Inter-Agent Communication Protocol for Adaptive Agents

Sreekanth V., S. Ramchandram, and A. Govardhan

***Abstract*—Mobile Agents no longer is limited to simple message communication but finds its applications in applications like distributed computing and network monitoring systems. The two key features expected from mobile agents are Location Transparency & Distributed Decision Making. Location Transparency is achieved through Inter-Agent communication protocol by ensuring the message is delivered to the remote host irrespective of its location on a network. At the same time agents can be programmed with limited amount intelligence so that the agents can take decisions on their own and react to certain situations while they are touring a remote host without depending on the instructions from the agent server. This paper explains how adaptive agents capable of performing multiple tasks on a single visit to remote host can communicate with other agents attached to the same or a different agent server. The complete protocol is explained along with each scenario based on the agent state on a host or a remote server. The paper also explains how agents can take decisions on their own without involving the agent servers.**

***Index Terms*—Mobile agents, agent communication, JADE agents, adaptive agents, distributed decision making.**

I. Introduction

Mobile agents are designed to perform simple tasks or collect a bid/message from a remote system and its features like mobility, autonomy and size bring several advantages to mobile agents and find its applications in many applications related to distributed computing and e-business applications. However, the current day expectations from mobile agents are far beyond simple tasks and communication. As the size of the agent and network bandwidth is not constraints anymore, the agents can be programmed with limited intelligence to take decisions on their own. This feature would further be transformed into a major utility by mobile agents called Distributed Decision Making. Decision Making is the sole responsibility of the monitoring system and adaptive agents were simply used to communicate the status. Now, with the agents having the capability to make decisions, the agents need not wait for the monitoring system to give instructions and the agents can by themselves react to certain situations and take decisions.

The other feature of major interest in this paper is Location Transparency of mobile agents. Location transparency can be achieved if the agents irrespective of their state and location can perform a given task and communicate on its own. The agents when on a remote networks should be able to communicate with the local agents and the required information. The example is a weather forecasting application on a mobile phone which is designed to show local weather of the location it's residing in. As the user moves from one location to the other, the agent should be able to detect its position through communicating with other trusted local agents or using Global Positioning System (GPS) and show the local weather rather than the weather from the location which the user configures.

The other example where location transparency of agents finds a good implementation is a simple search engine. The search results by the engine make more sense to the user who is trying to find a local restaurant or gas station if the mobile application can sort the search by location rather than the static search. The current location is always retained by the search engine and can be updated manually by the user when changes his location. The Fig. 1 shows how the search engine retains its location and the current location is used for the search. Both the examples would need the agent to communicate with other local agents and get the data through location transparency. Inter-agent communication requires a mobile agent platform that can be used to implement agents and set up inter agent communication protocol. Jade [1] is a convenient tool to be used for setting up the agent platform.

Fig. 1. Location services on search engines

II. Modeling Adaptive Agents

Adaptive Agents are programmed with intelligence to carry multiple operations during a single visit [2]. The agents can further programmed to perform limited action at a host as

Manuscript received January 14, 2013; revised April 2, 2013.

V. Sreekanth is with Jawaharlal Nehru Technological University Hyderabad, Hyderabad, AP 500085 India (e-mail: mail4sreekanth@gmail.com).

S. Ramchandram is with University College of Engineering, Osmania University and also chairman, BOE, Department of Computer Science & Engineering, Osmania University, Hyderabad, AP. 500007 (e-mail: schandram@gmail.com).

A. Govardhan is with Computer Science & Engineering and Director of Evaluations at Jawaharlal Nehru Technological University Hyderabad, Hyderabad. AP 500085. India (e-mail: govardhan_cse@yahoo.co.in).

need be. The structure of a Dynamic Adaptive Mobile Agent is represented in the Fig. 2. The Data, Execution State and Code section of the agent remains similar to a conventional mobile agent. However, the message section is configured as a dynamic queue. In the structure shown, the agent dynamically stores the data from 4 ports on host H1, 2 ports as it visits Host 2, 3 hosts when it visits host 3 and so on.

