行政院國家科學委員會專題研究計畫 期中進度報告

# 一個以 BDI 為基礎的多個代理程式系統之視覺化開發環境

(1/3)

計畫類別：個別型計畫
計畫編號：NSC91-2213-E-009-085-
執行期間：91 年 08 月 01 日至 92 年 07 月 31 日
執行單位：國立交通大學資訊工程研究所

計畫主持人：王豐堅

計畫參與人員：許嘉麟等研究生

報告類型：精簡報告

處理方式：本計畫可公開查詢

中 華 民 國 92 年 6 月 24 日

# 行政院國家科學委員會專題研究計畫期中報告

計畫編號：NSC 91–2213–E–009–085

執行期限：91 年 8 月 1 日至 92 年 7 月 31 日

主持人：王豐堅　　國立交通大學資訊工程系

計畫參與人員：許嘉麟等研究生

## -. 摘　要

「多個代理程式系統發展方法」對於分散式系統開發工程提供顯著的優點，例如：提高「互通性」、「擴充性」和「可重新組態性」。以代理程式爲基礎的解決方案是實用且吸引人的，因爲方案中的各種代理程式知道如何做許多事情。例如：代理程式知道如何與其它代理程式溝通，這使得系統發展者不再需要設計通訊協定和訊息傳遞的格式。這類的能力都被當作是基本的代理程式機制而提供。本計畫第一年的工作主要是分析現有的 Belief-Desire-Intention(BDI) 智慧型代理程式平台，並研究如何擴充現有的 BDI 代理程式系統，加入行動能力、名稱解析、代理程式之間的通訊、協調、合作等機制，訂定一套「擁有多個代理程式環境能力」的新 BDI 代理程式架構之規格並加以實作。另一方面，分析現有的開發多個代理程式的方法論，設計一套適合上述新 BDI 代理程式平台的開發多個代理程式系統的方法論。

**關鍵字**：行動代理程式、智慧代理程式、代理程式發展平台、代理程式導向

## Abstract

The multi-agent approach promises significant benefits to programming of distributed systems, such as enhanced interoperability, scalability, and reconfigurability. An agent-based solution is useful and attractive because the agents used in the solution inherently owns several capabilities. For example, agents can communicate with other agents. The system developers need not design communication protocols and message formats, because they are the agent's basic mechanisms. In the first year of the project, we have designed and implemented a new mobile BDI agent architecture and platform providing the essential capabilities, such as mobility, directory service, and inter-agent communication, collaboration, and coordination for multi-agent systems. Furthermore, we also proposed a multi-agent development methodology suited for the new BDI agent architecture and platform.

**Keywords**: mobile agent, intelligent agent, agent development platform, agent-oriented

## 1. Introduction

The multi-agent approach promises significant benefits to programming of distributed systems engineering, such as enhanced interoperability, scalability, and reconfigurability. An agent-based solution is useful and attractive because the agents used in the solution inherently owns several capabilities. For example, agents can communicate with other agents. The system developers need not design communication protocols and message formats, because they are the agent's basic mechanisms. Agents have the inherent capability to build the model of their environment, monitor the state of that environment, reason and make decisions based on that state. All the software developer needs to do is to specify what the agents do systematically.

Mobile agent technology has received a great deal of interest over the past few years in both academia and industry. Most models of mobile agent systems are mainly based on the concepts of agents, agent servers (also called agent systems or agent platforms) and places. A mobile agent is a software module responsible for executing some tasks that can autonomously migrate from place to place in a heterogeneous network. The state and code of an agent are transferred to the new place when the agent migration occurs. An agent server provides the execution environments, called places, for the safe execution of local agents as well as visiting agents. The server provides various functions, such as agent transport, security, communications of agents with the host, other agents, and their owners, and fault tolerance.

On each site, an agent interacts with other stationary agents, mobile agents and local resources in order to accomplish its tasks. Mobility and communication are important factors for cooperation among agents. Mobile agent platforms need an efficient and reliable communication system. Such a communication system needs a good procedure for locating mobile agents and a reliable message delivery mechanism. In general, both mechanisms are complicated than those in traditional distributed systems since agents may migrate anytime. For example, the receiver agent might migrate to another host after the message sender agent gets its location information. Since the location information is overdue now, the delivery mechanism needs additional efforts to deliver the message.

Most existing intelligent agent architectures use Belief-Desire-Intention (BDI) architecture. The BDI approach is based on the study of mental attitudes and tackles the problems arising when trying to use traditional planning in situations requiring real-time reactivity. However, the disadvantages of existing BDI architectures are that they mainly focus on single agent and thus lack mobility and some essential capacities in multi-agent environments, such as directory service and inter-agent communication, collaboration, and coordination. The objective of this report is to design a new BDI agent architecture and platform that support these features.

In this report, session 2 discusses naming, location tracking, and inter-agent communication issues in mobile agent systems. Session 3 presents our new mobile BDI agent and agent server architectures with mobility, directory service, and inter-agent communication support. Session 4 introduces our methodology for the development of multi-agent systems based on our platform. In the final session, we draw a conclusion and suggest the future work.

