APPLYING BROADCASTING/MULTICASTING/SECURED COMMUNICATION TO AGENTMOM IN MULTIAGENT-SYSTEMS

by

CHAIROJ MEKPRASERTVIT

B.A., Thammasat University, Thailand, 1997
B.S., Pittsburg State University, 2000

A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SOFTWARE ENGINEERING

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2004

Approved by:

Major Professor
Dr. Scott A DeLoach

ABSTRACT

agentMom is a framework for building multi-agent systems. The previous version of agentMom supports only the one-to-one communication using TCP/IP. The goal of this project is to integrate the broadcast, multicast and secured communication to agentMom. The new security features used in agentMom are Secure Socket Layers in one-to-one communication and Symmetric Key Algorithm in multicast communication. The entire project was implemented in java version 1.4. The total time spent in this project was 700 hours with 2200 source line of code, approximately.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 – PROJECT OVERVIEW

1. Background

1.1 Motivation

Communication is one of the critical parts in multi-agent systems because it enables the agents in multi-agent systems to exchange information and cooperate with each other. This raises the question of what communication technique to be used in multi-agent systems. An earlier implementation of agentMom developed by Dr. Scott A. DeLoach, allows only one-to-one communication using the TCP/IP protocol. However, in the situation where a single message is destined for many recipients, multiple copies of the same message have to be sent by using TCP/IP protocol. This is not very efficient when there are a large number of agents in the system. Broadcast/Multicast communication on the other hand prevents network overloading by generating a single message destined for multiple recipients. This technique also reduces the amount of conversations that an agent has to handle. Depending on the situations, an agent may choose to unicast over broadcast/multicast messages. For example, unicast message is more appropriate when there are very few recipients. Thus, by integrating broadcast/multicast communication capability, agentMom can provide a more flexible way for an agent to communicate with other agents

1.2 Multi-Agent Systems

Multi-Agent Systems are one of the most recent contributions in the field of software engineering in building distributed intelligent systems. It is described as a further abstraction of the object-oriented paradigm where agents are a specialization of objects. Agents are similar to objects; however, they have traits such as autonomy, cooperation, perception, and pro-activeness that imply characteristics that objects generally do not have. Basically, there are two differences:

1. Objects are passive. They react to external stimuli, but do not exhibit goal directed behavior.
2. Agents typically use a common messaging language between all agents whereas object messages are usually class dependent.

Therefore, an object is a logical combination of data structures and its corresponding methods while agents additionally support structures for representing mental state components such as attitudes, beliefs and goals.

1.3 Example of software agents

There are wide ranges of application domains that are making use of agent-oriented systems engineering. Software agents are being developed for fields as varied as entertainment, electronic commerce, user assistance, and information systems.

For example,

1. The animated paperclip agent in Microsoft Office
2. Computer viruses (destructive agents)
3. Artificial players or actors in computer games and simulations (e.g. Quake)
4. Trading and negotiation agents (e.g. the auction agent at Ebay)
5. Web spiders (collecting data to build indexes to used by a search engine, i.e. Google)

1.4 agentMom

Agents in Multi-Agent systems environment have to communicate with other agents to be able to cooperate and achieve the assigned goals. However, it is not an easy task for developers to manage all communications.

agentMom is a communication framework for multi-agent systems implemented in Java. It provides a framework for building agents, conversations between agents and messages passed in the conversations. Currently in agentMom version 1.2, it consists of four important classes: Agent, Conversation, Message and MessageHandler. The Agent class is an abstract class that defines the minimum set of requirements for an agent to use agentMom. The Conversation class is an abstract class that is basically used for sending and receiving message using the TCP/IP protocol. The Message class defines the field used in the message passing between agents such as host, port, sender and receiver. The The MessageHandler class is used to start a socket on the indicated port and wait for connection from another agent. Basically, it monitors the local port for establishing connection.

