, UAE
Email: leila@uaeu.ac.ae, b.belkhouche@uaeu.ac.ae

***Abstract*—Cooperation among autonomous mobile agents requires mechanisms to efficiently support communication. The interaction between mobility and communication and the constant re-localization of mobile agents raise several mobility management issues. To address these issues, we develop mechanisms for a full and autonomic mobility management of agents within an underlying model. In addition of being efficient and transparent to agents, our mechanisms take into account the interaction between mobility and communication among agents.**

I. INTRODUCTION

The use of mobile agents [1] as a model to structure distributed systems and applications has drawn a great attention in the past few years. Even though numerous researchers have been investigating the development of mobile agents systems [5], [6], [8], [10]–[12], most of the emphasis is focused on the communication infrastructure, leaving much of the mobility management efforts to be handled at the application level. In particular, we notice the absence of a full management of agent mobility. We refer to full management as the capacity of the system to move an agent and to manage the effects of this mobility on the communication links with other agents in the system. Full management frees the programmer and provides greater autonomy to agents. In this paper, we propose mechanisms for the management of agent mobility. In addition to being transparent to agents' applications, our mechanisms resolve the problem of the tension between communication and mobility with the introduction of dedicated run-time structures to support communication of mobile agents.

The rest of the paper is structured as follows. In section II we provide a brief background for this work. In section III, we present the localization problem of mobile agents. In section IV we describe the relationship between naming, localization, and communication in a mobile agent system. Requirements on communication are discussed in section V. In section VI, we describe the system architecture supporting our mechanisms. We conclude in section VII.

II. BACKGROUND

With the growing use of Internet applications by a wide community, distributed applications, such as information search on the web and electronic commerce applications [13], [14] are widely used. The use of the Internet as a platform to execute distributed applications introduces two major problems: the heterogeneity of machines and the lack of efficiency of the applications. First, we need to write programs that execute everywhere on the Internet. Second, applications slow performance becomes a major problem, as applications components are distributed over a large-scale network, and may run on a relatively slow distributed environment.

Mobile agent based programming [1] is an emerging paradigm for structuring distributed applications over the Internet. An agent is a process that can move with its code and execution context from site to site to perform its task. Mobile agents are mainly intended to address autonomy and efficiency problems of applications distributed over large scale and slow networks. They reduce communication costs by moving computation to the host on which the target data reside [15].

The research in the area of mobile agents programming is highly active. Several projects, such as Telescript [5], D'Agents ([6], [7]), Aglets ([8], [9]), Mole [10], MOA [11], and Voyager [12], have developed execution environments for mobile agents. Theoretical models based on the π calculus (e.g., [19]) treat communication at a very high level. They usually distinguish syntactically between location-dependent and location-independent communication. In either case a special process acting as as manager is needed to coordinate communication. In general, the proposed solutions address only partially the mobility issue of agents. In particular a full management of mobility is not handled by these systems.

In this paper, we call a client agent the agent that initiates a communication link with another agent, the server agent. As an example of full management capabilities of mobile-agents based systems, if a mobile client agent A wishes to obtain a service from a mobile server agent B, then this communication should take place independently of B's possible mobility. Full management consists then of two major services:

- Agent localization. A mobile agent can move from one site to another during its lifecycle. The agent

978-0-7695-3664-4/09 $25.00 © 2009 IEEE
DOI 10.1109/AFIN.2009.13

localization service consists of the mechanisms which allow finding the current location of an agent.
- Message tracking. A message can be sent to an agent that moved to another location, or is in the process of moving. The message tracking service consists of the mechanisms which allow delivering messages to their destination. Messages should not be lost.

In Mole [10], messages received by a mobile agent site and which are intended for a mobile agent in process of moving are not accepted. Mole does not suggest solutions for the communication links between the mobile agent and its clients. Management of communication links become the responsibility of the applications agents. MOA [11] provides another approach for the mobility management of agents. The approach is based on the migration of the communication channels established between the client agents and the server agent which moves and with which the clients communicate. Client agents receive an exception message for every communication with an agent whose movement has been initiated. It is the programmer's responsibility to deal with these exceptions messages, i.e., by locating the mobile agent and re-sending the messages. The Aglets [9] platform allows communications between agents using their corresponding proxies. A proxy becomes invalid if an agent moves. Then, it becomes the programmer's responsibility to locate the agent by communicating with a finder repository. The Mobile Agent System Interoperability Facility (MASIF) ([3], [4]) is a group of industries specifying a standard for interoperable mobile agents systems across the world. The MASIF specifies an interface for transferring and locating agents. However, it does not specify full management of agents mobility. In particular, the MASIF requires that a mobile agent system support the get_MAFFinder() operation. This operation retrieves a reference to the localization server MAFFinder which is responsible for locating agents.

