8-2002

Visual Execution Analysis for Multiagent Systems

Chong Kil

Follow this and additional works at: https://scholar.afit.edu/etd

Part of the Software Engineering Commons

Recommended Citation

Kil, Chong, "Visual Execution Analysis for Multiagent Systems" (2002). *Theses and Dissertations*. 4412.
https://scholar.afit.edu/etd/4412

Visual Execution Analysis for

Multiagent Systems

THESIS

Chong Kyung Kil, Captain, ROKA

AFIT/GCS/ENG/02-12

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/02-12

Visual Execution Analysis for Multiagent Systems

THESIS

Presented to the faculty of the Graduate School of Engineering & Management

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Computer Science)

Chong Kyung Kil, B.S. Computer Science

Captain, ROKA

August 2002

AFIT/GCS/ENG/02-12

The views expressed in this thesis are those of the author and do not reflect the official

policy or position of the Department of Defense or United States Government.
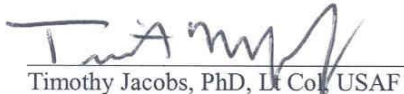
VISUAL EXECUTION ANALYSIS FOR

MULTIAGENT SYSTEMS

THESIS

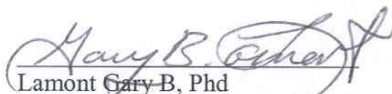Chong Kyung Kil, B.S. Computer Science

Captain, ROKA

Approved:

_____
Timothy Jacobs, PhD, Lt Col, USAF
Committee Chairman

12 AUG 02
date

_____
Karl Mathias, PhD, Maj, USAF
Committee member

12 AUG 02
date

_____
Lamont Gary B, Phd
Committee member

9 AUG 02
date

ACKNOWLEDGMENTS

AFIT/GCS/ENG/02-12

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Multiagent systems have become increasingly important in developing complex software systems. Multiagent systems introduce collective intelligence and provide benefits such as flexibility, scalability, decentralization, and increased reliability. A software agent is a high-level software abstraction that is capable of performing given tasks in an environment without human intervention. Although multiagent systems provide a convenient and powerful way to organize complex software systems, developing such system is very complicated. To help manage this complexity this research develops a methodology and technique for analyzing, monitoring and troubleshooting multiagent systems execution. This is accomplished by visualizing a multiagent system at multiple levels of abstraction to capture the relationships and dependencies among the agents.

Visual Execution Analysis for Multiagent Systems

I. Introduction

High-speed networks, Internet computing, and online communications expedite the progress of collaborative software systems. People in the early days of computer history did not expect resource sharing, distributed computing, and many other types of collaborative software systems. Nowadays, such cooperative software systems are very popular in industry and academic research areas. It is a common idea that sharing resources and information with other people helps solve complex problems using the combined knowledge.

Multiagent system technologies came from the same idea of collaborative software systems. A multiagent system comprises multiple software agents that perform given tasks, without direct human intervention, to achieve the overall system goal. In a multiagent system, each agent has well-structured roles with tasks to achieve a set of predefined goals. Agents can decide their behavior according to the knowledge given to them and they can communicate with each other via *conversations* to overcome the limitations of their knowledge and capabilities.

Before a multiagent system can be trusted to execute its behavior as expected, program execution analysis must be performed as is done during development of many other software systems. The execution analysis of multiagent systems includes profiling

run-time data, analyzing agent behavior, and analyzing system performance. This thesis effort designs and implements a methodology that enables developers to analyze, troubleshoot, and evaluate multiagent systems using visualization techniques along with agent technology.

1.1 Background

Multiagent system technologies are worth developing as solutions for military software systems where the collaboration of resource and information is essential for the military missions. Military software systems need to provide independent services to their own forces for tactical goals, and they need to perform various collaborative tasks with other systems to achieve high-level strategic goals. Flexibility is an important aspect of military software systems to adapt to the rapid changes of the battle environments. Distributed information management in the Joint Battlespace Infosphere (JBI) is a good example of how an agent-based system might be exploited for designing "fuselets." A fuselet is a key element for providing the timely and customized information required by the JBI [16; 17]. Fuselets can be mapped to agents that automatically manage a variety of information on behalf of human operators. Control of Agent-Based Systems (CoABS) is another example of the use of agent technology. CoABS utilizes agents to enhance the dynamic connection and operation of military planning, command, execution, and combat support systems [3].

The Air Force Institute of Technology has developed the Multiagent Systems Engineering (MaSE) methodology for designing and developing a multiagent system

[9:4]. This work includes a java-based graphical development tool, called agentTool, to support MaSE methodology. agentTool helps developers analyze, design, and implement multiagent systems by providing visual diagrams for describing complex multiagent systems behavior. Researchers have successfully used MaSE and agentTool for a number of multiagent systems, however, troubleshooting and analyzing these systems has proven difficult.

1.2 Problem Statement

The behavior of multiagent systems is extremely difficult to predict and analyze. A multiagent system has no global control of program executions and agents run on different processors to achieve given tasks independently. Complexity is increased by asynchronous agent interactions and complicated synchronizations for sharing resources and integrating each process's results. Even worse, an analyst has to deal with a large amount of data since a multiagent system entails thousands of message exchanges between agents.

Graphical representations of information are a powerful tool for understanding, analyzing, and relating large quantities of data. Visualization techniques can greatly improve program understanding and execution analysis in a distributed environment by allowing developers to see high-level abstract views of the system.

To ensure a multiagent system's functionality and behavior, developers must be able to analyze and troubleshoot the multiagent system. Therefore, the goal of this research is to *develop a visual program execution analysis methodology for analyzing,*

monitoring, and troubleshooting multiagent systems. This is accomplished by visualizing the program at multiple levels of abstraction to capture the relationships and dependencies among the processes.

1.3 Approach

To achieve the goals of this research, a generic program execution analysis methodology is developed for analyzing multiagent systems. To support this methodology this research includes an agent-based visualization system that demonstrates the capabilities and benefits of the methodology. The architecture of the agent-based visualization system provides a unique way to improve the visualization system's performance. Several visualization techniques are integrated into the visualization system to help developers understand, analyze, and troubleshoot multiagent systems.

1.4 Thesis Overview

Chapter 2 provides a review of literature including software agents, multiagent systems, and agent infrastructures. Chapter 3 outlines the methodology for obtaining the goal stated above in Section 1.2. Chapter 4 presents design considerations and related details of implementing a visual execution analysis system. The agent-based visualization system and its architecture are described in this chapter. Chapter 5 describes the application of the visual execution analysis methodology for analyzing and troubleshooting a multiagent system. Chapter 6 compares this research to previous works, and presents conclusions and future work.

II. Background

2.1 Overview

This chapter reviews agent related knowledge and techniques that have been developed up to now. Recent software development trends show that software systems are becoming much more complex to meet various users' requirements. Distributed systems are becoming popular to utilize computer networks and to increase available computing power by integrating many computers. Software development lifecycles are being shortened due to frequent modifications of systems.

Software agent technology is popular in distributed software development research. Software agent technology introduces a new way to manage software complexity by describing software systems with high-level abstractions such as organizations, tasks, roles, and so forth. A software agent is capable of operating as a standalone process and performing actions without user intervention. It is a very appealing idea that software agents can perform complex tasks on a user's behalf. However, designing agent-based system has proven difficult since developers need specialized skills and knowledge in a variety of areas including agent architecture, communications technology, knowledge representation, and agent communication languages and protocols.

Such agent related knowledge and related literature must be reviewed to effectively design a visual execution analysis system for agent-based systems. Section

2.2 addresses what software agents are, different kinds of software agents, and why they are preferred in the distributed software development research area. Section 2.3 outlines multiagent systems and their application areas. Section 2.4 explains agent platforms and specifically the agentMom programming interface that is applied for this research. Section 2.5 describes various types of agent communication languages used in agent-based systems. Section 2.6 presents agent conversations and how they are described. Section 2.7 reviews past works that relate to agent-based system visualization. Section 2.8 introduces the Multiagent Systems Engineering (MaSE) methodology and an automated tool for MaSE, agentTool, that is utilized in this research to develop a visualization system. Section 2.9 summarizes this chapter.

2.2 Agents

There is no standard definition of an agent, but an agent is considered as a self-controlling problem solving entity that has the following basic attributes [19:352]:

- Autonomy: agents perform given tasks without the intervention of humans or other agents. Agents should control their internal (software) and external (hardware) state by their own decisions.

- Social ability: agents interact with other agents to solve problems. This requires that agents must have communication capability to exchange views with related agents about a matter. Agents may collaborate or compete with other agents in different situations to share knowledge or to acquire limited resources for solving problems.

- Reactivity: agents perceive and respond to the given environment using their sensibility. The environment may be a variety of circumstances such as the physical world or the Internet.

- Proactiveness: agents are able to generate goals and execute tasks to achieve the goals. Agents decide their behaviors depending on the goal accomplishment conditions.

In addition to these basic attributes, agents may exhibit other attributes such as adaptability, mobility, and rationality [19; 35].

Agents can be classified into several categories depending on their characteristics or application areas [28:214-239]. *Collaborative* agents are the most common type of agent shown in agent related literature. Cooperation among agents is the major characteristic of collaborative agents since they are mainly used to solve problems that are too large or difficult to achieve by a single agent due to resource limitations or computing power. *Interface* agents facilitate developers' understanding of a particular application or an operating system. *Mobile* agents have a capability of traveling across the network to overcome limited local resources. Recently, much research has been contributed to solve the problems for developing mobile agents. Such problems include authentication, security, transportation, and interoperability of mobile agents. *Information* agents help developers manage lots of information such as information retrieval from World Wide Web documents or a large database. *Reactive* agents perceive the given environment and respond it. Reactive agents do not have knowledge

of their environment; instead, they act and respond in a stimulus-response manner to perform given tasks. *Hybrid* agents are the last type of agents. Hybrid agents are constructed by combination of two or more types of agents reviewed so far.

There are a number of compelling reasons to exploit agents for a distributed software system. Agents have many aspects that are consistent with the object-oriented paradigm. Agent-based systems can inherit all the benefits from object-oriented programming methodologies. A common concept for the future of Internet computing is that of intelligent software entities communicating and coordinating with each other over wide area networks. Agent-based systems are a good match for this paradigm. Self-configuration and decentralization are good aspects of using agents to provide fault-tolerance by replicating a disabled agent [25]. Furthermore, an agent-based system can have better scalability and modularity and it can be distributed over a large number of processors.

Although agent based technology shows many good characteristics for developing a complex distributed software system, it is still in need of maturing its methodologies to solve many technical hurdles, for example, agent communication infrastructures, knowledge representation, and interoperability between heterogeneous agents. There is considerable literature discussing such technical challenges and pitfalls for developing agent-based systems [15; 19; 36].

2.3 Multiagent System

A multiagent system is a group of agents that pursue some common high-level system goals. Generally, each agent has limited knowledge about overall problems and incomplete information to solve them. The entire system control and data are naturally decentralized and each agent's computation is asynchronous. To achieve overall system goals agents can cooperate on their activities, coordinate their knowledge, or compete with each other to achieve their given tasks. Interaction between agents may take place directly via an agent communication language (ACL) or indirectly via the system environment (Agents sense the actions of other agents and react accordingly). The benefits of a multiagent system are many and in most cases can include flexibility, scalability, decentralization, and robustness.

Complexity is highly increased in multiagent systems development. Developers must consider the problems of traditional distributed systems such as potential communication bottlenecks, weak security, deadlocks, resource sharing, and synchronizations. In addition, developers must consider additional issues for designing a multiagent system. Such additional issues include the following:

1) Agent representations: How are agents uniquely identified in a given environment? An agent's identity may contain name, IP address, or available services from the agent.

2) Organization structures (agent society): How does a multiagent system organize agents to achieve goals?

3) Task planning: How does a multiagent system distribute given tasks to agents and integrate the results?

4) Interaction protocols: How do agents interact with each other to coordinate tasks? Since agents may collaborate, compete, or negotiate to achieve given tasks, various interaction protocols must be considered.

Application domains in which multiagent system technology is appropriate typically have a naturally distributed system environment (military, banking, etc.) and the problems are too large and complex to be solved by a single, centralized system. Areas of application for multiagent systems can be divided into five main categories: problem solving in the broadest sense, collective robotics, multiagent simulation, the construction of hypothetical worlds, and kinetic design of programs [18]. Recently, multiagent systems have been used for education applications such as intelligent tutoring systems [14]. Numerous multiagent systems have been deployed in both academic and industrial areas ranging from patient scheduling in a hospital [1] to climate control of a building [37], and in areas as varied as Information Broadcasting via the Internet [38], supply chain integration [26], and an architecture for enterprise modeling and integration [32].

2.4 Agent Platforms

An agent platform is a software environment in which an agent lives. An agent platform provides a software environment for agents to execute their tasks, to access system resources, and to guarantee integrity and protection of agents and the platform

itself. An agent platform also provides various services for agents such as agent management, task distribution/integration, agent naming facility, message transport/handling mechanisms, and communication protocols.

Examples of agent platforms are DARPA's CoABS, Carolina, IBM's Aglets, General Magic's Odyssey, Object Space's Voyager, Grasshopper from GmbH Informations und Kommunikationssysteme (IKV++), and Mitsubishi's Concordia. Although there have been many agent platforms proposed in the software agent research areas, no generic standard agent platform has been established since the requirements of agent-based systems are largely varied across different software domains; however, there are two emerging standards for a generic agent platform in industry: 1) the Object Management Group's Mobile Agent System Interoperability Facility Specification (MASIF) and 2) the specifications promulgated by the Foundation for Intelligent Physical Agents (FIPA).

Eleven companies including 3Com, HP, and Sun organized the Object Management Group (OMG) in 1989. In 1995, OMG started working on the Mobile Agent Facility Specification (MAF) to support agent mobility and interoperability. The standard's name was changed from MAF to MASIF in 1997. The MASIF standardizes agent architecture, agent management, agent transfer, agent system types (names), and location syntax to promote interoperability among heterogeneous agent platforms [30].

FIPA was organized in 1996 to create generic software standards for agent-based systems. Currently FIPA has over 55 international organizations including British

Telecommunications, IBM, Toshiba, and Whitestein Technologies. FIPA announced the first specification in 1997 and FIPA 2000 is the latest version. The FIPA specifications standardize agent communication, agent management, agent/software integration, and human/agent interaction [11; 13]. The major difference between FIPA's specifications and MASIF is that FIPA doesn't specify agent internal architecture and agent implementation. Another major difference is the method of agent interactions. FIPA's specifications use the Agent Communication Language (FIPA-ACL), but MASIF uses Remote Procedure Call (RPC).