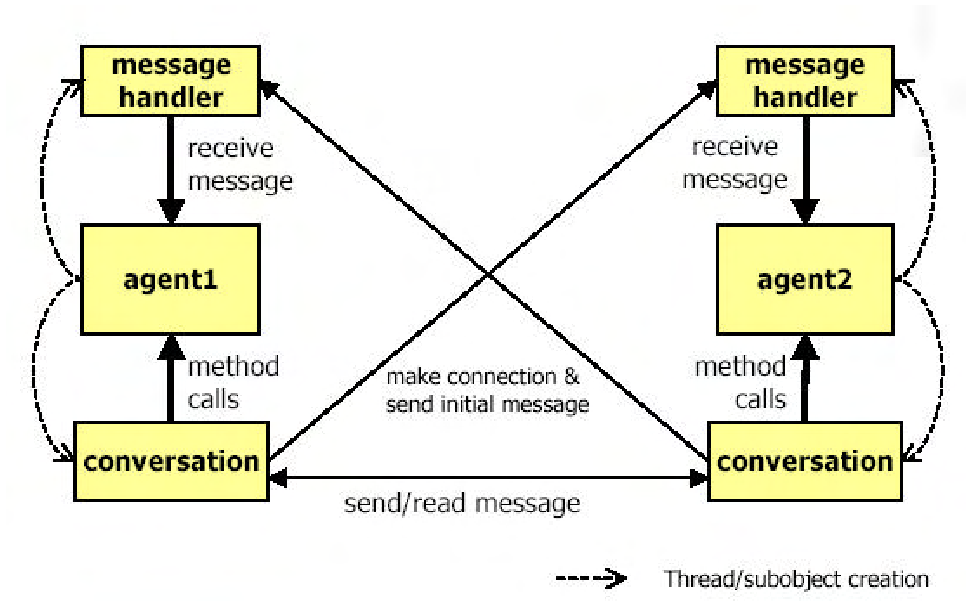
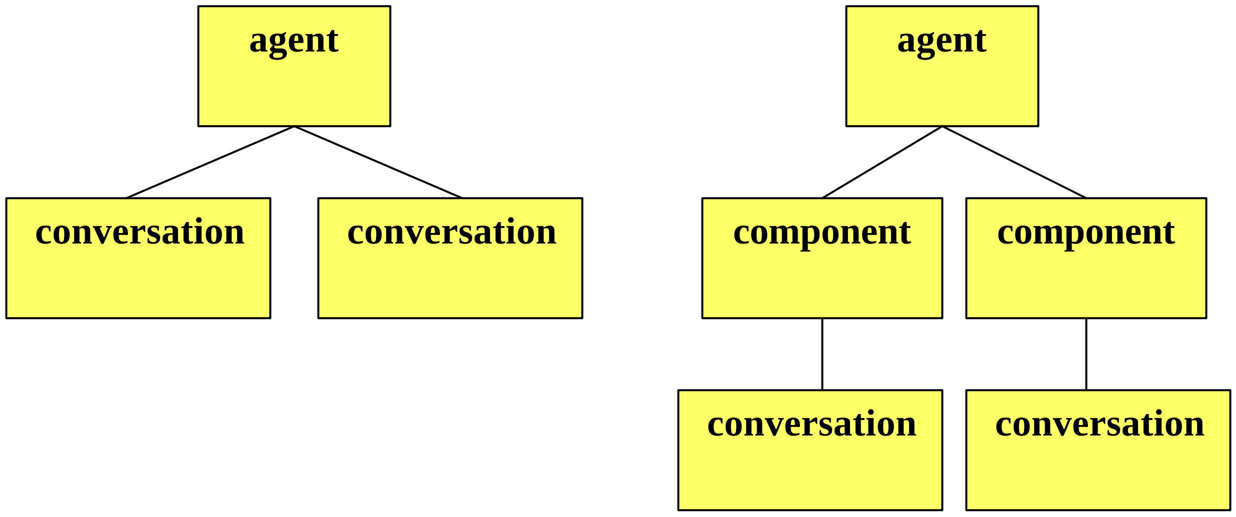


Figure 1. agentMom

An overview of how agentMom works is shown in Figure 1. When an agent wants to start a conversation with other agents, that agent starts the MessageHandler that opens the indicated port, and then waits for the connection to establish with the other agent's MessageHandler. An agent can start the conversation by establishing socket connection with another agent's MessageHandler. When connection has been established and the initial message has been validated, they can then start sending and receiving messages.



a. Agent directly controls conversations b. Component controls conversations

Figure 2. agentMom's architectures

Furthermore, there are two architectures that can be applied to agentMom. The first architecture is shown in Figure 2a. In the first architecture, agent directly controls the conversations. This architecture is very straightforward since conversations belong to agent. In the second architecture as shown in Figure 2b, an agent consists of one or more components, and the conversations belong to components, not directly to agents. Also, an agent can have multiple components and components can have multiple conversation. The difference from the first architecture is that component is responsible for making conversation with other agents. In the first architecture, agents are directly responsible for controlling the conversation. Having components separately from agent allows developers to map the agent role's tasks to the component. From now, we will refer to the first architecture as agent-based architecture and the second architecture as component-based architecture.

In this project, we will consider these two architectures in applying multicast, broadcast and security into agentMom.

2. Project Overview

2.1 Terms and Definitions

Unicast refers to one-to-one communication in such a way that a packet originates from a single Internet host, and it is destined to a unique location of another Internet host.

Multicast refers to one-to-many communication in such a way that a packet originates from a single Internet host, and it is destined to multiple receivers within the same multicast address.

Broadcast refers to one-to-many communication in such a way that a packet originates from a single Internet host, and it is destined to all receivers within the same local network.

Organization refers to a set of agents.

Institution refers to a set of the basic element required to build a particular type of organization consisting of goals, roles, rules and protocols.

Reorganization refers to a situation where the previous organization structure is not efficient to succeed the mission.
Group refers to a set of agents who agree to use the same multicast address to subscribe group message.
Time-To-Live (TTL) refers to the number of hops that multicast message is allowed to remain in the network before it is discarded by the router.

2.2 Overview

This MSE project is part of the research project "Autonomous Reorganization of Cooperative Robotic Teams for Robust Performance" supervised by Dr. Scott DeLoach. The main focus of this research is to provide autonomous cooperative robotic teams with enough knowledge of their team goals and organizational structure to allow them to autonomously organize and reorganize to achieve their team goal in the face of changing environmental conditions and individual team member failures.

