Enabling Mobile Agents Communication

Mohammad Samarah and Philip Chan
The division of Electrical and Computer Science and Engineering
Florida Institute of Technology
150 West University Boulevard, Melbourne, FL 32901-6988
msamarah@zach.fit.edu and pkc@cs.fit.edu

Florida Tech Technical Report CS-2000-3

Abstract - In this paper, we investigate the need for well-suited remote communication architectures to address communication issues in mobile agent environments. We study the implication of mobility for agent architectures – specifically, ways in which the architecture may facilitate agent communication. We present an architecture for inter-agent communication suitable for remote messaging, agent monitoring, agent tracing, and agent debugging for mobile agent environments. The architecture allows for the dynamic adaptation of communication components. It provides for a seamless and continuous active communication during the agent migration process. We have implemented an agent framework simulator that conforms to all these requirements. The paper presents the results illustrating the framework dynamic behavior.

Keywords: Agent models and architectures, mobile agents, communication languages and protocols, performance evaluation, distributed location registry.

1 Introduction

Mobile agents is an emerging technology attracting audiences from the fields of distributed systems, information retrieval, World Wide Web, electronic commerce and artificial intelligence. A mobile agent is an autonomous entity that can migrate from one machine to another in a heterogeneous network. The agent may suspend its execution at any point, transport itself to another machine and then resume execution. Mobile agents depart from the conventional client/server model and give rise to a paradigm shift in distributed systems in which the agents are autonomous and self sustained.

In order for mobile agents to flourish they need a software environment in which they can exist. A mobile agent environment is a distributed software system running over a network of heterogeneous computers. The primary task of the environment is to provide an execution framework for the agents. The mobile agent environment implements a large subset of the mobile agents' models. The environment may provide support services that relate to the mobile agent environment itself and support services pertaining to the environments on which the mobile agent environment is built. The environment also provides support services to access other mobile agent systems, and support services to provide open access to other non-agent based software environments.

Agent-to-agent communication is the key to realizing the potential of the agent paradigm, just, as the development of human languages was the key to the rapid progress of the human race. A well-defined communication architecture is a necessary component for the success and the wide deployment of mobile agents technology. Research has addressed the communication problem from a language perspective such as the interaction protocols [d'Iverno98] and the dialogue frames [Reed98]. Furthermore, a few have proposed formal communication models for mobile agent environments such as [Baumann98] on communication concepts and [Dong98] on open communication frameworks for software agents. Current mobile agent systems employ many communication mechanisms such as messages, local and remote procedure calls, but we are not aware of any framework based on communication types and not domain specific classification. Moreover, the agent ability to move while in active communication is not addressed in many of the communication mechanisms available today.

In this paper, we present an architecture for inter-agent communication suitable for remote messaging, agent monitoring, agent tracing, and agent debugging for mobile agent environments. Synchronous communication can be established for inter-agent interactions, while asynchronous communication addresses the need for mobile and group communication. The framework provides the ability of an agent to move while in active communication, by employing message buffering and forwarding.

Our framework provides seamless communication during migration. Communication is not interrupted and proceeds seamlessly during the migration process. This feature is not addressed in many of the current architectures such as the work done in [Baumann98], and most models assume the communication is interrupted and/or terminated at the initiation of the move and reestablished once the agent arrives at the destination. Using our feature an agent is not required to be aware of other agents network related activities. The agents collaborate on the task assigned, while the framework provides the low level details of mobility.

The remainder of the paper is structured as follows. In Section 2, we present an overview of the AGI communication architecture. Section 3 discusses the issues involved in distributing the location registry. An overview of our implementation is presented in section 4. Section 5 discusses related work. Section 6 concludes the paper with a summary of results and future research directions.

2 The AGI Communication Architecture

In this section we provide an overview of the **A**synchronous, **G**roup Oriented, and **I**nter-agent communication architecture, and a description of its key features.

The AGI architecture defines three models of communication. The first model of communication is based on an asynchronous event model. The second model allows for agent group collaboration. The third model allows for direct inter-agent messaging. The communication models provide the ability to perform call back messaging, asynchronous messaging with delayed retrieval, and direct synchronous messaging.

The architecture provides models for remote communication and messaging mechanisms. These models allow the owning entity of the agent (an end user, another mobile agent, or a stationary agent) to:

1. View the status of an agent.
2. Trace the agent execution.
3. Monitor the agent behavior and assign events triggers based on the agent state.
4. Initiate new directives to the agent such as: move to another location, change to a new state, or terminate.

The remainder of this section is structured as follows. Section 2.1 and section 2.2 define the communication models and types. Section 2.3 defines the messaging middleware and the system agents.

