An Adaptive Communication Mechanism for Highly Mobile Agents

JinHo Ahn

Dept. of Computer Science, Kyonggi University
San 94-6 Yiuidong, Paldalgu, Suwonsi Kyonggido 442-760, Republic of Korea
jhahn@kyonggi.ac.kr

Abstract. Agent mobility causes reliable inter-agent communications to be more difficult to achieve in distributed agent based systems. To solve this issue, three representative agent tracking and message delivery mechanisms, broadcast-based, home-based and forwarding pointer-based, were previously proposed. However, due to their respective drawbacks, none of them is suitable for efficient delivery of messages to highly mobile agents, which move frequently between service nodes. This paper introduces an adaptive forwarding pointer-based agent tracking and message delivery mechanism to alleviate their disadvantages. The proposed mechanism allows each mobile agent to autonomously leave tails of forwarding pointers on some few of its visiting nodes depending on its preferences. Thus, it is more efficient in terms of message forwarding and location management than the previous forwarding pointer-based one. Simultaneously, it considerably reduces the dependency on the home node in agent location updating and message delivery compared with the home-based mechanism.

1 Introduction

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[4]. Due to its beneficial characteristics, i.e., dynamicity, asynchronicity and autonomy, it has been primarily used as an enabling programming paradigm for developing distributed computing infrastructures in various application fields such as e-commerce, telecommunication, ubiquitous computing, active networks and the like[2,4]. However, as the size of these fields is rapidly increasing, several research issues related to the mobile agent technology such as communication, security, dynamic adaptation, etc., should be reconsidered to be suitable for their scale. Among them, it is most important to enhance the performance of the agent communication in Internet-scale infrastructures. For this purpose, some effective and efficient inter-agent communication mechanism is required in distributed agent-based systems. Agent mobility may lead to the loss of messages being destined to an agent on its migration. Thus, it causes reliable inter-agent communications to be not easy to achieve in the distributed agent based systems. Especially, guaranteeing the delivery of messages

© Springer-Verlag Berlin Heidelberg 2004

to highly mobile agents, which move frequently among service nodes, is a more challenging problem, which this paper attempts to address. To consider the agent mobility, three representative agent tracking and message delivery mechanisms, broadcast-based, home-based and forwarding pointer-based, were previously proposed. The broadcast-based mechanism[7] guarantees transparent and reliable inter-agent communication and can also provide multicast communication to a set of agents. But, to locate the message destination, the mechanism has to contact every visiting node in the network. Thus, its large traffic overhead makes broadcasts impractical in large-scale distributed agent systems.

The home-based mechanism[5] is borrowed from the idea of Mobile IP[8]. It requires that each mobile node has a home node, and forces the mobile node to register its current temporary address, called care-of-address, with its home node whenever it moves. Thus, when some messages are sent to a mobile node currently located at a foreign network, the messages are first directed to its home node, which forwards them to the mobile one. This mechanism is simple to implement and results in little mobile node locating overhead. However, it is unsuitable for highly mobile agents in distributed agent based systems because every agent location updating and message delivery are all performed around the home agent, which introduces centralization. Moreover, the Mobile IP generally assumes each mobile node's home node is a static one whereas distributed agent based systems don't have this assumption, i.e., the home node may be disconnected from the network. Thus, this mechanism cannot address the disconnection problem.

In the forwarding pointer-based mechanism[3,6], each node on a mobile agent's movement path keeps a forwarding pointer to the next node on the path. Thus, if a message is delivered to an agent not being at the home node, the message must traverse a list of forwarding nodes. Thus, this mechanism can avoid performance bottlenecks of the global infrastructure, and therefore improve its scalability, particularly in large-scale distributed agent-based systems, compared with the home based one. Additionally, even if a home node is disconnected from the rest of the network, the forwarding pointer based mechanism allows agents registering with the node to communicate with other agents. However, as highly mobile agents leads to the length of their chains of pointers being rapidly increasing, its message forwarding overhead may be significantly larger. Furthermore, the number of forwarding pointers each service node needs to keep on its storage may exponentially increase if a large number of mobile agents are running in the systems. In a previous work[6], a type of update message called $inform$ message was introduced to include an agent's current location for shortening the length of trails of forwarding pointers. In this case, a node that receives the message is allowed to update its table if the received information is more recent than the one it had. However, it introduces no concrete and efficient solutions for this purpose, for example, when update messages should be sent, and which node they should be sent to.