The main focus of this MSE project is on extending agentMom capability to manage unicast, multicast and broadcast communication and to provide secured communication such as message encryption and decryption. Currently, agentMom1.2 only supports unicast communication without multicast broadcast and security. There are many advantages in using broadcast/multicast communication in multi-agent systems environment. First of all, when there are many agents in the system and the use of network bandwidth is critical, then sending multiple copies of the same message to each receiving agent may not desirable. Broadcast or multicast communication can save network bandwidth by sending a single message destined for multiple receiving agents. Secondly, when an agent has to send the same message to hundreds of agent, then this agent may not be able to do anything else, but sending and receiving messages. Broadcast and multicast techniques can reduce agent's workload. Furthermore, in the situation where the sender does not know the address of all agents in the system, sender can choose to multicast or broadcast the message to find agents on service. In the bidding/marketing-based technique, agents may broadcast or multicast messages to other agents for some services. The recipients of these messages evaluate those requests, and then submit bids with directed message to the originating agents. The originating agents use this information to choose the appropriate agent to do the jobs, and then send directed message back to the desired agents. Lastly, there are many situations that need to divide agents into different groups such as search group and rescue group. Agents may want to receive messages only from the group they belong to. Multicast communication supports this implementation.

In a multi-agent system, security is also an important issue when message is sent over a public network such as Internet. It is undesirable if someone who is not specified to receive message can see the content of it. Message encryption can prevent this situation.

Therefore, integrating broadcast/multicast communication and security features to agentMom can provide a more flexible way for communication in multi-agent systems.

2.3 Goal

Integrate multicasting, broadcasting and secured communication capability in agentMom in order to provide more efficient and effective way for communication in multi-agent environment.

2.4 Purpose

1. Enable agents to broadcast a message to all the agents within the same local network.
2. Enable agents to multicast a message to all the agents within the same multicast address.
3. Allow agents to choose among unicast, multicast and broadcast communication.
4. Allow agents to join and leave multicast group
5. Reduce network bandwidth from multiple copies of the same message by using multicast and broadcast communication.
6. Reduce agent's workload by reducing the number of sending and receiving messages.
7. Provide message encryption and decryption techniques.

2.5 Feature

1. Support unicast, multicast and broadcast communication.
2. Allow agent to choose which communication method to be used (unicast/multicast/broadcast) to fit the needs.
3. Allow agent to join and leave multicast group.
4. Allow agent to choose to encrypt or not to encrypt message.

2.6 Risk

1. Reliable message delivery – multicast/broadcast packets are delivered with best effort. Thus, a packet may be delivered to all specified agents or none.
2. Security – we provide some basic mechanisms for security such as message encryption. However, there is no guarantee that the others cannot decrypt the encrypted messages.

2.7 Direction

1. Reliable message delivery – scalable reliable message delivery is an important issue in multicast and broadcast communication. It is a hot research area in communication network and there is no single solution to this problem. Thus, this can be further in the future work.
2. FIPA Agent Communication Language (ACL) – this project can be further to conform to FIPA ACL specification, including FIPA ACL messages represented in XML.

2.8 Environment

1. This software package will be compiled using Java 1.4.2.
2. Rational Rose 2000 will be used for creating various object diagrams..
3. Eclipse IDE 2.1 will be used for coding the software package.
4. This software package will be tested under Microsoft Windows XP/2000, Linux Debian Linux and Solaris 9.
5. USE 2.0 will be used for modeling the formal specifications, using UML/OCL methodology.

CHAPTER 2 – SOFTWARE REQUIREMENTS SPECIFICATION

1 Introduction

This section provides an overview of this project.

1.1 Purpose

The purpose of this document is to describe functionality and behavior of the new agentMom framework. This document is intended to be viewed only by project advisor and committee members.

1.2 Scope

This document covers the software requirements for the project "Applying Broadcasting/Multicasting/Secured Communication to agentMom in Multi-Agent Systems".

1.3 Definitions, Acronym & Abbreviations

agentMom 1.2 refers to current implementation of agentMom
New agentMom refers to this project, including agentMom with capability of broadcasting, multicasting and secured communication
Unicast refers to one-to-one communication in such a way that a packet originates from a single Internet host, and it is destined to a unique location of another Internet host.
Multicast refers to one-to-many communication in such a way that a packet originates from a single Internet host, and it is destined to multiple receivers within the same multicast address.
Broadcast refers to one-to-many communication in such a way that a packet originates from a single Internet host, and it is destined to all receivers within the same local network.
Organization refers to a set of agents.
Reorganization refers to a situation where the previous organization structure is not efficient to succeed the mission.
Group refers to a set of agents who agree to use the same multicast address to subscribe group message.
Time-To-Live (TTL) refers to the number of hops that multicast message is allowed to remain in the network before it is discarded by the router.

1.4 Overview

The remainder of this document provides a greater detail functionality and requirement of the software. Section 2 describes product perspective, overall functionality, intended users, constraints and assumption of this software. Section 3 provides general Use Cases and specific requirement of this software.

2 Overall Description

This section provides an overview of the project functionality and factors that affect this project and its requirements.

2.1 Product Perspective

This project will be a framework that provides reusability of agent's communication. It is implemented in Java and provides the basic building blocks for building agents, conversations between agents, and the message that are passed in the conversations.

2.1.1 Software Interface – java version 1.4.0 is required to use the software.

2.1.2 Communication Interface – TCP/IP is used in order to send unicast message. Multicast protocol is used in order to send multicast message. UDP is used in order to send broadcast message.