2.1 Communication Models

The AGI communication *models* represent a high abstraction from which new communication *types* can be devised. The communication *types* represent application-related capabilities. The AGI architecture is compromised of three models:

1. **Asynchronous Events Model:** The first model of communication is based on an asynchronous event model, in which agents may post events and messages and listen for events and messages from other agents. This model is used for normal priority messages, background tasks, and events not requiring immediate responses. Furthermore, this model is used to facilitate agent mobility by storing messages for agents in transient.

2. **Group Oriented Model:** The second model allows agents to cooperate and collaborate with each other toward a common set of goals. It is used to facilitate group communication and to provide a versatile communication conduit.

3. **Inter-Agent Synchronous Model:** The third model allows for direct inter-agent messaging that provides the ability for two or more agents to communicate directly with each other. It is used to facilitate real-time messaging and immediate delivery of alert and notification messages.

Furthermore, each model is compromised of one or more component. The components are integrated using a set of interfaces. The interfaces define the capabilities of each component, and provide a framework to describe availability, priorities and essential state information. Each agent may have one or more communication core. The communication core consists of one or more communication model components.

Figure 2.1 shows the AGI communication models. It depicts three agents with different communication handling. Agent A has two communication cores, the first core has three components and the second core has one component. Agent B and C have a single communication core consisting of component A (Asynchronous), G (Group), and I (Inter-Agent).

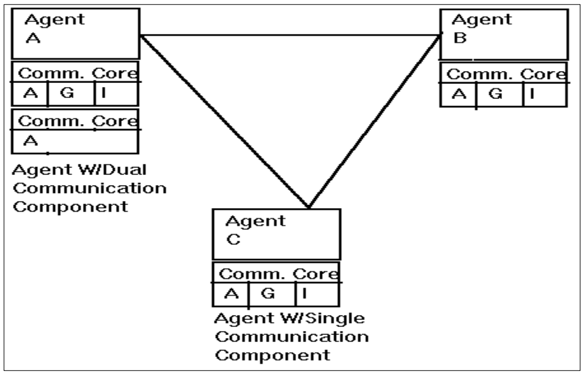

Figure 2.1 – The AGI Communication Models

In the next section we describe the communication types. The AGI architecture is not limited to those types; new ones can be created and integrated into the environment without impacting the overall architecture. The types discussed in the next section are directly mapped to our implementation.

2.2 Communication Types

The AGI architecture defines three types of agent-to-agent communication mechanisms.

2.2.1 Direct inter-agent messaging

This mechanism uses the shared messaging bus represented by the underlying network communication layer to deliver the message via single cast, multicast or broadcast as appropriate. It provides an efficient communication conduit while allowing for real-time agent-to-agent interactions. A message is delivered directly from one agent to one or more agents.

2.2.2 Indirect asynchronous messaging with return receipt

This mechanism uses the shared messaging bus to deliver the message to the messaging board. The messaging board is a persistent storage area where the messages are kept and maintained by a system agent. A return receipt is sent back to the originating agent upon the retrieval of the message. Furthermore, the arrival of the message triggers an update event that is sent from the messaging board to the recipient agent.

An agent group concept has been proposed in [Baumann97B]. This concept does not provide a solution to fault tolerance, but may be extended using group communication and voting. We define an agent group as a collection of agents working together on a common task. This mechanism provides group communication as well as asynchronous messaging.

2.2.3 Mobile Messaging

This mechanism uses the messaging board and the system agents to handle messages and events. Agents may subscribe to events and messages and may post additional messages. Messages may have a channel identifier that serves to categorize the message by subject, interest, or group. The subscription channels are created and destroyed dynamically by the submission and the deletion of the channel events. An agent may request subscription to a channel, and continue to receive updates, and at a later time turn off channel updates, or completely remove its subscription.

The communication types can be mapped into one or more communication models. The table below shows the mapping between the types above and the AGI models:

Type	Model
Indirect asynchronous messaging with return receipt Mobile Messaging	A
Indirect asynchronous messaging with return receipt Direct inter-agent messaging Mobile Messaging	G
Direct inter-agent messaging	I

Table 2.1 – Mapping Communication Types to Communication Models

In the next section, we discuss the design approach of the system agents and the middleware.

2.3 System Agents and the Messaging Middleware Design

The mobility of an agent is defined based on the code mobility and the migration model. As discussed in [Dömel97] and [Baumann97a], the different degrees of mobility can be distinguished. Our framework allows an agent to move while in active communication. The framework provides the following mobility services:

a. Location Registry Service: This service provides naming and location information. One or more location registry agent (LRA) provides this service.
b. Message Buffering and Forwarding Service: This service provides a persistent area to buffer and forward messages for agents in transient. One or more message relay agent (MRA) provides this service.
c. Messaging Events Management: This service provides monitoring and notification mechanisms of agent events. One or more messaging event agent (MEA) provides this service.
d. Reliable Delivery and Fault Tolerance: This is accomplished by replication of the LRA, MRA, and MEA agents.