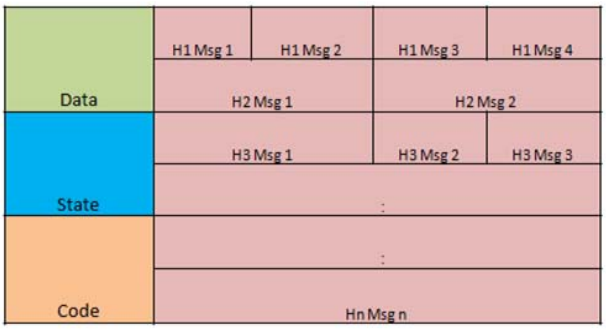

Fig. 2. Structure of adaptive agent

A. Inter Agent Communication

Agents need to communicate to facilitate faster decision making [3]. The modern day mobile agents are subjective, that are designed and programmed to accomplish a task. Agents communicate through agent servers. Though the virtual path of the agents appears to communicate to each other, the physical path of the message is always through an agent server [4]. Agents pass the message to its home agent server, which in turn identifies the address of the remote agent server and pass the message. The responsibility of the remote agent server is to pass the message to the mobile agent. The physical and virtual communication path of messages between agents is depicted in the Fig. 3.

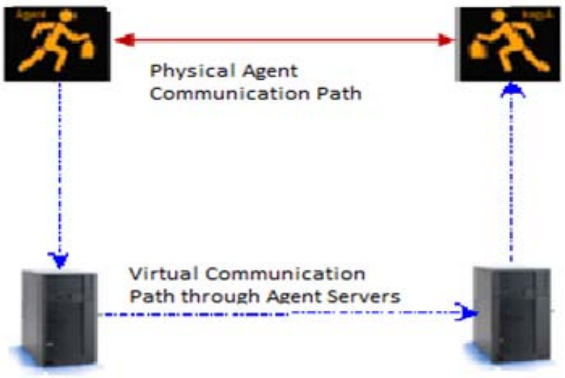

Fig. 3. Inter-agent communication

In order to implement the location-transparency and the reliability of message delivery, the peer agent must be located by some mechanism that maps an agent's unique name onto its current location; this mechanism is called Address Resolution. In a general way, addressing modes which are usually used include: searching mode, forwarding mode and registration mode [5], [6] and so on.

- Searching Mode: An agent usually is dispatched to visit all possible hosts or broadcasts the message to all the hosts on a network to search the target agent. This overhead is unaffordable when considering a large enterprise networks.
- Forwarding Mode: Because the agent leaves its migration track on the hosts when it passed by, its current location should be attained via following its trail. If the trail information is lost or if one of the hosts is down, the target agent would no longer be found.
- Registration Mode: An agent needs to update its location in a predefined directory server (e.g., its home host) that allows agent to be registered, deregistered or located.

The directory server can be either a central node, which may become the bottleneck of the system performance and/or a single Point of failure, or the agent's home host, which follows the idea of Mobile IP [7].

B. Agent Message Boxes

The messages to and from the agents are stored in a buffer on agent server called Agent Message Box (AMB) [3]. Agent message boxes have two sections of buffers allocated separately from a single buffer queue for incoming and outgoing messages called Receive Message Buffer (RMB) and Send Message Buffer (SMB). The incoming messages are saved in the Agent Message Box first and once the agent is located, the messages are sent to the agent either by a Pull mechanism or a Push mechanism. In short, the three assumptions being made in this section are

- Communication by means of asynchronous message delivering.
- Fault free in communication links and network hosts.
- Message not loss or damage during its transfer.

Fig. 4 shows the communication between two agents. The rectangle represents a host. Agent A located in Host1, Agent B located in Host2, and Bm is the message box of Agent B. When A sends a message to B, it sends the message to B's message box Bm, and then B uses a pull operation to fetch the message from its message box B_m.

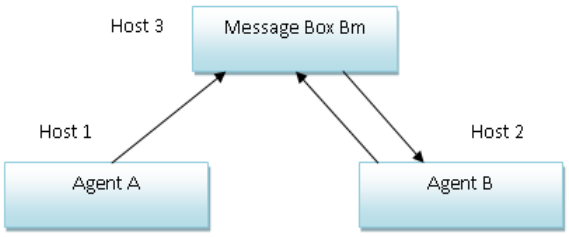

Fig. 4. Message box based agent communication

III. Inter-Agent Communication Protocol

The mechanism for Inter-Agent communication is to send messages between two different agents either on a same or a different enterprise network loops. In both cases, the messages are delivered to the destination agent servers through the destination agent routing tables. The agents are empowered to encrypt [8] the messages it receives using the public key of the remote host and are delivered to the remote agent. Both the agents (local and remote) can cooperate to form a multi agent system [9]. The physical representation of agent communication between two different network loops through agent servers is shown in the Fig. 5.

