Reliable Directory Service and Message Delivery for Large-scale Mobile Agent Systems

JINHO AHN
College of Natural Sciences, Kyonggi University
Department of Computer Science
San 94-6 Yiuidong, Yeongtonggu, Suwonsi Gyeonggido 443-760
Republic of Korea

Abstract: In this paper, we introduce a reliable directory service and message delivery mechanism for large-scale mobile agent systems to address both agent mobility and directory service node failures. The mechanism enables each mobile agent to keep its forwarding pointer only on the small number of its visiting nodes in an autonomous manner. As this desirable feature has every mobile agent's migration route be very considerably shortened, the time for forwarding each message to the agent becomes much smaller. Also, this feature causes each node to maintain much fewer forwarding pointers of mobile agents on its storage than in the traditional approach. Moreover, the mechanism allows each movement path of a mobile agent to be replicated in an effective way to preserve scalability of our proposed mechanism. This replication may require a little more agent location updating costs, but much more accelerate delivery of each message to the corresponding mobile agent.

Key–Words: Mobile agent system, Directory service, Message delivery, Reliability, Scalability

1 Introduction

As wireless devices such as PDAs and cellular phones and new Internet based technologies, for example, grid, ubiquitous computing and active networks, has been rapidly emerged, modern computing environments are becoming very complex[1, 7]. Also, mobility of users and devices leads to their softwares being executed on dynamically changing environments that support different capabilities and types of available local resources respectively. In terms of software development, these issues make it difficult to design the application programs because it is impossible to obtain completely accurate information about their dynamically changing runtime environments in advance. Thus, it is essential to develop a new middleware platform allowing software components to adapt to their local execution environments at runtime.

Mobile agent technology is gaining significant popularity as a potential vehicle for considering the complexity and variety[1, 7, 8]. Mobile agent is an autonomously running program, including both code and state, that travels from one node to another over a network carrying out a task on user's behalf. However, as the size of mobile agent systems rapidly increases, scalability is becoming the most important issue and forces some components of the systems, e.g., agent communication, deployment, monitoring and security, to be redesigned. Among the components, designing reliable inter-agent communication facility in a scalable manner is essential for the systems.

Generally, this facility consists of two components, directory service and message delivery. First of all, to address the scalability issue, the centralized dependency on the home node of each mobile agent, which is the critical drawback of home-based approach[8], should be avoided in case of its location updating and message delivery. In terms of this overhead, forwarding-pointer based approach[5, 9, 12] is more preferable than the home-based one. However, the forwarding-pointer based approach has three problems in case that it is applied to a large-scale mobile agent system. First, it may lead to a very high message delivery cost whenever each message is sent to a mobile agent. The approach has this undesirable feature because as mobile agents highly migrate, the length of their forwarding paths becomes rapidly increasing. Second, this approach requires a large size of storage where each directory service node maintains agent location information. The second problem results from greatly raising the number of forwarding pointers the node should keep on its storage because the system generally serves a large number of mobile agents running concurrently. To attempt to consider this problem, a previous mechanism[9] introduces a type of update message, inform message, to include an agent's current location for shortening the length of trails of

forwarding pointers. It enables a node receiving the message to update its agent location information if the received information is more recent than the one it had. However, it presents no concrete and efficient solutions for this purpose, for example, when update messages should be sent, and which node they should be sent to. The third drawback is that even if among all service nodes on a forwarding path of a mobile agent, only one fails, any message destined to the agent cannot be delivered to it. This feature may significantly degrade reliability of the inter-agent communication facility, which becomes unpractical. To consider this third issue, a fault-tolerant directory service for mobile agents using redundancy of forwarding pointers [10] was proposed, but doesn't address the first and the second problems stated previously.