In order to provide scalable location and message services, the network is divided into location zones; Figure 2.2 illustrates the zone topology. Each agent belongs to a zone. A zone is a collection of hosts connected together through the local network as depicted in Figure 2.2.

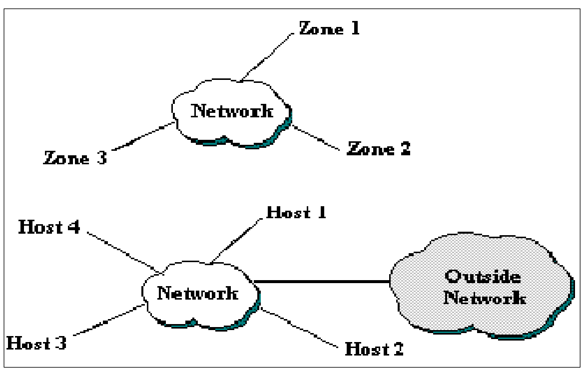

Figure 2.2 – Zone Topology: Interconnected zones and hosts within a Zone.

The system agents interact with each other to provide messaging and mobility services to user agents. The interaction is carried out through the messaging middleware (MMW).

Each agent is equipped with the messaging middleware. The middleware carries out the communication among the agents. Furthermore, all interactions among the service agents and user agents are carried through this component. The user agent performing a high-level application task is not aware of the detail interaction among the messaging middleware. The middleware maintains the system agent names and the status of the middleware as illustrated in table 2.2.

LRA Name	LRA Address	MRA Name	MEA Name	State
LRA	158.147.130.40	MRA	MEA	Idle

Table 2.2 – Information maintained by the messaging middleware

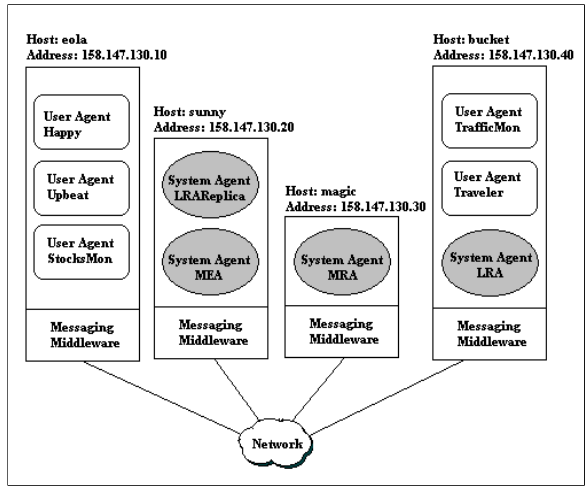

Figure 2.3 – The AGI Architecture Hosts Configuration

Each user agent undergoes a discovery phase through the messaging middleware to select its system agents. When an agent joins the framework, its messaging middleware probes the network and determines the most appropriate service agents. As the agent moves from one machine to another the middleware may select another system agent to take advantage of resource availability and proximity.

Figure 2.3 shows a typical host configuration. In this example the location zone consists of four hosts. Host eola has three user agents, while host sunny has two system agents: the MEA agent and the LRA replica agent. Furthermore, host magic has one system agent: the MRA agent, while host bucket has two user agents and one system agent.

The system agents collaborate to provide communication mobility. Fault tolerance and reliable delivery of messages are provided through the replication of system agents. In the next sections, we describe the system agents in more details.

2.3.1 The Location Registry Agent (LRA)

The location registry agent keeps track of the location of each agent and their current state. This agent may reside on one or more hosts on the network. The registry implementation may utilize a central registry, a fully replicated registry or a distributed registry. More details on the registry design are presented in section 3.

The LRA agent is equipped with a special registry to maintain location information. The registry keeps track of three tables: the transient table, the user agent location table, and the system agent location table. The transient table has two attributes:

1. Agent Name: A unique name that identifies the agent in transient.
2. Target Address: The network address where the agent is migrating.

The location table for user agents has five attributes:

1. Agent Name: A unique name that identifies each agent.
2. Physical Address: The network address where the agent is currently executing.
3. MRA Agent Name: The name of the agents' MRA service provider.
4. MEA Agent Name: The name of the agents' MEA service provider.
5. Agent State: The agent mobility state. This attribute is used by the middleware to decide how to route messages. The state attribute takes one of these values: stationary, in transient, arrival, and off-line.

The MEA and the MRA attributes provide the ability to load-balance the message forwarding and events management services. In the simplest environments, a single MEA agent and a single MRA agent carry these services.

The location table for system agents has four attributes:

1. Agent Name: A unique name that identifies each agent.
2. Physical Address: The network address where the agent is currently executing.
3. Agent State: The agent running state.
4. Utilization: The agent utilization load.