In this research, a specific agent platform, called agentMom, is selected to develop the visual execution analysis system. The agentMom platform is developed at AFIT to provide a simple and basic architecture for building agents and for specifying communication methods between agents [7]. An overview of how agentMom works is shown in Figure 1.
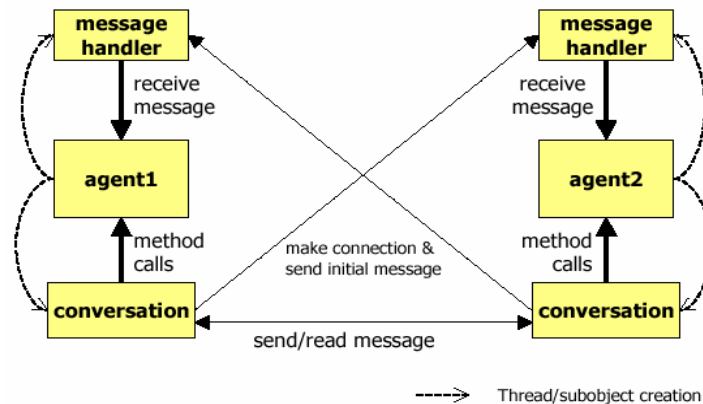
Figure 1. agentMom Architecture

In agentMom, agents communicate through a *Message Handler*. All agent communications are performed as conversations, which define a sequence of message exchanges between agents to coordinate their actions. The *Message Handler* is similar to a personal mailbox for an agent to receive messages from other agents.

An agent allows itself to coordinate with other agents by starting a *Message Handler* that receives messages from other agents. A *Message Handler* monitors a network communication port to receive incoming messages from other agents. When one agent wants to communicate with another agent, it starts one of its conversations as a separate Java thread. The conversation then establishes a TCP/IP socket connection with the other agent's *Message Handler* and sends the initial message in the conversation. When the *Message Handler* receives a message, it passes the message to the agent's *receiveMessage* method that compares the message against its known list of allowable message types to see if it is the start of a valid conversation. If the conversation is valid, the agent starts its side of the appropriate conversation, also as a separate Java thread. If the conversation is not valid, the agent replies with *Sorry* message to the sender agent. After two agents establish a valid conversation, all communications between agents are controlled by the two different conversation threads. During the conversation, agents can send multiple messages to each other using built in *readMessage* and *sendMessage* methods. While conversations handle the message passing between agents, they still must have a way to communicate with their parent agents. This is accomplished using method calls from the conversations back to their parents. The agentMom is platform independent since it is implemented in Java.

2.5 Agent Communication Language (ACL)

Once agents are deployed in a distributed environment, they need to communicate to coordinate their actions and exchange information. Without communications, an agent is merely an isolated computation entity that has limited capability to achieve overall system goals.

To share knowledge between agents efficiently, a communication language and an interaction methodology are needed. An ACL plays a key role in agent communications. An ACL is a set of messages and their descriptions. It includes semantic and syntactic specifications for communicative acts between agents such as ask, inform, tell, reply, and so forth. An ACL message contains a set of one or more message elements. Precisely which elements are needed for effective agent communication will vary according to the situation; the only element that is required in all ACL messages is the *performative* (type of communicative act), although it is expected that most ACL messages will also contain *sender*, *receiver* and *content* elements. An example of an ACL message structure is shown in Figure 2.

```
(inform
    :sender agent1
    :receiver hpl-auction-server
    :content
        (price (bid good02) 150)
    :in-reply-to round04
    :reply-with bid04
    :language sl
    :ontology hpl-auction)
```

Figure 2. Example of an ACL Message

In Figure 2, `agent1` informs `hpl-auction-server` to `bid good02` with the price `150`. The *in-reply-to* element denotes earlier action (`round04`) to which this message is a reply. The *reply-with* element denotes an expression (`bid04`) that will be used by the responding agent to identify this message. The *ontology* element describes a meaning to the symbols in the content expression. The *language* element denotes the language in which the content element is expressed.

Although the standard for ACL has yet to emerge, two major ACL standards have been proposed and exploited in many agent applications: 1) KQML (Knowledge Query and Manipulation Language) and 2) FIPA-ACL.

KQML is a communication protocol that includes both a message format and a message handling procedure to support knowledge sharing between agents [10]. KQML can be used as a language for an application program to interact with an intelligent system or for sharing knowledge between many intelligent systems in support of cooperative problem solving. KQML focuses on an extensible set of *performatives*, which defines the permissible operations that agents may attempt on each other's knowledge and goals [22]. Although such an effort for developing a high-level communication standard is certainly valuable, the KQML has some drawbacks. Cohen and Levesque [4] discussed some drawbacks. They pointed out some *performatives* are ambiguous and incoherent such as *achieve, broker* and *stream-all*. They then proposed minimum set of *performatives* as fundamental *performatives* to improve KQML.

Another emerging communication standard is FIPA-ACL. FIPA-ACL is the agent communication language associated with FIPA's agent architecture. FIPA-ACL comprises about 20 basic types of communication, using a rigorous semantic specification [11]. FIPA-ACL focuses on the interoperability between heterogeneous agents. To improve the interoperability, FIPA-ACL allows agents to utilize different message transportation methods.

The fundamental difference between FIPA-ACL and KQML is that FIPA-ACL does not allow an agent to directly manipulate another agent's internal state. Therefore, some of KQML's performatives are not meaningful in FIPA-ACL. Since Cohen and Levesque's criticisms, work has been done in connection with KQML that has produced more precise forms of semantics [23], and the differences between KQML and FIPA-ACL are diminishing. Both standards have many common aspects in the recent specifications [11; 23].

2.6 Agent Conversation

An agent conversation is a sequence of ACL message exchanges between agents. Multiple agents can engage in a conversation to share information, request services, or negotiate limited resources. Designing agent conversations is important in agent-based system development to minimize network overload caused by redundancy.

There are number of ways to describe an agent conversation using a finite-state automaton, Petri nets, or a sequence diagram. Figure 3 shows the finite-state automaton corresponding to an agent conversation initiated by an agent A.
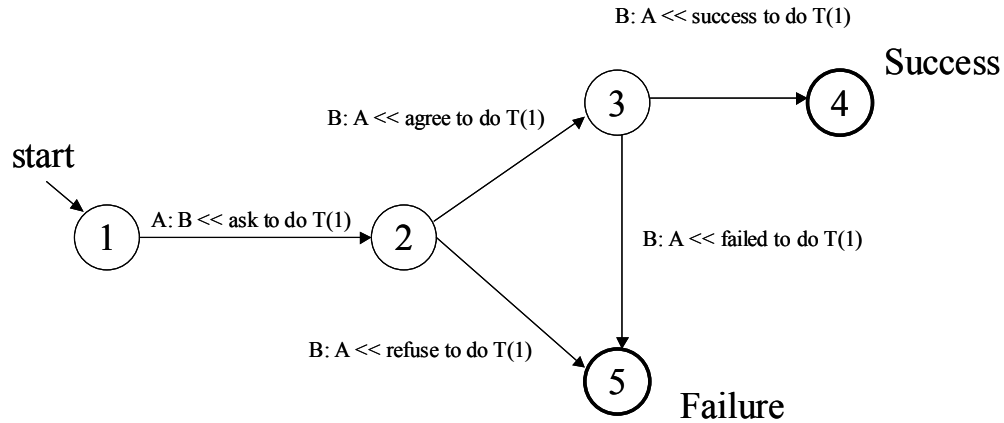
Figure 3. A Finite State Automaton for an Agent Conversation

Initially, the conversation is in state 1.   Then agent A starts the conversation by asking B to perform task T (1).   The conversation then passes into state 2 and two possibilities open up.   Agent B may accept agent A's request or reject it (if for example, it is not competent to carry out the task T (1)).   If agent B rejects the request, the conversation will pass into state 5 and the conversation is considered as being a failure. If agent B accepts the request, the conversation will pass into state 3 to wait for the task result.   Depending on agent B's task result, the conversation will pass into state 4 with successful completion of task T (1) or state 5 indicating failure.

2.7 Visualization of Agent-Based Systems

Visualization of agent-based systems can be divided into two categories: single agent visualization and multiagent visualization.   Although they are related to each other, these categories focus on different aspects of the agent-based system.   While single agent visualization mainly focuses on an agent's internal state and interactions with other

agents, multiagent visualization focuses on external workings of the distributed, heterogeneous agents in a given environment [31].

Considerable research has focused on the development of agent platforms, agent internal architectures, and agent communication protocols; however, visualization of multiagent systems for analysis is largely neglected. One of the few systems that incorporates some analysis and visualization features is ZEUS [29]. ZEUS is an agent building toolkit that allows system developers to use visual editing tools to construct the multiagent systems and to specify the interactions between the agents. The system developers can monitor concurrent tasks and messages between agents by employing the visualization tool, called Visualizer. The Visualizer is comprised of Society Viewer, Reports Tool, Agent Viewer, Control Tool, and Statistic Tool. Society Viewer shows predefined agents relationships such as peer-to-peer or superior-subordinate. Users are required to define an agent's relationship when designing agent organization. Society Viewer also shows message exchanges between agents. The Reports Tool visualizes task distribution and the execution state of tasks. The Agent Viewer enables users to observe an agent's internal states. The Control Tool is used to remotely review and/or modify the internal states of individual agents. The Statistic Tool provides various statistical data about an agent and the system. Figure 4 shows an example of Society Viewer. In Figure 4, each agent is displayed by graphic icons in a rectangle. Agent relationships and message exchanges are shown as color-coded arrows between agents.
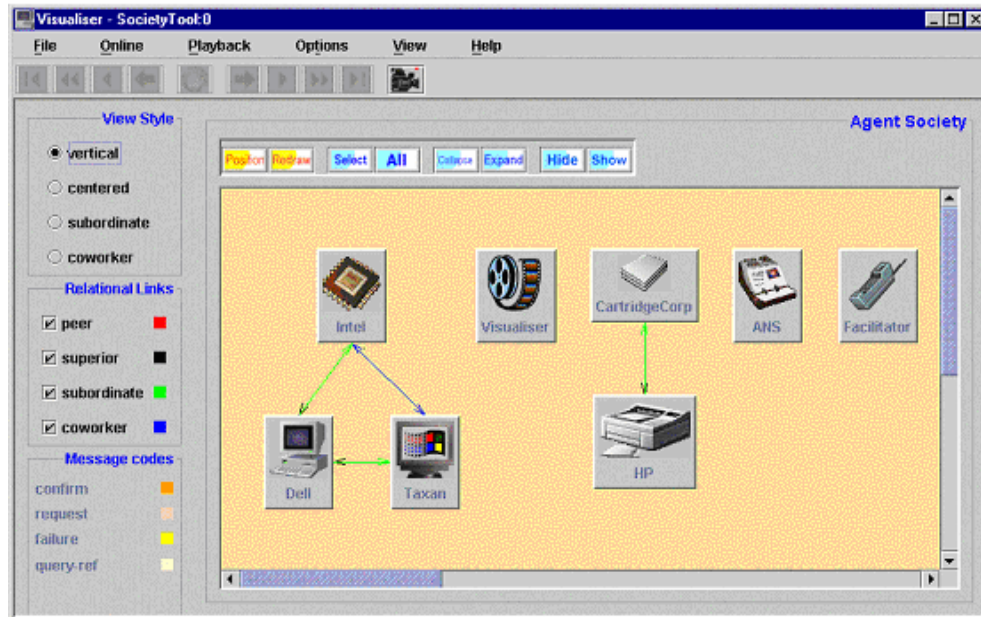
Figure 4. Example of ZEUS Society Viewer

Schroeder and Noy [31] developed a methodology to visualize agent messaging for various agent types. They developed a distance metric to describe an agent's messaging behavior. They also exploited various distance metrics such as Euclidean, Hamming distance, and edit distance. Figure 5 shows the agent messaging visualization. Agents are shown as spheres and the distances between spheres represent the number of message exchanges. Figure 5(a) shows equal number of messages sent by three agents and Figure 5(b) shows equal number of messages sent by two agents and no messages sent by third agent.

(a)   (b)
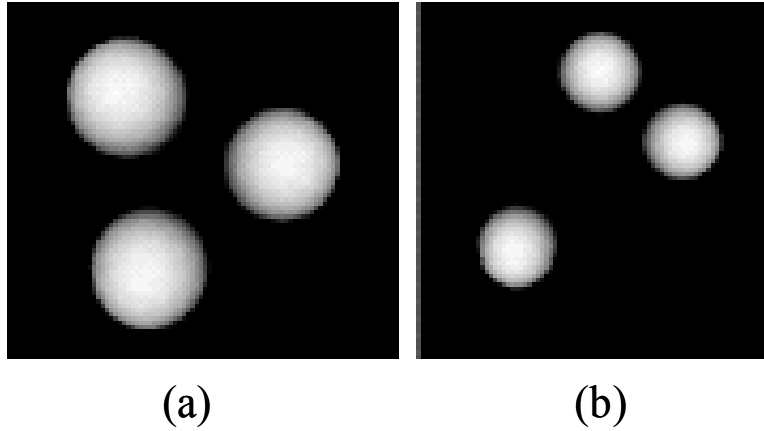
Figure 5. Visualization of Agent Messaging Behavior

Nowostawski et al used a Colored Petri Net (CPN) to visualize complex agent conversations [27]. A conversation is modeled as a whole Petri Net composed of a set of subnets, where at least one role has *Start* place and is connected to an arbitrary number of other conversation participants. An example of CPN for *Request* conversation is shown in Figure 6.
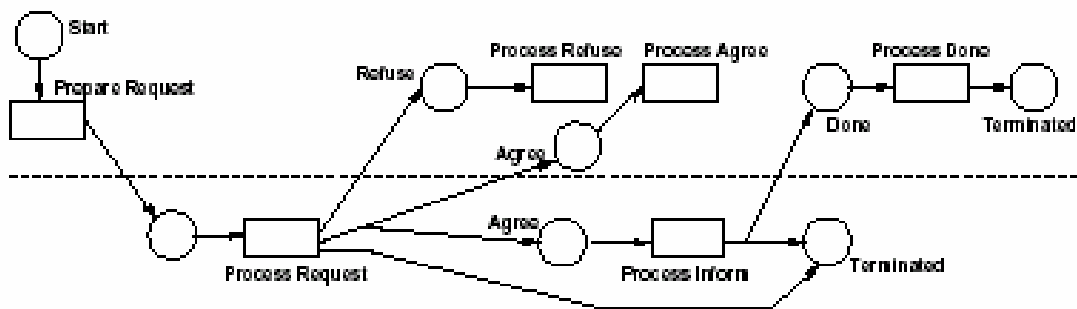
Figure 6. Colored Petri Net for *Request* Conversation

Kaminka et al developed the plan-recognition tool, OVERSEER, to monitor a previously deployed multiagent system [21]. They employed the probabilistic algorithm, YOYO, to reduce the uncertainty when the tool tracks an agents' state.

Although not directly discussing agents, some researchers have addressed visualization for distributed systems. Georgia Tech developed a visualization environment called PARADE [39] for developing animations and visualizations of parallel and distributed programs. They also developed Gthreads [40] for visualizing threads-based parallel programs on a shared memory parallel computer. PARADE and Gthreads utilize an animation toolkit, POLKA [41] to visualize programs from different languages and architectures. Pablo Research Group at the University of Illinois developed performance analysis techniques and a visualization environment for performance visualization of parallel and distributed systems. They exploit SvPablo [42] to capture performance related data from the observed system and provide the data to Virtue [43] for visualizing the system's dynamic behavior and optimizing the system's performance. ParaGraph [44], SPCview [45], and Medea [46] are similar works for visualizing message passing parallel distributed systems. These tools visualize inter-processor communications, message passing paths, or message routing performance against various network topologies.

