On Efficient Communication and Service Agent Discovery in Multi-agent Systems

Myeong-Wuk Jang and Gul Agha
Department of Computer Science
University of Illinois at Urbana-Champaign
{mjang, agha}@uiuc.edu

Abstract

The paper studies two closely related problems: how to support efficient message passing in large-scale agent systems given that agents are mobile, and how to facilitate the discovery of service agents in an open environment where agents may enter and leave. The Actor Architecture *has been designed to study simulations of large-scale agent systems where agents obey the operational semantics of actors. We describe the solutions to these two problems that have been adopted in the Actor Architecture. The problem of efficient message-passing is partially addressed by using dynamic names for agents. Specifically, a part of the name of a mobile agent changes continuously as a function of the agent platform that it is currently hosted by. This enables the agent platform of a sender to use location information about the receiver agent in order to optimize message delivery. The problem of agent discovery is addressed by using a broker agent. Moreover, the sender agent may reduce the communication that is required between the sender itself and a broker agent by sending the broker an agent to localize the search for the appropriate service agents. In order to mitigate security problems, this search agent is very restricted in what operations it is allowed to perform and is transmitted in the form of a passive object. A description of the Actor Architecture is given, focusing on these two ideas and their preliminary evaluation.*

Keywords: Open Distributed System, Multi-agent System, Actor System, Message Passing, Brokering Service.

1. Introduction

A number of multi-agent systems, including EMAF [3], JADE [4], InfoSleuth [5], and OAA [6], support an *open agent systems,* i.e., systems in which agents may enter and leave at any time. Moreover, the growth of computational power and networks has made large-scale open agent systems a promising technology. However, before this vision of scalable open agent systems can be realized, two closely related problems must be addressed:

- How can an agent efficiently discover service agents which are previously unknown? In an open agent system, the mail addresses or names of all agents are not globally known; agents may not have the addresses of other agents with whom they need to communicate. This suggests that *middle agent services* such as *brokering* and *matchmaking* are necessary [14]. As we scale up agent systems, efficiently implementing these services is a challenge.

- How to efficiently send messages to agents which have potentially moved? In mobile agent systems, efficiently sending messages to an agent is not simple because they move continuously from one agent platform to another. For example, one obvious solution, viz. requiring the *agent platform* on which a mobile agent is created to manage location information about that agent, may double the message passing overhead.

We address the message passing problem for mobile agents in part by providing a richer structure on names which allows the names to dynamically evolve. Specifically, the names of agents include information about their current location. Moreover, rather than simply sending data as messages, we allow an agent system to use the data to find the location of an appropriate receiver agent.

We have implemented our ideas in a Java-based agent system called the *Actor Architecture* (or *AA*). AA supports the *actor semantics* for agents: each agent is an autonomous process with a unique name (address), message passing between agents is asynchronous, new agents may be dynamically created, and agent names may be communicated [1]. AA is being used to develop tools to facilitate large-scale simulations, but it may be used for other large-scale open agent applications as well; AA has been designed with a modular and extensible, application-independent structure. The primary features of AA are to provide a light-weight implementation of agents, minimize communication overhead between agents, and enable service agents to be located efficiently.

This paper is organized as follows. Section 2 introduces the overall structure and functions of AA and the agent life cycle model in AA. Section 3 explains how to reduce the communication overhead in AA, and Section 4 shows how to improve the middle agent service in AA. Section 5 describes our experiments with AA and evaluation of our approaches. Finally, in Section 6 we discuss our preliminary conclusions and research directions.

2. The Actor Architecture

AA provides a light-weight implementation of agents as active objects or actors [1]. Actors can provide the infrastructure for a variety of agent systems; they are social and reactive, but they are *not* explicitly required to be "autonomous" in the sense of being proactive [16]. However, autonomous actors may be implemented in AA and many of our experimental studies require proactive actors. Although the term agent has been used to mean proactive actors, for our purposes, the distinction is not critical. In this paper, we use terms 'agents' and 'actors' as synonyms.

The Actor Architecture consists of two main components:

1. Actor execution environments called *AA platforms*. AA platforms provide the system environment in which actors exist and interact with other actors. Specifically, AA platforms provide actor state management, actor communication, actor migration, and middle actor services.