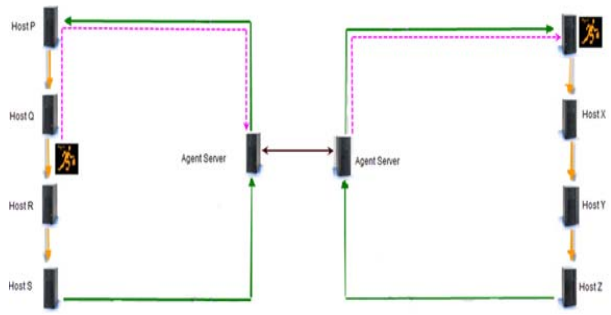

Fig. 5. Inter-agent communication through agent servers

A. The Setup

Algorithm:
Originator Agent

- When an agent is initiated, it would create Sending Message Box & Receiving Message Box along with the Host Message Routing Tables.
- The agent ID and the Agent Server information is communicated to the other agent servers already active in the enterprise network.
- When an originator agent has to send a message to another agent, the agent saves it current execution state, and then send the message

  sendMsg(Dest Agt, Msg, Priority)
  waitforAck()
- Once the message is received by the agent server, the server sends the acknowledgement to the agent before it moves to the next host.

B. Case – 1 Destination Agent in Dormant State

Once the message is received by the destination server and the agent server updates the Destination Agent Routing Table and starts looking for the agent. The agent is in dormant state at the agent server, the message is directly delivered to the agent after decryption. Fig 6 shows the agent is at the destination agent server and ready to take the message addressed for the agent. When the agent is in dormant state, the agent periodically checks the destination routing table if there are any messages available for the agent.

Fig. 6. Destination agent in dormant state

Algorithm:

Originator Mobile Agent sends the encrypted message to Agent Server with appropriate priority and the destination address

- Originator Agent Server decrypts the message and populates the Destination Agent Routing Table for the Destination Mobile Agent
- The Originator Server encrypts the message in the HMRT with Destination Servers RSA key that is already shared between the two servers
- The encrypted message is sent to the Destination Agent Server
- As the Destination Agent is in dormant state and at the Agent Server, the message is pushed to the Agent

pushMsg(Src, AgtID, Message,Exptime)

The Push Mechanism is used to push the message to the agent without considering the priority of the message since in dormant state, the host address is always the agent server and the valid flag is set to 1 (active). So in order to make the agent active, the message is pushed to the agent. The agent determines the action to be performed either by itself or through a set of instructions from the server and performs the action.

C. Case – 2 Destination Agent at Remote Host

The agent moves from originator to each of the host to perform a defined task at each host. When the host is at a remote host away from the agent server, the message intended for the agent arrives in the agent message box. Fig 7 represents the agent is at a remot host and to deliver this message to the agent the agent server takes into consideration the priority of the message and also the address flag validity. The priority of the message would define whether it would employ a push or pull mechanism to deliver the message. The valid flag would determine if the agent is stationary on a host or mobile. A message cannot be delivered to an agent when it is mobile.

Each message arriving at the agent server message box would first be loaded into the Receiving Message Box (RMB) and the data is updated in the Destination Agent Routing Table (DART). The important parameters that would be updated are the priority of the message and the timestamp along with the pointer to the message from RMB. The address of the agent is currently located is available in the Destination Agent Server ID field. When the agent is at the agent server, the agent server location is loaded into the column and when it migrates to a remote host, the remote host address is available in the column.

A low priority message would be saved in the DART until agent pulls the message from the message box. Agent periodically communicates its position to its agent server and would also check if there are any unread messages in its message box. All the unread low priority messages would be read by the agent using pull mechanism periodically. When a high priority message arrives at the agent message box, the agent server checks the valid flag, if the address is valid the message is pushed to the agent. If the valid flag is set to 0 (inactive), the agent server has to wait till the address field is set to 1 (active) and deliver the messages to the host.