System agents inform the LRA of updated states by utilizing the state attribute. It allows for dynamically enabling or disabling of the system agents. The utilization attribute is used by the middleware in the discovery phase as a selection mechanism to select from several system agent candidates. These two columns are optional and are used to provide load balancing and fault tolerance. Optionally the state and utilization attributes can be set to null in which case the LRA agent queries the system agent for state and utilization information. The information returned is cached for future queries. The middleware may retrieve those columns from the LRA or ask the LRA to refresh its information.

2.3.2 The Message Relay Agent (MRA)

The message relay agent is responsible for storing asynchronous messages. The MRA agent buffers messages for agents in transient.

The MRA agent is equipped with a special registry to maintain message information. The registry keeps track of one table. The message table has five attributes:

1. Agent name: A unique name that identifies the recipient of the message.
2. Message ID: A unique serial message identifier used for indexing and text retrieval.
3. Message envelope: The message sender and recipient envelope. Essentially this attribute represents the message header.
4. Message contents: The message body represented by the actual text of the message.
5. Time stamp: Message arrival date and time.

Buffering a message is triggered by an event that is posted by the middleware of the agent in transient. At arrival the middleware may instruct the MRA agent to deliver its messages or it may retrieve the messages itself. The MRA reassembles the message from the message envelope and the message content fields and routes the message to the recipient.

The MRA agent also serves as a messaging board that stores asynchronous messages. Agents independently post and retrieve messages relevant to their task.

2.3.3 The Messaging Events Agent (MEA)

The messaging events agent is responsible for receiving, maintaining and triggering message events.

The MEA agent is equipped with a special registry to maintain event information. The registry keeps track of one table. The event table has three attributes:

1. Agent Name: A unique name that identifies monitored agent.
2. Monitored Event: Registry field triggering callback message.
3. Recipient Name: A unique name that identifies the recipient of the event.

3 Distributing The Registry

One of Mobile Agent systems target application areas are geographically distributed applications. For such applications scalability is a major hurdle.

To scale the system agents, the core component of the agent must be scalable namely the registry. There are at least three approaches for the implementation of the registry. The registry types are contrasted in table 3.1.

Registry Type	Advantages	Disadvantages
Centralized Global Registry	Easy to use and implement.	Does not scale well.
Replicated Registries Everywhere	Easy to use and provides fault tolerance.	Replication may overwhelm the network. Must deal with concurrency and coherency issues.
Distributed (non-overlapping or slightly overlapping) Registries.	Scales well and provides fault tolerance.	Difficult to implement. Must deal with concurrency and mobility issues.

Table 3.1 – Comparing the Registry Types

We model our design of the registry for a non-overlapping distributed registry. In this context, the system agents employ discovery mechanisms to share status and state information among each other and provide mechanisms to find and update the registry entries. The agents collaborate among each other to keep the registries up-to-date. Periodic messages are sent out to indicate agent activity and status. A system agent can probe another for activity status and determine its registry state.

When an agent moves, the registry entries associated with that agent may move to another registry to take advantage of geographical proximity. Upon arrival to the destination machine, the messaging middleware through the LRA agent determines if the registry entries need to be moved to a system agent closer to the new destination. If such host is available, the registry information is copied to the new location, and immediately removed from the previous registry.

The DNS protocol provides a distributed hierarchical registry, but does not address mobility. We model our distributed registry based on the DNS protocol, and provide mechanisms to address mobility.

The remainder of this section is structured as follows. Section 3.1 describes the organization of the location registry. Section 3.2 provides a description of the registry events at the system and user agent levels.

3.1 The Location Registry

The location registry agent is the logically central but physically distributed repository for information about agents. Agents register themselves with the LRA so that other agents may find them. The location registry agent maintains a database that contains descriptions of the capabilities of the agents.

Each agent has a name and belongs to its birth zone. In a single zone environment as described in section 2 only the agent name is significant, however in multi-zone environments the agent name and its birth zone information are necessary to locate the agent. We term the agent name and its zone information as the *agent ID* (*AID*). The format of the AID is:

*Agent-name***:***birth-zone*

The *birth zone* is the place to locate the agent if it can not be found otherwise. The AID provides location transparency; it is independent of the agent physical address, and the agent ID does not change throughout the life cycle of the agent.

Agent names are unique within each birth zone. The agent ID space is the collection of user agent names in all the zones within the execution environment. Zone names are globally unique throughout the environment. The zone name space is the collection of all zone names available.

3.1.1 The Registry Organization

The location registry is organized as a tree hierarchy. The hierarchy employs location zones at the system and user agent levels. Zones are subdivision of the naming space. Zones may represent geographical locations, country codes, data center servers, a collection of LANs (Local Area Networks), or a subset of an organization private network. The primary zones constitute the entire global naming space. Zones are non-overlapping and are organized in a hierarchical tree. Non-leaf registries have a list of LRA agents serving that zone, and leaf registries have a list of user agents that are served by this location registry.