Therefore, we observe these respective drawbacks of the three previous mechanisms may be critical obstacles to efficient communications between highly mobile agents in large-scale distributed agent systems. This paper introduces an adaptive forwarding pointer-based agent tracking and message delivery mechanism to avoid their disadvantages. The proposed mechanism allows each mobile agent to autonomously leave trails of forwarding pointers only on some few of its visiting nodes depending on its preferences such as location updating and message delivery costs, security, network latency and topology, communication patterns, etc.. Thus, it is more efficient in terms of message delivery and location management than the previous forwarding pointer-based one. Additionally, it considerably decentralizes the role of the home node in agent location updating and message delivery. This feature alleviates the two problems of the home-based mechanism.

Due to space limitation, our system model, formal descriptions and correctness proof of the proposed mechanism, and related work are all omitted. The interested reader can find them in [1].

2 The Adaptive Communication Mechanism

As mentioned in section 1, the proposed adaptive communication mechanism is designed to have the following features unlike previous ones.
• Require small size of storage for location management per service node.
• Result in low message forwarding overhead.
• Reduce considerably the dependency on home node in agent location updating and message delivery.

First of all, let us define two important terms, *forwarder* and *locator*. Forwarder of an agent means a service node keeping a forwarding pointer of the agent on its storage. Thus, depending on the behavior of agent communication mechanisms, there may exist various number of forwarders of each agent in the system. Locator of an agent is the forwarder managing the identifier of the node where the agent is currently located. In this paper, it is assumed that there is only one locator in the system. To satisfy all the three requirements, our adaptive mechanism forces only some among all visiting nodes of each agent to be forwarders. This behavior can considerably reduce both the amount of agent location information each node needs to maintain and the delivery time of each message because the length of its forwarding path may be much more shortened. Also, since there exist multiple forwarders, not only one, in this mechanism, the home node centralization can be avoided. But, as a part of the vising nodes are forwarders in this mechanism, a new method is required to consistently manage location of each agent and enable each sent message to be delivered to the agent despite its migrations unlike in the previous forwarding pointer-based one. In the following, two components of the proposed mechanism, agent location management and message delivery algorithms, are explained in detail respectively.

Agent Location Management. For the agent location management algorithm, every node N_i should maintain the following data structures.

• $RunningAgents_i$: A table for saving location information of every agent currently running on N_i. Its element is a tuple ($agent\_id$, $locmngr\_n$, $agent\_t$). $locmngr\_n$ is the identifier of agent $agent\_id$'s locator. $agent\_t$ is the timestamp associated with agent $agent\_id$ when the agent is located at N_i. Its value is incremented by one every time the corresponding agent migrates. Thus, when agent $agent\_id$ migrates to N_i, N_i should inform $locmngr\_n$ of both its identifier and $agent\_t$ of the agent so that $locmngr\_n$ can locate the agent.
• $AgentLocs_i$: A table for saving location information of every mobile agent which is not currently running on N_i, but of which N_i is a forwarder. Its element is a tuple ($agent\_id$, $destination\_n$, $agent\_t$, $ismanaging\_f$, $ismigrating\_f$). $destination\_n$ is the identifier of the node where N_i knows agent $agent\_id$ is currently located and running. $agent\_t$ is the timestamp associated with the agent when the agent is located at node $N_{destination\_n}$. It is used for avoiding updating recent location information by older information[6]. $ismanaging\_f$ is a bit flag indicating whether N_i is agent $agent\_id$'s locator or not. In the first case, its value is $true$ and otherwise, $false$. $ismigrating\_f$ is a bit flag designating if the agent is currently migrating to another node(=$true$) or not(=$false$).