In this paper, we present a new fault-tolerant and efficient mobile agent communication mechanism based on forwarding pointers to solve the three problems as follows. For the first and the second problems, each mobile agent enables its forwarding pointer to be saved only on the small number of its visiting nodes in an autonomous manner. As this desirable feature has every mobile agent's migration route be very considerably shortened, the time for forwarding each message to the agent is much smaller. Additionally, this feature causes each node to maintain much fewer number of forwarding pointers of mobile agents than in the traditional approach. For the third problem, each movement path of a mobile agent is replicated in an effective way to preserve the previously stated scalability of our proposed mechanism. Also, this replication much more accelerates delivering each message to the corresponding mobile agent whereas resulting in a little more agent location updating costs.

2 System Model

A distributed agent based system assumed in this paper is asynchronous: each agent has no global memory and no global clock, and is executed at its own speed and communicates with each other only through messages at finite, but arbitrary transmission delays. This system is augmented with a unreliable failure detector [4] in order to solve the impossibility problem on distributed consensus [6]. The system consists of a set of agent service nodes. Each service node supports an environment in which agents can operate safely and securely, and provides a uniform set of services that visiting agents can access its local resources in a limited way regardless of their locations. An agent is initially created on a service node, called $home\ node$ of the agent, and is given a unique identifier within the node. So, each agent can be identified as a globally unique object in the system by using the combination of its local identifier and the identifier of its home node. When an agent migrates in the system, its code and state information are captured and then transferred to the next node. After arriving at the node, the mobile agent resumes and performs its task, if needed, by interacting with other agents. In order to perform an assigned task on behalf of a user, a mobile agent a executes on a sequence of $l(l > 1)$ service nodes $I_a = [N_{home}, N_1, \cdots, N_{(l-1)}]$ according to its itinerary, which may be statically determined before the mobile agent is launched at its home node or dynamically while progressing its execution. It is assumed that communication channels support standard asynchronous message passing and are immune to partitioning, and reliable and FIFO. Agents can migrate and messages be passed along these channels. Finally, we assume that each service node has crash-failure semantics, in which they lose contents in their volatile memories and stop their executions [13].

3 Related Work

A broadcast-based mobile agent communication protocol was proposed by Murphy and Picco[11]. The protocol guarantees transparent and reliable inter-agent communication and can also provide multicast communication to a set of agents. But, to locate the message destination, it has to contact every visiting host in the network. Thus, its large traffic overhead makes broadcasts impractical in large-scale mobile agent systems.

Belle[2] proposed a hierarchical structure-based mechanism to form a location directory consisting of a hierarchy of servers. The location server at each level keeps lower-level object location information. For each object, the information is either a pointer to an entry at a lower-level location server or the agent's actual current location. However, this hierarchy cannot always be easily formed, especially in the Internet environment. Moreover, this mechanism may cause useless hops to be taken along the hierarchy.

Feng[3] introduced a mailbox-based mechanism to provide location-independent reliable message delivery. It allows messages to be forwarded at most once before they are delivered to their receiving agents. Also, the movement of agents can be separated from that of their mailboxes by determining autonomously whether each mailbox is migrated to its owner agent. However, uncertainty of message delivery to mailboxes may result in useless early pollings. On the other hand, even if urgent messages are forwarded to a mailbox on time, they can be delivered to its corresponding agent very late depending on the

agent's polling time. Moreover, whenever each mailbox moves, its new location information should be broadcasted to every node where the mailbox has visited. This may incur high traffic overhead if assuming most agents are highly mobile.

All the works stated above doesn't consider failures of forwarding nodes for mobile agents.

4 The Proposed Mechanism

4.1 Directory Service

First of all, let us define two important terms, $forwarding\ node$ and $locator$. Forwarding node of an agent is a directory service node that maintains a forwarding pointer of the agent on its storage. Thus, there may be the various number of forwarding nodes of each agent in the system according to which agent communication mechanism is used. Locator of an agent is a special forwarding node managing the identifier of the service node that the agent is currently running on. Assuming every node is failure-free, our mechanism requires only one locator for each mobile agent to address agent mobility. But, if the mechanism intends to tolerate up to $F(F \geq 1)$ node failures, $(F + 1)$ locaters of each mobile agent should exist. Thus, every service node N_i needs to keep the following data structures to enable our fault-tolerant directory service algorithm to satisfy the goal mentioned in the previous section.