2. An *actor library* which supports the development of agents that are executed on AA platforms.

We describe the structure of AA in greater detail. An AA platform consists of eight components (see Figure 1): Message Manager, Transport Manager, Transport Sender, Transport Receiver, Delayed Message Manager, Actor Manager, Actor Migration Manager, and ATSpace.

A *Message Manager* (MM) handles message passing between actors. Every message passes through at least one Message Manager. If the *receiver* actor of a message exists on the same AA platform, the MM of the platform directly delivers the message to the receiver actor. However, if the receiver actor is not on the same AA platform, this MM delivers the message to the MM of the platform where the receiver currently resides, and finally the MM delivers the message to the receiver. A *Transport Manager* (TM) maintains a public port for message passing between different AA platforms. When a *sender* actor sends a message to a receiver actor on a different AA platform, the *Transport Sender* (TS) residing on the same platform as the sender receives the message from the MM of the sender actor and delivers it to the *Transport Receiver* (TR) on the AA platform of the receiver. When there is no a built-in connection between these two AA platforms, the TS contacts the TM of the AA platform of the receiver actor to open a connection so that the TM can create a TR for the new connection. Finally, the TR receives the message and delivers it to the MM on the same platform.

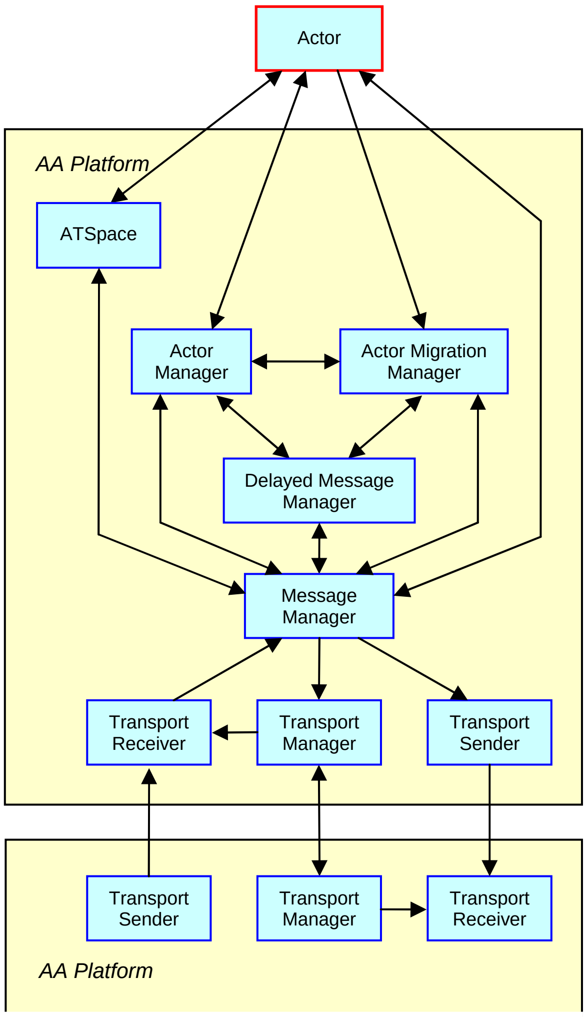


Figure1. The Architecture of an AA Platform

A *Delayed Message Manager* (DMM) temporarily holds messages for mobile actors while they are moving from their AA platform to other AA platforms. An *Actor Manager* (AM) manages states of the actors that are currently executing and the locations of the mobile actors created on the AA platform. An *Actor Migration Manager* (AMM) manages actor migration.

An *ATSpace* provides middle actor services, such as matchmaking and brokering services. Unlike other system components, an ATSpace is implemented as an actor. Therefore, any actor can create an ATSpace, and hence, an AA platform may have more than one ATSpaces. The ATSpace created by an AA platform is called the *default ATSpace* of the platform, and all actors can obtain the actor names of default ATSpaces. Once an actor has the name of an ATSpace, the actor may send the ATSpace messages in order to use its services.