## 2. Design Issues for Supporting Mobility

### 2.1 Mobility

A mobile agent consists of code, state, and attributes. Mobile agent's code is a program that defines agent's behavior. Besides code, the state of the agent contains all contents and values of the agent's runtime state and object state. The state contains all the information required to resume execution at the suspended point after migration. The third part of a mobile agent consists of attributes that describe the agent, its requirements, and its history for the infrastructure. They include data such as a unique agent identifier, the agent's owner, error messages, and movement history.

An agent can request its host server to transport it to a remote destination. After receiving the commands, the agent server deactivates the agent, captures its state, and transmits it to the server at the remote host. The destination server then restores the agent state and reactive it at the remote host, thus completing the migrating.

In order to let code be executed on heterogeneous machines, there are many languages used for implementing agent system before Java was shown in the world [1]. For example, Agent Tcl [3] and Ara (Agent for remote access) [4] are based on the Tool Command Language, and Telescript [5] is from General Magic [10] Inc. Java is an appropriate choice of languages for agent systems, because it has some features not found in other languages for supporting mobile agent systems. For example, by using object serialization in Java, objects can be easily "serialized" and sent over the network or written to disk for persistent storage.

### 2.2 Directory Service: Naming scheme and Name Service

**A naming scheme and a name service are needed for addressing various entities in a mobile agent system, such as hosts, agent servers, agents, and other global resources. A naming scheme is location transparent [12] if the agent name does not contain any site-specific information. For example, a name comprising the site to which the agent belongs plus an agent identifier (e.g. dssl.csie.nctu.edu.tw/MyAgent) is not location-transparent. On the contrary, a naming scheme according to an agent's properties or functions may be location transparent. An example is the name "MySearchAgent". Furthermore, a naming scheme is location independent if an agent's name does not change after being created and might be used to identify and reach the agent independently of its current location. Note that "location-independent" property does not indicate that an agent name cannot contain any site-specific information. For example, an agent name might contain the name of its creator server to record the current location of the agent. Once an agent migrates, its current location record is updated**

**correspondingly. Thus, by contacting the creator server, it is possible to locate the agent much easier. This scheme is location independent but not transparent, as it requires the name presence of the creator server to form an agent name.**

Location-dependent naming schemes may allow simpler implementation of name service systems than location-independent ones. Platforms like Agent Tcl, Aglets, and Concordia use location-dependent scheme to name agents. In these systems a mobile agent is named based on the host name, port number, and an identifier. Name resolution is based on DNS. When an agent migrates, its name would change. However, the location-dependent naming schemes make the implementation of agent location tracking cumbersome. On the contrary, a location-independent naming scheme requires a name service to map the symbolic name to the agent's current location. A simpler solution is to use a unique name server to keep track of all agents. However, this is only suitable for small scale systems and lacks of scalability. Another approach taken by Voyager design uses proxies object. A remote mobile agent is located by contacting its creator server to obtain its local references called proxies. Such proxies are updated by the runtime environment when the agent migrates. However, this method creates a strong binding between application level names and network level names and raises the issue of performance if there are a lot of proxies for an agent in the network.

Because entities such as agents are mobile in the network, it is desirable to allow accessing them in a location independent manner. Table 1 gives examples of possible naming schemes for agents and in which platforms that have been implemented.

| Scheme | Example | Transparency | Independence | Used by |
|---|---|---|---|---|
| Properties or Functions | MyTestAgent | Yes | Yes | |
| Home-Host + Name | Home host/TestAgent | No | Yes | Ajanta |
| Home-Host + ID | Home host/756721 | No | Yes | ARCA, MOA |
| Current-Host + Name | Current host/TestAgent | No | No | Voyager, AgentTcl |
| Current-Host + ID | Current host/756721 | No | No | Aglets, Concordia |

Table 1: Naming Scheme

### 2.3 Inter-agent Communication

The design challenges for inter-agent communication mechanisms arise due to the mobility of the agents. There are several design choices: connection-oriented communication (such as TCP/IP), connection-less communication (RMI, RPC, or CORBA-IIOP), or indirect communication based on the event publisher/subscriber model and shared mailboxes or meeting objects. In TCP/IP or RMI based communication, agents need to know each other's network address in order to establish communication. Connection-oriented schemes raise the issue of connection disruption due to a participating agent's migration. In other words, connection-oriented communication can be location-independent as long as the agents do not move during communications. Otherwise a new connection for communications has to be established. In comparison, RMI based request-reply model throws an exception when a remote invocation fails due to the migration of the server agent; the client agent only need to re-execute the lookup and binding protocol to re-establish communication with the migrated agent at its new location. However, it may become hard to establish communication if the invoked agent moves very frequently. The indirect communication model using stationary objects to hold events/messages/tuples is more appropriate for such cases. The tuple-space mode is suitable for agent coordination, but not applicable for bulk data exchange.