Although multiagent systems are implemented as multi-threaded, parallel, or distributed systems, it is difficult to apply distributed system visualization tools for analyzing multiagent systems since they do not address the knowledge level messaging infrastructures and task synchronizations typically associated with multiagent

systems [29]. These distributed and parallel visualization tools do, however, provide a good foundation of knowledge for this research.

2.8 Multiagent System Engineering (MaSE)

In this research, the visualization system is developed using the Multiagent System Engineering methodology (MaSE). MaSE is the result of ongoing work by a number of researchers. Deloach developed the major concepts of the MaSE [6] and Wood implemented the MaSE in agentTool that supports design of multiagent systems and produces basic source code for further implementations [34]. Lacey extended the MaSE methodology by creating a formal method to verify the communication protocols in multiagent systems [24]. Raphael developed a Multi-Agent Markup Language (MAML) for representing multiagent systems design knowledge [33].

MaSE is an end-to-end methodology for the design and implementation of multiagent systems. MaSE uses a number of graphical models to define different types of agents, to specify individual agent behavior, and construct an agent's interactions with other agents using conversations. MaSE can be viewed as an extension of object-oriented paradigm where agents are specialization of objects. The primary focus of MaSE is to help a developer take an initial set of requirements and analyze, design, and implement a working multiagent system. The MaSE methodology is independent of a particular system architecture, programming language, or communication framework [9].

MaSE is comprised of analysis and design phases as shown in Figure 7. The analysis phase consists of *Capturing Goals*, *Applying Use Cases*, and *Transforming*

Goals to Roles.   The design phase consists of *Creating Agent Classes*, *Assembling Agent Classes*, *Constructing Conversations*, and *System Design*.
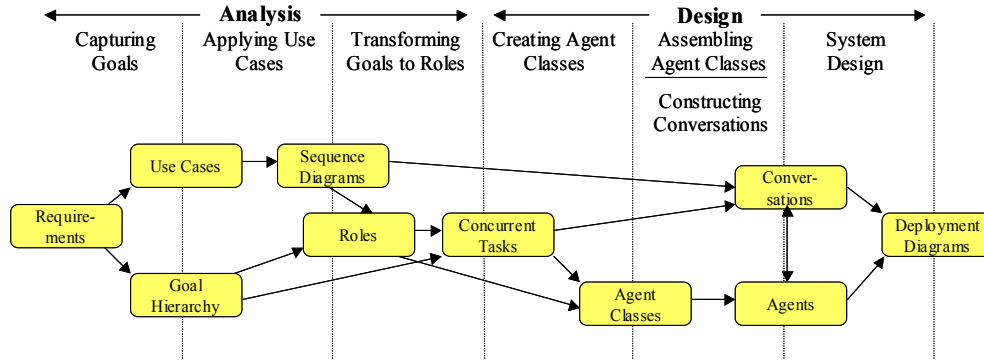
Figure 7. MaSE Analysis and Design Phases [8]

A major strength of MaSE is the ability to track changes throughout the process. Every object created during the analysis and design processes can be traced forward or backward through the different steps to their corresponding constructs.   For instance, a goal derived in *Capturing Goals* step can be traced to a specific role, task, and agent class in the agent deployment diagram.   Likewise, agent classes can be traced back through tasks and roles back to the system level goals they were designed to satisfy.

agentTool is a software tool to support MaSE using visual diagrams based on underlying formal semantics.   agentTool allows users to describe a multiagent system graphically, specify the necessary properties, check the design for correctness such as verification of conversations, and plan the system deployment.   To reduce users' effort to construct multiagent systems with minimum knowledge of agent related theories, agentTool supports automatic-code generation for the basic architecture of agents and

conversations based on graphical design documents such as the agent template diagram (Figure 8) and the conversation state diagram.

An agent template diagram for a package express system is shown in Figure 8. Agents are shown as rectangles and conversations are shown as arrows between agents. The system is comprised of five agents and eleven conversations. Roles of each agent are shown inside the rectangles; for example, *Airline Manager* agent has two roles, *Regional Manager* and *Manager*.
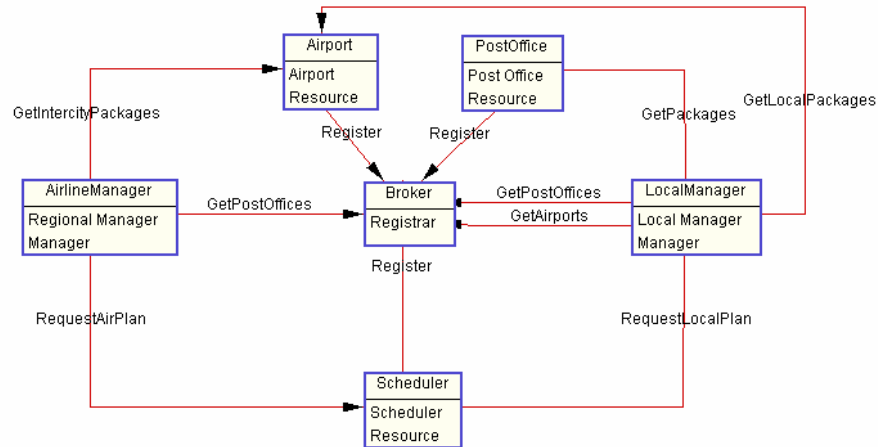
Figure 8. MaSE Agent Template Diagram for a Package Express System

2.9 Summary

Agent-based systems are an interesting area of research for developing complex intelligent distributed software systems. However, developing a multiagent system is difficult since developers need to acquire specialized skills with knowledge about software agents. In addition, software agent technologies still have many problems to resolve.

To effectively construct agent-based systems, developers must consider both the problems of traditional distributed software systems and the problems of designing complex agent infrastructures for agent interactions, goal achievement strategies, and agent organizations.

Currently, a number of agent development tools are available for agent-based system designers to construct agents and conversations. Most tools provide a graphic user interface to help users specify agent tasks, organizations, and conversations. Zeus, JAFMAS [5], JATlite, and FIPA-OS [12] are examples of such agent development tools. Others may be found at the World Wide Web [47]. While such agent development tools mostly focus on the multiagent systems development environment, visualization for analyzing and monitoring a multiagent system's execution has rarely been considered. This makes it difficult for developers to understand and debug multiagent systems. As agent-based technologies are becoming widespread, visualizations for analyzing and troubleshooting multiagent systems are needed.

III. Methodology

3.1 Introduction

As stated in Chapter 1, the goal of this research is to develop a visual program execution analysis methodology for multiagent systems. The primary focus of the visual execution analysis is to help developers analyze, validate, and troubleshoot dynamic multiagent system behavior. In this way, developers can produce a better system design to get a robust, validated, enhanced performance system. The visual execution analysis consists of *Profiling Run-Time Data*, *Behavior Analysis*, and *Semantic Performance Analysis*. Figure 9 depicts the visual execution analysis. Each step is described in the rest of this chapter.

```
                    ┌─────────────────────────────────┐
                    │   Visual Execution Analysis      │
                    │  ┌───────────────────────────┐   │
                    │  │   Profiling Run-Time Data │   │
                    │  └───────────────────────────┘   │
  Initial                                               
  system   ⇒         │  ┌───────────────────────────┐   │    ⇒   Robust,validated,
  design             │  │     Behavior Analysis     │   │        enhanced performance
                     │  └───────────────────────────┘   │        system design
                    │  ┌───────────────────────────┐   │
                    │  │   Semantic Performance    │   │
                    │  │        Analysis           │   │
                    │  └───────────────────────────┘   │
                    └─────────────────────────────────┘
```
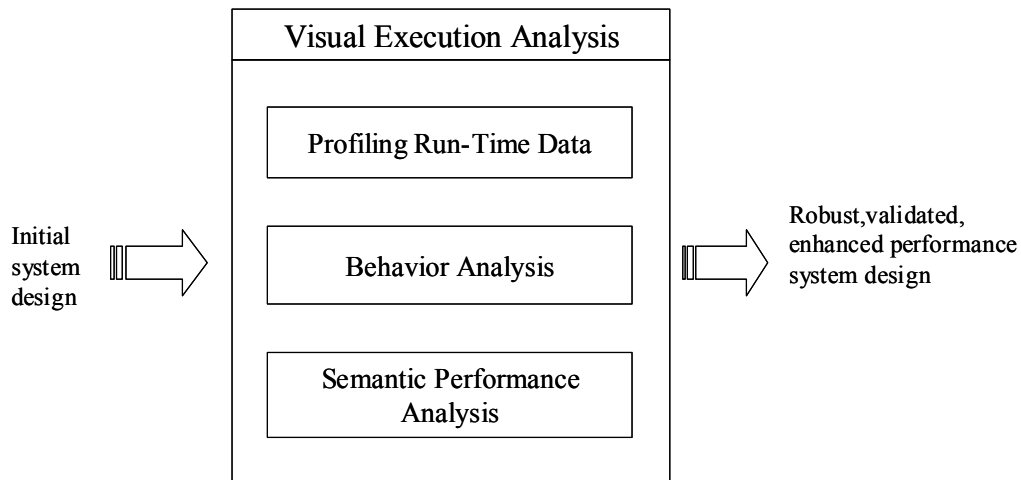
Figure 9. The Visual Execution Analysis

26

3.2 Profiling Run-Time Data

Profiling run-time data from a multiagent system is the first step of the visual execution analysis of multiagent systems.   In this step, developers define desired data to analyze the system execution behavior and to create high-level abstract views of the system.   Such data will be collected and presented by the visualization system.   As with any distributed software systems, it is a challenging task to extract such relevant information since agents perform their tasks asynchronously and dynamically within a distributed environment.

To accomplish this step, developers must consider the following:

1) *Content*: What subset of information from a multiagent system is needed to analyze the system's execution behavior?

2) *Collection Method*: How do developers collect the *content* in an agent environment?

The content can vary widely over the context of the multiagent systems and the designer's perspective.   The content of a travel scheduling system can be different from a factory automation system.   An advanced designer and a novice designer may need different content to understand the execution of multiagent systems.

System execution should be visualized or animated in a fashion similar to the original high-level design to reduce the effort required by developers to understand the

execution and to identify errors by comparing the actual execution against the expected behaviors. To accomplish this, finding the appropriate content for generating high-level abstract views is essential. For multiagent systems, message passing is the major characteristic of system behavior and performance. In an agent environment, messages are expressed in an agent communication language and exchanged via a message transportation protocol. Although messages can be encoded in different agent communication languages, it is expected that they will contain *sender*, *receiver*, and *content* elements. Messages can also include *performative* and *ontology* elements for further collaboration. A *performative* element denotes the type of the communication activity such as ask, reply, and inform. An *ontology* element describes the type of the information that is shared between agents. Such *ontology* elements in the messages show various kinds of information flows in the multiagent system.

Consequently, by focusing on the message exchanges along with their contained information rather than tracing individual agent executions, developers can gain a comprehensive understanding of the complex and sophisticated system behavior. In addition, by collecting knowledge and data from the message analysis along with the total number of messages, total size of messages, and total elapsed time for message transportation, one can gain considerable insight into system performance.

After the desired content is defined, developers need a method to acquire the content from multiagent systems. Collecting run-time data from multiagent systems can be divided into two major categories: 1) perturbation methods and 2) non-perturbation methods [21]. Perturbation methods include various kinds of intrusion techniques to

collect the desired data from multiagent systems. Having agents report their execution information or exploiting a remote debugging tool to step through an agent's execution are two possible ways of perturbation methods. Obviously, while these intrusion techniques can provide accurate information about a multiagent system's behavior, they may suffer several problems such as additional system overload, undesired impacts on an agent's original behavior, and modifications of an agent's infrastructure and the message transportation protocol.

Non-perturbation methods try to minimize intrusion by exploiting the system execution plan-recognition algorithms [21] or agent's task achievement pattern-recognition techniques. Using broker (intermediate) agents to intercept messages or exploiting an algorithm to find out agents' message exchanges in the network are two examples of non-perturbation methods. Non-perturbation methods have very little impact on agent's original behavior and require little or no modifications on the existing agent infrastructures. However, non-perturbation methods suffer large uncertainty and low accuracy for acquiring desired data. For example, examining a large amount of network packets for searching an agent's message costs a lot of computation. In addition, there is no guarantee for finding the desired data. Although non-perturbation methods are preferred for monitoring multiagent systems, they may not be good for debugging and performance analysis purposes due to the uncertainty and low accuracy to get the desired data from large information sources.

Consequently, in this research, a perturbation method is applied to capture the message exchange data between agents. This is accomplished by exploiting agent

technologies.   A special agent, called the *InfoGathering* agent, is developed for gathering message exchange data.   Data collection is accomplished by requesting a copy of received messages from the observed agents.   Although this approach requires small modifications on the existing agent infrastructure, it provides exact, well-timed information for both analyzing system executions and troubleshooting the system performance.   After the data is prepared by *InfoGathering* agents, another agent, called the *VisAnalysis* agent, generates high-level abstract views of the system execution behavior.   Detailed design and implementation of these agents are described in Chapter 4.

3.3 Behavior Analysis

Behavior analysis is the second step of visual execution analysis.   After the agents' message exchange data is collected by *InfoGathering* agents and presented by *VisAnalysis* agents, developers start analyzing the system execution behavior to discover defects of the system and check if the system performs given tasks as intended. Although the behavior analysis of multiagent systems is a challenging task, it is an important process to validate a system's functionality and construct a robust, scalable multiagent system.

Analyzing execution behavior of a multiagent system consisting of many agents is difficult for a number of reasons.   First, the overall system behavior emerges from complex agent interactions that can lead to unexpected or undesired system behavior. Second, agents are irregular and dynamic.   By their nature, there is no global system

control, data is decentralized, and agents perform their task by their own decisions in an asynchronous manner. Third, the complexity of the system grows dramatically as the number of agents increase. Finally, multiagent systems are more prone to errors than other software systems. Since multiagent systems are distributed and concurrent software systems, many types of errors can happen during the system execution such as message loss, deadlocks, and infinite loops. Even worse, the system can appear to be working while an undetected major problem exists. Limited capability of multiagent systems development tools makes it difficult to build a robust multiagent system. Currently available tools, including agentTool, provide partial support for generating a multiagent system architecture. Constructing multiagent systems mainly depends on the developer's experience and skill.

To analyze an agent's behavior, developers must know how an agent's behavior is modeled. In a typical multiagent system, an agent's behavior is represented by tasks. An agent's tasks are usually modeled with a visual diagramming language such as state transition diagrams to define an agent's behavior in each state. In each state, an agent may perform certain computations or communicate with other agents to share knowledge of the problem to solve.