Based on the current literature, we classify mobility management mechanisms into two categories:

- Mechanisms based on the migration of communication channels. In these mechanisms the agent informs all its collaborators of its intention to move. Subsequently, all the messages addressed to the agent are not delivered directly to the agent, but stored by the receiving agent site in a vector of non-read messages. Once an intent to move is communicated by an agent to the mobile agent system, the latter sends an acknowledgment and closes all its communication channels with the agent in the process of moving. When the agent requesting a move receives an acknowledgement from all its clients, the agent is suspended from reading messages. Whenever a client agent communicates with the mobile agent, an exception is returned to the client application. The client has to deal with the exception at the programming level. All the communication channels are re-established after the move is completed successfully. This solution does not handle well scalability, that is, as the number of communicating agents and the corresponding communication channels increase, its efficiency suffers. In addition, each server agent has to keep a list of all the client agents with which communication channels are established. The management of this list degrades seriously the overall performance of the mobile agent system.
- Mechanisms based on a complete separation between mobility and localization. In these mechanisms the mobility of an agent is supported by the system. However, the localization is a supplementary service offered by the mobile agent platform. The programmer of agents can then use this service to locate a mobile agent. This solution requires programming efforts at the application level for agent localization. We argue that programmers would rather concentrate on the application logic rather than on the programming efforts of services that ought to be provided by the system.

When designing our agent mobility mechanisms, we consider the following requirements:

- Transparency in maintaining communication links via transparent localization. A client agent must be able to maintain its communication links with an agent server despite the server agent mobility. The re-localization of mobile agents must be transparent to clients.
- Transparency in managing messages. The management of messages for an agent in transit must be transparent to client agents. Delivery of messages must be guaranteed.
- Management efficiency. The mechanisms for the management of messages must be devised in a way to minimize the management overhead.

Existing mobile agents systems propose partial solutions to the above problems. To ensure communication with a mobile agent, the system must include proper designation, mobility and localization mechanisms. As mobile-agents systems are interesting in large-scale distributed environments like the Internet, efficiency should be considered while designing these mechanisms.

The mechanisms we propose take into consideration the above requirements. These mechanisms provide agents in a transparent fashion the ability to communicate regardless of mobility.

III. LOCALIZATION PROBLEM

The following issues motivate our design choices for the localization mechanisms of mobile agents:

- Transparency. Applications should not include localization-related instructions. The localization and re-localization processes are implemented within the underlying mobile agent system. The programmers of

mobile agents-based applications should not include code to locate agents explicitly.
- Efficiency. The localization mechanisms should not generate a large number of system messages exchanged through the network.
- Modularity. The localization mechanisms should be implemented in a separate layer from the agent's application code. Therefore, the programmer would concentrate on the application programming logic rather than on localization issues.
- Portability. The localization mechanisms should be general and applicable to all mobile agents systems. The mechanisms should not include features that are operating-system-dependent or language-dependent so that they can be ported to any mobile agent system.

IV. NAMING AND LOCALIZATION

Naming is an integral part of the mobility management mechanisms. The choice of a naming scheme for mobile agents impacts drastically the mobility management of agents. In this section, we present our solution for the naming issues. We consider an agent as a group of objects where the root object represents the entry points to other objects internal to that agent. The root object represents the agent itself.

A. Naming and Localization

To be able to communicate with a particular agent, we need to give the agent a name. Generally, in object-oriented languages, the language associates a name with every created object. Such a name is local to the execution context of the process creating the object. An agent is assimilated to an object in a global networked environment. Therefore, a naming mechanism is needed to identify an object in a global context. A name associated with an agent should allow for the establishment of a communication link with that agent. In other words, it should allow locating that agent.