2.2 Product Functions

2.2.1 Enable agents to broadcast a message to all the agents within the same local network.

2.2.2 Enable agents to multicast a message to all the agents within the same multicast address.

2.2.3 Enable agents to unicast a message to other agents within organization.

2.2.4 Allow agents to choose among unicast, multicast and broadcast communication.

2.2.5 Allow agents to join and leave multicast group.

2.2.6 Provide message encryption and decryption techniques for secured communication.

2.2.7 Allow agents to choose to encrypt or not to encrypt message.

2.3 User Characteristics

Users who want to implement multi-agent systems based on this framework are expected to have general knowledge of Java programming, object-oriented programming and Multi-Agent Systems Engineering Methodology.

2.4 Constraints

2.4.1 Reliable message delivery – multicast/broadcast packets are delivered with best effort. Thus, a packet may be delivered to all specified agents or none.

2.4.2 Security – we provide some basic mechanisms for security such as message encryption. However, there is no guarantee that the others cannot decrypt the encrypted messages.

2.4.3 Multicast Protocol – in order to send multicast message, network environment such as router, network card and operating systems must support multicast protocol.

2.4.4 Broadcast Message – in many network, only system administrator is allowed to send broadcast message.

2.5 Assumptions and Dependencies

2.5.1 We assume that each agent knows the address of destinating agents in order to send unicast message.

2.5.2 We assume that each agent has enough knowledge to decide the best way to communicate with the other agents.

2.5.3 In the case of using secured multicast communication, we assume that there is an agent whom each agent can request for the same encryption and decryption key. This agent should maintain a list of agents who are allowed to get the keys.

2.5.4 We assume that each agent knows the multicast address in order to send multicast message.

3 Specific Requirements

This section provides all of the project requirements in detail.

3.1 Use cases

Use Case 1: Notify join/leave multicast group

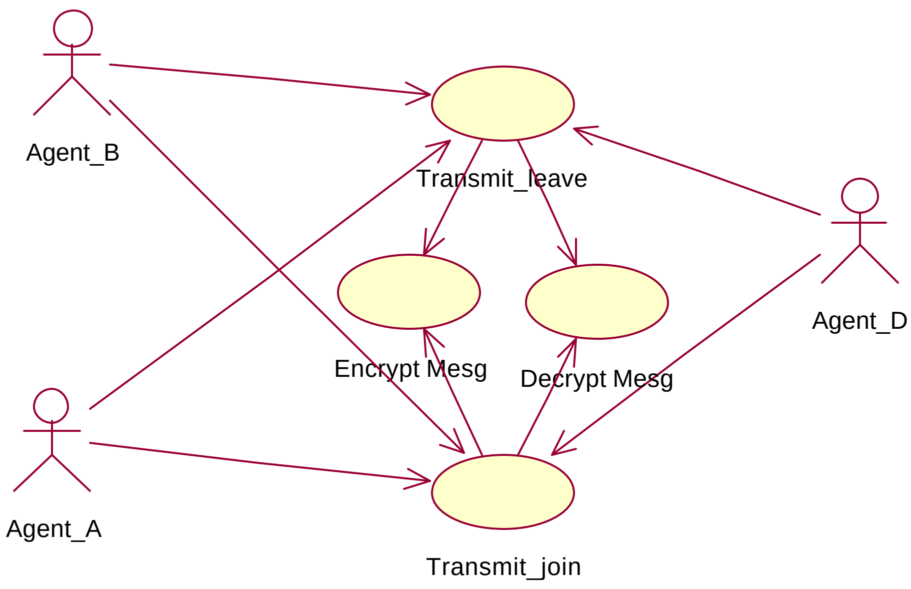


Figure 3. leave/join

1. Message is encrypted or Message is not encrypted.
2. An agent sends notify to join/leave multicast group.
3. Message is decrypted only if Message is encrypted.

For example, Agent_B and Agent_D belong to the same group, and then Agent_B wants to leave the group and Agent_A wants to join the group. In this situation, when reorganization occurs, Agent_A who is previously not part of the group may send notify message to join the group, and Agent_B who is previously part of the group, may send notify message to leave the group. For instance, Agent_B suffers a failure in one of its capabilities and does not want to receive any further message from the group. Agent_A who may have capability to substitute Agent_B is needed to be part of the group. This involves sending notify of join and leave the group.

Use Case 2: Send/Receive Unicast

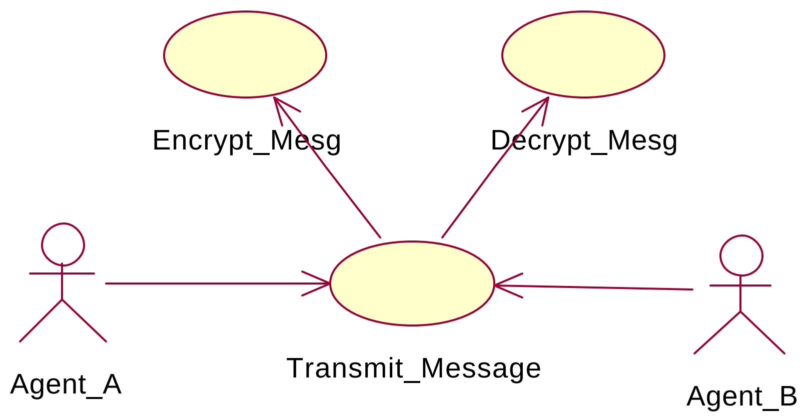


Figure 4. unicast

1. Message is encrypted or Message is not encrypted.
2. An agent sends unicast message to another agent.
3. Another agent receives message.
4. Message is decrypted only if Message is encrypted.