Effective behavior analysis can be achieved by observing how information flows in the system, how agents cooperate to produce desired and undesired behaviors, and how agents influence one another. By tracing message exchanges between agents, developers can gain a comprehensive understanding of information flow in the system. Developers can identify undesired agent behaviors by comparing an agent's behavior

design model along with messages that the agent utilized. Although developers cannot see each computation process of an individual agent, they can analyze the result of the computation by examining the *content* element of messages. A caveat is that focusing on the individual computation process is inefficient since the overall system behavior emerges from complex agent interactions.

In MaSE, behavior analysis can be accomplished by analyzing message exchanges between agents with a task diagram and a role diagram. A task diagram depicts an agent's task state transitions and a role diagram displays an agent's collaborations with other agents. An example of role diagram is shown in Figure 10 and an example of a task diagram is shown Figure 11.
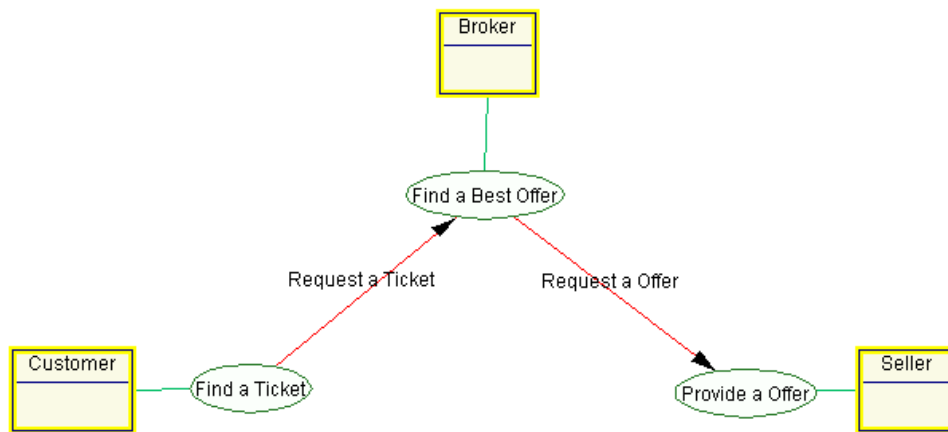

Figure 10. a Role Diagram in a Ticket Searching System

In Figure 10, a ticket searching system is described by three roles (Broker, Customer, and Seller) and three tasks (Find a Best Offer, Find a Ticket, and Provide a Offer). The system begins with a request for a ticket from a customer to a broker (Request a Ticket). The broker then asks available

ticket sellers about the ticket price (`Request a Offer`). After the ticket sellers

propose different prices of the ticket to the broker, the broker find out the best offer and

sends the ticket price to the customer.   A customer can ask multiple brokers to find out

the best price of the ticket.

A task diagram for the `Find a Ticket` task is shown in Figure 11.   There are

three types of messages required to complete the `Find a Ticket` task.   The task

starts by sending a `request a ticket` message to a broker and ends after a customer

sends a `deny` or `confirm` message to the broker.   In the meantime, a customer

compares different prices of the ticket from various brokers to select the best price offer.
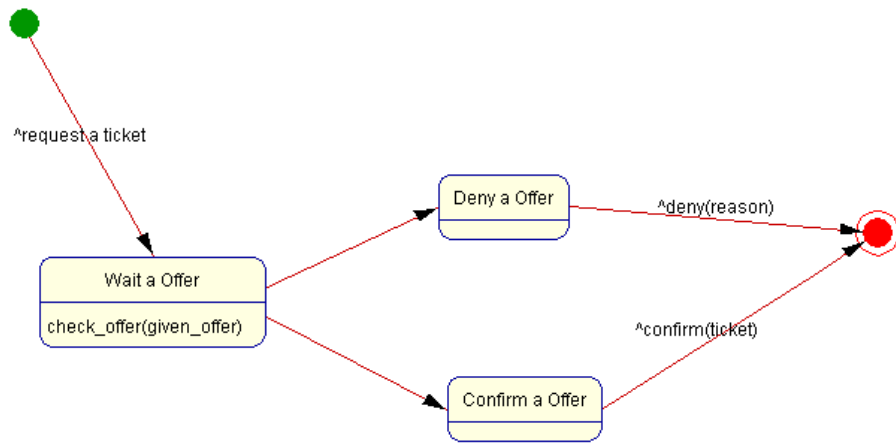

Figure 11. "Find a Ticket" Task Diagram

As shown in Figure 10 and Figure 11, one can easily understand the system

behavior by analyzing message exchanges between agents with role and task diagrams.

For example, users can identify what types of tickets and how many tickets are traded in

the system by analyzing `request a ticket` messages between `Customer` agents

and `Broker` agents. Users can check if the Broker agent performs its task as intended by analyzing `request a ticket` and `request a offer` messages.

3.4 Semantic Performance Analysis

Through the previous two steps, run-time data (especially message exchange data) of the system are presented and the developer evaluates the system's behavior and agents' collaborations against existing system behavior design models. In the semantic system performance analysis step, different developers may define different measurement criteria to evaluate the system's performance. For example, while data security may be a good measurement for military intelligence or online banking systems, it may not be good for a public system where data security is not a critical issue for the system. Advanced developers and novice developers may have different criteria to evaluate system performance.

Although there can be many criteria for performance evaluation of multiagent systems, there is a common factor that affects multiagent systems' performance: an agent's task throughput. Each agent's task throughput is closely related to the overall system performance since the overall system goal is accomplished from the collection of each agent's task results. There are many factors affecting an agent's task throughput including an agent's run-time environment, system structures (agent society), message transportation protocol, data models, task planning, the agent's action selection algorithm, and resource allocation. It is generally considered that delayed information exchange

(delayed message exchange), complex task planning, and high-computational action selection algorithms decrease the system's performance.

After the system performance evaluation criteria are decided, developers begin a performance evaluation. Developers may then consider modifications of system configuration, the agents' organization, and other factors such as system execution environment to optimize the system performance. To achieve the best performing multiagent system, running and analyzing the system executions in different configurations is necessary. Off-line replaying capability can be beneficial to help developers compare different results of the system's performance.

3.5 Summary

This chapter describes the visual execution methodology for understanding, analyzing, troubleshooting multiagent systems. The process began by profiling run-time data from the observed system. The collected data was then transformed to multiple visual presentations for helping users analyzing the system behavior and evaluating the system performance. Consequently, developers can produce a better system design form the initial design to construct a robust, validated, enhanced performance system.

IV. Design and Implementation

4.1 Introduction

This chapter covers the visualization system architecture and the visualization process to achieve the visual execution analysis described in Chapter 3. The visualization system helps developers capture the run-time data from the observed system and generates multiple views for analyzing the system behavior and performance. The visualization system architecture consists of two types of agents: (1) *InfoGathering* agents and (2) *VisAnalysis* agents. These agents are developed in Java (jdk 1.3) and designed to run on any platforms or processors that support the Java Run-time Environment (JRE). These agents provide a dynamic, selective visualization environment that is well suited for the visualization process. Section 4.1 discusses the design considerations for developing a visualization system for multiagent systems execution analysis. Section 4.2 discusses the use of agents for developing the visualization system and how *InfoGathering* agents and *VisAnalysis* agents are configured to produce a dynamic interactive visualization environment. Section 4.3 describes how developers create their own visualization sessions with *InfoGathering* agents and *VisAnalysis* agents. This section also presents an example multiagent system that is used to help understand of the visualization process.

4.2 Design Consideration

Designing a visualization system for multiagent systems execution analysis requires careful thought. To develop an efficient visualization of multiagent systems execution behavior, one must take into account both multiagent systems properties and data visualization characteristics.

Key design considerations are:

- Scalability: A multiagent system may consist of a large number of agents. A visualization system must deal with many agents to collect their execution data and present the large amount of data effectively.

- Minimal invasion: A visualization system must try to reduce disturbance of agent execution to maintain the original system behavior.

- Distributed data collection: Agents are distributed and they may run on different hardware platforms. A visualization system needs strategies for gathering desired data from distributed agents run on different processors.

- Accuracy: A visualization system must present correct views of multiagent systems behavior.

- Efficiency: Maximizing data-ink ratio and reducing chart-junk are important to utilize the limited display area.

- Adaptability: The visualizations can be adjusted to serve users' multiple needs.

- Effectiveness: Minimal effort should be required of users to generate visualizations.

To meet the above design considerations, this research developed an agent-based visualization system. The agent-based visualization system exploits agents to collect distributed data and visualize multiagent systems behavior. Details of this agent-based visualization system are discussed in the subsequent sections.

4.3 Agent Based Visualization

Card *et al* identified several steps necessary in information visualization to transform raw data into the specific views for different types of users [2]. The visualization steps are shown in Figure 12.

```
┌──────────┐      ┌──────────┐      ┌──────────┐      ┌──────────┐
│   Raw    │ ───▶ │   Data   │ ───▶ │  Visual  │ ───▶ │  Views   │
│   Data   │      │  Tables  │      │Structures│      │          │
└──────────┘      └──────────┘      └──────────┘      └──────────┘
              Data              Visual            View
          Transformations     Mappings       Transformations
                 ▲                ▲                ▲
                 └────────────────┼────────────────┘

                        Human Interaction
```
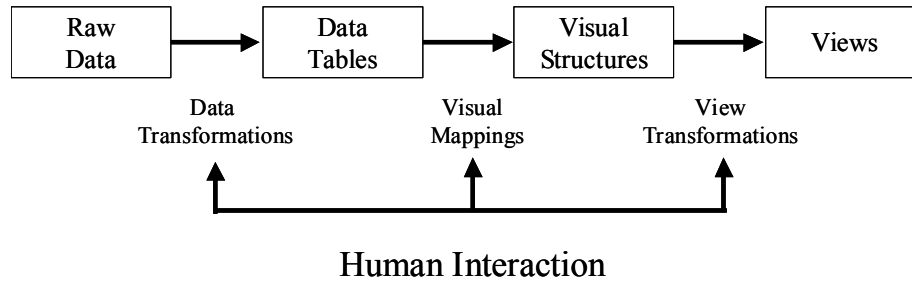
Figure 12. Information Visualization Steps

Agent-based visualization follows such steps to produce high-level views of multiagent systems' behavior. First, the raw data (message exchanges between agents) are collected from multiagent systems. This requires gathering the data from the selected agents and conversations. The gathered data is then transformed into a data table. The data table stores gathered data into a specific format for easy mapping to the

visual structures.   Visual mapping follows the data transformation.   In the visual mapping, the formatted data is converted to visual structures such as glyphs, labels, figures, and other graphical objects.   View transformation generates multiple views by displaying graphical objects in different layouts such as the *Agent Relationship View* or the *Conversation Flow View*.   These views are described in Chapter 5.   Developers can interact with the visualization system to select desired data, create graphical objects, and generate views.

This research utilizes agent technology to implement this information visualization process.   The agent-based visualization comprises the same steps as traditional visualization process except it uses agents to perform each step.   In the agent-based visualization, each visualization step is performed as a task in an agent.   Figure 13 shows the agent-based visualization.
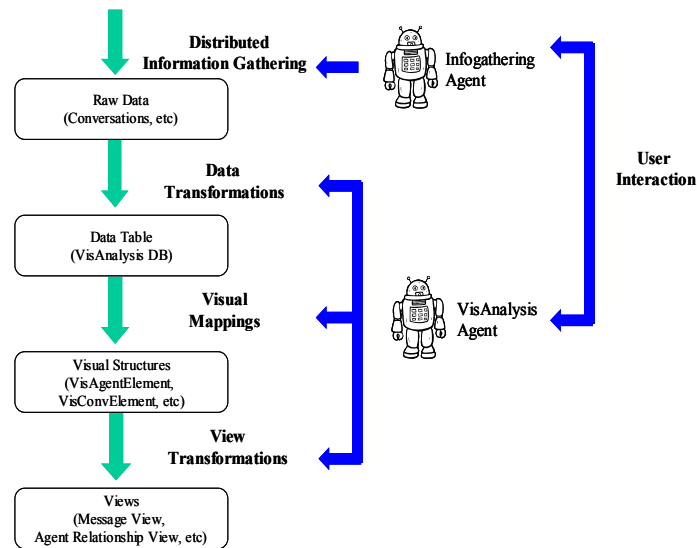

Figure 13. Agent-based Visualization

In Figure 13, *InfoGathering* agents and *VisAnalysis* agents are created to achieve the agent-based visualization. Both agents are developed using agentTool. They perform given tasks and communicate with other agents based on the agentMom infrastructure. Both agents can be integrated into any type of multiagent system since they can communicate with agents through agent conversations. To achieve the visualization process, *InfoGathering* and *VisAnalysis* agents are assigned to one or more tasks. *InfoGathering* agents perform the distributed information gathering task. *VisAnalysis* agents perform the data transformation, visual mapping and view transformation tasks. Developers interact with both agents to collect data from the observed systems, specify visual structures, and generate desired views that show key aspects of the system.

This agent-based visualization architecture provides a number of benefits. First, it improves the visualization system performance and helps to resolve major problems that may occur when the visualization system runs on a single machine. Such problems can include a data collection bottleneck due to gathering data on a single machine, and limited scalability for visualizing numerous objects in a limited display area. An example of the agent-based visualization system configuration is shown in Figure 14.
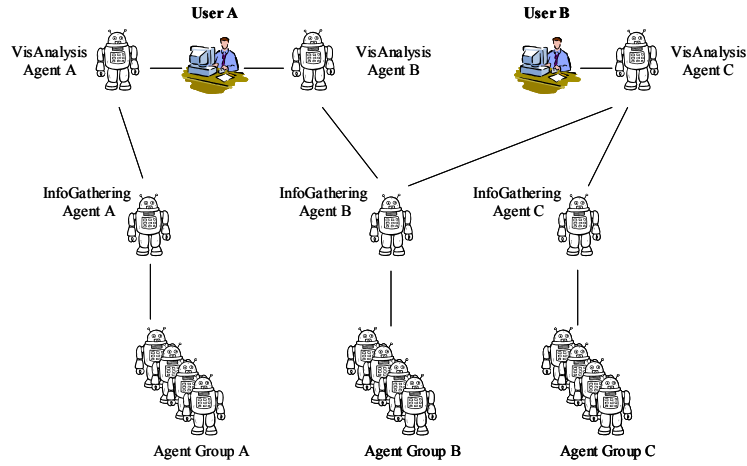
Figure 14. An Example of Agent Based Visualization System Configuration

In Figure 14, *User A* instantiates two *VisAnalysis* agents and two *InfoGathering* agents on different machines to collect data and generate views for analyzing execution of *Agent Group A* and *B* in the system. Since the *User A* separated data collection task loads into two *InfoGathering* agents on different machines, the data collection bottleneck for gathering message exchange data from many agents is avoided or decreased. Scalability is improved by assigning each agent group to different *VisAnalysis* agents for visualizing agents execution behavior.