We divide the naming mechanisms into two categories: relative naming scheme and absolute naming scheme. In the relative naming scheme, the agent name has only a meaning in the current execution context of the agent. The main advantage of having relative names is their length. Relative names have small length thus saving storage. However, if the agent moves, then there is a necessity to have a mechanism which locates an agent within its new destination execution context. Forwarding pointers have been traditionally used for migrating objects in Operating Systems and later in distributed systems [16]. In the forwarding pointer mechanisms, when an agent moves, it leaves behind it its current location. For instance, if a server agent moves from $site_1$ to $site_2$, it will leave on $site_1$ the address of $site_2$. A client agent wishing to communicate with that server agent must send its message along with the forwarding pointers until the message request reaches the current server agent location. The reply message will have to follow the same chain back to the requesting client agent. This algorithm is very costly as messages may have to traverse a long chain of the forwarding pointers. The possibility of race conditions increases, as the agent might move during this localization time. This algorithm can be optimized by sending with the reply message the current location of the server agent. Consequently, the chain length is shortened for future communication requests. This algorithm must also provide a garbage collection algorithm to delete invalid forwarding pointers.

In the global naming scheme, an agent is given a name which is globally unique. We classify the localization mechanisms using a global naming scheme into two classes of algorithms: the mechanisms using the forwarding pointers algorithm, and the mechanisms using a centralized localization server, which has the latest location of a mobile agent. Aglets [9] uses a naming scheme based on the notion of a proxy. The proxy includes a unique global name associated with an agent. A proxy is returned to an agent creator when an agent is first created. This proxy becomes invalid when an agent moves. The clients have the possibility to re-validate the proxy by consulting a localization server which updates the proxy with the current location of an agent. Voyager [12] identifies its agents by a globally-unique name. In order to find the current location of an agent, the forwarding pointers algorithm is used.

In our mechanisms, we propose a naming scheme that integrates the relative and absolute naming schemes. For each agent, represented by a graph of objects, a global name is generated for the root object of the graph, while a relative name is generated for an internal object to the graph. A global name for an agent depends on the location of the home site creating that agent. More precisely, an agent name has the following format:

home-site-address:home-site-port-number:id

Where *home-site-address* and *home-site-port-number* are the address and the port number of the mobile agent system site where the agent is created, respectively. *id* is an automatically-generated serial number identifying the agent in a unique way within its creation site. Each internal object to the agent graph of objects is then identified globally by using the name of the agent and the agent local identifier.

The combination of the address of the home site of the agent and its name provides the corresponding localization server with a unique address. Consequently, there is no need for an application to know explicitly about an agent localization server. Our design choice of integrating the identity of the creation site of the mobile agent in the agent global name allows the mobile agent system to have the address of the localization server responsible for locating the agent.

We view the Internet as consisting of regions. A region is defined as a network domain for sites under the

same authority. For efficiency and robustness, we include one localization server per region instead of having one global centralized localization server. This is consistent with MASIF specifications for mobile agents [2]–[4]. Each localization server keeps up-to-date information about the location of agents created within its region. This mechanism is less costly from a client's perspective than the forwarding pointer mechanism. In particular, in a dynamic system, where agent movements are frequent, the forwarding pointers chain becomes very long making its traversal an unacceptable overhead. The use of a localization server requires a communication message between the migrating agent and its localization server after each movement. Such a communication results is less traffic and no forwarding pointers.

V. Requirements for Agent Communication

When designing our mechanisms for the management of agent mobility, the following issues are considered:

- Management of messages sent to an agent which has moved. The messages sent from other agents to an agent which has moved from its original location should not be lost.
- Management of messages sent to an agent in the process of moving. When an agent is in moving state, its state is saved. Such an agent is unable to accept messages sent to it. Again, these messages should not be lost.

To solve the first problem, two techniques have been used in the literature of mobile agents: the forwarding technique and the exception technique. In the forwarding technique, messages sent to an agent follow the agent to its current migrated location [11]. Each message is first sent to a localization server, which in turn sends it to the current location of the agent. A serious drawback of this technique is that messages are first addressed to the localization server, which processes them and re-sends them to their corresponding agents in their new locations. The localization server becomes then a bottleneck in the system. The exception technique, used by Aglets [9], consists of sending an exception to the client agent sending the message. On reception of the exception, the client agent application must locate the agent using a localization server and sends the message again. The main disadvantage of this algorithm is the lack of transparency to client applications, because localization is being taken care at the application level.