In this situation, Agent_A wants to communicate with Agent_B. This direct communication can happen between any two agents within organization.

<u>Use Case 3</u>: Send/Receive Multicast

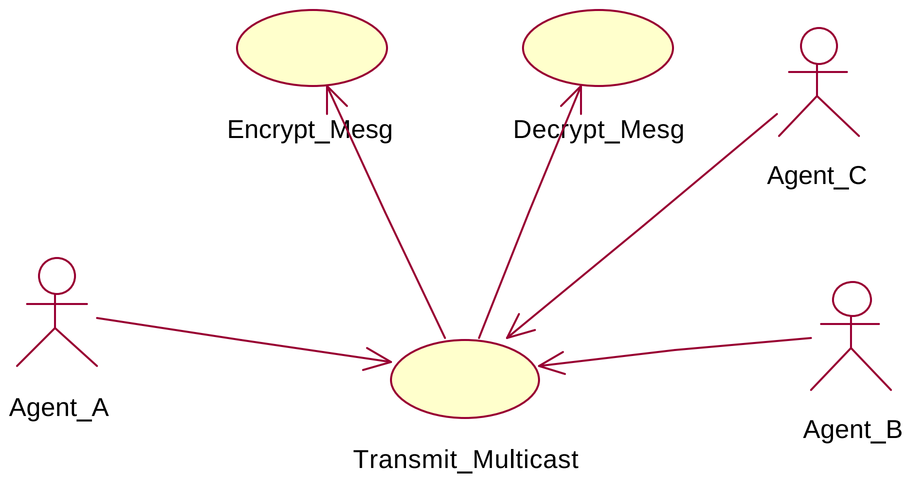


Figure 5. multicast

1. Message is encrypted or Message is not encrypted.
2. An agent sends multicast message to the group (multicast address).
3. Other agents in the group receive message.
4. Message is decrypted only if Message is encrypted

In this situation, Agent_A wants to send a message to everyone within the group, assuming that Agent_A, Agent_B and Agent_C subscribe to the same multicast address. This involves the multicast communication since other agents who do not belong to the group cannot receive this message. For instance, an agent may want to inform everyone in the group when the assigned tasks are completed. This is more effective than in unicast communication since only one copy is sent. Also, using bidding/market-based protocols fit well with this kind of communication. An agent may request a bid from other agents for doing some tasks.

Use Case 4: Send/Receive Broadcast

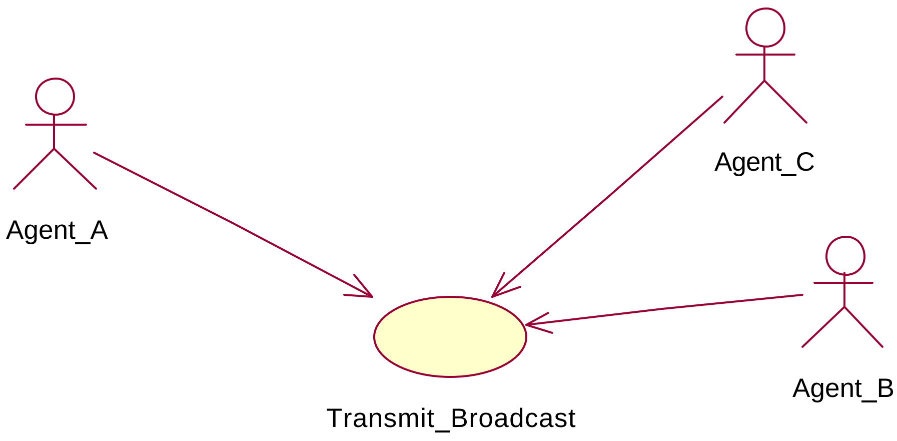


Figure 6. broadcast

1. An agent sends message to everyone in the same local network
2. Other agent in the same local network receive message

In this situation, Agent_A wants to send a message to everyone within the same local network that agent A belongs to. This involves the broadcast communication since any agent in the same local network as Agent_A can receive this message. For instance, when a new agent who does not previously exist in that local network wants to announce the existence to other agents.

3.2 Specific Requirement

3.2.1 Unicast Communication

3.2.1.1 *agentMom shall support the ability to send unicast message.
3.2.1.2 *agentMom shall support the ability to receive unicast message.
3.2.1.3 Unicast message shall only be received by the specified address.
3.2.1.4 Unicast message shall arrive at the specified address and in order.

3.2.2 Multicast Communication

3.2.2.1 *agentMom shall support the ability to send multicast message.
3.2.2.2 *agentMom shall support the ability to receive multicast message.
3.2.2.3 *agentMom shall support the ability to send request to join multicast group.

3.2.2.4 *agentMom shall support the ability to send request to leave multicast group.

3.2.2.5 agentMom shall not allow receiving multicast message from a group before joining that multicast group.

3.2.2.6 agentMom shall not allow receiving multicast message from a group after leaving that multicast group.

3.2.2.7 agentMom shall support the ability to set time-to-live for multicast message.

3.2.2.8 agentMom shall support the ability to set multicast address and port for sending and receiving multicast message.

3.2.2.9 agentMom shall support the ability to receive multicast message from multiple groups.