The algorithm for managing each agent's location on its migration is informally described using figure 1. This figure shows message interactions between nodes occurring in a's location updating and location information maintained by each node while migrating from its home node to N_1 through N_5. In figure 1(a), a is created on N_{home} and then an element for a (id_a, $home$, 0) is saved into $RunningAgents_{home}$ in the first step. If a attempts to move to N_1 after having performed its partial task, in the second step, it inserts into $AgentLocs_{home}$ a's element (id_a, 1, 1, $true$, $true$) indicating N_{home} is a's locator and a is currently moving to N_1. Then, N_{home} dispatchs the agent with the identifier of the node and a's timestamp to N_1. When receiving these, N_1 increments the timestamp by one. In this case, as a wants N_1 to be its locator, it inserts a's location information (id_a, 1, 1) into $RunningAgents_1$ in the third step. At the same time, N_1 sends N_{home} a message *changelmngr* including a's timestamp in order to inform N_{home} that N_1 is a's locator from now. On receiving the message, N_{home} updates a's location information on $AgentLocs_{home}$ using the message and sets two fields of a's element, $ismanaging\_f$ and $ismigrating\_f$, all to $false$. If the messages destined to a have been buffered in N_{home}'s message queue due to the migration, they are transmitted to N_1. When a attempts to migrate to N_2 after a has performed a part of its task in the forth step, N_1 puts a's element (id_a, 2, 2, $true$, $true$) into $AgentLocs_1$ and then dispatchs agent a to N_2. In this case, N_2 increments a's timestamp by one and then inserts a's element (id_a, 1, 2) into $RunningAgents_2$ like in the fifth step because a wants N_2 to be just a visiting node. Also, the node registers a's current location with its locator N_1 by sending a message *update* including the timestamp to N_1. If there are any messages sent to the agent in the queue of N_1, they are forwarded to N_2.

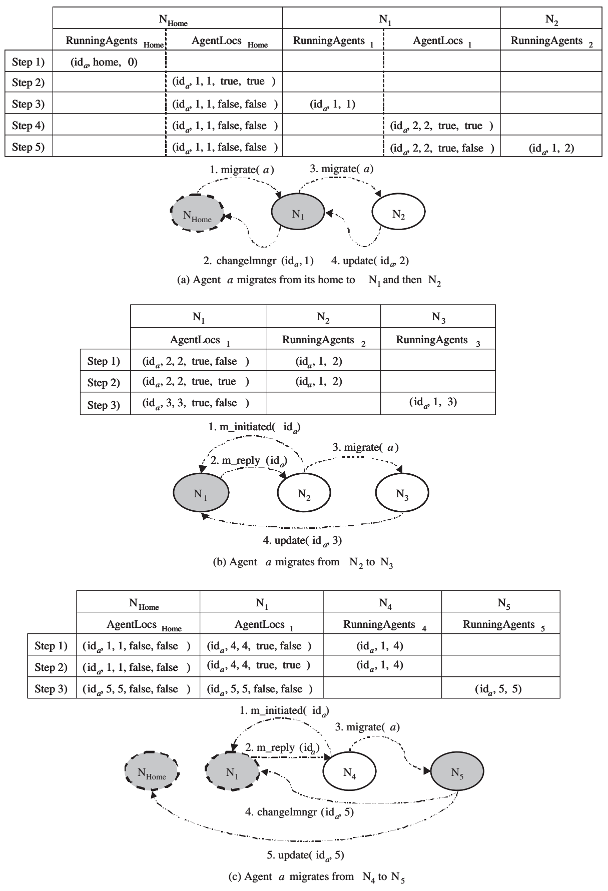
Figure 1(b) illustrates that agent a moves from N_2 to N_3. In this example, N_2 first sends N_1 a message *m_initiated* indicating that a's migration process begins from now. When receiving the message, N_1 sets one field of a's element *ismigrating_f* to *true* in the second step and then send a message *m_reply* to N_2. Suppose the migration is started without the execution of this invalidation procedure. If N_1 receives any message destined to a in this case, it forwards the message to N_2 because it doesn't know whether the migration continues to be executed. But, neither a may be currently running on N_2 nor N_2 keep a's location information on $AgentLocs_2$ because N_2 isn't a's forwarder. In this case, the message cannot be delivered to a. Therefore, N_2 has to push a to N_3 after having received the message *m_reply* from a's locator, and then remove a's element from $RunningAgents_2$. Afterwards, a's visiting node N_3 increments a's timestamp and saves a's element $(id_a, 1, 3)$ into $RunningAgents_3$ in the third step, and then sends a message *update* to N_1. On the receipt of the message, N_1 updates a's element in $AgentLocs_1$ to $(id_a, 3, 3, true, false)$ using the message.