• $R\text{-}Agents_i$: It is a vector which records the location information of every agent currently executing on N_i. Its element consists of three fields, $agent\_id$, l_fwdrs and $agent\_t$. l_fwdrs is a set of identifiers of agent $agent\_id$'s forwarding nodes which N_i guesses are alive. $agent\_t$ is the timestamp associated with agent $agent\_id$ when the agent is running on N_i. Its value is incremented by one whenever the agent moves to a new node. Thus, when agent $agent\_id$ migrates to N_i, N_i should inform only locators of the agent in l_fwdrs of both its identifier and $agent\_t$ so that the locaters can locate the agent.

• $AgentFPs_i$: It is a vector which maintains the location information of every mobile agent which is not currently running on N_i, but which N_i is a forwarding node of. Its element consists of five fields, $agent\_id$, $next\_n$, $agent\_t$, $manage\_f$ and $migrate\_f$. $next\_n$ is a set of identifier(s) of the node(s) which N_i thinks agent $agent\_id$ is currently running on or are the locators of the agent. $agent\_t$ is the timestamp associated with the agent when being located at the latest among the node(s). It is used for avoiding updating recent location information by older information[9]. $manage\_f$ is a bit flag indicating if N_i is a locator of agent $agent\_id$ or not. In the first case, its value is set to $true$ and otherwise, $false$. $migrate\_f$ is a bit flag designating if the agent is currently moving to another node(=$true$) or not(=$false$).

Then, we assume that the value of F is 1 in all examples shown later for explaining. First, let us see how to perform failure-free operations of the algorithm using figure 1. This figure illustrates basic agent migration procedures in order and their corresponding state changes in mobile agent location information maintained by each node when agent a moves from N_4 to N_6 via N_5 in case no node fails. In figure 1(a), agent a was running on N_4 and its locators were N_{home} and N_1 before its migration procedure initiated. On attempting to move to N_5, agent a first should send its two locators each a message $migr\_init(Id_a)$, which indicates a's migration procedure starts from now. If N_{home} and N_1 receive this message, they change the value of field $migrate\_f$ of a's record in their $AgentFPs$s from $false$ to $true$, and then acknowledge the message to N_4. This invalidation procedure must be executed to prohibit wrong message forwarding from being performed during the agent migration operation. Otherwise, the following problem might occur. If N_{home} or N_1 receives any message destined to a, it forwards the message to N_4 as it cannot recognize whether the migration procedure for a is currently being executed. Unfortunately, in this case, a may be gone from N_4 while N_4 isn't a forwarding node and so has no location information of a in $AgentFPs_4$ in our algorithm. If so, agent a cannot receive the message. Therefore, after having completed the invalidation procedure, N_4 should push agent a to N_5 and then remove the element of a from $R\text{-}Agents_4$. Then, agent a increments a's timestamp, whose value becomes 5. In this case, it has N_5 become the latest locator of a, creates and inserts a's record into $R\text{-}Agents_5$, and changes the value of field l_fwdrs of the record from $\{1\}$ to $\{5,1\}$. Afterwards, N_5 transmits two different messages to its previous locators, N_{home} and N_1, respectively. First, it sends N_{home} a message $changelm(Id_a,\{5,1\},5)$ to inform N_{home} that N_{home} is a's locator no longer and N_1 and N_5 become two locators of a from now. Thus, N_{home} updates the values of fields $next\_n$ and $agent\_t$ of a's record in $AgentFPs_{home}$ to $\{5,1\}$ and 5 using the message, and resets both fields, $manage\_f$ and $migrate\_f$, to $false$. Second, as N_1 still continues to play the role of a's locator, agent a sends a message $update(Id_a,5)$ to N_1. When receiving the message, N_1 updates the state of a's record in $AgentFPs_1$ to $(Id_a, \{5\}, 5, true, false)$ using the message. Finally, in both cases, if the messages destined to a have been buffered in their message queues during the agent mi-