3.2.3 Broadcast Communication

3.2.3.1 *agentMom shall support the ability to sent broadcast message.

3.2.3.2 *agentMom shall support the ability to receive broadcast message.

3.3.3.3 *Broadcast message shall be sent to all possible hosts under the same local network.

3.2.4 Security

3.2.4.1 *agentMom shall support the ability to encrypt unicast message.

3.2.4.2 *agentMom shall support the ability to decrypt unicast message.

3.2.4.3 agentMom shall allow an agent to decide whether or not to encrypt a message.

3.2.4.4 agentMom shall automatically decrypt encrypted message.

3.2.4.5 agentMom shall support the ability to encrypt multicast message.

3.2.4.6 agentMom shall support the ability to decrypt multicast message.

3.2.5 Architecture

3.2.5.1 *agentMom with shall support the use of the architecture that agent directly controls the conversations.

3.2.5.2 *agentMom shall support the use of the architecture that agent's components control the conversations.

3.2.6 Compatibility

3.2.6.1 The new built agentMom shall be compatible with the agentMom 1.2.

Note: The " * " indicates Driving Requirements that need to be demonstrated by the end of phase II.

CHAPTER 3 – PROJECT PLAN

1 Introduction

This section provides an overview of project plan

1.1 Purpose

The purpose of this document is to provide cost estimation and architecture elaboration plan for the project "Applying Broadcasting/Multicasting/Secured Communication to agentMom in Multi-Agent Systems". This document is intended to be viewed only by project advisor and committee members.

1.2 Scope

This document covers project plan for the project "Applying Broadcasting/Multicasting/Secured Communication to agentMom in Multi-Agent Systems", including time frame, cost estimation and architecture elaboration plan. Time frame provides the phases, iterations and milestones that will comprise the project. Cost estimation provides a detailed estimate on the size, cost and effort required for the project. Architecture elaboration plan provides details of activities and actions that must be accomplished prior to the Architecture presentation.

2 Time Frame

Deliverable	Estimated Date
Phase I: Objectives	March - April
Project Overview 1.0	March 24 – March 30
Software Requirements Specification	March 24 – March 30
Project Plan	March 31– April 06
Software Quality Assurance Plan	March 31– April 06
Prototype I	March 31– April 06
MSE homepage	April 07 – April 13
First presentation	April 14 – April 25
Phase II: Architecture	April - November
Update Documents	April 26 – April 30
Formal Requirement Specification 0.1	May 19 – May 25
Architecture Design 0.1	May 26 – May 31
Test Plan 0.1	June 01 – June 07
Formal Technical Inspection 0.1	June 08 – June 14
Executable Architecture Prototype 0.1	June 08 – June 14
Formal Requirement Specification 1.0	September 01 – September 07
Architecture Design 1.0	September 08 – September 14
Test Plan 1.0	September 15 – September 21
Formal Technical Inspection 1.0	September 22 – September 28
Executable Architecture Prototype 1.0	October 01 – October 15
Second Presentation	November 17 – November 21
Phase III: Implementation	November – February

Update Documents	November 22 – November 30
Component Design	December 01 – December 07
Final Product	December 08 – December 31
Javadoc	December 08 – December 31
Assessment Evaluation	January 01 – January 18
User Manual	January 19 – January 25
Project Evaluation	January 19 – January 25
References	January 26 – January 31
Formal Technical Inspection Letters	January 26 – January 31
Final Presentation	February 24 – February 28

For a graphical representation of the proposed project plan, consult the included *Gantt chart*.

3 Cost Estimation

3.1 Function Point

First, the different types of program features must be identified. These include the following:

a) Internal Logical Files – A file is a major logical group of user data or control information, which could be in a large database or a separate file. This is zero for the agentMom.
b) External Interfaces Files – Normally considered files passed or shared between systems. This is zero for the agentMom.
c) External Inputs – Unique user data or user control input that enters the external boundary of the system and adds or modifies a logical internal file. The inputs are unicast message, multicast message, broadcast message, secured multicast message and secured multicast message. Thus, there are five external inputs.
d) External Outputs – Each user data or control output type leaving the external boundary of the system is counted. The outputs are unicast message, multicast message, broadcast message, secured multicast message and secured multicast message. Thus, there are five external outputs.
e) External Inquiry – Each input-output combination is counted, when input causes an immediate output. This is zero for the agentMom.

Total Unadjusted Function Points

Type	**Complexity**			**Function Points**
	Low	Average	High	
Internal Logical Files				0
External Interfaces Files				0
External Inputs	5x3			15
External Outputs	5x4			20
External Inquiry				0
Total				**35**

3.2 COCOMO I

Estimation is based upon the Organic mode in the Constructive Cost Model (COCOMO) cost model developed by Barry Boehm. Since this project is fairly simple and very flexible, we can assume using the Organic mode. Also, original COCOMO model is used since COCOMO II is more appropriate with large team development project with large number of developers.