Several systems (such as Aglets, Grasshopper, and Voyager) have supported synchronous, asynchronous with a reply, and asynchronous without a reply (or one-way) communication models. When an agent sends a synchronous message, its thread blocks until the receiver replies to the message. When sending an asynchronous message, the agent does not block and a return handler is used to get the reply via waiting, polling, or call back. The last type of message is one-way message, i.e. asynchronous without a reply; it is useful when a message sending agent that does not expect reply from message receiving agent. Table 2 compares several existing mobile agent systems according to naming, transport protocol, and inter-agent communication issues.

| System | Naming | Transport protocol | Communication protocol | Messaging modes | Multicast messages |
|---|---|---|---|---|---|
| Aglet | URLs based on DNS names. (location-dependent) | ATP based on TCP/IP | Plain TCP/IP and RMI | Synchronous, asynchronous, and one-way | Yes |
| Concordia | Location-dependent (based on DNS). Directory service | RMI | RMI | Synchronous and asynchronous, based on | Yes |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
|  |  |  |  | collaboration points |  |
| D`Agents (Agent Tcl) | Location-dependent name based on DNS, and optional symbolic alias. | Proprietary protocol based on TCP/IP or on e-mail | Proprietary protocol based on TCP/IP or on e-mail | Asynchronous and one-way | No |
| Grasshopper | Location-independent global names. | Plain TCP/IP+SSL, RMI+SSL, CORBA IIOP and MAF IIOP | Plain TCP/IP+SSL, RMI+SSL, CORBA IIOP and MAF IIOP | Synchronous, asynchronous, and one-way | Yes |
| Odyssey | Use process name | RMI, CORBA IIOP and DCOM | RMI, CORBA IIOP and DCOM | Through meeting places | No |
| Voyager | Location-independent global ID, as well as local proxy. | Proprietary Voyager ORB based on TCP/IP and CORBA | Proprietary Voyager ORB, by use of messengers | Synchronous, asynchronous, and one-way | Yes, enhanced |

Table 2: comparisons of naming, transport protocol, and inter-agent communication

## 3. Mobile BDI Agent Architecture & Platform

### 3.1 Mobile BDI Agent Architecture – MBDI Agent

The intelligent agent architecture we have proposed is based on BDI-theories [16]. A generic agent is shown in Figure 1. The generic agent includes the following components:

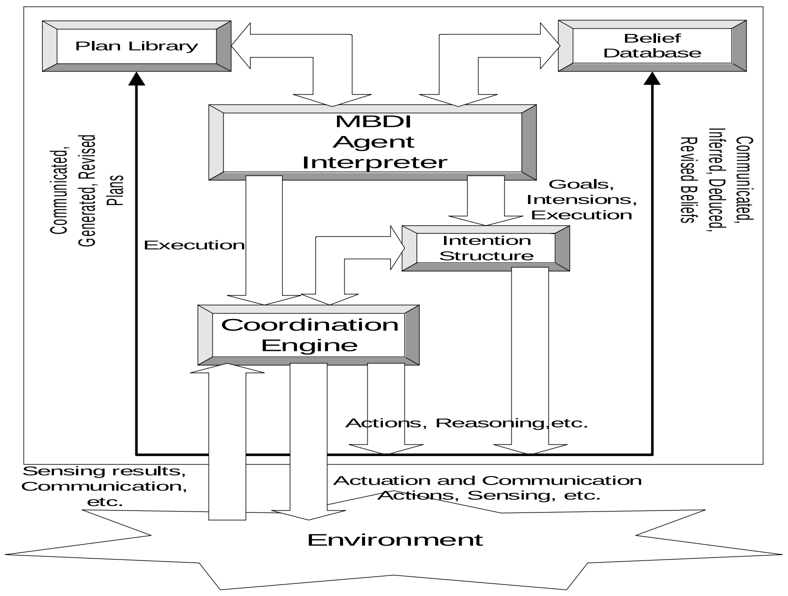
Plan Library
Belief Database
MBDI Agent Interpreter
Communicated, Generated, Revised Plans
Communicated, Inferred, Deduced, Revised Beliefs
Goals, Intensions, Execution
Execution
Intention Structure
Coordination Engine
Actions, Reasoning, etc.
Sensing results, Communication, etc.
Actuation and Communication Actions, Sensing, etc.
Environment

Figure 1: MBDI agent architecture

- **Belief Database** represents what the agent knows about itself and the world. For example, the belief may contain information that describes the agent's relationships with other agents and the capabilities of those agents.
- **Goals** specify what the agent is trying to achieve.
- **Plan Library** defines the

sequences of actions and tests to be performed to achieve a certain goal or react to a specific situation.