To solve the second problem, two techniques have been used in the literature of mobile agents: the forwarding technique and the exception technique. In this context, the forwarding technique consists of keeping the messages received by the destination site and retransmitting those messages once the agent has completed its mobility. MOA [11] and Aglets [8] use this technique. The main disadvantage of this technique is the burden imposed on the destination server forcing it to store and manage messages for all its visitor agents in a global network. The exception technique consists of returning an exception to the client sending the message. Here, it is the client application responsibility to locate the agent and retransmit the message. The main problem of this technique is that client applications are forced to handle the exception message at the application level.

VI. Architecture

In this section, we present our proposed architecture for the management of agent mobility in a mobile agent system.

A. Overall Architecture

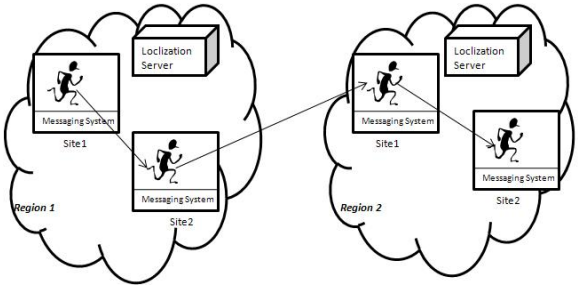

Figure 1. Overall Architecture of a Mobile Agent Platform

A mobile agent platform includes the following components as show in figure 1:

- Agent. An agent is a mobile object which can move from site to site under its own control to achieve tasks on these sites. In general, in order to move to a site, an agent must explicitly invoke a *move(site)* message. An agent is composed of its code, its execution thread, and its data which correspond to the values of the agent global variables. When an agent moves from one site to another, it continues its execution on the destination site at the instruction which immediately follows the invocation of the *move* operation. To communicate with other agents, agents invoke methods which are translated into a message by the underlying messaging system.
- Execution Environment. Each site, as part of the mobile agent platform, runs an execution environment. This execution environment implements facilities for creating agents, executing them concurrently, suspending them, destroying them, etc.
- Messaging System. A messaging system is part of an agent execution environment. It provides facilities for agents to communicate both locally and remotely. It establishes communication links between communicating agents. It is constantly aware of the status of links and when these are broken, it takes the necessary actions to establish new links for re-routing the messages to their destination.

- Localization Server. There is one localization server per region. Each localization server has up-to-date information about the locations of the agents created within its region. When an agent is created, it is registered by a naming server and then its name and location information are registered with the localization server.

B. Agent Communication

In our proposal, agents communicate using method invocations as in ORB (Object Request Broker) [18]. A communication link between a client agent and a remote server agent is translated to a link between the client agent stub and the server agent skeleton. A remote method invocation in our architecture is similar to the Java Remote Method Invocation (RMI) [17] (see figure 2).

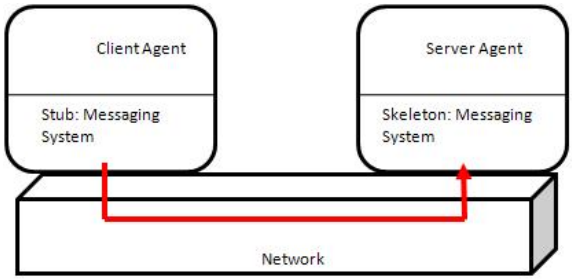

Figure 2. Communication Infrastructure for Mobile Agents

To establish a link between a client agent and a server agent, the stub in our architecture includes the following information:

- The unique global name of the mobile server agent, in addition to the relative names of the internal objects that are to be access by the client. These are needed by the mobile server agent messaging system for messages delivery.
- The current address of the mobile server agent which is obtained from the localization server. This address is transparently obtained when the client agent consults a naming service which associates an agent human-friendly symbolic name to its corresponding unique global name.

When a client agent communicates with a server agent, the address in the client stub is used to send the message to the destination server agent. In addition to the message that should be delivered to the server agent, the message also includes the global name of the agent to which the message is to be delivered, and the name of the object internal whose method to be invoked.

C. Mobile Agent State Automaton

When messages are received by the recipient site messaging system system, those message must be delivered to their destination agents. Whether to immediately deliver the messages to the agent or not depends on the agent state. The mobile agent system should know whether an agent is in a stable state allowing it to receive messages, or it has moved, or it is in process of mobility. Consequently, messages can be dealt with accordingly. In particular a message cannot be delivered to an agent which is in process of mobility. Otherwise, an inconsistent state of the agent would move. Therefore, we tag an agent with a dynamic state which changes during its lifecycle. Figure 3 shows the automaton state transition of an agent state during and after moving.