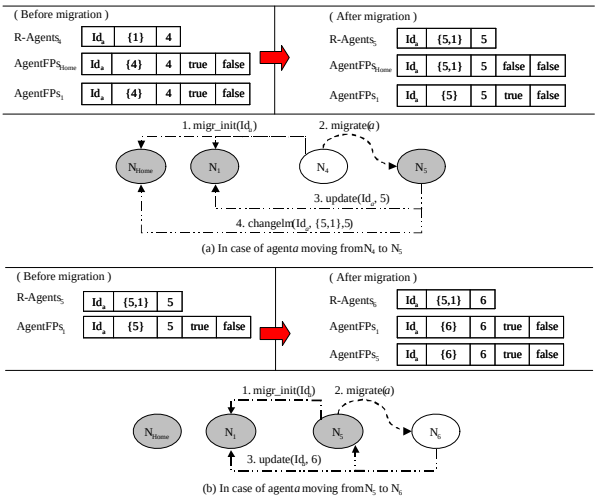


Figure 1: In case agent a moves from N_4 to N_6 on its movement path without any node failures

gration, the messages are forwarded to N_5.

Figure 1(b) shows an example that a attempts to migrate from the latest locator N_5 to N_6 after a has finished its partial task at N_5. In this case, the invalidation procedure is first executed like in figure 1(a). Then, as N_5 is a's locator, it creates and inserts a's record (Id_a, {6}, 5, $true$, $true$) into $AgentFPs_5$ and then dispatchs agent a to N_6. After that, N_6 increments a's timestamp and then saves a's record (Id_a, {5,1}, 6) into $R\text{-}Agents_5$ like in the right-hand side of this figure because a wants N_6 to be just a visiting node. Then, N_6 sends a message $update(Id_a,6)$ to a's locators, N_1 and N_5 to notify them that a's agent migration procedure has terminated. After updating a's location information, the locators forward each all the messages in their message buffers, which should be sent to the agent, to N_6. Next, we attempt to informally describe our fault-tolerant directory service algorithm in case some nodes fail using figure 2. This figure shows an example that an FN N_5 crashes while agent a migrates from N_7 to N_8. In figure 2(a), N_5 has failed before N_7 informs two locators of a, N_5 and N_1, that a attempts to move to N_8 by sending message $migr\_init(Id_a)$ to them. In this case, to keep the number of a's locators being F+1, N_7 sends message $migr\_init(Id_a)$ to N_{home}, which is currently a forwarding node of a, not its locator, to allow N_{home} to play a role of a's locator from now. On receiving the message, N_{home} changes the values of two fields $manage\_f$ and $migrate\_f$ of a's record in $AgentFPs_{home}$ to $true$ and $false$. Then, it sends an acknowledgement message to N_7. After finishing all the invalidation procedures, N_7 enables agent a to migrate to N_8, and then N_8 sends each a message $update(Id_a,8)$ to a's current locators, N_1 and N_{home}, to inform them of the termination of a's agent migration procedure. In this case, the two locators update the state of a's record in their $AgentFPs$s to (Id_a, {8}, 8, $true$, $false$) respectively.

Figure 2(b) indicates an example that N_5 has failed after agent a migrates to N_8 and before N_8 sends message $update(Id_a,8)$ to the current locators of a, N_5 and N_1, for notifying them of the completion of a's movement process. In this case, N_8 sends message $update(Id_a,8)$ to N_{home} to force the number of a's locators to be still F+1. When it receives the message, N_{home} has the state of a's record in $AgentFPs_{home}$ become (Id_a, {8}, 8, $true$, $false$) and then plays a role of a's locator from now on.