As a second benefit of the agent-based architecture, developers can select or deselect observed agents dynamically at runtime. This is achieved by allowing developers to send a data-gathering request or cancel message anytime to *InfoGathering* agents via *VisAnalysis* agents. In addition, developers can reduce additional network overload for gathering data and decrease interference with the multiagent system's original behavior by selecting only those necessary agents for analysis.

As a third benefit, agent platform dependence is minimized. Since the data collection is achieved by asking agents to report the copies of received messages via agent conversations, the agent-based visualization does not require any specific agent platforms.

Last, multiple developers can analyze different parts of the same observed system by using different *VisAnalysis* agents and *InfoGathering* agents. Cognition effects can be maximized by employing familiar glyphs and colors to encode data for visualization. In Figure 14, the *User A* analyzes *Agent Group A*, and *B* and the *User B* analyzes *Agent Group B*, and *C* in the system using different *VisAnalysis* and *InfoGathering* agents.

4.4 The Visualization Process

The visualization process allows developers to create their own visualization sessions for analyzing multiagent systems execution. The visualization process consists of six steps as shown in Figure 15. Developers interact with multiple *VisAnalysis* agents and *InfoGathering* agents via a graphic user interface to complete each step. Each step is described in more detail in the following paragraphs.
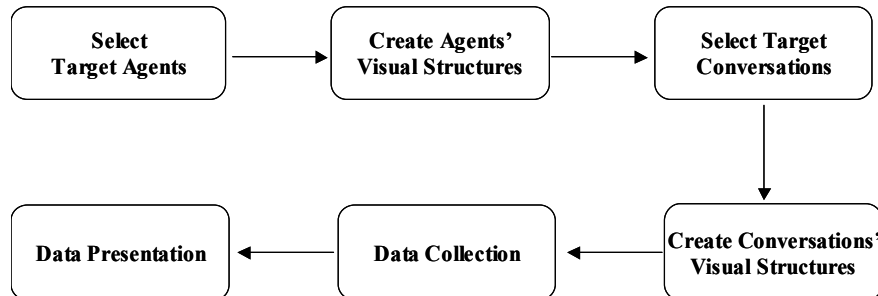

Figure 15. The Visualization Process

4.4.1 Example Multiagent System

To explain the visualization process and to demonstrate the benefits and usability of our agent-based visualization system, this research developed a multiagent system for Command, Control, Communication and Intelligence (C3I) simulation as depicted in Figure 16.
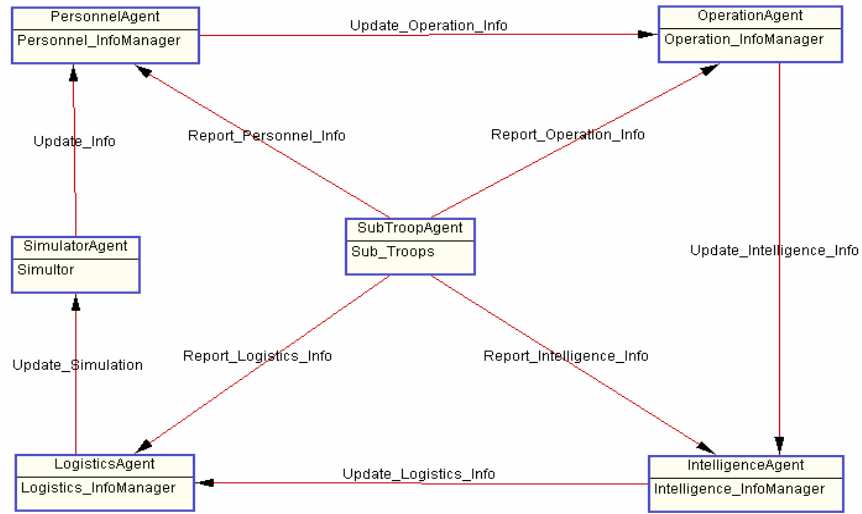
Figure 16. C3I Simulation System Architecture

The goal of the C3I simulation system is to support a commander's decision-making process by integrating various sources of information using distributed agents. The system is constructed using agentTool.   The system consists of six types of agents and nine types of conversations between agents.   In Figure 16, agents are shown as rectangles and conversations are shown as arrows between agents. *SubTroop* agents collect battlefield information and report this information via conversations to a higher level agent according to the information categories such as *Report Personnel Info* or *Report Operation Info*. *Personnel*, *Operation*, *Logistics*, and *Intelligence* agents

manage different battlefield data based on the *SubTroop* agents' reports.   A *Simulator* agent presents the latest battle situations to the commander.   To update the battle simulation, a *Simulator* agent queries new battlefield data via an `Update Info` conversation with a *Personnel* agent.   The *Personnel* agent then starts a series of conversations (*Update Operation Info*, *Update Intelligence Info*, *Update Logistics* Info, and *Update Simulation*) to report the latest battlefield information to the *Simulator* agent.

This research instantiated a C3I simulation system consisting of ten agents: five *SubTroop* agents and one of each type of the remaining agents.   *SubTroop* agents are named 1^{st}_Div, 2^{nd}_Div, 3^{rd}_Div, 4^{th}_Div, and 5^{th}_Div.   The *Simulator* agent is named *CommandPost* and other agents are named after their type such as *Personnel*, *Operation*, *Intelligence*, and *Logistics*.   All agents are implemented using agentMom and distributed across different machines on a local area network.

4.4.2 Select Target Agents

Developers start the multiagent system visualization process by initializing one or more *VisAnalysis* agents.   Based on the total number of agents that the developer wants to analyze in the observed system, the developer needs to determine how many *VisAnalysis* agents are required.   Multiple *VisAnalysis* agents may be required if developers need to analyze many agents in the observed system.   Due to the limited display area of the Agent Relationship and Conversation Flow views that are produced by an *VisAnalysis* agent, the recommended rate is one *VisAnalysis* agent for 20 to 30 observed agents.

After the *VisAnalysis* agents are initialized, developers are required to select target agents and input the target agents' registry data with the *VisAnalysis* agents. The visualization system allows developers to choose target agents dynamically for various analysis purposes. Developers may need to see different parts of the system depending on the different analysis situations. A developer may want to observe the overall system behavior to seek performance bottlenecks, to understand the system execution behavior, or to find out major actors in the system. Major actors in the system can be the agents that involve many interactions with other agents, or the agents that have important resources that should be shared among agents. Another developer, on the other hand, may want to observe a part of the system to debug errors, or to focus on agents of interest.

The last task in this step is to input the selected agents' registry data into the *VisAnalysis* agents. The agent registry data describes the agent's name, type, physical (network) location, and communication port. *InfoGathering* agents use the agent registry data to communicate with the target agents for message exchange data collection.

Since the visualization system is independent from specific agent platforms, an agent directory facilitator or agent management services are not provided to help input the selected agents' registry data. Developers interact with the *VisAnalysis* agents to manually input the agents' registry data or automatically load this data from agentTool system deployment diagram if available.

Figure 17 shows the results of an agent selection process in which a developer initialized a VisAnalysis agent called `C4I_VisAnalysis`. The developer then

selected *CommandPost*, *Intelligence*, *1ˢᵗ_Div*, and *Personnel* agents (right pane) and input the *Personnel* agent's registry data.
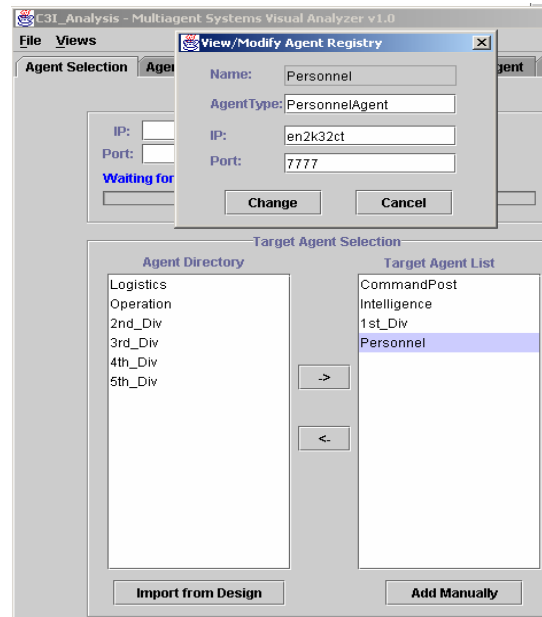

Figure 17. Select Target Agents with a VisAnalysis Agent

Developers can select / deselect agents dynamically using right or left arrow buttons located in the center between two panes. The selected agents' registry data will be transmitted to the selected *InfoGathering* agents in the `Data Collection` step for gathering the target agents' message exchange data.

4.4.3 Create Agents' Visual Structures

In this step, developers map the selected agents to graphical attributes for colors, shapes, lines, and rendering options. The visualization system allows developers to select colors, shapes, lines and rendering options to differentiate agents in the Agent Relationship, Strip, and Conversation Flow views. In this way, developers can increase

the cognition effect for recognizing the agents in such views.   The visualization system also enables developers to save the agents' visual structures into a file to be reloaded at a later time.   This reduces the developers' efforts to create the agents' visual structures in different visualization sessions.

Figure 18 shows a session in which a developer selected a green, thin lined triangle with stroke plus fill rendering option for the *Intelligence* agent.   The target agent is highlighted in the left pane when the developer creates the agent's visual structure.   Developers can preview the visual structure for an agent using the "Display" box in the right.
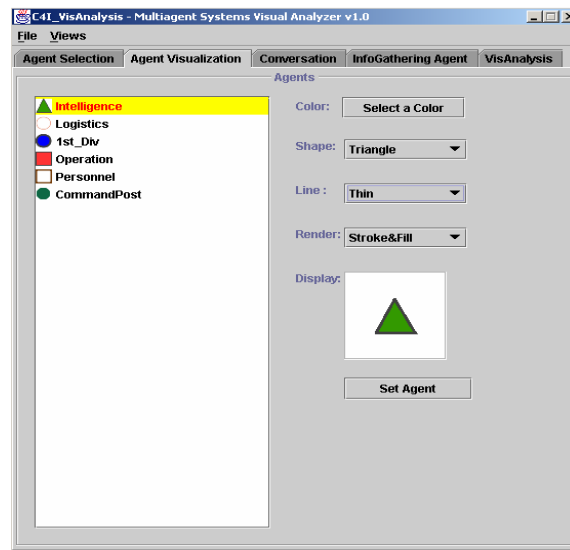

Figure 18. Create Agents Visual Structures

4.4.4 Select Target Conversations / Create Conversations' Visual Structures

Developers select target conversations and create the selected conversations' visual structures in this step.   Selecting the conversations has similar purposes as

selecting target agents in the previous step. Developers may want to observe all conversations among agents at the beginning and narrow down to an interesting subset of the conversations later on. Different developers may want to focus on specific conversations for other purposes, for example, analyzing the conversations that access shared variables.

To help developers' recognize selected conversations in the Agent Relationship, Strip, and Conversation Flow views, the visualization system allows developers to select a color and line type to represent the conversations. In Figure 19, the developer selected a red, flat line for displaying `Update_Info_Simulator_Agent_I` conversation (highlighted in the left pane). Developers can select / deselect conversations dynamically using the `Add Manually` or `Remove Conv` buttons shown in the top right side of this figure.
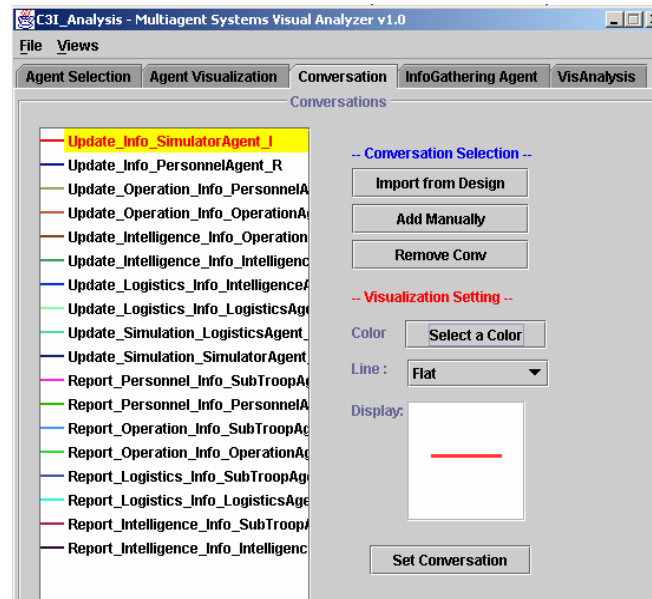

Figure 19. Select Target Conversation / Create Visual Structures

4.4.5 Data Collection

After the target agents and conversations are selected and their visual structures are constructed, developers need one or more *InfoGathering* agents for acquiring the message exchange data from the target agents. Developers need to consider how many *InfoGathering* agents are required based on the number of target agents. Since assigning too many agents to an *InfoGathering* agent may decrease the *InfoGathering* agent's data collection performance, developers may need to distribute the data collection task loads across multiple *InfoGathering* agents. In addition, developers may encounter a performance bottleneck by collecting and presenting too much information on the same machine. To prevent such bottlenecks, the visualization system allows developers to separate the data collection and the data presentation processes by running *InfoGathering* agents and *VisAnalysis* agents on different machines.

Figure 20 shows a scenario in which a developer employed three *InfoGathering* agents for collecting data, `Local_IGAgent`, `Remote_IGAgent_1`, and `Remote_IGAgent_2`. Developers can initialize local *InfoGathering* agents (run on the local machine) or connect to remote *InfoGathering* agents (run on different machines) by using the graphic user interface in the *VisAnalysis* agent (two boxes in the top of the InfoGathering Agent tab).
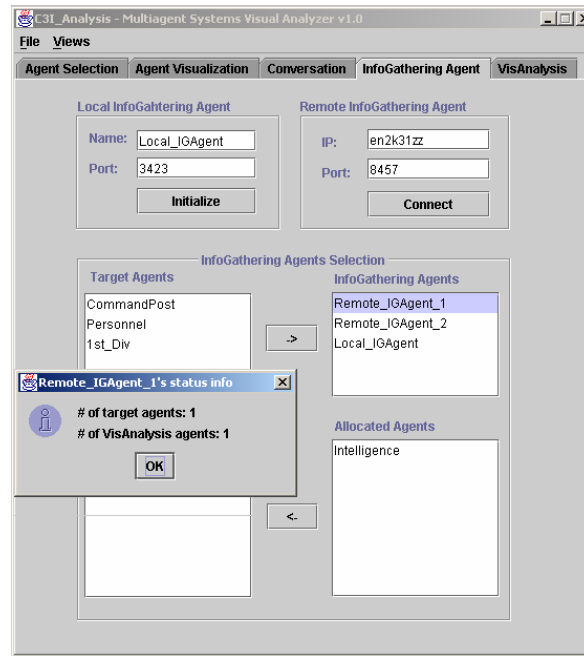
Figure 20. Employ InfoGathering Agents

To begin the data collection process, developers need to assign the target agents to *InfoGathering* agents. Target agents are displayed in the left pane and available InfoGathering agents are shown in the right top pane. In Figure 20, a developer assigned the *Intelligence* agent to the `Remote_IGAgent_1` to collect message exchange data from the *Intelligence* agent. Once an agent is assigned to an *InfoGathering* agent, the *VisAnalysis* agent automatically starts an `InfoGathering Request` conversation to ask the *InfoGathering* agent to collect the incoming messages to the agent.