In AA, actors are implemented as active objects and executed as threads; actors on an AA platform are executed with that AA platform as part of one process. Each actor has one actor life cycle state at any time (see Figure 2). An actor may be *static*, meaning that it exists on its original AA platform, or it may be *mobile*, meaning that it has migrated from its original AA platform. The state information of a static actor appears within only its original AA platform while that of a mobile actor appears both on its original AA platform and on its current AA platform. When an actor is ready to process a message its state becomes `Active` and stays while the actor is processing the message. When a mobile actor initiates migration, its state is changed to `Transit`. Once the migration ends and the actor restarts, its state becomes `Active` on the current AA platform, and `Remote` on the original AA platform. Following a user request, an actor in the `Active` state may be `Suspended` state. In contrast to other agent life cycle models (e.g. [7, 12]), AA's life cycle model uses the `Remote` state to indicate that an agent that was created on the current AA platform is working on another AA platform.

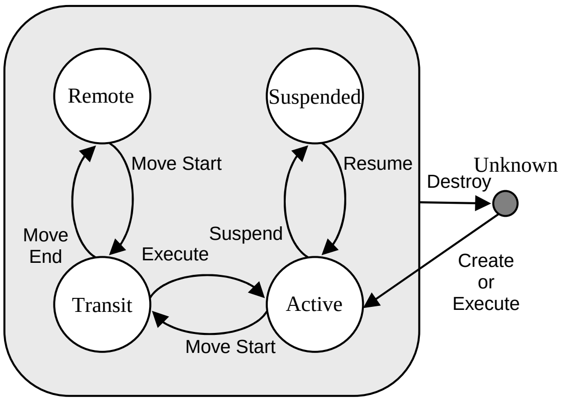


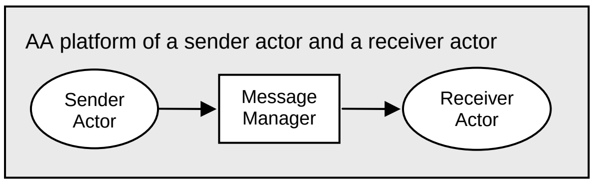
Figure 2. The Actor Life Cycle Model

3. Optimized Communication

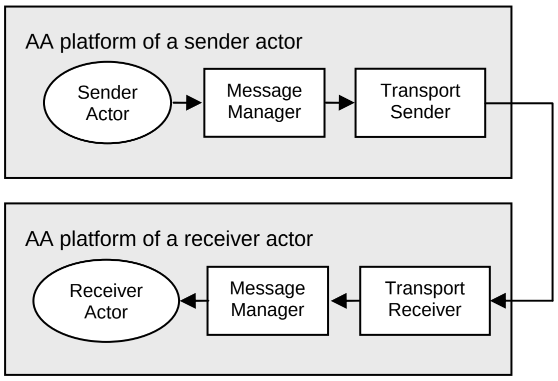
We describe the mechanisms used to support actor communication. Specifically, AA uses two approaches to reduce the communication overhead for mobile actors that are not on their original AA platforms: namely, *location-based* message passing and *delayed* message passing.

3.1. Message Passing between Actors

After a message has been created, the message is managed by the Message Manager. When the receiver actor of a message is located on the same AA platform where the sender actor exists, the message is directly delivered to the receiver actor by the Message Manager (Figure 3a). However, if the receiver is on a different machine, the message is delivered to the receiver through the Message Manager and the Transport Sender of the sender actor, and the Transport Receiver and the Message Manager of the receiver actor (Figure 3b). Although these two approaches of message passing are different at the system level, they are transparent to actors, and hence, actors always use the same operator to send their messages.



a. Internal Actor Communication



b. External Actor Communication

Figure 3. Procedure for Actor Communication

3.2. Location-based Message Passing

Before an actor can send messages to other actors, it must know the names of the intended receiver actors. In AA, each actor has its own unique name called *UAN* (*Universal Actor Name*). The UAN of an actor includes the *location information* and *unique identification number* of the actor as follows:

```
uan://128.174.245.49:37
```

From the above name, we can infer that the actor exists on the host whose IP address is `128.174.245.49`, and that the actor is distinguished from other actors on the same platform with its unique identification number `37`.