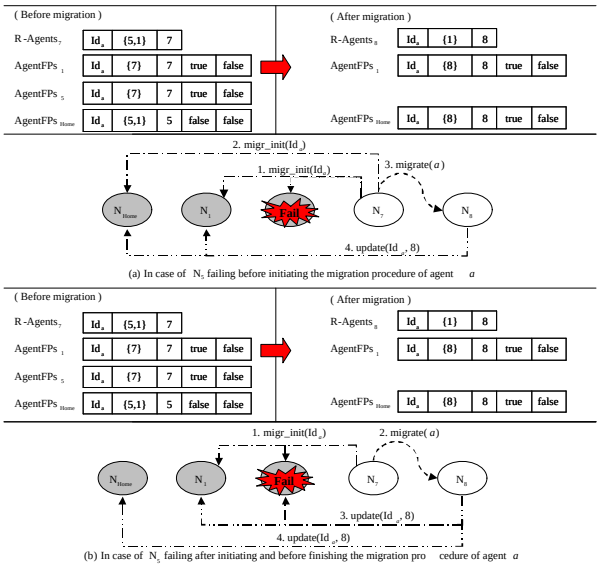


Figure 2: In case an FN N_5 fails while a moving from N_7 to N_8

4.2 Message Delivery

To support reliable message delivery despite forwarding node failures, our mechanism requires the following agent location cache of every node N_i.

• $L\text{-}Cache_i$: It is a vector which temporarily keeps location information of each mobile agent which agents running on N_i communicate with. Its element consists of three fields, $agent\_id$, $fwdrs$ and $agent\_t$. $fwdrs$ is a set of identifiers of the nodes which N_i guesses are locators of agent $agent\_id$. Thus, to send messages to agent $agent\_id$, an agent on N_i forwards them to the latest among live locaters in $fwdrs$. If there is no live locater in $fwdrs$, the messages are sent to the home node of agent

$agent\_id$. $agent\_t$ is the timestamp assigned to agent $agent\_id$ when the agent registered with the latest among all locators in $fwdrs$.

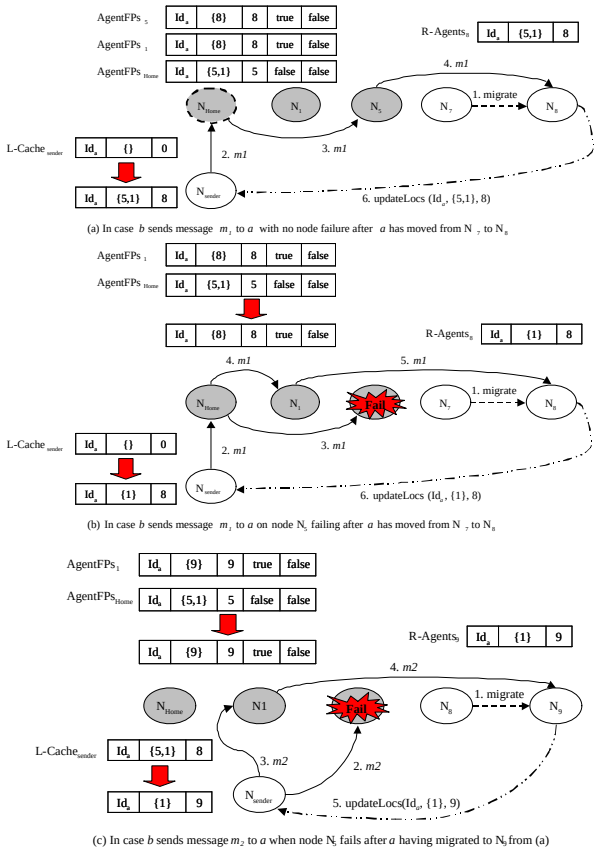
To illustrate how our message delivery algorithm based on the fault-tolerant directory service achieves the goal, figure 3 shows an example that agent b sends two messages, m_1 and m_2, to agent a in this order while a is moving from N_7 to N_9 according to its itinerary. In figure 3(a), agent b at N_{sender} attempts to deliver a message m_1 to agent a with no node failure after a has migrated from N_7 to N_8. In this case, as N_{sender} maintain no location information for a in its agent location cache $L\text{-}Cache_{sender}$, it creates and inserts a's element (Id_a, {}, 0) into $L\text{-}Cache_{sender}$. After that, it sends the message m_1 to N_{home}. On receiving the message, N_{home} retrieves a's element from $AgentFPs_{home}$. In this case, as the value of the bit flag $manage\_f$ in the element is $false$, N_{home} isn't a's locator. Thus, it consults the element and forwards the message m_1 to the next forwarder N_5 that N_{home} guesses agent a is currently running on or that is the latest locator of a. On the receipt of the message, N_5 obtains a's element from $AgentFPs_5$ and then checks the flag $manage\_f$ in the element. In this case, N_5 is a's locator because the value of the flag is $true$. Also, as the value of the second flag $migrate\_f$ is $false$, it directly forwards the message to a's currently running node N_8 by consulting the element. After receiving the message, N_8 sends N_{sender} a message $updateLocs$(Id_a, {5,1}, 8) containing the identifiers of a's current locators(=N_5,N_1) and timestamp(=8) because N_{sender} doesn't correctly know which nodes are a's current locators. Receiving the message, N_{sender} updates a's element in $L\text{-}Cache_{sender}$ using the message like in this figure. Thus, when agent b communicates with agent a from now, N_{sender} can directly send messages to the latest locator of a, N_5, with the assumption of N_5 never failing.