3.2 The Registry Events

In this section we describe the main events that take place in the registry. The events are divided into three categories: LRA agent events, middleware events, and user agent events. The main events for LRA agents and the middleware are startup and termination. The main events for user agents are startup, local name lookup, external name lookup, migration, and termination. Due to space limitation, we only describe details on name lookup and agent migration.

3.2.1 Name Lookup

Name-to-address lookup queries can be for local agents, or agents in a remote zone. From a user perspective, the agent name lookup process is transparent. The process is performed on behalf of the user agent using the middleware. The middleware calls a name-to-address lookup function that queries an LRA agent, which returns the network address of the destination agent to the calling middleware.

At lookup time, the primary zone registry for the agent submitting the request is consulted, if this registry can not be contacted, the replica registry is contacted, if the replica is down, its replica in turn is contacted. If the process fails, we consult the root registry for another agent providing name services for that zone, and we continue this process until some timeout value is reached, or the name lookup has been resolved.

In the next two subsections, we describe in greater details the two scenarios of lookup: local name lookup and external name lookup.

3.2.1a Local Name Lookup

When the name lookup is performed, the client's middleware asks the local LRA agent for the network address of the destination agent. If the agent name is found, then the query is for a local agent. Figure 3.1 illustrates this process. The middleware queries the local LRA agent for information about an agent within the local zone.

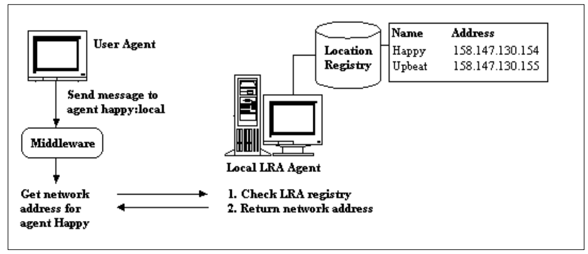

Figure 3.1 – The Local Name Lookup

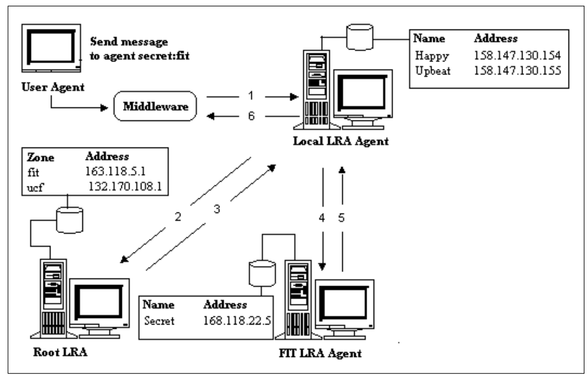

Figure 3.2 – The External Name Lookup

3.2.1b External Name Lookup

The local lookup process is quite simple and straightforward, involving very little work. However, once these lookups are for zones outside of the local zone, they get much more complicated very quickly.

Because the local LRA agent only knows about the local zone, any agent queries for external data must be forwarded to the LRA agents responsible for those external zones. Because the registry is distributed, the remote LRA agent must be located using the LRA queries as well.

When a user agent issues a lookup for a remote zone, it begins by sending a query to the middleware, the middleware in turn forwards the query to the local LRA agent. If the local LRA agent does not have the information, then it will check its top-level zone list and its cache of recently requested zones for the name of the remote LRA responsible for the birth zone in the agent ID, and then issues a request to the remote LRA agent on behalf of the client. If the local LRA agent does not know the network address of the remote LRA agent, then it must issue a query to the root LRA agent asking for the network address of the LRA agent responsible for the remote zone. The root LRA looks up its registry for an LRA serving the zone requested. Once this information is returned, the LRA agent will then issue a query to the remote zone's LRA agent asking for the network address of the destination agent. Finally, this information is returned to the user agent middleware that issued the original query. Figure 3.2 illustrates this process. The local LRA must query remote LRA agent for information about an agent in a remote zone.

3.2.2 User Agent Migration

At the agent migration event the target address of the destination machine is checked by the middleware. The user agent may move within the same zone or to another zone.

3.2.2a Intra-Zone Migration

If the target address of the destination machine belongs to the same zone as the one the agent is currently in, then the agent is moving within the same zone. In this case, no physical move of the registry entries is necessary. The agent network address is updated upon the arrival to the destination machine.

3.2.2b Inter-Zone Migration

To begin migration, the local middleware checks to see if the target host is ready by querying the middleware on the destination machine. If a connection was made, the destination middleware returns its zone name as an acknowledgment to begin the migration process. If the zone name returned is not the same as the local zone name, then the agent is migrating to another zone. In this case, the entries are marked with "in transient" attribute in the birth zone, and at arrival are copied to the destination zone. Once the move is complete, the birth zone registry entries are updated with the new host name.