- **Intention Structure** contains those plans that have been chosen for eventual execution.
- **Interpreter (Reasoner)** selects appropriate plans based on agent's current belief, goals, and intentions. Then the interpreter places the selected plans on the intention structure, and executes them.
- **Co-ordination Engine (CE)** is a lightweight plan that the agent executes between plan steps. CE is responsible for processing the incoming/outgoing messages and events that coordinating the agent's interactions with agent servers and other agents by using the information stored in the belief database. The CE manages the agent's problem solving behavior, especially for those involving multi-agent collaboration, i.e. team plans. It provides several predefined co-ordination protocols, such as master-slave for delegating tasks to subordinate agents, contract-net for contracting tasks out to peer agents, and various auction protocols for buying or selling resources. The CE also provides a number of methods that enable to customize the behavior of the CE.

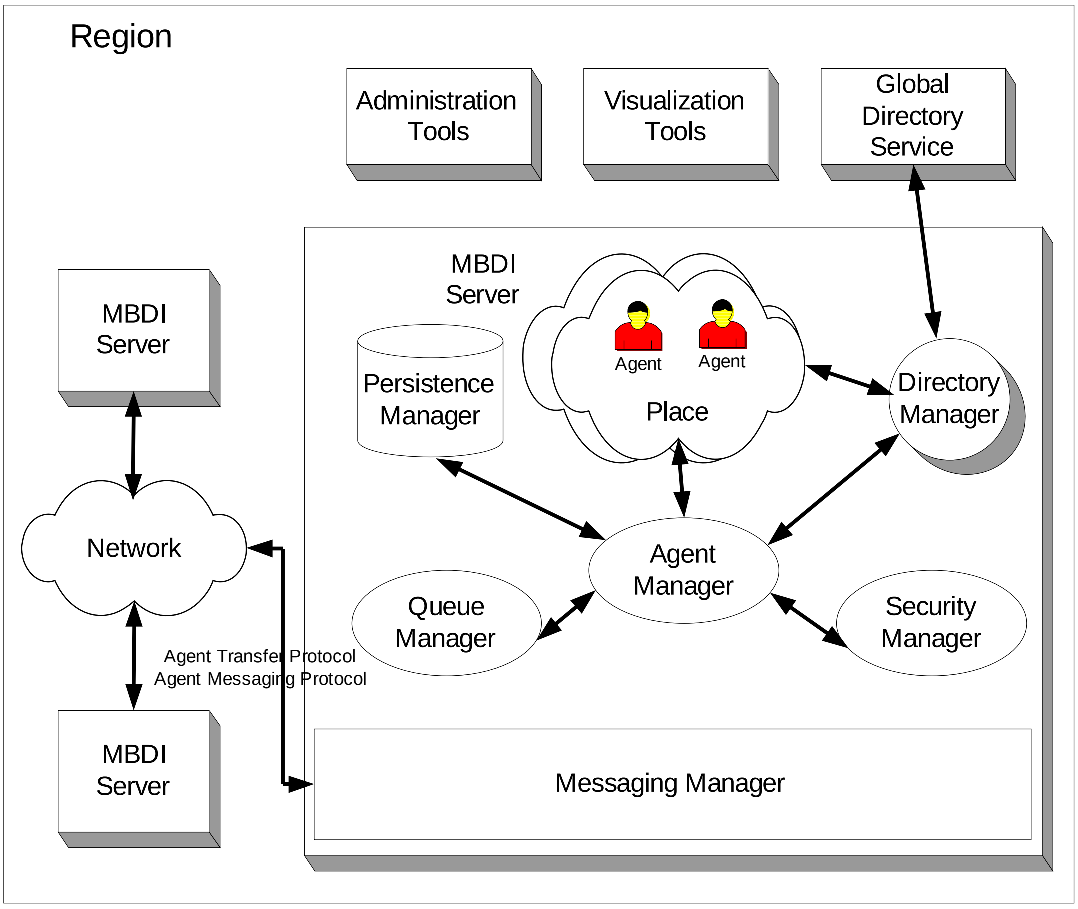

Figure 2. MBDI Server architecture

## 3.2 Mobile BDI Agent Platform – MBDI Server

Our Mobile BDI Agent Platform – MBDI Server provides the environment called *place* for the safe execution of local agents as well as visiting agents. The MBDI Server provides various functions, such as agent management, transport, security, communications of agents with the host, other agents, and their owners, and fault tolerance. The MBDI Server accepts incoming agents, authenticates the identity of the owner, and passes the authenticated agent to the execution environment. The MBDI Server also keeps track of the agents running on its machines and answers queries about their status. Furthermore, the MBDI Server allows an authorized user to suspend, resume and terminate a running agent, and allows agents to communicate with each other through message passing and direct connections. A MBDI server can join a region. A region is a set of agent systems that have the same authority, but not necessarily of the same agent system type. Global directory service is provided if a MBDI Server joins a region. As in Figure 2, MBDI Server consists of the following components:

- **Agent Manager (AM)** is responsible for providing fundamental agent management operations. The operations include the creation, suspending, resuming, transferring, receiving, termination, and getting status of agents.
- **Directory Manager (DM)** provides local directory service. DM maintains two lists of agents, one is for the agents that are created locally and the other is for visitor agents. DM provides white pages (address books) services to agents. Each newborn agent created locally must register with local DM in order to get a valid and unique name and the DM is also in charge of keeping the location of agents created locally. An agent may register with the DM to announce its address or query DM to find out current locations of other agents created locally. When the agent migrates to another MBDI Server, the agent informs the DM to update its location information. DM also provides yellow pages services to agents. An agent may register with the DM to announce its capabilities or query the DM to find out what capabilities are offered by other agents. For an MBDI Server joining a region, the global directory service is described in latter session.
- **Security Manager (SM)** is responsible for protecting hosts and agents from malicious entities. SM provides mechanisms for identifying users, authenticating and authorizing their agents, and data encryption.
- **Persistent Manager (PM)** is required to ensure that the agents can recover from the MBDI Server crash successfully. It maintains the state of agents in transit around the network. As a side benefit, it allows for the checkpoint and restart of agents in the event of MBDI Server failure. Additionally, it can checkpoint objects upon request by agents, to provide finer granularity of reliability guarantees for critical procedures.
- **Queue Manager (QM)** is responsible for the scheduling and guaranteed delivery of mobile agents between MBDI Server.
- **Message Manager (MM)** is responsible for managing incoming/outgoing messages and events. The MM has two queues, one for incoming messages/events, and the other for outgoing messages/events. For incoming messages, the MM forwards these messages one at a time to the corresponding object. It ensures

that the next message is not forwarded until the current message has been handled. For an outgoing message, it will be kept in the outgoing queue until it has been delivered successfully or failurely in a fixed time.

### 3.2.1 The mobility

—The agent can migrate with its own will, forced by another authorized agent, or by the agent system or the user via the GUI. Our system provides two basic mobility patterns, sequential and parallel migrations, which can be used to derive (other) different travel plans.

- Sequential Migration:

Here an **itinerary** object maintains a list of destinations, including the next the agent will move to, defines a routing scheme, and handles special cases such as what to do if a destination place does not exist. Objectifying the itinerary allows programmer to save and reuse it later. It is similar to use bookmarks.

- Parallel Migration:

The **Master-Slave** Pattern [2] allows an agent to spawn several *slave* agents, which move to the places of different locations for execution in parallel. A slave agent is delegated a task by the master agent. After finishing its task, the slave returns to the place created to report the results to the master agent.

### 3.2.2 Naming scheme

Our naming scheme contains three characteristics:

1. Adopt hierarchical naming that provides ease of maintenance and delegation of namespaces.
2. Provide location independent naming for mobile object.
3. Use existing name resolution infrastructures.

To comply with the location independent naming requirement of mobile objects, we adopted Uniform Resource Name (URN) scheme. A URN is a persistent, location-independent resource identifier that can be used for accessing the characteristics of the resource, or the resource itself. The format is as follows:

URN:MBDI:NS/SubNS

Here "MBDI" is the Namespace Identifier (NID) and "NS/SubNS" is the Namespace Specific String (NSS). The NID indicates the unique name-space for which a URN is created. In our case, it is "MBDI", which stands for our Mobile BDI Agent Name Space. The representation of NSS part is NID specific. We use a hierarchical naming where sub name-spaces are separated by a slash, '/'. For example, an agent server's NSS part is RegionName/AgentServerName and an agent's NSS part is RegionName/AgentServerName/LocalAgentName. For the agent name, RegionName represents the region where the agent was created. We refer RegionName as the agent's *home region*. AgentServerName is the name of the agent server that the agent is created and LocalAgentName is the name of the agent chosen by generator or programmer. The unique name requirement imposes that there is no more than one agent, which is born in the same agent server, to have the same LocalAgentName. The string expressing the agent's characteristics can be used for agent's LocalAgentName for better readability. The following is a sample agent name: URN:MBDI:CsieRegion/MyAgentServer/MySellerAgent.

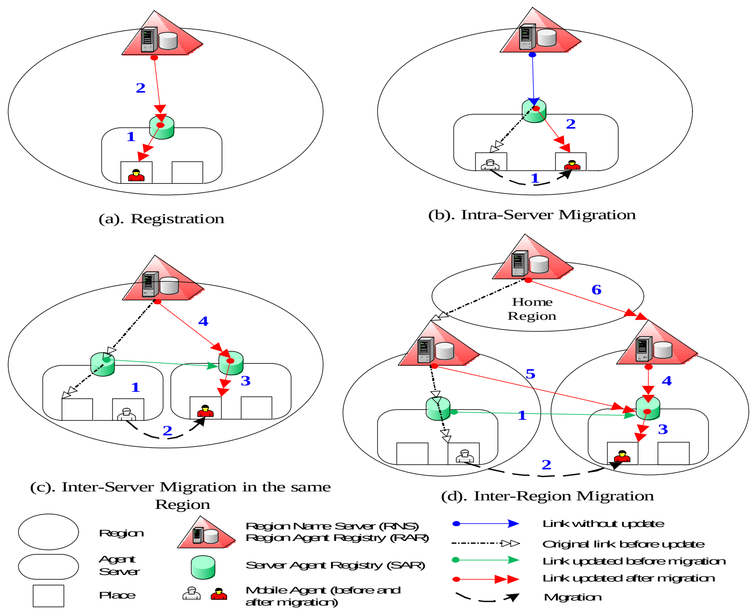
(a). Registration
(b). Intra-Server Migration
(c). Inter-Server Migration in the same Region
(d). Inter-Region Migration
Home Region
Region
Agent Server
Place
Region Name Server (RNS) Region Agent Registry (RAR)
Server Agent Registry (SAR)
Mobile Agent (before and after migration)
Link without update
Original link before update
Link updated before migration
Link updated after migration
Migration