Figure 1(c) shows an example that N_5 becomes a's locator when agent a moves from N_4 to N_5. In this case, after N_5 creates a's location information $(id_a, 5, 5)$ and inserts it into $RunningAgents_5$ in the third step, the node sends N_1 a message *changelmngr* for notifying the previous locator N_1 that N_5 is a's locator from now. Also, a attempts to register its current location with N_{home} in order to reduce the message delivery time incurred when another agent initially sends a message to a via N_{home}. For this purpose, N_5 sends a message *update* to N_{home}. If a recognizes this consideration helps no performance improvement, it doesn't perform the home node update procedure. After that, N_{home} and N_1 update a's location information respectively like in the third step.

Message Delivery. For the message delivery algorithm, every node N_i should contain an agent location cache, $ALocsCache_i$, as follows.

• $ALocsCache_i$: A cache for temporarily storing location information of each mobile agent which agents running on N_i communicate with. Its element is a tuple $(agent\_id, forward\_n, agent\_t)$. $forward\_n$ is the identifier of the node where N_i knows agent *agent_id* is currently located and running. Thus, when attempting to deliver messages to agent *agent_id*, each agent on N_i forwards them to $forward\_n$ regardless of whether this address is outdated. *agent_t* is the timestamp assigned to agent *agent_id* when the agent was located at node $N_{forward\_n}$.

We intend to use an example in figure 2 to clarify the algorithm to enable every sent message to be reliably delivered to its target agent despite agent migrations. This example illustrates agent b sends three messages, $msg1$, $msg2$ and $msg3$ to agent a in this order while a is migrating from its home node to N_1 through N_6 according to its itinerary. In figure 2(a), after a has moved from N_{home} to N_2, b at N_{sender} will deliver the first message $msg1$ to a. In this case, N_{sender} has no location information for a in its location cache $ALocsCache_{sender}$. Thus, N_{sender} creates and saves a's element $(id_a, home, 0)$ into $ALocsCache_{sender}$. Then, it sends the message $msg1$ to N_{home}. Receiving the message, N_{home} retrieves a's element from $AgentLocs_{home}$. In this case, as the value of the bit flag *ismanaging* in the element is $false$, N_{home} isn't a's



	N_{Home}		N_1		N_2
	RunningAgents $_{Home}$	AgentLocs $_{Home}$	RunningAgents $_1$	AgentLocs $_1$	RunningAgents $_2$
Step 1)	(id_a, home, 0)				
Step 2)		(id_a, 1, 1, true, true)			
Step 3)		(id_a, 1, 1, false, false)	(id_a, 1, 1)		
Step 4)		(id_a, 1, 1, false, false)		(id_a, 2, 2, true, true)	
Step 5)		(id_a, 1, 1, false, false)		(id_a, 2, 2, true, false)	(id_a, 1, 2)

(a) Agent a migrates from its home to N_1 and then N_2

	N_1	N_2	N_3
	AgentLocs $_1$	RunningAgents $_2$	RunningAgents $_3$
Step 1)	(id_a, 2, 2, true, false)	(id_a, 1, 2)	
Step 2)	(id_a, 2, 2, true, true)	(id_a, 1, 2)	
Step 3)	(id_a, 3, 3, true, false)		(id_a, 1, 3)