As an agent moves from one zone to another the registry must be updated to reflect the current location. Several update schemes may be used to maintain the registry records. In all schemes the birth zone records are continuously updated to reflect the current host name. We describe the following update schemes:

1. Greedy Update: In this scheme when an agent moves, its location entry is removed in the local LRA and a new entry is added in the remote LRA. Its entry in the birth zone LRA is updated. That is, only the LRAs of the current zone and the birth zone have the location changes of the agent. There are only 3 changes: One add, one delete, and one update.

2. Deep Update: In this scheme when an agent moves, all the LRA records in zones where the agent has resided are updated. The scheme requires an LRA to store a back link to the previous LRA where the agent resided. By following the back links, all LRAs in previously visited zones are updated. There are $m+1$ updates for m previous migrations, one add is needed for the new residence.

3. Delayed Update: in this scheme when an agent moves, its location entry is updated in the local LRA and a new entry is added in the remote LRA. Its entry in the birth zone LRA is updated. That is, only the LRAs of the previous zone, current zone and the birth zone have the location changes of the agent. There are only 2 changes: One add, and two updates.

The first scheme is the simplest to implement and may increase external lookups while reducing the registry size. The second scheme blindly updates the previously visited zones, reducing external lookups while increasing the registry size. The third scheme amortizes the cost of updating all visited zones over the path of travel by updating two zones for each move, while increasing the registry size. In terms of update cost the first scheme performs best, however, in terms of lookup cost the second scheme is best. The third scheme reduces external lookups at the cost of extra updates. The ideal scheme will vary according to geographic proximity between zones and the application communication cost requirements versus storage requirements for each registry.

4 Simulation Results with Base Agent Behavior

In this section, we describe the AGI framework implementation. We have implemented the system agents and the messaging middleware and created a simulator to validate the AGI framework. The simulator consists of the user agents and the execution environment. Our implementation supports communication within one zone, inter-zone communication will be feasible using the design approach of section 3.

We have measured three metrics to test the simulation. The goal of the measurements is to test the agent architecture and to simulate a real-life execution environment. Furthermore, the experiments are needed to validate the scalability of the system.

The test environment consisted of a network of ten workstations comparably configured. Each workstation had an instance of the middleware, all agents communication were external, and the underlying network speed was 100 Mbits/Second. The interaction among the agents was driven randomly and recorded in a script for multiple replay of each test.

In the next section we describe each metric in greater detail.

1. Message Throughput

The goal of this experiment is to measure the overall aggregate message throughput rate of the system. Throughput is defined as the number of messages the system can handle within a sample interval of one minute. The metric is given by this formula:

$$\textit{Message Throughput} = M_s / I_s$$

Where:

M_s = Number of messages delivered.
I_s = Sampling interval in minutes.

In this experiment we vary two parameters: Number of agents, and number of messages per agent. The number of messages varies from 1 to 1024 messages, the number of agents was fixed at 10, 50, 100, and 150. The message length of 2K was used, and a sample interval of 1 minute.

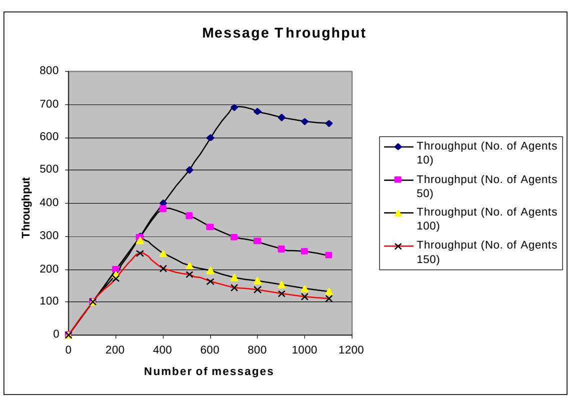

Figure 4.1 – Message Throughput: Effect of the number of messages

Figure 4.1 shows a plot of message throughput versus the number of messages in the system. Each line represents the throughput with a fixed number of agents 10, 50, 100, and 150 respectively. The results show that the throughput peak point is mainly related to the number of agents. The smaller the number of agents the higher the peak point value; however the throughput decline rate appears to be independent of the agents size and the number of messages in the system. The throughput decline rate is uniform regardless of the two parameters in this experiment.

2. Message Latency

The goal of this experiment is to measure the delays in transmission of messages. Latency is defined as the delay in receiving a message due to traffic, overhead, etc. The metric is given by this formula:

$$\begin{aligned}\textit{Message Latency} &= T_t - T_{itf} \\ &= T_t - M_l / N_S\end{aligned}$$

Where:

T_t = Total end to end time
T_{itf} = Ideal Transfer time = M_l / N_S
M_l = Message Length
N_S = Network Speed

In this experiment we vary two parameters: Number of messages and length of message. The number of messages varies from 1 to 1024 messages, the message length was fixed at 8 bytes, 2K, 4K, and 8K.