The COCOMO estimating equations follow this simple form:

$\text{Effort} = C1*\text{EAF}*(\text{Size})^{P1}$

$\text{Time} = C2*(\text{Effort})^{P2}$

where:

Effort = number of person-months

C1 = constant scaling coefficient for effort

C2 = a constant scaling coefficient for schedule

P1 = an exponent that characterizes the economics of scale inherent in the process used to produce the end product

P2 = an exponent that characterizes the inherent inertia and parallelism in managing a software development effort

EAF = an effort adjustment factor that characterizes the domain, personnel, environment, and tools used to produce the artifacts of the process

Size = size of the end product (in human-generated source code), measured by the number of delivered source instructions

Time = total number of months

As in Organic mode,

C1 = 3.2

C2 = 2.5

P1 = 1.05

P2 = 0.38

Since EAF value is difficult to determine, the EAF effect is not considered at this point. For instance, the EAF of Programmer capability is range from 1.42 – 0.70. It is hard to specify the value when there is no database that refers to the number of years in programming experience for each value. The estimation of size is defined as human-generated source line of code, excluding comments. The SLOC per Function Point for java is 46, so SLOC is 35 x 46 = 1610.

Therefore, the total effort and time are:

$\text{Effort} = 3.2*(1.610)^{1.05} = 5.3$ person-months (4.9 previously)

$\text{Time} = 2.5*(5.3)^{0.38} = 4.7$ months (4.6 previously)

Productivity = 1610/4.7 = 343 LOC-month (330 previously)

Staff = 5.3/4.7 = 1.13 person (1.07 previously)

As the number shown above, this project requires one person to complete in 4.7 months with 343 SLOC per month, or one person works 4.7*152 = 715 hours

Note: As described by Boehm, there are 152 working hours in a month. Therefore, time to complete this project may vary depend on number of working hours in a month.

4 Architecture Elaboration Plan

4.1 Vision Document (revision)

After the first presentation, suggestions shall be provided by the committee and these shall be used to revise the Vision document. The revised document shall be approved by the major professor. Vision document consists of Project Overview document and Software Requirements Specification document.

4.2 Project Plan (revision)

After the first presentation, suggestions shall be provided by the committee and these shall be used to revise the Project Plan document. The cost estimation shall be updated as appropriate. Also, the implementation plan shall be included. The revised document shall be approved by the major professor.

4.3 Formal Requirement Specification

The class diagram from architecture design shall formally be specified using UML/OCL methodology. The tool USE, a UML-based Specification Environment, shall be used. For more information about USE, please refer to "www.db.informatik.uni-bremen.de/projects/USE/ "

4.4 Architecture Design

The completed class diagrams and use cases diagrams shall be produced and well document. This design shall be implemented based upon the class diagram and use cases presented in Vision document. Also, this architecture design shall be undergo formal technical inspection.

4.5 Test Plan

Test plan shall be produced to show that all requirements specified in vision document are satisfied. Unit testing, integration testing, and system testing shall be conducted. Unit testing shall be class-based. Two or more related classes shall be used for integration testing. Finally, the whole system shall be used for system testing. Reliability shall involve in the testing to measure successful rate of message delivery.

4.6 Formal Technical Inspection

The architecture design shall be undergo formal technical inspection. The group of inspector consists Madhukar Kumar of and Acharaporn Pattaravanichanon. The developer shall develop a formal checklist and provide it to inspectors. The inspectors shall provide a formal report on the result of their inspection during Phase III.

4.7 Executable Architecture Prototype

All driving requirements identified in vision document shall be implemented. This prototype shall be implemented based on the first prototype from phase I. Specifically, The executable architecture prototype shall be integrated into agentMom, and it shall have all driving requirements capabilities.

CHAPTER 4 – SOFTWARE QUALITY ASSURANCE PLAN

1 Purpose

The purpose of this document is to specify how the software quality assurance plan will be handled in the software development life-cycle of the project "Applying Broadcasting/Multicasting/Secured Communication to agentMom in Multi-Agent Systems". The intended use of this to software project is to enhance the pre-exist agentMom (agentMom 1.2) to be capable of providing multicasting conversation and basic message encryption for security purpose. This document is based on the IEEE standard for Software Quality Assurance Plan, IEEE Std 730.1-1995. This document is intended to be used in partial fulfillment of the requirements for the Master of Software Engineering Project's Portfolio. Furthermore, this document will be reviewed and evaluated by the major professor and the supervisory committee.

2. Management

2.1 Organization

Supervisory Committee consisted of:

Dr. Scott A. DeLoach
Dr. David Gustafson
Dr. William Hankley

Major Professor:

Dr. Scott A. DeLoach

Developer:

Chairoj Mekprasertvit

Formal Technical Inspector consisted of

Madhukar Kumar
Acharaporn Pattaravanichanon

2.2 Responsibilities

2.2.1 Supervisory Committee

Primary responsibilities include reviewing each milestone deliverable at the requirements, architecture, and implementation phases. After reviewing, each committee member should provide feedback and suggestions to the software developer.

2.2.2 Major Professor

In addition to the responsibilities as one of the committee member, the major professor will supervise and evaluate all artifacts submitted by developer. Reviews and walkthroughs of related materials will be conducted on weekly basis.

2.2.3 Software Developer

Since the project is being developed individually, developer is responsible for ensuring quality of the project. The software developer is also responsible for producing the required artifacts for MSE projects.

2.2.4 Formal Technical Inspectors

The major responsibility of inspectors is to provide a formal report on their inspection result from architecture design artifact produced by developer.