Developers can stop the data collection process anytime by releasing target agents from an *InfoGathering* agent. Once an agent is released from an *InfoGathering* agent,

50

the *VisAnalysis* agent automatically starts an `InfoGathering Cancel Request` conversation to ask the *InfoGathering* agent to cancel the data collection from the agent.

Assigning and releasing agents can be simply done by mouse clicking on the right or left arrow button in Figure 20.   Developers also can check the assigned agents and current state of *InfoGathering* agents.   The dialog box in Figure 20 shows the result of a developer checking the `Remote_IGAgent_1`'s current state.

To collect data from the example multiagent system (C3I simulation system), this research modified the example system's infrastructure, agentMom, for acquiring message exchanges between agents.   Figure 21 shows the modified agentMom and the data collection process.
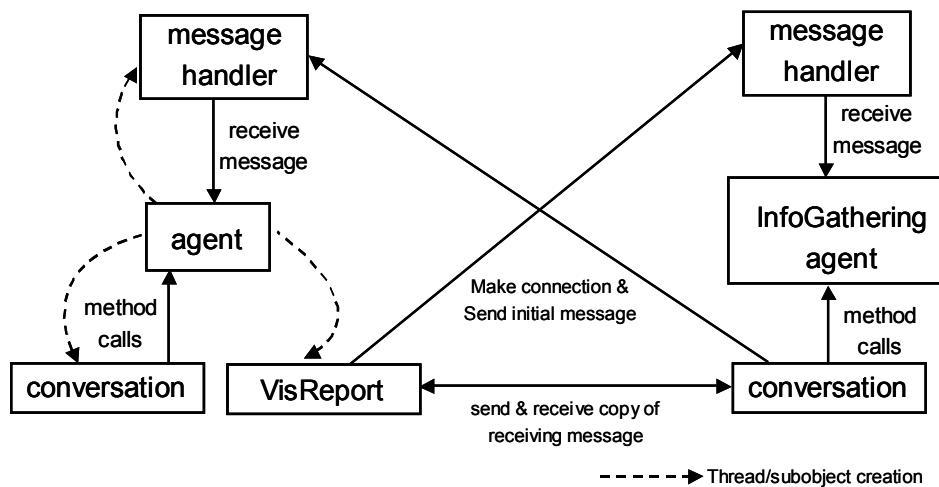

Figure 21. Modified agentMom for data collection

A new conversation, `VisReport`, is added to agentMom for sending a copy of receiving message to *InfoGathering* agents.   The *agent* class is modified to instantiate `VisReport` conversation with *InfoGathering* agents and to manage the *InfoGathering*

51

agent's registry data. Figure 21 shows the modified agentMom and the data collection process. Once an agent received data collection request from *InfoGathering* agents, it starts sending a copy of receiving message to the *InfoGathering* agents until *InfoGathering* agents send a data collection cancel request to the agent.

4.4.6 Data Presentation

Data presentation in multiple views is the last step of the visualization process. Once the message exchange data arrives at the *VisAnalysis* agents from the *InfoGathering* agents, the *VisAnalysis* agents automatically start processing the data to map the data into the Agent Relationship, Conversation Flow, Strip, and Statistics views. *VisAnalysis* agents use predefined visual structures of selected target agents and conversations for presenting message exchange data in such views. Developers begin a visual execution analysis session of the system by interacting with the multiple views that are linked by message exchange data. The details of visual execution analysis using the multiple views are described in Chapter 5.

4.5 Summary

This chapter describes the agent-based visualization system that supports the visual execution analysis methodology. The visualization system is comprised of InfoGathering agents and VisAnalysis agents. The visualization system enables users to configure the visualization system dynamically for improving the visualization performance. Developers interact InfoGathering agents and VisAnalysis agents to

collect data from the observed system, specify visual structures of agents and conversations, and observe the multiple views.

V. Visual Execution Analysis

5.1 Introduction

This chapter describes how developers apply visual execution analysis for analyzing multiagent systems. Section 5.2 discusses the requirements of multiagent system execution analysis. Section 5.3 describes how developers exploit the multiple views to analyze the observed system's behavior and evaluate the system performance.

5.2 Requirements

This research derives the requirements of visual execution analysis from the characteristics of multiagent systems [28; 19; 20]. First, agents are distributed and they may be mobile. Developers need to know where agents are physically located since their hardware platform can affect the agent's execution. Second, agents are communicative. Keeping the history of agent communications is important for analyzing multiagent systems. In addition, developers often need to see a large number of message exchanges simultaneously for tracing the system's evolution. Third, agents work together for a common purpose and they organize dynamic relationships depending on the given problems and tasks. Analyzing the dynamic organizational structures of different problem solving situations is often beneficial for optimizing the system's performance and evaluating the agents' role assignment. Fourth, the multiagent system's performance varies as the system's configuration changes. Developers need to compare the system performance with different configurations to search for an optimal

system configuration. Providing statistical data and offline replay capability can significantly reduce a developer's effort for comparing different system performance. Last, multiagent systems can be more easily understood and analyzed when they are viewed at a high level. However, sometimes developers need to focus on the detailed level of analysis such as inspecting the source code, tracing an agent's execution line by line, or monitoring many variables in the system. Therefore, a visual execution analysis tool needs to show a multiagent system with various levels of abstraction.

To summarize, visual execution analysis of multiagent systems requires:

- Visualizing agents physical locations

- Keeping the history of agents communications

- Visualizing a large number of message exchanges simultaneously

- Showing dynamic agent relationships

- Providing statistic data and offline replaying

- Describing multiagent systems with multiple levels of abstraction

To achieve efficient multiagent systems execution analysis, these requirements must be satisfied.

5.3 System Analysis Using Multiple Views

In this research, the visualization system collects and displays multiagent system execution data at various levels of abstraction. The visualization system provides multiple views based on the collected messages from multiagent systems.

- The *Agent Relationship View* displays acquaintance relationships between selected agents and animates message exchanges between agents.

- The *Conversation Flow View* depicts the sequencing of messages between specified agents during a certain period of system execution.

- The *Strip View* presents the history of agent communications according to a certain order selected by the developer.

- The *Message View* shows the details of a selected message.

- The *Content View* shows the details of an object passed within a message.

- The *Statistic View* shows various statistical summaries of system performance results.

Since different views have individual advantages, they may be applied for different purposes in evaluation tasks. Developers can interact with these views to configure the visualization, to obtain detailed information, or to arrange the information more conveniently.

To demonstrate the benefits and usability of the visualization techniques, this research used message exchange data from the C3I simulation system described in section 4.3.1. The system was instantiated with ten agents that are distributed across different machines on a local area network. Then one *VisAnalysis* agent and two *InfoGathering* agents were initialized to analyze the system executions. After only a few minutes, hundreds of messages were generated among agents and collected by the *InfoGathering* agents. The *VisAnalysis* agent then created multiple views using the message exchange data.

5.3.1 Agent Relationship View

The Agent Relationship View is the main view for understanding, evaluating, and analyzing multiagent systems. This view satisfies the following visual execution analysis requirements: (1) visualizing an agent's physical locations, (2) describing system structures with multiple levels of abstraction, and (3) showing dynamic agent relationships.

The Agent Relationship View displays agents and their message exchanges simultaneously. By displaying conversations between agents, the Agent Relationship View correlates the system execution with the system design. Figure 22 displays seven agents and their relationships. Individual agents are identified by icons with different shapes and colors that are selected by the developer. Agents are labeled with the agent name, the communication port, and the system identifier. Message exchanges are displayed by the gray lines between agents with the thickness of the line (along with a

label) depicting the total number of messages exchanged between two agents. In Figure 22, the example view resembles the system architecture design shown in Figure 16. The visualization system automatically connects the appropriate agent icons and conversations. Developers can then move agents and choose display options in the Agent Relationship view to enhance understanding of the system structure and the system executions. This enables developers to recognize the system structure without knowledge of the design documents. If developers already know the system structure, they can validate the system design by comparing the Agent Relationship View with the system structure in the design documents. Developers also can discover undesired agent interactions by comparing the Agent Relationship View with the system design. For example, it is an undesired agent interaction if *Operation* agent exchanges messages with *Logistics* agent since there is no conversation between these agents in the system design document shown in Figure 16.
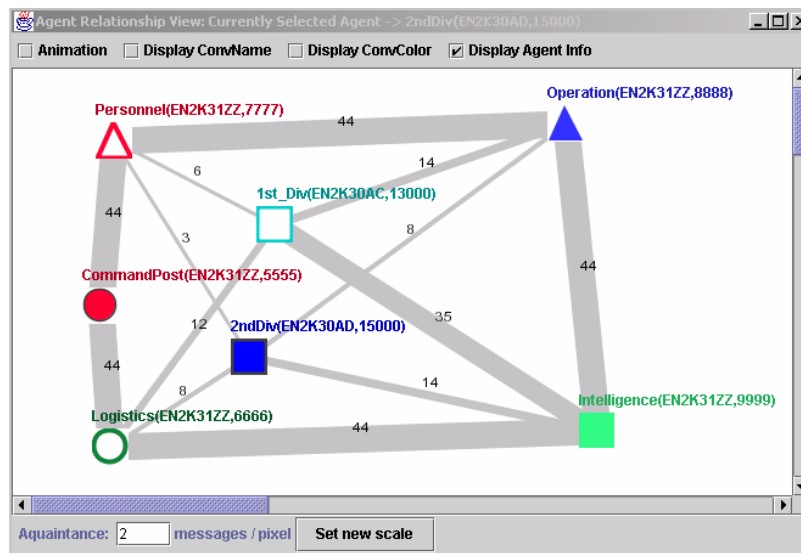

Figure 22. Agent Relationship View of C3I System

The Agent Relationship View provides a variety of other information.   The view enables developers to identify potential bottlenecks of the system.   In Figure 22, developers can recognize that the *Intelligence* agent is sending and receiving many more messages than other agents.   *Intelligence* agent's performance may be decreased by too many message transactions.

Developers can identify task synchronization and coordination between agents by monitoring conversations between agents.   For example, the sequence of messages "request to do task2" $\rightarrow$ "accept t2" $\rightarrow$ "notification of end t2" between agents shows that a task t2 is coordinated and completed between the agents.   In addition, developers can inspect the progress or results of the task by checking the detail of the messages. The Message View is developed for enabling developers to check the sequence of messages and the details of a message.   The Message View shows a list of all the messages that are exchanged between two agents.   Figure 23 shows a message list between *1ˢᵗ_Div* and *Intelligence* agents.   When developers select a message in the list, the details are shown in "Message Info" box.   In Figure 23, the message is sent from *1ˢᵗ_Div* agent to *Intelligence* agent for reporting intelligence information.   The message sending time, receiving time, and host information (address and port number) are also shown in this figure.   If the message contains object content, further drill down reveals the object structure as shown in Figure 24.
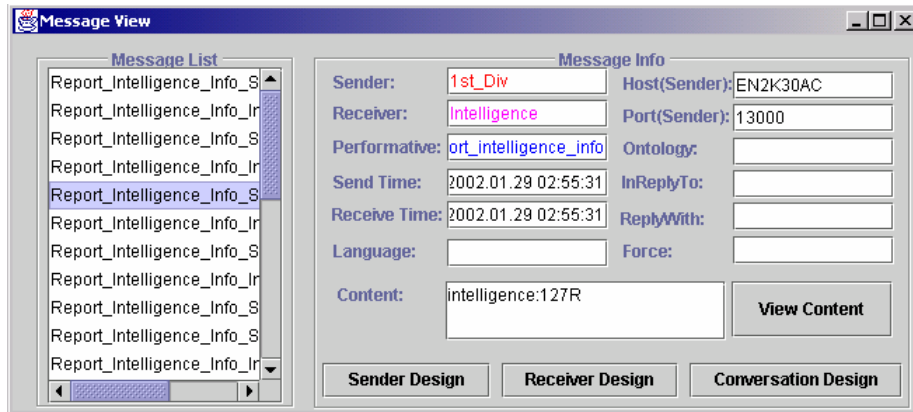
Figure 23. Message View listing all messages between 1st_Div and Intelligence
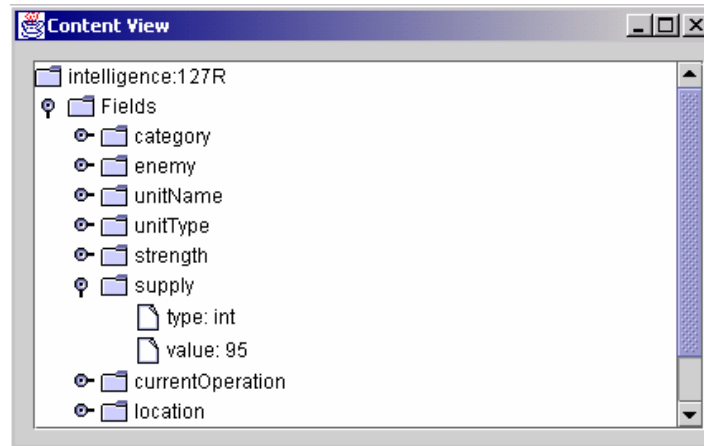

Figure 24. Content View depicting details of the object in the message

The Agent Relationship View shows conversations using a label on the message line or a color-coded line that is selectable by the developer. In Figure 25, two conversations are labeled and other conversations are color-coded. I^{st}_Div is in the `Report Intelligence Info` conversation with *Intelligence* agent to report intelligence data. To examine the messages in the conversation, developers bring up the Message View by clicking on the message line.

Developers can filter messages by setting the acquaintance variable in the Agent Relationship View. Filtering is beneficial when developers analyze large amounts of message exchange data. In Figure 25, a developer resets the acquaintance variable equal to 20 for checking the message lines that involve more than 20 messages. Resetting the acquaintance variable also changes the message line thickness. Six message lines out of thirteen are displayed by the new acquaintance variable. In this way, developers can easily find the links with large communication traffic in the system. Developers also can recognize the major actors that have many message exchanges with other agents.
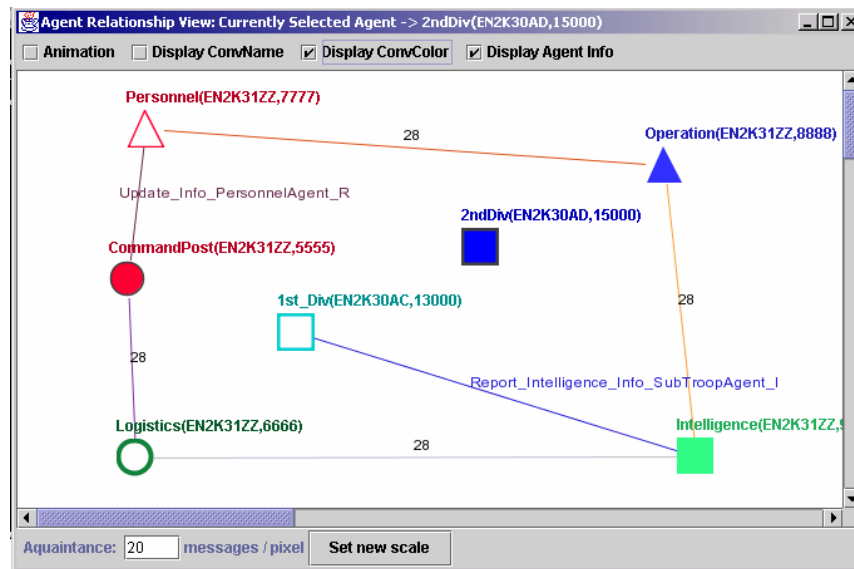

Figure 25. Agent Relationship View displaying various kinds of information

As a message is sent, an animated line is drawn from the sending agent to the receiving agent, providing a mechanism for following information or control flow through the entire system.