When the Message Manager of a sender actor receives a message whose receiver actor has the above name, it checks whether the receiver actor exists on the same AA platform. If they are on the same AA platform, the Message Manager finds the receiver actor on the AA platform and delivers the message. If they are not, the Message Manager of the sender actor delivers the message to the Message Manager of the receiver actor. In order to find the AA platform where the Message Manager of the receiver actor exists, the location information `128.174.245.49` in the UAN of the receiver actor is used. When the Message Manager on the AA platform with IP address `128.174.245.49` receives the message, it finds the receiver actor and delivers the message.

The above actor naming and message delivery scheme works correctly when all actors are static. However, because an actor may migrate from one AA platform to another, we extend the basic behavior of the Message Manager with a *forwarding* service; when a Message Manager receives a message for a mobile actor, it delivers the message to the current AA platform of the mobile actor. To facilitate this service, an AA platform maintains a table providing the current locations of mobile actors that were created on the AA platform.

The problem with using only the universal names of actors for message delivery is that every message for a mobile actor that has moved from the original AA platform where the actor was created still has to pass through the original AA platform (Figure 4a). This kind of indirection may happen even in case the receiver actor exists on an AA platform that is close to (or the same as) the AA platform of the sender actor. In fact, message passing between actor platforms is relatively expensive. AA uses *Location-based Actor Naming* (LAN) for mobile actors in order to generally eliminate the need for this kind of indirection. Specifically, a LAN of an actor consists of its current location and its UAN as follows:

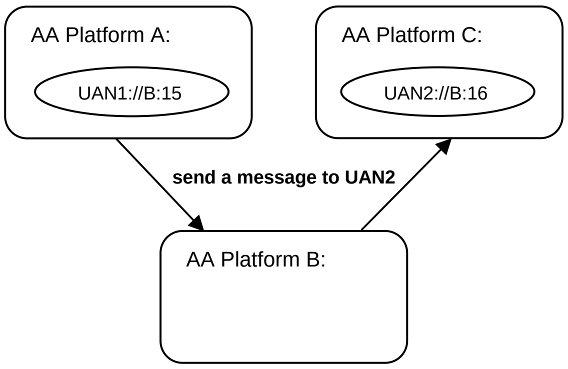
```
lan://128.174.244.147//128.174.245.49:37
```

The current location of a mobile actor is set by an AA platform when the actor arrives on the AA platform. If the current location is the same as the location where an actor was created, the LAN of the actor does not have any special information beyond its UAN.

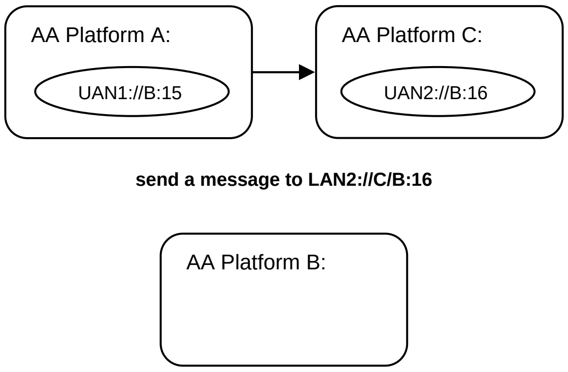
Under the location-based message passing scheme, when the Message Manager of a sender actor receives a message for a remote actor that exists on the different AA platform, it checks the current location of the receiver actor with its LAN and delivers the message to the AA platform where the receiver actor exists (Figure 4b). The rest of the procedure for the message passing is similar to that in UAN-based message passing scheme.

With location-based message passing, the system is more fault-tolerant; since messages for a mobile actor need not pass through the original AA platform of the actor, the messages may be correctly delivered to the actor even when the actor's original AA platform is not working correctly.

In order to use LAN address scheme, the location information in a LAN should be correct. However, mobile actors may move repeatedly, and a sender actor may have old LANs of mobile actors. Thus a message for a mobile actor may be delivered to the *previous AA platform* from where the actor left. This problem is addressed by having an old AA platform deliver the message to the original AA platform where the actor was created; the original platform always manages the current address of an actor. There is no guarantee that the location-based message passing will perform better than the UAN-based message passing. Therefore, AA allows an actor to decide which addressing scheme is better for the current situation.



a. UAN-based Message Passing



b. Location-based Message Passing

Figure 4. Message Passing to a Mobile Actor

3.3. Delayed Message Passing

While a mobile actor is moving from one AA platform to another, the current AA platform of the actor is not well defined. Therefore, when the Message Manager of the original AA platform receives a message for a mobile actor, it sends the message to the Message Manager of the old AA platform. After the Message Manager of the old AA platform receives the message, it forwards the message to the Message Manager of the original AA platform because it no longer has information about the mobile actor's current location. An AA platform manages location information about only the mobile actors that are created on it. Thus, a message is continuously passed between these two AA platforms until the mobile actor updates location information with its new AA platform by informing the Actor Manager of the original AA platform.