Figure 3: Location management

### 3.2.3 The Mechanism to Track an Agent in a Region

In each Region there are one or more hosts acting as Region Name Servers (RNS) for that Region to provide global naming service. Each RNS contains a Region Agent Registry (RAR), the entry in which provides the location where the target agent is to be found. Similarly, each agent server's Directory Manager (see session 3.2) contains a Server Agent Registry (SAR) used to store the place name where the target agent is to be found. Both RAR and DAR contain information about all the agents that have been created in their scope or have transited through their scope.

An entry of RAR is in the form of {AgentName, CurrentLocation, IsHome, IsMigrating}. AgentName is our URN name of an agent. IsHome indicates that whether the agent is created in this region or just a visitor transiting through this region. Moreover, IsMigrating is used to indicate whether the agent have started to migrate or not. If IsMigrating is false, the CurrentLocation stores URN name of the region or agent server where the agent stays currently; otherwise, it indicates the target region or agent server to where the agent have started to migrate. Similarly, an entry of SAR is also in the form of {AgentName, CurrentLocation, IsHome, IsMigrating}. The only difference is that the CurrentLocation of SAR stores the URN name of place where the agent is to be found.

The management of agents' location information in our system could be described in four separate phases: *registration*, *migration*, *getLocation*, and *deregistration* phase.

- *Registration(Deregistration)*
  When an agent is created or arrives at an agent server, the agent will perform a registration operation to declare its existence. The agent registers its birth in the former case (birth registration) while it registers as a visitor in the latter case (visitor registration). Both cases use the same procedure. The difference is the information used to register: the field IsHome is set to True in the former case while it is set to False in the latter. The registration operation is as follows. First, the agent registers its name and current place name in the SAR of current agent server and then

registers its name and current agent server name in the RAR of current region. Note that for birth registration, the SAR will check the agent name to guarantee the requirement of name uniqueness (see Figure 3(a)).

- *Migration*
  The operations carried out during this phase depend on whether the source place and destination place belong to the same agent server, different agent servers but the same region, or different agent servers and regions. For intra-server (the same agent server) migration, only the SAR is updated with the new place name (Figure 3(b)). For inter-server migration in the same region, before migration, the agent updates the SAR with the URN name of destination agent server. After migration, the agent performs a visitor registration (Figure 3(c)). For inter-region migration, firstly, the agent updates the SAR with the URN name of destination agent server. After successful migration, the agent performs a visitor registration. Finally, the agent updates the RAR of source region with the URN name of destination agent server and the RAR of the agent's home region with the URN name of destination region (Figure 3(d)).
- *getLocation*
  The *getLocation* procedure follows the links that the agent has left in the agent registry on the regions or agent servers it has visited.
  1. The name of agent's home region is extracted from its URN name and the existing resolution framework of the Domain Name System (DNS) for URN resolution is used to find the relative RNS of the agent's home region.
  2. The RNS is contacted and the RAR entry for the agent is retrieved. This (Home) RAR entry contains an indication of
     A. The target agent server the agent is on if it is still within its home region.
     B. The target region the agent is on if it has migrated to the outside of its home region.

  In case A, the agent then contacts the target agent server's SAR to find on which place the agent stays, whereas in case B the RNS of target region is contacted and then the similar procedure in case A is repeated until the place on which the agent stays is found.
- *Deregistration*
  When the agent terminates, it informs the home SAR and the home RAR to clear the entry of the agent.

If all the update operations in the migration phase have been successful, the target agent can be found by at most six messages (Request/Reply with home RAR, current RAR, and current SAR).

### 3.2.4 Communication

In our system, agents can send messages either synchronously or asynchronously, locally or remotely, peer-to-peer or multicast. Messaging is done through the passing of message objects. The message object can contain anything, from a simple data type to a serializable object. The actual message passage is performed by obtaining a reference to the receiver agent object via the name service and then calling *sendMessage* method with the message object as an argument. A *messenger* component is responsible for reliable message delivery. The receiver collects the messages in a queue managed by the receiver's *messageManager* component. Through the messageManager, priority levels of messages can be set for faster processing for messages of more importance. When the receiver invokes its *handleMessage* method, the message at the head of the

queue is dequeued and processed. After the message response has been determined, the receiver invokes one of several *sendReply* methods of the message object. These methods take the reply as an argument and send it automatically back to the original sender; addressing and location are transparent and handled automatically by the name service.
The messaging system supports three types of message modes: (1) synchronous, (2) asynchronous with a reply, and (3) one-way, i.e. asynchronous without a reply. These messaging mechanisms work not only with the agents running in the same place, but between remote agents as well. However, care must be taken to ensure that the sender knows where the intended receiver is and the message is guaranteed to be delivered to the receiver, which might even migrate anytime. It is done in our system through the name service and messenger.