The Algorithm:

- When the Destination Agent Server receives a message, the Destination Agent Server gets the Address of the Remote Host and the Valid Flag.
- If (Valid_Flag = ‘Active’)

then

```
    If (Priority = 'High')
    pushMsg(SrcAgtID, Message, Exptime)
       else
       waitForPull()
  else
    ## If the agent is in Transit Valid Flag is Inactive
  waitForValidFlag()
```

Two wait functions are designed for the agent server to hold on the message before passing it to the agent. The function waitForPull() is used in this section that would wait for the message to be pulled by the agent either when the message priority is low. Similarly when the Valid flag is inactive, the function waitForValidFlag() will wait for the valid flag to be active. The high priority messages are pushed using the same function pushMsg(SrcAgtID, Message, Exptime) discussed in the earlier section.

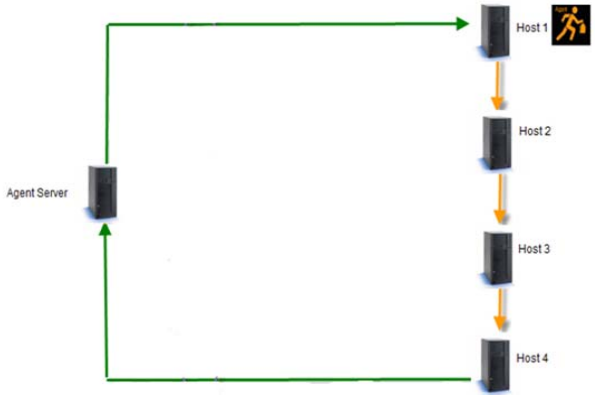

Fig. 7. Case 2- agent on remote host

Fig. 8. Case 3- agent in transit

D. Case –3 Destination Agent in Transit

When an agent moves from one host to the other, it performs the set of operations viz., save the current state of execution, encrypt the message collected from the host using the host RSA key, then communicate to the agent server that the agent would be moving to the next location, set the Valid flag to inactive then migrate to a new host. The first two operations are part of the generic agent operations; however, the next steps are designed as part of this algorithm. The agent is aware of the next host to visit, and this need to be communicated to the agent server. Hence before moving to the new host, the agent server sends the current and next host address to the agent server, set the Valid flag to 'Inactive' and wait for the acknowledgement from the agent server. Once the agent receives the acknowledgement the agent migrates to the new host and retrieves it execution state. Now the agent sends another message to the agent server with the new host address and instructions to set the Valid flag to 'Active'. The agent server sets the new address location and Valid flag and send acknowledgement to the agent. The agent wait for the acknowledgement and once it is received; the agent pulls the messages from the agent server message box. As discussed earlier, first the high priority messages are pulled followed by the lower priority once. Fig 8 describes the agent migration from Host 2 to Host 3 and the algorithm represents the steps to be taken during the migration.

The Algorithm:

- Agent sets Valid Flag to Inactive

```
setInactive(AgentID, prevHostID, newHostID)
waitforAck()
```

- Once the Agent reaches the new host,

```
setActive(AgentID, newHostID)
```

- Agent Server responds

```
setAck(ack, msgCount)
```

- Pull the messages from Agent Message Box ordered by priority

```
while (msgCount <> 0)
pullMsg(SrcAgtID, Message, Exptime)
```

The valid flag is set to inactive and reset by two functions setInactive(AgentID, currHostID, newHostID) & setActive(AgentID, newHostID) The setInactive function would require both current host id and the new host id sent to make sure agent server understands the current and new locations of the agent. This new address is validated when the setActive message is sent by the agent. The waitforAck() function would make the agent wait at the current host until it receives the acknowledgement from the agent server. If the acknowledgement is not received in a specified period of time, the agent would resend the setInactive() message.

Once the agent migrates to the new host, the setActive() message would be sent to the agent server to make the flag valid. The acknowledgement message sent by the agent server would have both acknowledgement as well as the unread message count. The agent before going ahead with the its next set of operations at the host, will pull all the unread messages from the agent server message box using the function pullMsg(SrcAgtID, Message, Exptime).