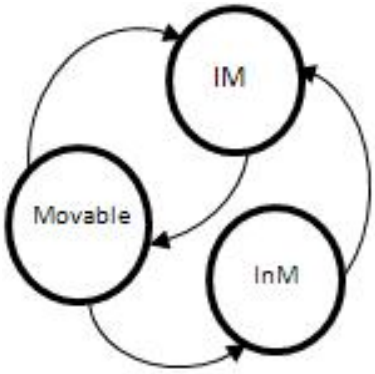

Figure 3. States of a Mobile Agent

- Movable. The agent is in a stable state. Neither the agent, nor any object internal to that agent is executing an operation. Then, if the agent calls for a move operation, the underlying mobile agent system will execute the operation and the agent then moves. When an agent is created, it is in the Movable state.
- IM (Impossible Mobility). This state corresponds to an agent in in process of executing operations (called by other agents). The agent cannot move. Otherwise, an inconsistent copy of the agent would move.
- InM (In Mobility). The agent is in process of moving. When an agent decides to move, its state changes from form a Movable state to an InM state. A message cannot then be delivered to an agent in the InM state. The mobile agent system should then return to the caller agent the new address of the agent. No operations should be executed by an agent during its move.

Note that not delivering messages to agents during moves allows for a consistent state to progress. In addition, an agent should not start to save its state before completing all the operations which are in the process of execution.

D. Localization Algorithm

LOCATE($ServerAgent_j$)

```
1  reply ←send(locateRequest, LSR_i)
2  if reply
3     then
4          establishLinkWithServerAgent(reply)
5     else
6          if reply IsException
7            then send(locateRequest, LSR_j)
```

As stated previously, there is one localization server per region. Agents created within a region should inform their home localization server of the location of their destination. Thus, a localization server within a region has an up-to-date information about its agents current locations. When a client agent wishes to communicate with a mobile server agent, the messages go through the client agent messaging system. The client messaging system will then send a localization request to the localization server which is within the same region as the client agent. If the mobile server agent is co-located with the client agent, a communication link is established between agents and local communication takes place. If the agent is not available within the client agent region, then a localization request is sent to the localization server of the mobile server agent. As stated this localization server has an up-to-date information about the mobile server agent current location. A communication link is established between agents. Agents communicate remotely.

E. Re-localization Algorithm

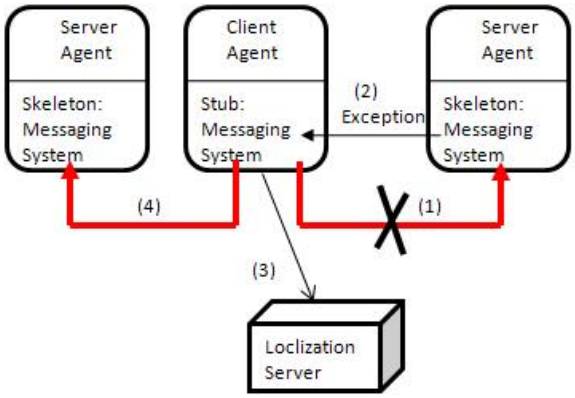

Figure 4. Communication Links between Mobile Agents

As stated previously, when a client agent locates a mobile server agent, a communication link is established between them. However, when an agent moves, then this communication link is broken. In other words, the current address of the agent server within the client agent stub is no more valid. Subsequent messages which are sent over a broken link should not be lost. In our mechanisms (see figure 4), we propose to dynamically update the location information of the mobile server agent within the client agent stub in a lazy way rather than in an eager way. An eager update means that the localization information within the client agent stub is immediately updated. This is done immediately after each movement of the mobile server agent that the client is communicating with. Although an eager approach contributes in keeping the link permanently valid, it has scalability and performance issues. As the number of client agents increases, all the stubs of the client agents should be updated before after agent movement, even if some of the clients would not need to communicate with the migrated server agent anymore. On one hand, this increases the number of messages exchanged over the network before each agent move. On the other hand, the mobile server agent has to keep a list of all the clients with which it is communicating. The management of this list, i.e., adding and removing clients would impose a performance problem in case of frequent changes of the number of clients. Consequently, the eager mechanisms are not adequate for a dynamic environment like the mobile agent system, in which agents can join a group of agents for exchanging information and then leave the group in a dynamic way.