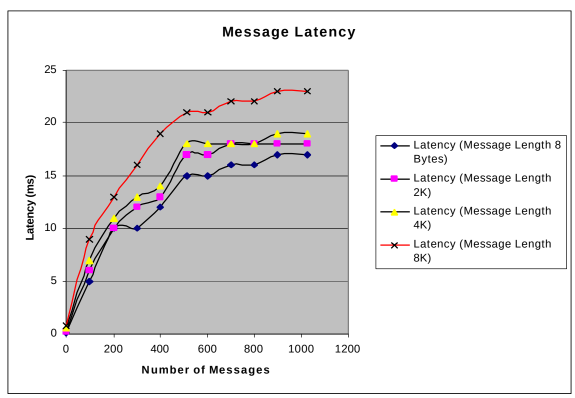

Figure 4.2 – Message Latency: Effect of the number of messages

Figure 4.2 shows a plot of message latency versus the number of messages in the system. Each line represents the latency with a fixed message length of 8 bytes, 2K, 4K, and 8K respectively. As one can see, as the number of messages is increased the overall latency is increased. The increase is proportional to the message size. The results then show that the overall latency is not directly related to the number of the messages in the system, but not surprisingly, it is related to the message size.

3. Messaging Middleware Overhead

The goal of this experiment is to measure the cost of the middleware on communication. The overhead is defined as the delay added by the messaging middleware (MMW) to the communication cost. The metric is given by this formula:

$$\textit{MMW Overhead} = (S_{mwo} + R_{mwo}) / T_t$$

Where:

S_{mwo} = The time elapsed while the message is being processed by the sender's middleware.

R_{mwo} = The time elapsed while the message is being processed by the receiver's middleware.

T_t = The end to end time from the time the message enters the sender's middleware until it leaves the receiver's middleware as depicted in figure 4.3.

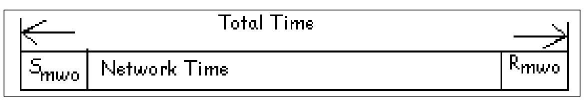

Figure 4.3 – The middleware overhead metric

In this experiment we vary two parameters: Number of agents, and length of message. The number of agents varies from 1 to 150, while the message length was fixed at 8 bytes, 2k, 4k, and 8K.

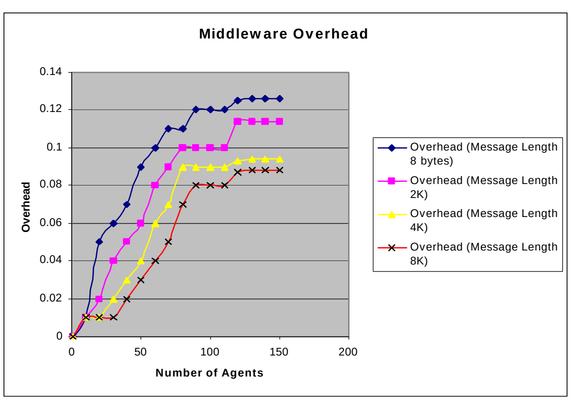

Figure 4.4 – Middleware Overhead: Effect of the number of agents

Figure 4.4 shows a plot of the middleware overhead percentage versus the number of agents in the system. Each line represents the overhead with a fixed message length of 8 bytes, 2K, 4K, and 8K respectively. As one can see, the longer the message, the lower the middleware overhead. Not surprising. What is interesting is that the overhead appears to stay about the same for messages of 8K or larger.

5 Related Work

In this section, we discuss research efforts related to this paper.

Baumann discussed two communication concepts based on session and global event management [Baumann98]. While the communication concepts introduced in Baumann where general, it did not allow for the mobility of the agents while in active communication. Our scheme allows an agent to move while in active communication and provides for message buffering and forwarding.

Dong in [Dong98] proposed a communication framework from a language perspective, based on the various types of cooperation among the agents. While Reed in [Reed98] introduced a framework based on dialogue types and the distinction between persuasion and negotiation. Also, D'Iverno Agentis framework [D'Iverno98] is based upon a model of agent interaction whose key element is services and tasks. Our scheme is general and application neutral, and employs high-level communication mechanisms to provide agent to agent communication and group collaboration. Our scheme is independent of the agent task and any agent group classification

Chess discussed communication portals that are responsible for managing the arrival and departure of itinerant agents. The portal may support either session-oriented connection or messaging based protocols [Chess95]. Rus's transportable agents have network-sensing tools that allow the agent to adapt to the network configuration and to navigate under an alternate plan [Rus97]. Our scheme borrows from Rus's concepts, and allows the agent to dynamically acquire or offload communication components as it moves through its life cycle.

Tambe have studied the problem of agent tracking in multi-agent worlds. Although, the paper [Tambe96] discusses the ability of one agent to execute models of another agent, and provides for dynamic and simultaneous execution of models, the architecture does not address communication issues, but instead assumes a high bandwidth inter-model communication.