Unlike figure 3(a), figure 3(b) shows the case that the latest locator of a, N_5, fails after a has moved from N_7 to N_8. In this case, N_{sender} creates and keeps a's element on $L\text{-}Cache_{sender}$, and then forwards the message m_1 to N_{home}. Then, N_{home} attempts to transmit the message m_1 to the first latest locator N_5 by consulting a's element. However, as N_5 has failed, N_{home} cannot receive any acknowledgment message from N_5. Thus, N_{home} chooses the second latest locator of a, N_1, as the next forwarding node and sends the message m_1 to N_1. On receiving the message, N_1 finds a's element from $AgentFPs_1$ and then is able to know it is the locator of a. Thus, it sends the message to N_8 where a is currently running by looking up the second field of the element. When receiving the message, N_8 delivers it to agent a. Simultaneously, as N_8 becomes aware of N_5's failure, it forces N_{home} to play a role of a's locator by sending message $update$(Id_a,8) to N_{home} like in figure 2(b). In this case, N_{home} changes the state of a's record in $AgentFPs_{home}$ to (Id_a, {8}, 8, $true$, $false$) like in figure 3(b). Then, N_8 updates the second field of a's element in $R\text{-}Agents_8$ from {5,1} to {1} and then informs N_{sender} of this update by sending a message $updateLocs$(Id_a, {1}, 8). Receiving the message, N_{sender} updates a's element in $L\text{-}Cache_{sender}$ using the message like in this figure.

Figure 3(c) illustrates a different failure case that a's latest locator, N_5, crashes after a has migrated to N_9 from figure 3(a). In this case, N_{sender} first finds a's element (Id_a, {5,1}, 8) from $L\text{-}Cache_{sender}$, and then attempts to transmit the second message m_2 to the latest locator N_5. But, N_{sender} recognizes N_5's failure because of receiving no acknowledgment message from it. Thus, N_{home} forwards message m_2 to the second latest locator of a, N_1, by looking up a's element in $L\text{-}Cache_{sender}$. On receiving the message, N_1 consults the second field of a's element in $AgentFPs_1$ and then sends the message m_2 to a's currently running service node, N_9. After having received the message, N_9 can recognize N_5 has crashed. Thus, it allows N_{home} to be a locator for a by sending message $update$(Id_a,9) in the same manner as in figure 3(b). In this case, values of all the fields of a's element in $AgentFPs_{home}$ becomes (Id_a, {9}, 9, $true$, $false$) like in figure 3(c). Then, N_9 changes the state of the corresponding element in $R\text{-}Agents_9$ to (Id_a, {1}, 9) and sends a message $updateLocs$(Id_a, {1}, 9) to N_{sender}.