SENDMESSAGE($Message, ServerAgent_j$)

```
1  ▷ The client's stub has an information
2  ▷ about the server agent's location
3  ▷ which may be outdated.
4  reply ←send(Message, ServerAgent_j, location)
5  if reply IsResult
6     then deliverToClient(reply)
7  if reply IsLocalizationtype
8     then if reply NOTexception
9     then establishLinkWithServerAgent(reply)
10         send(Message, ServerAgent_j)
11    else location ←send(locateRequest, LSR_j)
12         Send the message to the agent.
13         Goto 4.
```

In the lazy approach (see above algorithm), when a client agent messaging system sends a messages to the messaging system of a mobile server agent which has moved, an exception message is returned to the client agent stub by the mobile agent underlying system. The client agent stub localization module will then deal with the exception by sending a localization request to the localization server to find the mobile server agent. The localization server is available on the creation site of the mobile server agent. The client agent stub localization module will detect the address of the localization server from the agent global name which is stored within the client's stub. This name includes the address of the creation site of the mobile agent, where a localization server can be found. We assume that the global unique name of the agent includes an indication about the address of its localization server.

F. Agent in Process of Mobility State

The question here is: what happens when the messaging system of a client agent sends a message to the messaging system of a mobile server agent which is in the process of movement? Our approach guarantees that the message is not lost, but delivered to its destination. An exception message is sent from the messaging system of the mobile server agent to the messaging system of the client reporting the broken link. Consequently, the former will issue a localization request to locate the agent in the process of movement. During the time the server agent is in the process of moving, the localization server cannot provide the new location of its requested agent. Two mechanisms can be use here. The first mechanism called a polling mechanism, consisting of systematically sending the localization request until a valid reply is obtained. The polling mechanism solution results in a considerable number of localization messages sent over the network with the inherited network delay problem. This delay increases in case the setup of the agent move takes a long time, for instance due to the waiting time for the current operations that the agent should terminate before its movement. Also the delay increases if the server agent state is large; the agent state must be loaded by the mobile agent system before the agent can move. Therefore, we propose that only one localization request be issued to the localization server. The localization server will block the localization request until a valid reply can be sent to the messaging system of the client agent.

G. Reception of Messages

When a message is sent from the client's messaging system to the messaging system of the mobile server agent, the latter will consult the server agent's state.

RECEIVEMESSAGE($Message, ServerAgent_j, ClientAgent_i$)

```
1  ▷ The server's skeleton Knows about
2  ▷ the server agent's current state.
3  state ←checkState(ServerAgent_j)
4  if state isstable
5     then
6          ▷ BrokenLink
7          deliver_to_ServerAgent_j(Message)
8  if state isInMAndCreatedByLocalRegion
9     then reply ←send(LocalizationRequest, LSRJ_j)
10         send(reply, ClientAgent_i)
11    else
12         ▷ The server agent is not
13         ▷ created locally
14         send(exception, ClientAgent_i)
```

We identify three possible cases (see above algorithm):

- The mobile server agent is stable. In this case, the message is delivered to the mobile server agent by its messaging system.
- The mobile server agent has moved. In this case, a "link broken" message is sent from the messaging system of the mobile server agent to the messaging system of the client agent server.
- The mobile server agent is in process of moving. There are two cases here.
  - In the first case, the mobile server agent which is in process of moving does not belong to the same domain as the messaging system on which it is running i.e it is a visitor agent. A "link broken" message is then sent from the recipient messaging system to the client agent messaging system. The "broken link" message is dealt with by the localization module of the client agent's messaging system as described in previous sections.
  - In the second case, the mobile server agent which is in process of mobility belongs to the same domain as the messaging system on which it is running. In this case, a valid localization reply is sent back to the client agent's messaging system, which in turn resubmits the message.

VII. Conclusion and Perspectives

With the wide spread use of Internet by a large public, new classes of applications have appeared such as information search on the web and electronic commerce. The apparition of such types of applications makes the Internet a programming platform, where the heterogeneity of the underlying machines and the efficiency of a large-scale distributed environment have to be considered. Mobile agent based programming is an emerging paradigm for structuring distributed applications over the Internet, which is considered a solution to the efficiency problems of distributed applications over the Internet. While many mobile agents systems have been developed, only partial solutions have been advanced to support full management of agent mobility and communication. To efficiently manage mobility and communication involves addressing two types of mechanisms: the localization of mobile agents and the management of messages among mobile agents.