## 4. A Methodology for Analysis & Design of Multi-Agent Systems

Our methodology is based on role models. A role model, a collection of roles, is an abstraction for modeling and designing multi-agent systems. A role is an abstract description of an agent's expected function and encapsulates the system goals that it has been assigned the responsibilities of fulfilling. In our methodology, a role is described by five attributes: *authorities, responsibilities, collaborators, relationships,* and *protocols.* A role's *Authority* identifies the person or organization for whom the role acts and the rights associated with the role. The *Responsibilities* of a role define what the role should do, i.e. the system goals that a role should achieve. In other words, a responsibility of a role defines a relationship between a set of roles (the role and its collaborative roles if any) and a goal to be achieved. The *collaborators* of a role are the roles which interact with the role. The *Relationships* of a role identify correlations and dependencies among the role and other roles. The methodology for the analysis of multi-agent systems is summarized as follows:

**Analysis phase:**

1. Identify key roles of the application domain according to organizational or functional views. There are two kinds of roles: domain independent roles and domain specific roles. Domain independent roles, such as information maintenance and concurrent control, can be reused through minor changes. The major task is to analyze domain specific roles.
2. Identify the authority and lifetime of each role and analyze dependency and relationship between roles to develop a role hierarchy.
3. For each role, identify its associated responsibilities. Each responsibility designates the role and associated collaborators (if any) to achieve a goal. For each responsibility, identify the interactions among the role and its collaborators, the performatives (speech acts) required for those interactions, and the coordination protocols according to their relationship. (Identify events and conditions to be noticed, and actions to be performed.)
4. Iterate steps (1)－(3)
5. Refine the role hierarchy.

**Design phase:**

In general, each role identifies a particular type of agent that will be in the system. However, the designer may combine multiple roles to make a single agent type. In our design, an agent can play multiple roles and can change dynamically. Furthermore, each role identified from the analysis phase must be played by at least one agent. An agent class is a template for a particular type of agent that will be in the system. In this phase, our methodology for the development of an agent class begins from selecting the roles it will play. This defines the agent’s purpose, and determines the agent’s top-level goals to be achieved. Analysis and decomposition of the goals into subgoals might result in the identification of different plans by which a goal can be achieved. The methodology for the development of an agent class is summarized as follows:

1. Determine the roles that an agent will play. Identify the top level goals of the agent by collecting responsibilities (goals) of these roles.
2. Construct the goal hierarchy as follows. For each goal, analyze the different contexts in which the goal has to be achieved and for each of these contexts, decompose the goal into activities, represented by subgoals, and actions. Analyze, in corresponding order and conditions, the activities and actions to be performed, specify how failure should be dealt with, determine the performance measures that need to be collected (for meta-level planning), and then generate a plan to achieve the goal in this context. Repeat the analysis for subgoals.
3. Determine the belief structure of the system — the information requirements for each plan and goal. Analyze the input and output data requirements for each subgoal in a plan and make sure that this information is available either as beliefs or as outputs from prior subgoals in the plan.
4. Iterate step (1)—(3)
5. Refine the agent classes by introducing a new class which existing agent classes can specialize and composing agent classes via inheritance or aggregation.
6. Use deployment diagram to show the numbers, types, and locations of agent instances within the system.

## 5. Conclusions

In the first year of the project, we have analyzed the existing BDI agent architectures and mobile agent platform to design and implement a new mobile BDI agent architecture and platform that provides essential capabilities in multi-agent environments, such as mobility, directory service, and inter-agent communication, collaboration, and coordination. Furthermore, we have analyzed existing methodologies for analysis and design of multi-agent systems and have proposed our own methodology suited to the new platform.

The major work of the next year is to: (1) continue accomplishing implementation and testing of the new platform; (2) use our methodology to analysis and design several example applications to verify its practicability; (3) implement the examples on the new platform to verify the platform’s practicality; (4) analyze existing development environments for multi-agent systems; (5) design and implement a visual programming environment which supports analysis, design, implementation, and simulation for multi-agent systems.