In order to avoid unnecessary message passing, a Delayed Message Manager in AA platform is used; the Message Manager of the old AA platform delays the message passing for a mobile actor while the state of the actor is `Transit`. For this operation, the Actor Manager of the old AA platform manages the state of the mobile actor and the Delayed Message Manager holds messages for the mobile actor until the actor reports that its migration has ended. After an actor finishes its migration, the new AA platform of the actor sends its old AA platform and its original AA platform a message to inform that the migration process has ended. Whenever one of these two AA platforms receives a message, the original AA platform changes the state of the mobile actor from `Transit` to `Remote` while the old AA platform removes information about the mobile actor.

4. Active Brokering Service

A brokering service is useful for supporting *attribute-based communication* between agents that are in an open multi-agent system. Recall that in open multi-agent systems, service agents that support a specific service may not be known to client agents; with attribute-based communication, client agents may use the attributes of the service they require instead of using the names of the service agents. The attributes of the service are delivered to a middle agent as a *tuple template*, and the middle agent tries to find a service agent or a set of service agents whose attributes are matched with the tuple template. The agents selected by the middle agent receive the message sent by the client agent through the middle agent. This service is very effective in open multi-agent systems, but the searching ability of the middle agent is often very restrictive for efficiency reasons; a middle agent typically provides only template-based exact matching or regular expression matching [2, 8, 11]. If a client agent requires a more powerful search, the client agent must use a matchmaking service instead of a brokering service; the client receives all the information about service agents and utilizes its own searching algorithm to locate proper service agents. For example, consider a middle agent that has information about seller agents with their products and prices, and a buyer agent wants to find seller agents that sell a computer with price greater than $500 and less than $1,000. If the exact matching service of the middle agent is not powerful enough to support this function, the buyer agent has to obtain all information about seller agents from the middle agent, and then choose seller agents that sell their computers within the price range. This sequence of operations requires moving of all information about seller agents from the middle agent to the buyer agent through the network.

In order to reduce the communication overhead, AA provides an active brokering service through an ATSpace actor. An ATSpace actor allows a sender actor to send its own search algorithm instead of a simple description for attributes of the service to locate receiver actors, and the algorithms are executed in the ATSpace actor. In Figure 5, the seller actors with UAN2 and UAN3 are selected by the search algorithm, and the ATSpace actor delivers `sendComputerBrand` message to the actors. Finally, they will send information about brand names of their computers to the buyer actor.

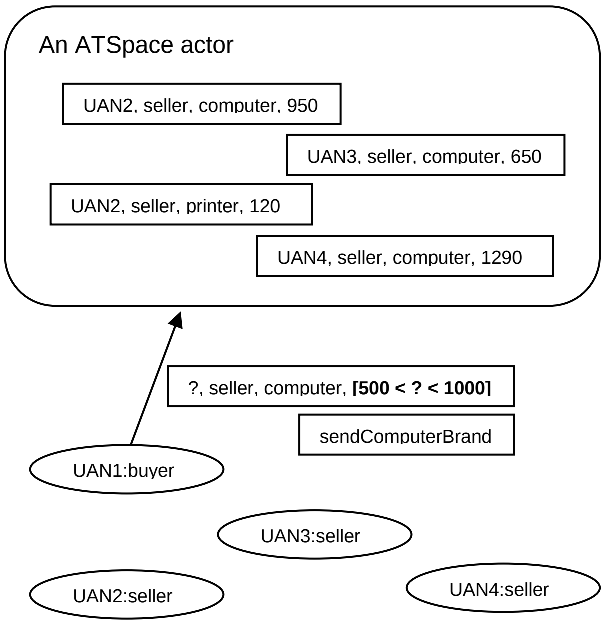

An ATSpace actor
UAN2, seller, computer, 950
UAN3, seller, computer, 650
UAN2, seller, printer, 120
UAN4, seller, computer, 1290
?, seller, computer, [500 < ? < 1000]
sendComputerBrand
UAN1:buyer
UAN3:seller
UAN2:seller
UAN4:seller