5.3.2 Conversation Flow View

Although the Agent Relationship View is useful for comprehending the overall structure and the information flows in a multiagent system, it is often desirable to focus on the timing and sequencing of message exchanges during a specific period of system execution. The Conversation Flow View was created to facilitate identification of sequential dependencies between messages, long delays for any given message, and overall timing patterns among messages. The Conversation Flow View also meets one of the visual execution requirements: Visualizing a large number of message exchanges simultaneously.

When developers generate the Conversation Flow View, they can select agents and conversations for filtering out unnecessary information and clarifying the view. To help developers with selection, the Conversation Flow View Dialog (Figure 26) provides information about the currently available agents, conversations, and time frame.
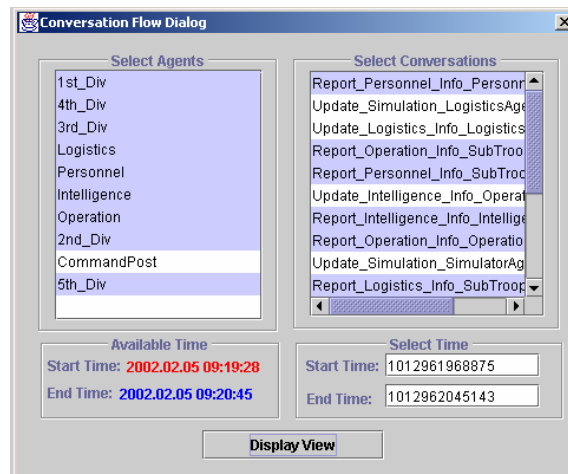

Figure 26. Conversation Flow View Dialog

Using the Conversation Flow View Dialog, developers can generate multiple Conversation Flow Views with different selections of agents, conversations, and time frames.

The Conversation Flow View (Figure 27) consists of two panes, the overview pane (top pane) and the detail view pane (bottom pane).   In both panes, agents are represented by color-coded horizontal lines and a message is depicted as a line starting at the send time on the sending agent line and ending at the receive time on the receiving agent line.   An *S* at the end of the message line indicates the sending agent and a small colored rectangle at the other end of the message line indicates the receiving agent of the message.   Time slots are shown with different time labels and vertical lines above the agent lines.   When the mouse is moved over a message, that message is highlighted and a text bubble is displayed summarizing the message.   Mouse clicking on the message brings up the Message View for displaying the details of the message.
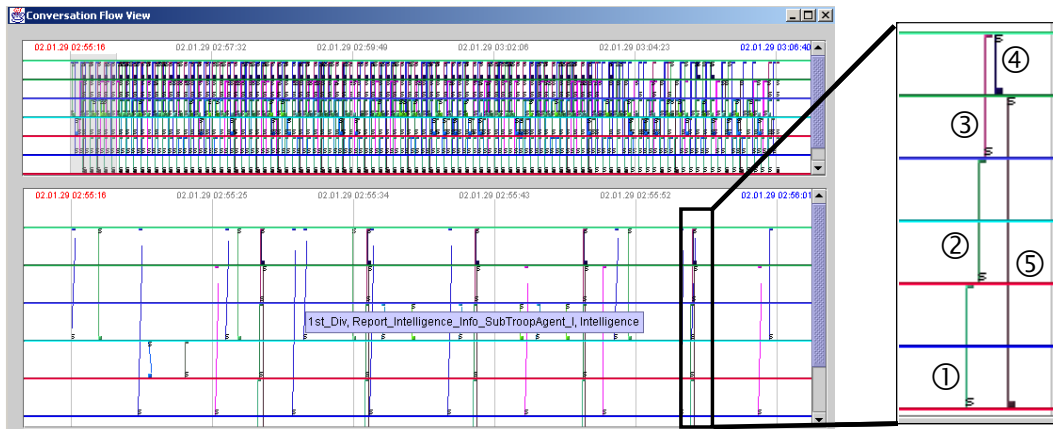

Figure 27. Conversation Flow View displaying 1220 messages

In Figure 27, messages are represented by nearly vertical lines since all agents are hosted within a local area network and message delay times are very short. Lengthy delays between sending and receiving a message would result in diagonal lines. This is useful for identifying delayed information in the system and for debugging and optimizing performance since the delayed information complicates agent behavior and lowers performance of the whole system

In a typical system with hundreds of thousands of messages, it can be difficult to identify individual messages when displayed all at once as shown in the overview pane of Figure 27. For this reason, the view provides an interactive zooming capability that enables more expanded plotting of areas of interest in the detailed view pane. Developers can move the selection window in the overview pane and change its size to observe an appropriate level of detail.

The Conversation Flow View also provides agent interaction pattern information. In Figure 27, a repetitive communication pattern is shown in the detailed view pane (enlarged in the right side of the view). Using the zooming capability, the developer can check the sequence of messages between agents and the content of the messages. The sequence of messages is ① Update Info, ② Update Operation Info, ③ Update Intelligence Info, ④ Update Logistics Info, and ⑤ Update Simulation. This sequence of messages matches the agent communication design as shown in Figure 16. Developers can utilize this information to evaluate the correctness of the system execution behavior as compared to the system design.

5.3.3 Strip View

The Strip View tracks the history of agent communications.   The Strip View depicts all messages in the order of sender, receiver, sending time, receiving time, or conversation name.   In the Strip View, messages are color coded by sending agent, conversation name, and receiving agent.   Figure 28 shows a Strip View with messages sorted by the sender.   For each sender, messages are further sorted by sending time. When the mouse is moved over a message, that message is highlighted and a text bubble is displayed summarizing the message.   Developers can drill down to view the details by clicking on the message in a manner similar to the Agent Relationship and Conversation Flow views.
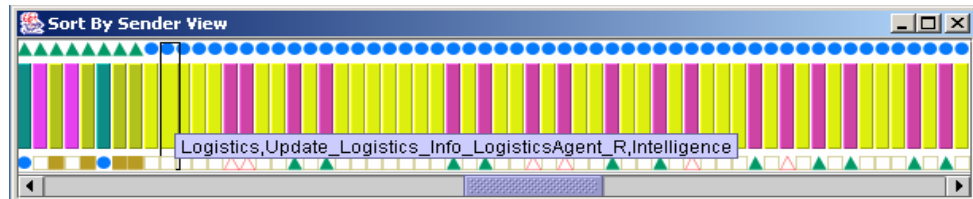

Figure 28. Strip View displaying messages sorted by Sender

5.3.4 Statistics View

A variety of Statistics Views provides statistical analysis from a multiagent system execution.   These views present summary information (left side of Figure 29) about the system such as the total number of messages among selected agents and the longest, shortest, and average message delay.   A series of charts (right side of Figure 29) also depicts the distribution of messages, tasks, agents, and conversations in the system. Since agents are autonomous objects and they may show different behaviors on every

system run, such information is useful for comparing system behavior between execution runs and for identifying messages, agents, and tasks that are frequently executed.
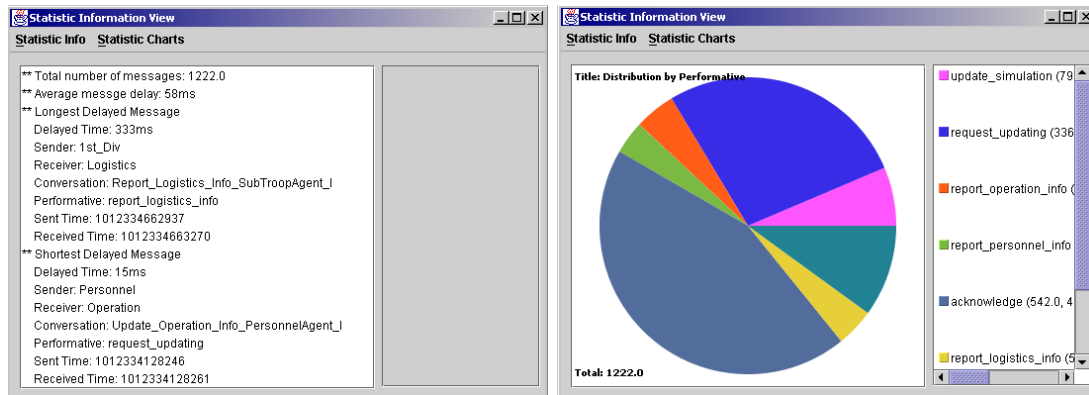

Figure 29. Statistic View presenting message delay summary (left) and message usage summary (right)

Developers also can utilize statistical information to optimize system performance. In the example shown, the most frequently executed task was `acknowledge` (44.3 %). Therefore, optimizing the `acknowledge` task can be the possible solution to increase the system's performance.

5.3.5 Visualizing the Errors

With larger numbers of agents in multiagent systems, the probability of an agent or communications failure increases. Since these agents may be separated geographically, it may be difficult to determine when a failure occurs. The Agent Relation and the Strip Views deal with this problem by tracking when a message fails to reach its destination. When this occurs, the message sending and receiving agents are marked with a large X until they correctly receive or send another message in the Agent Relationship View (top of Figure 30). The Strip View presents error messages in the

same manner as the Agent Relationship View except it uses a large E to mark errors (bottom of Figure 30).
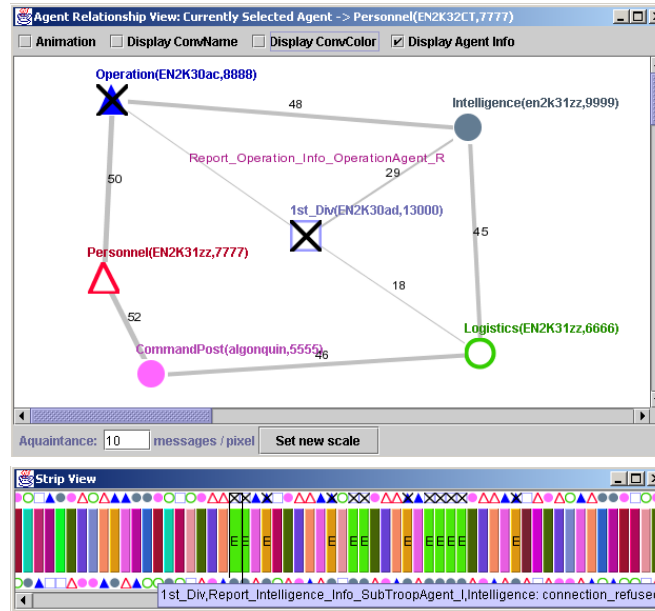
Figure 30. Agent Relation and Strip Views displaying Errors

If an agent has failed, it cannot send or receive any messages so the X remains. The large X and E immediately draw one's attention to the error. This is important since errors are likely to have a dramatic impact on system behavior and performance.

In addition to errors caused by system failures, an agent may receive a wrong message that was sent out of order or incorrectly defined by the sender. If the receiving agent recognizes a message as an error, it may respond with a message describing the error. Such error messages are also identified by the visualization tools and annotated with a large X.

5.3.6 Replaying System Execution Behavior

It is very hard to generate the same program execution behavior repeatedly from multiagent systems since agents perform their tasks asynchronously. For this reason, the visualization system records all information during system execution and developers can save the information to a file for replaying the visualization session. By replaying different scenarios, developers can analyze performance across different program execution environments. If a problem is overlooked during initial visual monitoring, replaying capability enables the analyst to replicate the problem during later analysis. The Message Loader (top of Figure 31) enables the analyst to trace message exchange data in a forward or backward direction.
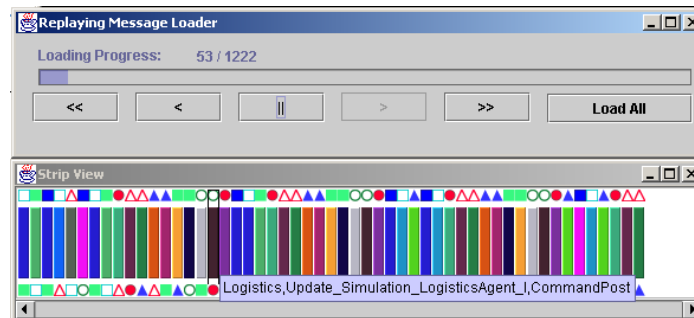
Figure 31. Replaying System Execution with Strip View

5.4 Summary

This chapter demonstrates how developers apply visual execution analysis for multiagent systems. The requirements of multiagent system execution analysis are derived according to multiagent systems' characteristics. Multiple views are generated from VisAnalysis agents to satisfy the requirements and to present various aspects of the

system's execution and performance.   The Message Loader provides record and playback capability for non-real-time analysis to compare different system execution results.

VI. Results

6.1 Introduction

The previous chapters of this thesis demonstrate how the application of visual execution analysis helps developers understand, analyze, and troubleshoot multiagent systems. This chapter compares this research effort with previous works, summarizes this research, and suggests areas of future work that will enhance and extend this research.

6.2 Analysis

As discussed in Chapter 2, a limited amount of work has addressed the visualization of multiagent systems. This research differs from the above works by providing an advanced generic program execution analysis methodology for any type of multiagent system.

For data collection, previous works focused on the message exchanges between agents; however, the visualizations are tightly integrated with system implementation or a specific agent platform. These aspects can be a major limitation for adapting these visualizations to other types of multiagent systems built on different agent platforms. This research provides a more flexible and adaptable data collection methodology since *InfoGathering* agents can be integrated into any multiagent infrastructure in which the observed system is running to capture communications between any agents in the observed system. Data collection is achieved by using duplicate messages without changing the sequence of messages. Unlike existing approaches, this research separates

data collection and data presentation processes by using *InfoGathering* and *VisAnalysis* agents. Scalability and reliability are enhanced since multiple *InfoGathering* and *VisAnalysis* agents can be instantiated and distributed throughout the system to avoid platforms or network links that might cause bottlenecks.