(b) Agent a migrates from N_2 to N_3

	N_{Home}	N_1	N_4	N_5
	AgentLocs $_{Home}$	AgentLocs $_1$	RunningAgents $_4$	RunningAgents $_5$
Step 1)	(id_a, 1, 1, false, false)	(id_a, 4, 4, true, false)	(id_a, 1, 4)	
Step 2)	(id_a, 1, 1, false, false)	(id_a, 4, 4, true, true)	(id_a, 1, 4)	
Step 3)	(id_a, 5, 5, false, false)	(id_a, 5, 5, false, false)		(id_a, 5, 5)

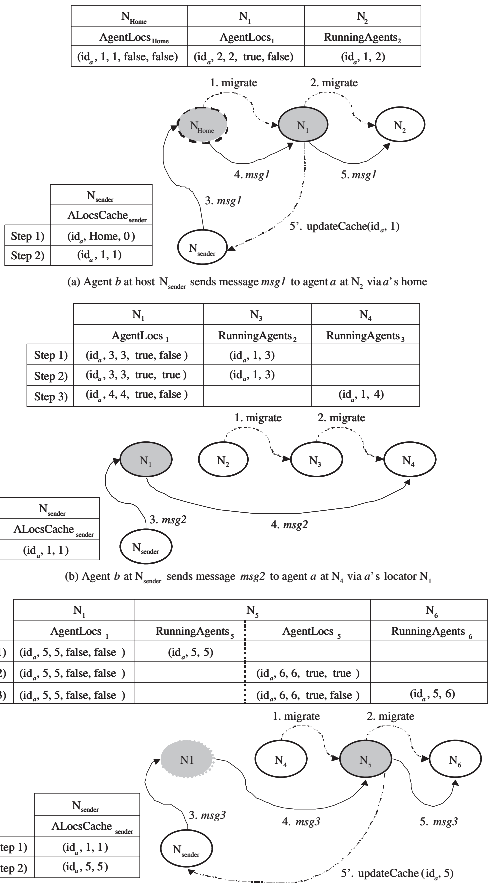
(c) Agent a migrates from N_4 to N_5

Fig. 1. An example of agent a's location updating on its migration according to its itinerary

locator. Thus, it consults the element and forwards the message $msg1$ to the next forwarder N_1. On the receipt of the message, N_1 obtains a's element from $AgentLocs_1$ and then checks the flag *ismanaging* in the element. In this case, N_1 is a's locator because the value of the flag is *true*. Also, as the value of the second flag *ismigrating* is $false$, it forwards the message to a's currently running node N_2 by consulting the element. At the same time, as N_{sender} has the outdated identifier of a's locator, N_1 sends N_{sender} a message *updateCache* containing the identifier of a's current locator($=N_1$) and timestamp($=2$) in figure 2(a). When receiving the message, N_{sender} updates a's element in $ALocsCache_{sender}$ using the message in the second step.

Figure 2(b) shows that a migrates from N_2 to N_4 and then b at N_{sender} sends the second message $msg2$ to a. In this case, N_{sender} finds a's element from $ALocsCache_{sender}$ and then forwards $msg2$ to N_1. On the receipt of *msg*2, N_1 can see that it is *a*'s current locator because the value of the bit flag *ismanaging* of *a*'s element in $AgentLocs_1$ is *true*. Also, as the value of the flag *ismigrating* is *false*, N_1 can sends *msg*2 to *a*'s currently running node(=N_4). At this time, as *b* knows the identifier of *a*'s current locator, N_1 doesn't sends any message *updateCache* to *b*.