Figure 5. An Example of Active Brokering Service

In many situations, moving the search algorithm from the seller agent to the middle agent is less expensive than moving all the information about certain agents from the middle agent to the sender agent. Since a matching algorithm is provided by a sender agent and the algorithm is executed on a middle agent, the middle agent called an *ATSpace* actor can provide application oriented brokering service more efficiently. Moving the search algorithm may be accomplished by sending an agent incorporating the search algorithm. However, this extension introduces security threats for the data in the ATSpace actor. AA provides some solutions to mitigate such threats, in particular by not allowing arbitrary agents to be sent; agents are sent as passive objects and their functions are controlled by the ATSpace actor [9].

5. Experiments and Evaluation

AA platforms and actors have been implemented in Java language to support operating system independent actor mobility. We are using our actor system for large-scale UAV (Unmanned Aerial Vehicle) simulations. In these simulations, we investigate the effects of different collaboration behaviors among the large number of micro UAVs during their surveillance missions on the large number of targets [10]. For our experiments, we execute more than 5,000 actors on four computers: 2500 micro UAVs, 2500 targets, and other simulation purpose actors.

The delayed message passing removes unnecessary message passing for moving agents. When the delayed message passing is used, the old AA platform of a mobile actor needs to manage its state information until the actor finishes its migration, and the new platform of the mobile actor needs to report the migration state of the actor to its old AA platforms. In our experience, this overhead is more than compensated; without the delayed message passing the same message may get delivered seven or eight times between the original AA platform and the old AA platform in the local network environment while a mobile actor is moving. If a mobile actor takes more time for its migration, this number may be even greater. Moreover, the extra hops also make the message log files more complex and reduce their readability.

The performance benefit of the active brokering service can be measured by comparing it with the matchmaking service that provides the same service along four different dimensions: the number of messages, the total size of messages, the total size of memory space on two AA platforms for client and middle actors, and the time for the whole operation. First, in the matchmaking service, the number of messages is $n + 2$, where n is the number of service actors, while it is $n + 1$ in the active brokering service. In the former, the number of messages for this operation includes a service request message from the client actor to the middle actor, a reply message from the middle actor to the client actor, and multicast messages from the client actor to n service actors. The active brokering service does not require the reply message, and hence, one message is unnecessary. It is a small difference, but more significantly, the total size of messages is very different. The service request message in the active brokering service is a little larger than that in the matchmaking service, because it includes the code for a searching algorithm and the message to be delivered to service actors. However, the reply message in the matchmaking service to be communicated across the network may be much larger than the difference of service request messages in two approaches. Moreover, the total size of storage space for the active brokering service is less than that in the matchmaking service; in the matchmaking service case a copy of the data exists in the client actor, while in the active brokering service such a copy need not exist in the client actor. However, for the data safety, the active brokering service may still keep a copy of the data. Finally, the difference in operation times except communication times is relatively small. Mainly, the computation in matchmaking is off-loaded to the server side. However, since the communication time is proportioned to the total size of messages, the active brokering service is more efficient in the time for the whole operation.

6. Conclusions

The location-based message passing scheme in AA reduces the number of hops (AA platforms) that a message for a mobile actor goes through. The basic mechanism of the location-based message passing is similar to the message passing in Mobile IP [13], although its application domain is different from ours. The original and current AA platforms of a mobile actor correspond to the home and foreign agents of a mobile client in Mobile IP, and the UAN and LAN of a mobile actor are similar to the home address and care-of address of a mobile client in Mobile IP. However, while the sender node in Mobile IP manages a binding cache to map home addresses to care-of addresses, the sender AA platform in AA does not have a mapping table, and while the home agent communicates with the sender node to update the binding cache, it does not happen in AA.