For data presentation, this research provides unique, detailed, and dynamically configurable views for analyzing multiagent systems behavior. Schroeder and Noy's Agent Messaging View and ZEUS's Society View provide a high-level viewpoint of agent relationships similar to the Agent Relationship View in this research. However, neither of these works captures the timing and sequencing of the various types of messages and the dynamic relationships between agents. These works only visualize certain message types and predefined agent relationships. This research, on the other hand, can handle any types of messages and it shows an agent's relationships by message exchanges. In addition, one can recognize information flows in the system by animating the sequence of the messages in the Agent Relation View. Task dependency between agents and delayed information in the system can be recognized in the Conversation Flow View that captures both timing and sequencing of messages. The Strip View displays the entire history of the agent communications with different sorting orders. Drill-down and filtering capabilities in this various views enable one to easily navigate the system's behavior for further inspection. Such features do not appear in previous works. Critical events such as communication error or undesired behavior are highlighted in the Agent Relationship and Strip views to focus one's attention and provide detailed information about the events.

Similar to this research, ZEUS supports the statistical analysis of system execution and offline replaying capability; however ZEUS requires more effort to regenerate visualizations since it does not support preserving both visual structures of agents and message exchange data. This research makes it easier to revisit different visualization sessions by saving visual structures and message exchange data simultaneously.

ZEUS provides the Reporting Tool to trace the progress of an agent's current tasks. This research does not provide such a feature since it requires additional messages for reporting the task states and brings more interference to the observed agent's original behavior. However, such a tool seems to be necessary when the developer wants to analyze each agent's internal states. This view may be considered as a future addition to this research.

6.3 Summary

This research addresses execution analysis, a critical need in the development of multiagent systems. Program execution analysis is important to improve initial system design by monitoring, analyzing, and troubleshooting the complex multiagent system's behavior. This research has described and implemented a visual execution analysis methodology for multiagent systems. The visualization system is extensible to any type of agent-based systems with only a small modification to the agent conversation infrastructure in the observed system.

Using the visual presentations, developers are able to observe thousands of messages simultaneously in multiple views showing various aspects of the system's behavior. These views provide the overview of the system, relationships among displayed agents, dependencies among agents, and the history of agent communications. Beginning with a high-level summary of the message exchange data, an analyst can progressively focus on smaller subsets of the data to be displayed in more detail by graphical techniques such as zooming or drilling down to the individual data of interest. The timing and sequencing of the messages are captured to identify the information flow in the system and to optimize the system performance. Critical errors are highlighted for debugging. Separation of data collection and data presentation processes provides better performance and a dynamically configurable visualization system.

Initial observations indicate that these capabilities significantly improve one's ability to analyze and evaluate multiagent systems. To conclusively validate the benefits of these visual presentations, however, requires additional experimentation with additional developers and different multiagent systems.

6.4 Future Work

Although the information gathering and visual analysis specification in this research requires little or no modification to the multiagent system implementation, it does require modifications to the agent messaging infrastructure to send duplicate messages. It is likely that these modifications and duplicate messages can be eliminated by exploiting the capabilities of the Java virtual machine or underlying network protocols.

To this point, this research has focused primarily on the communications between agents. For a comprehensive analysis and troubleshooting solution, it is necessary to monitor individual agent execution. Java debugging capabilities should facilitate this enhancement.

This research to this point has focused on exploring different visual techniques for multiagent system monitoring. Empirical and experimental user studies are needed to improve and validate the visualization techniques.

Bibliography

1   Aknine S. and H. Aknine. "Contribution of a Multi-agent Cooperation Model in a Hospital Environment," *Proceedings of the Third Annual Conference on Autonomous Agents*. 406-407. Seattle, WA, May 1-5, 1999.

2   Card, Stuart K. and others. *Readings in information visualization : using vision to think*. San Francisco: Morgan Kaufmann Publishers Inc, 1999.

3   DARPA. Control of Agent-Based Systems. http://coabs.globalinfotek.com/. 2002.

4   Cohen, P.R. and H.J. Levesque. "Communicative Actions for Artificial Intelligence'. *First International Conference on Multi-Agent Systems (ICMAS'95)*. San Francisco, V.Lesser (Ed.), MIT Press.

5   Deepika Chauhan. *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation*. PhD thesis, ECECS Department, University of Cincinnati, 1997.

6   DeLoach, Scott A.   "Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems," *Agent-Oriented Information Systems '99 (AOIS'99)*. Seattle WA, 1 May 1998.

7   DeLoach, Scott A. Class handout, CSCE 623, AI System Design. Graduate School of Engineering and Management, Air Force Institute of Technology, Wright-Patterson AFB OH, July 2000.

8   DeLoach, Scott A. "Analysis and Design using MaSE and agentTool," *12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*. Miami University, Oxford, Ohio, March 31 – April 1, 2001.

9   DeLoach, Scott A. and Mark Wood. "Developing Multiagent Systems with agentTool," *The Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*. Boston, MA, July 2000.

10   Finin, T., Y. Labrou and J. Mayfield. KQML as an Agent Communication Language. In: Software Agents, J.M. Bradshaw (Ed.), Menlo Park, Calif., AAAI Press, 1997, pages 291-316.

11   FIPA. "Part 2: Agent Communication Language." FIPA 97 Specification, Foundation for Intelligent Physical Agents, Geneva, Switzerland, November 28, 1997.

12   FIPA. FIPA-OS, http://www.nortelnetworks.com/products/announcements/fipa/. 1999.

13   FIPA. FIPA 2000, http://www.fipa.org/repository/fipa2000.html. 2000.

14   Hamburger, H., and G. Tecuci. "Toward a unification of human computer learning and tutoring," *Conference on Intelligence Tutoring System (ITS'98)*. B. P. Goettl, H. M. Halff, C. L. Redfield, and V. J. Shute, Eds. Springer, 1998.

15   Huhns, M.N. and M.P. Singh. *Agents and Multi-agent Systems: Themes, Approaches, and Challenges*. In: Readings in Agents, Huhns, M.N. and Singh, M.P. (Eds.), San Francisco, Calif., Morgan Kaufmann Publishers, 1998, pages 1-23.

16   Timothy M. Jacobs. "Visualization of Collaborative Software Systems." Research proposal of Air Force Institute of Technology, Wright-Patterson AFB OH, September 2000.

17   Timothy M. Jacobs. and Sean Butler. "Collaborative visualization for military planning." In *Java/Jini Technologies,* Sudipto Ghosh, Editor, Proceedings of SPIE Vol. 4521, 42-51 (2001).

18   Jacques Ferber. *Multi-Agent Systems*. New York:Addison-Wesley, 1999.

19   Jennings, N.R. and M. Wooldridge. "Applying agent technology", *Applied Artificial Intelligence, Vol. 9(4),* pp.351 – 361, 1995.

20   Jennings, N.R., K. Sycara and M. Wooldridge. "A Roadmap of Agent Research and Development," *In: Autonomous Agents and Multi-Agent Systems Journal*, N.R. Jennings, K. Sycara and M. Georgeff (Eds.), Kluwer Academic Publishers, Boston, 1998, Volume 1, Issue 1, pages 7-38.

21   Kaminka Gal A., David V. Pynadath and Milind Tambe. "Monitoring Deployed Agent Teams," *Proceedings of the fifth international conference on Autonomous agents*. 308-315. Montreal, Quebec, Canada, May 28 - June 1, 2001.

22   Finin Tim, Richard Fritzson, Don McKay and Robin McEntire. "KQML as an agent communication language," *Proceedings of the third international conference on Information and knowledge management*. 456-463. Gaithersburg, MD USA, November 29 – December 2, 1994.

23   A Proposal for a new KQML Specification, Yannis Labrou and Tim Finin, TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD 21250. February 1997.

24   Timothy H. Lacey. *A Formal Methodology and Technique for Verifying Communization Protocols in a Multi-agent Environment*. MS thesis, AFIT/GCS/ENG/00M-12. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

25   Marin O., P. Sens, J-P. Briot, and Z. Guessoum. "Towards Adapative Fault Tolerance for Distributed Multi-Agent Systems," *Proceedings of ERSADS'2001*. Bertinoro, Italy, May 14-18, 2001.

26   Nissen M. E. and A. Mehra. "Some Intelligent Software Supply Chain Agents," *Proceedings of the Third Annual Conference on Autonomous Agents*. 374-375. Seattle, WA, May 1-5, 1999.

27   Nowostawske Mariusz, Martin Purvis, and Stephen Cranefield. "Modelling and Visualizing Agent Conversations," *Proceedings of the fifth international conference on Autonomous Agents*. 374-375. Montreal, Quebec, Canada May 28-June 1, 2001.

28   Nwana Hyacinth. "Software agents: an overview", in *Knowledge Engineering Review*, vol. 2 N. 3, pp.205-244, October/November 1996..

29   Nwana Hyacinth, Divine Ndumu, and Lyndon Lee. "ZEUS: An advanced tool-kit for engineering distributed multi-agent systems", *Proceedings of the Third International Conference on the Practical Applications of Intelligent Agents and Multi-agent Technology*. PAAM98, London, UK, 1998.

30   OMG MASIF, formal/2000-01-02,
http://www.omg.org/technology/documents/formal/mobile_agent_facility.htm. 2000.

31   Schroeder Michael and Penny Noy. "Multi-Agent Visualization Based on Multivariate Data," *Proceedings of the fifth international conference on Autonomous agents*, 85-91. ACM Press, New York, NY, USA, 2001.

32   Peng, Y., T. Finin, Y. Labrou, B. Chu, Long, J., W.J. Tolone and A. Boughannam. "A Multi-Agent System for Enterprise Integration," *Proceedings of the Third International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology*, H.S. Nwana and D.T. Ndumu (Eds.), London, UK, March, 1998, pages 155-169.

33   Rapahel Marc J. *Knowledge base Support for Design and Synthesis of Multi-Agent Systems*. MS thesis, AFIT/GCS/ENG/00M-21. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

34   Wood Mark F. *Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems*. MS thesis, AFIT/GCS/ENG/00M-26. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.

35   Wooldridge Michael and Nicholas R. Jennings. "Agent Theories, Architectures and Languages: A Survey" in Wooldridge and Jennings Eds., *Intelligent Agents*, Berlin:Springer-Verlag, 1-22, 1994

36   Wooldridge Michael and Nicholas R. Jennings. "Pitfalls of Agent-Oriented Development", *Proceedings of the second international conference on Autonomous agents*. 385-391. Minneapolis, MN USA, May 10 - 13, 1998.

37   Ygge F. and H. Akkermans. "Making a Case for Multi-Agent Systems", *Multi-Agent Rationality: 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW '97),* Ronneby, Sweden, May 13-16, 1997, Eds. M. Boman and W. Van de Velde, Springerman 1997.

38   David Esther, Sarit Kraus. "Agents for Information Broadcasting," Agent Theories, Architectures, and Languages (ATAL-1999) 91-105, Orlando, Florida, USA 1999.

39   Stasko John T. The PARADE Environment for Visualizing Parallel Program Executions: A Progress Report. Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-GVU-95-03, January 1995.

40   Zhao, Qiang A. and John T. Stasko. *Visualizing the Execution of Threads-based Programs.* Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-GVU-95-01, January 1995.

41   Lawrence W. Andrea, Albert N. Badre, John T. Stasko. "Empirically Evaluating the Use of Animations to Teach Algorithms," *Proceedings of the 1994 IEEE Symposium on Visual Languages*, St. Louis, MO, 1994.

42   DeRose L., Y. Zhang, and D. Reed. *SvPablo: A Multi-Language Performance Analysis System,* Computer Performance Evaluation Modeling Techniques and Tools, 352-355.   Springer-Verlag, R. Puigjaner, N. Savino, and B. Sera (Eds), September 1998.

43   Shaffer Eric, Shannon Whitmore, Benjamin Schaeffer, and Daniel A. Reed. "Virtue: Immersive Performance Visualization of Parallel and Distributed Applications," *IEEE Computer,* 44-51, December 1999.

44   Heath M. T. and J. A. Etheridge. "Visualizing the Performance of Parallel Programs," *IEEE Software,* 29-39, September 1991.

45   Wu Xingfu and Wei Li. "Visualizing the Network Activity among Node Processors," *Proceedings of International Symposium on Transmission & Switching New Technology (ISTST'96),* China, September 1996.

46   DeRose L., M. Pantano, R.A. Aydt, E. Shaffer, B. Schaeffer, S. Whitmore, and D. A. Reed. "An Approach to Immersive Performance Visualization of Parallel and Wide-Area Distributed Applications," *Proceedings of the Eight IEEE International Symposium on High-Performance Distributed Computing,* 247-254, August 1999.

47   Vidal José M. "Multiagent Systems." n.pag. http://www.multiagent.com.   December 1998.

Vita

Captain Chong Kyung Kil was born in August 1971 in Seoul, Korea. He graduated from Kyung-Hee High School, Seoul in February 1990. He entered the Korea Military Academy in March 1990. In March 1994, he completed his undergraduate studies with a Bachelor of Science degree in Computer Science and he was commissioned in the Republic of Korea Army as a communications officer.

Captain Kil's first assignment was at the the 26th Mechanized Division as a communication platoon leader in June 1994. In December 1995, he changed his specialty as a computer officer and he was assigned as a systems analysis officer to the 5th Infantry Division. In August 1997, he was assigned as a program officer to the 3rd Army Headquarters. In July 2000, he entered the Graduate School of Engineering's Computer Systems Engineering program, Air Force Institute of Technology, Wright Patterson Air Force Base, Ohio. Upon graduation, he will be assigned to the Army Headquarters as a program officer.

1. REPORT DATE (DD-MM-YYYY) Aug 2002	2. REPORT TYPE **Master's Thesis**	3. DATES COVERED (From – To)

4. TITLE AND SUBTITLE VISUAL EXECUTION ANALYSIS FOR MULTIAGENT SYSTEMS	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Chong Kyung Kil, Captain, ROKA	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Building 640 WPAFB OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/02-12
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Office of Scientific Research Robert L. Herklotz, Ph.D. Program Manager: Software and Systems 801 N. Randolph St., Room 732 (703) 696-6565 Arlington, VA 22203-1977 e-mail: robert.herklotz@afosr.af.mil	10. SPONSOR/MONITOR'S ACRONYM(S) 11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Multiagent systems have become increasingly important in developing complex software systems. Multiagent systems introduce collective intelligence and provide benefits such as flexibility, scalability, decentralization, and increased reliability. A software agent is a high-level software abstraction that is capable of performing given tasks in an environment without human intervention.

Although multiagent systems provide a convenient and powerful way to organize complex software systems, developing such system is very complicated. To help manage this complexity this research develops a methodology and technique for analyzing, monitoring and troubleshooting multiagent systems execution. This is accomplished by visualizing a multiagent system at multiple levels of abstraction to capture the relationships and dependencies among the agents.

15. SUBJECT TERMS

Software Visualization, Multiagent Systems, Execution Analysis, Agent-based Visualization, Messages, Agent Conversations, Inter-agent Relationships, agentTool, agentMom

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Timothy M. Jacobs, Lt Col, USAF
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	UU	91	19b. TELEPHONE NUMBER (Include area code) (937)-255-6565 ext 4279, timothy.jacobs@afit.af.mil