5 Conclusion

In this paper, we presented a new fault-tolerant and scalable mobile agent communication mechanism based on forwarding pointers to address both agent mobility and directory service node failures. The mechanism enables each mobile agent to keep its forwarding pointer only on the small number of its visiting nodes in an autonomous manner. This feature can significantly shorten the path for routing each message to the corresponding mobile agent. Thus, the average message delivery time is much more reduced. Also, our mechanism requires a very small size of storage per each directory service node compared with the previous ones because the amount of agent location information the node keeps significantly decreases in the mechanism. Moreover, the mechanism allows $F + 1$ locators of each agent to know its current location for tolerating a maximum of F failures



(a) In case b sends message m_1 to a with no node failure after a has moved from N_7 to N_8

(b) In case b sends message m_1 to a on node N_5 failing after a has moved from N_7 to N_8

(c) In case b sends message m_2 to a when node N_1 fails after a having migrated to N_8 from (a)

Figure 3: An example agent b at N_{sender} sends two messages m_1 and then m_2 to agent a

of forwarding nodes. This behavior enables the inherent scalability of the proposed mechanism to be reasonably preserved. Also, our effective agent location cache much more speeds up the delivery of each message to the final destination even in case of node failures.

Choosing the proper forwarding nodes of each mobile agent among its visiting nodes and the optimal degree of redundancy of forwarding pointers is very important for our mechanism to work effectively. For this, we are currently implementing the mechanism in a lightweight mobile agent platform and will perform various experiments to evaluate its scalability with respect to the two performance factors.

References:

[1] P. Bellavista, A. Corradi and C. Stefanelli. The Ubiquitous Provisioning of Internet Services to Portable Devices. *IEEE Pervasive Computing*, Vol. 1, No. 3, pp. 81-87, 2002.

[2] W. Belle, K. Verelst and T. D'Hondt. Location transparent routing in mobile agent systems merging name lookups with routing. *In Proc. of the 7th IEEE Workshop on Future Trends of Distributed Computing Systems*, pp. 207-212, 1999.

[3] J. Cao, X. Feng, J. Lu and S. Das. Mailbox-based scheme for mobile agent communications. *IEEE Computer*, Vol. 35, No. 9, pp. 54-60, 2002.

[4] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43:225-267, 1996.

[5] J. Desbiens, M. Lavoie and F. Renaud. Communication and tracking infrastructure of a mobile agent system. *In Proc. of the 31st Hawaii International Conference on System Sciences*, Vol 7., pp. 54-63, 1998.

[6] M. J. Fischer, N. A. Lynch and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32:374-382, 1985.

[7] M. Fukuda, Y. Tanaka, N. Suzuki, L.F. Bic and S. Kobayashi. A Mobile-Agent-Based PC Grid. *In Proc. of the Fifth Annual International Workshop on Active Middleware Services*, pp. 142-150, 2003.

[8] D. Lange and M. Oshima. *Programming and Deploying Mobile Agents with Aglets*. Addison-Wesley, 1998.

[9] L. Moreau. Distributed Directory Service and Message Router for Mobile Agents. *Science of Computer Programming*, Vol. 39, No. 2-3, pp. 249-272, 2001.

[10] L. Moreau. A Fault-Tolerant Directory Service for Mobile Agents based on Forwarding Pointers. *In Proc. of The 17th ACM Symposium on Applied Computing*, pp. 93-100, 2002.

[11] A. L. Murphy and G. P. Picco. Reliable Communication for Highly Mobile Agents. *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 5, No. 1, pp. 81-100, 2002.

[12] ObjectSpace. Voyager. *http://www.objectspace.com/*.

[13] R. D. Schlichting and F. B. Schneider. Fail-stop processors: an approach to designing fault-tolerant distributed computing systems. *ACM Transactions on Computer Systems*, 1:222-238, 1985.