6 Conclusion and Future Work

Mobile agents have several advantages over the traditional client/server model. Mobile agents consume fewer network resources since they move the computation to the data rather than the data to the computation and do not require a continuous connection between machines. Mobile agents allow clients and servers to extend each other's functionality by programming each other.

There are many alternative techniques to mobile agents such as queued RPC, proxy servers, etc. that have many of the same advantages. The problem with these techniques is that each one is only suitable for certain domain specific applications [Harrison95, and Green97]. A mobile agent system on the other hand is a unified framework in which a wide range of distributed applications can be implemented easily and efficiently.

Mobile agents offer a new paradigm for very large scale distributed heterogeneous applications. The paradigm focuses on the interactions of autonomous, cooperating and adaptable processes. Communication is of central importance to agents, and, in particular, establishing common agent communication languages and protocols is essential. This paper argues that if mobile agents are to successfully use complex and dynamic networks, they must obtain architectural support for remote agent communication – an important capability required for agent interactions. The key implications of agent communication for agent architectures include open and flexible framework, extendable and modular communication models, and the ability to communicate while in migration.

One of the main contributions of this research is an open communication architecture based on communication types. The AGI (**A**synchronous, **G**roup Oriented, and **I**nter-agent communication) architecture is an open framework not tied to a particular agent execution environment, a particular implementation, or the underlying network protocol. Furthermore, the framework is based on components employed through out the framework. This design choice provides for scalability and adaptability. Well-defined interfaces are employed among the components to create a robust framework.

We have built an initial implementation of the AGI agent communication architecture. Agents based on this implementation have been used in a simulated environment. We ran several automated agent plan tests. Our initial results show that the AGI architecture is capable of providing conventional and mobile messaging. Furthermore, our tests show that the overhead of the middleware is negligible and does not impact the overall system performance.

Among issues for future work, we shall integrate the design approach of section 3 for an implementation of a distributed registry in which the location registry is distributed across many location zones.

Other issues for future work, we are looking into an implementation for a collaboration application in which participants may subscribe to message channels and listen for events. The application will provide for active conversation, background conversation, and passive interactions among participants. The application will consist of a client component, and a distributed server component.

Other future works involve integration with commercially available agent execution environments. Current state of the arts execution environments includes Object-Space Voyager, IBM's Aglets, Agent Tcl, and Mitsubishi Concordia. One problem with these execution environments is that the source code is not available due to their commercial nature. As a result augmenting and enhancing the built in communication mechanism may not be feasible for research purposes. Nevertheless, we shall study the viability of the current execution environments and compare their mobile communication features. Additionally, we shall investigate the role of Java RMI and object serialization in providing mobile agent communication facilities.

References

[Baumann97A] J. Baumann, F. Hohl, K. Rothermel, and M. Strasser: Mole – Concepts of a Mobile Agent System, submitted to WWW Journal, Special issue on Applications and Techniques of Web Agents, 1997.

[Baumann97B] J. Baumann, and N. Radouniklis: Agent Groups in Mobile Agent Systems, The IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems, DAIS'97.

[Baumann98] J. Baumann, F. Hohl, N. Radouniklis, K. Rothermel, and M. Strber: Communication Concepts for mobile agent systems, Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany, April 1997.

[Chess95] Chess, D. et al: Itinerant Agents for Mobile Computing, IEEE Personal Communications, Volume 2, Number 5, Pages 34-49, October 1995.

[d'Iverno98] M. d'Inverno, D. Kinny and M. Luck: Interaction Protocols in Agentis, Third International Conference on Multi Agent Systems, 1998.

[Dömel97] P. Dömel, A. Lingnau and O. Drobnik: Mobile Agent Interaction in Heterogeneous Environments, First International Workshop on Mobile Agents Berlin, Germany, April 1997.

[Dong98] H. Dong, J.H. Ding, X. Li and J. Lu: On Open Communication Framework for Software Agents, Proceedings of the Technology of Object-Oriented Languages and Systems, 1998.

[Green97] Green, S. et al: Software Agents: A review, IAG Technical Report, Trinity College, May 1997.

[Harrison95] C. Harrison, D. Chess and A. Kershenbaum: Mobile agents: Are they a good idea?, IBM Research Report, IBM T.J. Watson Research Center, 1995.

[Reed98] C. Reed: Dialogue Frames in Agent Communication, 3rd International Conference on Multi-Agent Systems (ICMAS 1998).

[Rus97] D. Rus, R. Gray, and D. Kotz: Transportable Information Agents, International conference on autonomous Agents, Feb. 1997.

[Tambe96] M. Tambe & P.S. Rosenbloom: Architectures for agents that track other agents in multi-agent worlds, Intelligent Agents, Vol II Springer Verlag Lecture Notes in Artificial Intelligence (LNAI 1037), 1996.