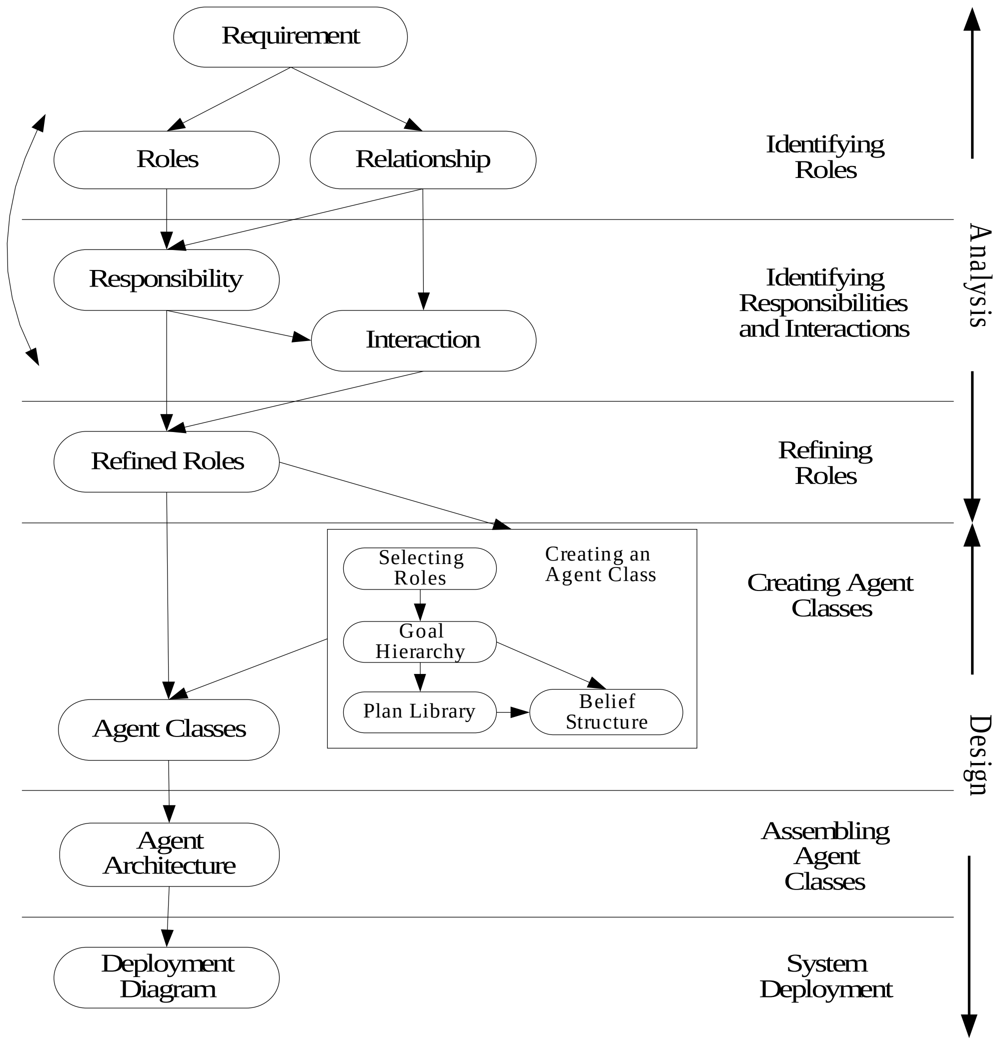

Figure 4: Methodology for the Development of Multi-Agent Systems

## Reference

[1]"Software Agents: A review", Shaw Green, Leon Hurt etc.

[2] Programming and Deploying Java Mobile Agents with Aglets, Danny B. Lange and Mitsuru Oshima

[3] Agent Tcl was developed by Robert S. Gray and colleagues at the Dartmouth College Computer Science Department.

[4] Ara is a project within the Distributed Systems Group in the Computer Science Department of the University of Kaiserslautern, Germany.

http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/index_e.html

[5] James E. White; Telescript technology: The foundation for the electronic market place; General Magic White Paper.

[6] IBM Aglets, http://www.trl.ibm.co.jp/aglets

[7] Voyager 3.1, http://www.objectspace.com

[8] Concordia, http://www.meitca.com/HSL/Projects/Concordia

[9] MASIF-The OMG Mobile Agent System Interoperability Facility; Mobile Agents–Second International Workshop, MA'98 (Stuttgart, Germany, September 1998); Published as Kurt Rothermel and Fritz Hohl, editors, Lecture Notes in Computer Science, 1477. Springer, September 1998.

[10]Mobile Agents White Paper, General Magic, http://www.genmagic.com/technology/techwhitepaper.html/

[11] Agent system Development Method Based on Agent Pattern; Yasuyuki Tahara, Akihiko Ohsuga and Shinichi Honiden; 21st International Conference on Software Engineering, 16-22 May 1999.

[12] Distributed Systems: concepts and Design; George Coulouris, Jean Dollimore, and Tim Kindberg; second edition 1994

[13] Distributed Operation Systems & Algorithms; Randy Chow and Theodore Johnson at university of Florida; publisher Addison Wesley 1997

[14] HOSTNAME Server; Tech. Report RFC 953; ftp://nic.ddn.mil/user/pub/RFC

[15] Mobile Objects and Agents (MOA); Dejan S., William LaForge and Deepika Chauhan; The Open Group Research Institute.

[16] Marcus J. Huber. JAM: A BDI-theoretic mobile agent architecture. In Proceedings of the Third International Conference on Autonomous Agents (Agents'99), pages 236--243, May 1999.