Our work may also be compared to SALSA. In SALSA, a sender actor may use a middle actor called Universal Actor Naming Server to locate the receiver actor [15]. SALSA's approach requires the receiver actor to register its location at a certain middle actor, and the middle actor must manage the mapping table. With the location-based message passing scheme in AA, a LAN of an actor is changed automatically as a function of an AA platform, and the mapping table does not exist at any single place.

We are currently implementing and testing new message passing mechanisms for mobile agents. For example, the location-based message passing may be modified to allow a mobile agent to set its future location address in its LAN and announce this to other agents. For the delayed message passing, instead of the old AA platform of a mobile agent, the new AA platform of the mobile agent may hold messages for the agent, and hence, when the agent finishes its migration it receives the messages managed by the Delayed Message Manager of the AA platform. We plan to investigate various trade-offs and methods for automatically selected best estimated message-passing mechanism for a given situation.

Acknowledgements

This research is sponsored by the Defense Advanced Research Projects Agency under contract number F30602-00-2-0586.

References

[1] G. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press, Cambridge, Mass, 1986.

[2] G. Agha and C.J. Callsen, "ActorSpaces: An Open Distributed Programming Paradigm," *Proceedings of the 4th*

ACM Symposium on Principles & Practice of Parallel Programming, May 1993, pp. 23-32.

[3] S. Baeg, S. Park, J. Choi, M. Jang, and Y. Lim, “Cooperation in Multiagent Systems,” *Intelligent Computer Communications* (*ICC ’95*), Cluj-Napoca, Romania, June 1995, pp. 1-12.

[4] F. Bellifemine, A. Poggi, and G. Rimassa, “JADE - A FIPA-compliant Agent Framework,” *Proceedings of Practical Application of Intelligent Agents and Multi-Agents* (*PAAM ’99*), London, April 1999, pp. 97-108.

[5] R.J. Bayardo Jr., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk, “InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments,” *ACM SIGMOD Record*, Vol. 26, No. 2, June 1997, pp. 195-206.

[6] P.R. Cohen, A.J. Cheyer, M. Wang, and S. Baeg, “An Open Agent Architecture,” *AAAI Spring Symposium*, March 1994, pp. 1-8.

[7] Foundation for Intelligent Physical Agents, *SC00023J: FIPA Agent Management Specification*, December 2002. http://www.fipa.org/specs/fipa00023/

[8] N. Jacobs and R. Shea, “The Role of Java in InfoSleuth: Agent-based Exploitation of Heterogeneous Information Resources,” *Proceedings of Intranet-96 Java Developers Conference*, April 1996.

[9] M. Jang, A. Abdel Momen, and G. Agha, “ATSpace: A Middle Agent to Support Application-Oriented Matchmaking and Brokering Services,” Technical Report UIUCDCS-R-2004-2430, Department of Computer Science, University of Illinois at Urbana-Champaign, April 2004.

[10] M. Jang, S. Reddy, P. Tosic, L. Chen, and G. Agha,"An Actor-based Simulation for Studying UAV Coordination," *15th European Simulation Symposium* (*ESS 2003*), October 2003, pp. 593-601.

[11] D.L. Martin, H. Oohama, D. Moran, and A. Cheyer, “Information Brokering in an Agent Architecture,” *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, April 1997, pp. 467-489.

[12] D.G.A. Mobach, B.J. Overeinder, N.J.E. Wijngaards, and F.M.T. Brazier, “Managing Agent Life Cycles in Open Distributed Systems,” *Proceedings of the 2003 ACM symposium on Applied computing*, Melbourne, Florida, 2003, pp. 61-65.

[13] C.E. Perkins, “Mobile IP,” *IEEE Communications Magazine*, Vol. 35, No. 5, May 1997, pp. 84-99.

[14] K. Sycara, K. Decker, and M. Williamson, “Middle-Agents for the Internet,” *Proceedings of the 15th Joint Conference on Artificial Intelligences* (*IJCAI-97*), 1997, pp. 578-583.

[15] C.A. Varela and G. Agha. “Programming Dynamically Reconfigurable Open Systems with SALSA,” *ACM SIGPLAN Notices: OOPSLA 2001 Intriguing Technology Track*, Vol. 36, No. 12, December 2001, pp. 20-34.

[16] M. Wooldridge, *An Introduction to MultiAgent Systems*, John Wiley & Sons, Ltd, 2002.