In this paper, we proposed mechanisms for the localization of mobile agents and the management of messages. Unlike existing mechanisms, ours mechanisms are more efficient and transparent to clients' applications. We also described an optimization to our algorithms by eliminating some exceptions sent to clients' localization module. Clients' applications will receive a valid reply about the current location of mobile server agents.

References

[1] C. Harrison, D. Chess, and A. Kershenbaum. "Mobile Agents: Are They a Good Idea?". IBM Research Division, T. J. Waston Research Center, Yorktown Heights, New York, March 1995. Proceedings of the ACM Operating System Review, 33(3), August 1999, pages 7-13.

[2] Open Management Group. "Mobile Agent Facility Specification". OMG TC Document cf/96-08-01, August 1996.

[3] Open Management Group. "Mobile Agent Facility Specification", January 2002. http://www.omg.org/docs/formal/00-01-02.pdf.

[4] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, D. Kosaka, D. Lange, K. Oshima, C. Tham, S. Virdhagriswaran and J. White. "MASIF. The OMG Mobile Agent System Interoperability Facility". Lecture notes in computer science, ISSBN 0302-9743, Springer Verlag, 1998.

[5] J. E. White. "Telescript Technology: The Foundation for the Electronic Market Place". General Magic Inc., Mountain View, CA.

[6] Robert S. Gray, George Cybenko, David Kotz, and Daniela Rus. "Agent Tcl. Itinerant Agents: Explanations and Examples with CDROM. William Cockayne and Michael Zypa (editors), Manning Publishing and Prentice Hall, 1997.

[7] B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko and D. Rus. "Mobile agents in distributed information retrieval". Intelligent Information Agents, edited by Matthias Klusch, Springer Verlag, chapter 15, pp. 355-395, 1999.

[8] D. Lange and O. Mitsuru. "Programming and Deploying Java Mobile Agents with Aglets. Addison-Wesley Pub Co, ISBM: 0-201-32582-9, August 1998.

[9] Mitsuru Oshima, Guenter Karjoth, and Kouichi Ono. Aglets Specifications 1.1 Draft 0.65. September 1998. http://www.trl.ibm.com/aglets/spec11.htm.

[10] J. Baiumann, F. Hohl, K. Rothermel, M. Straer. "Mole, Concepts of a Mobile Agents System". World Wide Web, vol. 1, No. 3, pp. 123-137, 1998.

[11] D. S. Milojici, W. LaForge, D. Chauhan. "Mobile Objects and Agents, Design, Implementation and Lessons Learned". Distributed systems Engineering IEEE, pp. 1-14, 1998.

[12] G. Glass. "Object Space Voyager - The Agent ORB for Java". Lecture Notes in Computer Science, no.1368, pp. 38-55, 1998.

[13] L. Moser. "Electronic Commerce using Mobile Agents Technology". 1998-1999 Final Report for MICRO Project 98-107 for implementing MAgNET (Mobile agent for Networked Electronic Trading).

[14] P. Dasgupta, N. Narasimhan, L. Moser, and P. Melliar-Smith. "MAgNET: Mobile Agents for Networked Electronic Trading". IEEE Transactions on Knowledge Data Engineering 11(4):509-525 (1999).

[15] L. Ismail, D. Hagimont. "A Performance Evaluation of the Mobile Agents Paradigm". In Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), 1999 ISBN = "1-58113-238-7, 0-201-48561-3 (Addison Wesley Longman)", pp. 306-313.

[16] A. Tanenbaum, M. van Steen. "Distributed Systems: Principles and Paradigm". ISBN: 0130888931, 2001.

[17] A. Wollrath, R. Riggs and J. Waldo. "A Distributed Object Model for the Java System". Computing Systems, vol. 9, no. 4, pp. 291-312, 1996.

[18] R. Grimes. "DCOM Programming: A guide to creating practical applications with Microsoft's Distributed Component Object Model". ISBN 1-861000-60-X. Wrox Press, 1997.

[19] P. Sewell, P. Wojciechowski, and B. Pierce, "Location-Independent Communication for Mobile Agents: a Two-Level Architecture". In *Internet Programming Languages*, H. Bal, B. Belkhouche, and L. Cardelli (eds.), LNCS 1686, pp. 1-31, Springer, 1999.