Figure 2(c) illustrates that after *a* has migrated from N_4 via the next



N_{Home}	N_1	N_2
$AgentLocs_{Home}$	$AgentLocs_1$	$RunningAgents_2$
(id_a, 1, 1, false, false)	(id_a, 2, 2, true, false)	(id_a, 1, 2)

	N_{sender}
	$ALocsCache_{sender}$
Step 1)	(id_a, Home, 0)
Step 2)	(id_a, 1, 1)

(a) Agent *b* at host N_{sender} sends message *msg1* to agent *a* at N_2 via *a*'s home

	N_1	N_3	N_4
	$AgentLocs_1$	$RunningAgents_2$	$RunningAgents_3$
Step 1)	(id_a, 3, 3, true, false)	(id_a, 1, 3)	
Step 2)	(id_a, 3, 3, true, true)	(id_a, 1, 3)	
Step 3)	(id_a, 4, 4, true, false)		(id_a, 1, 4)

N_{sender}
$ALocsCache_{sender}$
(id_a, 1, 1)

(b) Agent *b* at N_{sender} sends message *msg2* to agent *a* at N_4 via *a*'s locator N_1

	N_1	N_5		N_6
	$AgentLocs_1$	$RunningAgents_5$	$AgentLocs_5$	$RunningAgents_6$
Step 1)	(id_a, 5, 5, false, false)	(id_a, 5, 5)		
Step 2)	(id_a, 5, 5, false, false)		(id_a, 6, 6, true, true)	
Step 3)	(id_a, 5, 5, false, false)		(id_a, 6, 6, true, false)	(id_a, 5, 6)

	N_{sender}
	$ALocsCache_{sender}$
Step 1)	(id_a, 1, 1)
Step 2)	(id_a, 5, 5)

(c) Agent *b* at N_{sender} sends message *msg3* to agent *a* at N_6 via *a*'s forwarder N_1

Fig. 2. An example of agent *b* sending three messages *msg*1, *msg*2 and *msg*3 to agent *a* in order while *a* migrating to several nodes according to its itinerary

locator N_5 to N_6, *b* transmits the third message *msg*3 to *a*. In this figure, when the migration has been completed, three nodes N_1, N_5, N_6 have each *a*'s location information like in the step 3. First, N_{sender} sends *msg*3 to N_1 by consulting *a*'s element in $ALocsCache_{sender}$. In this case, as the value of the flag *ismanaging* of *a*'s element in $AgentLocs_1$ is *false*, N_1 forwards *msg*3 to the next forwarder N_5. When N_5 receives the message, it transmits the message to N_6 because it recognizes that it is a's locator and a isn't currently migrating. Concurrently, N_5 informs N_{sender} that a's current locator is N_5 by sending a message *updateCache* to N_{sender}.

3 Conclusion

In this paper, an adaptive agent communication mechanism was introduced to considerably reduce both the amount of agent location information maintained by each service node and the delivery time of each message while reducing the dependency on the home node. To achieve this goal, this mechanism forces only a small part of all visiting nodes of each agent to becomes forwarders. In this mechanism, each agent should register its location with its current locator on every migration until it arrives at the next locator of the agent. This method may result in slightly higher location update cost per agent migration compared with that of the previous forwarding pointer-based mechanism. However, if each agent determines some among its visiting nodes as forwarders by properly considering several performance factors, the gap between the two costs may be almost negligible. Moreover, the mechanism allows the identifier of each agent's locator to be kept on the agent location cache of a node. This behavior highly reduces the cache update frequency of the node compared with the previous mechanism.

Future work is focused on selecting appropriate forwarders of each agent according to the changes which the agent senses in its environment related to location updating and message delivery costs, security policies, network latency and topology, communication patterns.

References

1. J. Ahn. Adaptive Communication Mechanisms for Mobile Agents. *Technical Report KGU-CS-03-050*, Kyonggi University, 2003.
2. 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.
3. 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.
4. A. Fuggetta, G.P.Picco and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp. 342-361, 1998.
5. D. Lange and M. Oshima. *Programming and Deploying Mobile Agents with Aglets.* Addison-Wesley, 1998.
6. L. Moreau. Distributed Directory Service and Message Router for Mobile Agents. *Science of Computer Programming*, Vol. 39, No. 2-3, pp. 249-272, 2001.
7. 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.
8. C. Perkins, IP Mobility Support. *RFC 2002*, October 1996.