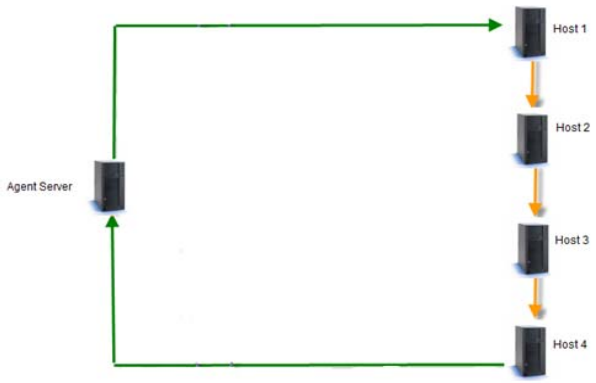

Fig. 9. Case 4- agent is dead

E. Case –4 Destination Agent is Dead

When a message is received by an agent server for an agent

that is no longer active, the agent server would respond back to the originator that the agent is dead. The dead agent cannot be seen on the network as represented in Fig 9. The agent server first checks the Destination Agent Routing Table (DART), for the address of the agent. If the agent is not found, a negative acknowledgement is sent to the originator. The originator on receiving the negative message would update its Host Message Routing Table (HRMT) by deleting the address associated with the agent. Though the agent the agent server is looking for is dead, the other agents that use the same Agent Server could still be operational and performing their task.

The Algorithm:

When destination Agent Server receives a message intended for an Agent,

- Agent Server looks for the Agent ID in the Destination Agent Routing Table (DART)
- When the Agent doesn't exist in the DART

  sendNegative(OrgAgentServerID)
- Wait till the expiry time of the message,

  delMsg(MsgID)

The message is restored in the Receiving message box till the expiry time and then deleted from both the message box and the DART using the fuction delMsg(MsgID).

IV. Conclusion and Future Scope

The demonstration Inter-Agent communication protocol is tested on a trusted network where a standard RSA Key encryption protocol is used and the keys are exchanged between trusted hosts in advance. However, the implementation in real time would require the keys to be exchanged over internet which is risky. The protocol would undergo few changes while implementing on public network.

References

[1] Jade. Java Agent Development Framework. [Online]. Available: http://jade.cselt.it

[2] V. Sreekanth, S. Ramchandram, and A. Govardhan, "A novel approach for security and integrity of mobile agents," in *Proc. ICCBN 2008*, IISc, Bangalore, India.

[3] J. Cao, X. Feng, J. Lu, and K. Sajal, "Mailbox based scheme for mobile agent communications," *IEEE Computer*, vol. 35, no. 9, pp. 54-60, September 2002.

[4] P. Braun and W. Rossak, *Mobile Agents Basic Concepts, Mobility* Models*, and the Tracy Toolkit*, Morgan Kaufmann Publishers, 2004.

[5] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, 1998.

[6] P. Braun, D. Trinh, and R. Kowalczyk, "Integrating a new mobility service into the jade agent toolkit," in *Proceedings of the Second International Workshop on Mobility Aware Technologies and Applications*, Montreal/Canada, 2005, Springer, Montreal/Canada.

[7] M. Lunge and D. B. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*, Addison Wesley, 1998.

[8] Roth, "Empowering mobile software agents," in *Proc. 6th IEEE Mobile Agents Conference*, vol. 2535 of Lecture Notes in Computer Science, pp. 47-63, Springer.

[9] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, 2005, pp. 387-434.

V. Sreekanth hails from Warangal, AP, India. He completed his Diploma in Electronics & Communication Engineering in 1996 in Distinction from Govt. Polytechnic, Warangal and then his BE in Computer Technology in year 2000 from Nagpur University, Nagpur. Later he worked as Lecturer at Kakatiya Institute of Technology & Sciences, Warangal for 2 Years. He joined Acharya Nagarjuna University for his M.Tech in Computer Science & Engineering and completed with Distinction in year 2005.

The authors' zeal towards Networki ng and mobile computing from the days of graduation has encouraged him to take up research in the field of mobile computing. During the same period, the author met with Professor S.Ramchandram and Dr. A Govardhan and shared his research Ideas and was readily accepted and encouraged by both and accepted to guide him in the research program. The Author registered as a research scholar in JNTU Hyderabad in year 2006 and had published 5 papers in different International Conferences and Journals.