Furthermore, the following tasks will be conducted in order to ensure quality assurance:

1. Driving Requirements: The developer should ensure that the software requirement specification in the vision document clearly states the functionality of software and unambiguously declares the requirements that must be satisfied. In addition, descriptions of the scope should clearly outline what the software will allow and not allow.
2. Design: The developer and major professor will conduct reviews and analyses of the construction of the software. Strengths and weaknesses of various design techniques will be discussed and scrutinized.
3. Implementation: Informal code reviews will be conducted by the developer on a regular basis to ensure consistency with the design and the detection of any error. Also, JavaDoc will be produced for purpose of maintainability and future work.
4. Testing: Developer will conduct tests as presented in the Software Test Plan to ensure the requirement satisfaction and reliability of the software.

3. Documentation

The following documentation will be generated and updated throughout the duration of software life cycles:

Phase I:

1.) Vision Document - provides detailed description of the entire project, goals of the software, constraints and requirements for the software to satisfy.
2.) Project Plan - illustrates the major milestones and provides a rough timeline for the project and estimation on the size and effort of the project.
3.) Software Quality Assurance Plan – provides plan for software quality assurance

Phase II:

1.) Formal Requirement Specification – UML/OCL methodology will be used to produce this document.
2.) Test Plan - provides description of test cases during testing
3.) Architecture Design – Object Model and Use Cases will be produced.
4.) Formal Technical Inspection - two MSE students will participate in formal technical inspection, and developer will also provide an inspection checklist.

Phase III:

1.) User Manual - instructions on how to use software
2.) Final source code - actual implemented documented source code
3.) Assessment Evaluation - assessment of reliability and performance of software
4.) Project Evaluation - review of the entire project

4. Standards, Practices, Conventions and metrics

4.1 Standards

- Documents – MSE portfolio requirements, CIS Dept., Kansas State University
- Coding – Java 1.4.0 (commenting will follow JavaDoc standards)
- Testing – IEEE Standard for Software Test Documentation

4.2 Metrics

- SLOC – source lines of code will be primarily used for measuring the size of the software
- COCOMO I – cost estimation will be calculated based on COCOMO I model.

5. Reviews and audits

Two formal MSE students will perform a formal technical inspection on the architecture design document and provide a formal report. Also, each committee member will review the produced documentation and make comments and suggestions during each presentation. Each milestone must be approved by each committee member to proceed to the next milestone. Each milestone is indicated by the presentation of each phase. There are three presentation described as follow:

Presentation I at the end of phase I includes project overview, software requirements, project plan, SQA plan and prototype demonstration.

Presentation II at the end of phase II includes formal requirement specification, architecture design, test plan and architecture prototype demonstration.

Presentation III at the end of phase III includes component design, assessment evaluation, project evaluation, result from formal technical inspection and completed software demonstration.

6. Problem reporting

If any problems are encountered throughout the duration of the project, the software developer can report and discuss the problems with the major professor. If any conflicts or problems are discovered by one of the committee members during a presentation, the developer will then correct the errors.

7. Tools, Techniques and Methodologies

For determining whether the software requirements were satisfied, a software test plan will be written during phase II. This plan will provide an overview of the methodologies, timetables, and resources for testing the software. Testing will commence in three primary ways:

7.1 Unit Testing

Individual classes will be tested to ensure reliability and functionality within a unit-level. Furthermore, testing module will be created before the tested code to ensure that the code is testable. Junit 3.8 will be the tool to perform testing. Unit testing will be performed before alpha and beta release.

7.2 Integration Testing

Several classes will be tested together to ensure sufficient execution and compliance with the requirements after integration. Integration testing will be performed before beta release.

7.3 System Testing

The whole system shall be used for system testing to ensure all requirements is satisfied, and reliability will be included in the testing to measure successful rate of message delivery. System testing will be performed before beta release.

The following tools are used in creating, testing and debugging software

- Java 1.4.2 will be the language used for coding the software.
- USE 2.0 will be used for modeling the formal specifications, using UML/OCL methodology
- Rational Rose will primarily be used for producing object models.
- Eclipse IDE 2.1 will be used for coding the software package.
- This software package will be tested under Microsoft Windows XP/2000, Linux Debian and Solaris 9.

8. Media control

All the required documentation generated throughout the course of the project is available at the software developer's personal website (http://www.cis.ksu.edu/~cme6556).

Upon project completion, a CD containing the entire project document, prototype, and final product is created.

9. Document and Software Version Control

Each document version or software version is incremented by 0.1 after it is approved by major professor. Also, version is move to the next digit after it is approved by all committee members. For example, version 1.2 will be changed to version 2.0 after all committee approval or version 1.2 to 1.3 after major professor approval.
Alpha version is released after the software is passed all unit tests.
Beta version is released after the software is passed all unit tests, integration tests and system tests. Final version is released after the software is approved by major professor.

10. Training

CIS 740 Software Engineering
CIS 748 Software Management
CIS 771 Software Specifications
CIS 890 Agent-Oriented Software Engineering

CHAPTER 5 – ARCHITECTURE DESIGN

Architecture Design

1 Introduction

The purpose of this document is to provide the architecture design including class diagram, description of class diagram, sequence diagram and description of class diagram for the project “Applying Broadcasting/Multicasting/Secured Communication to agentMom in Multi-Agent Systems”. The architecture design of this project is defined by driving requirement stated in Software Requirements Specification version 1.0. This document is intended to be viewed only by project advisor and committee members.

2. Class Diagram

2.1 agentMom 1.2

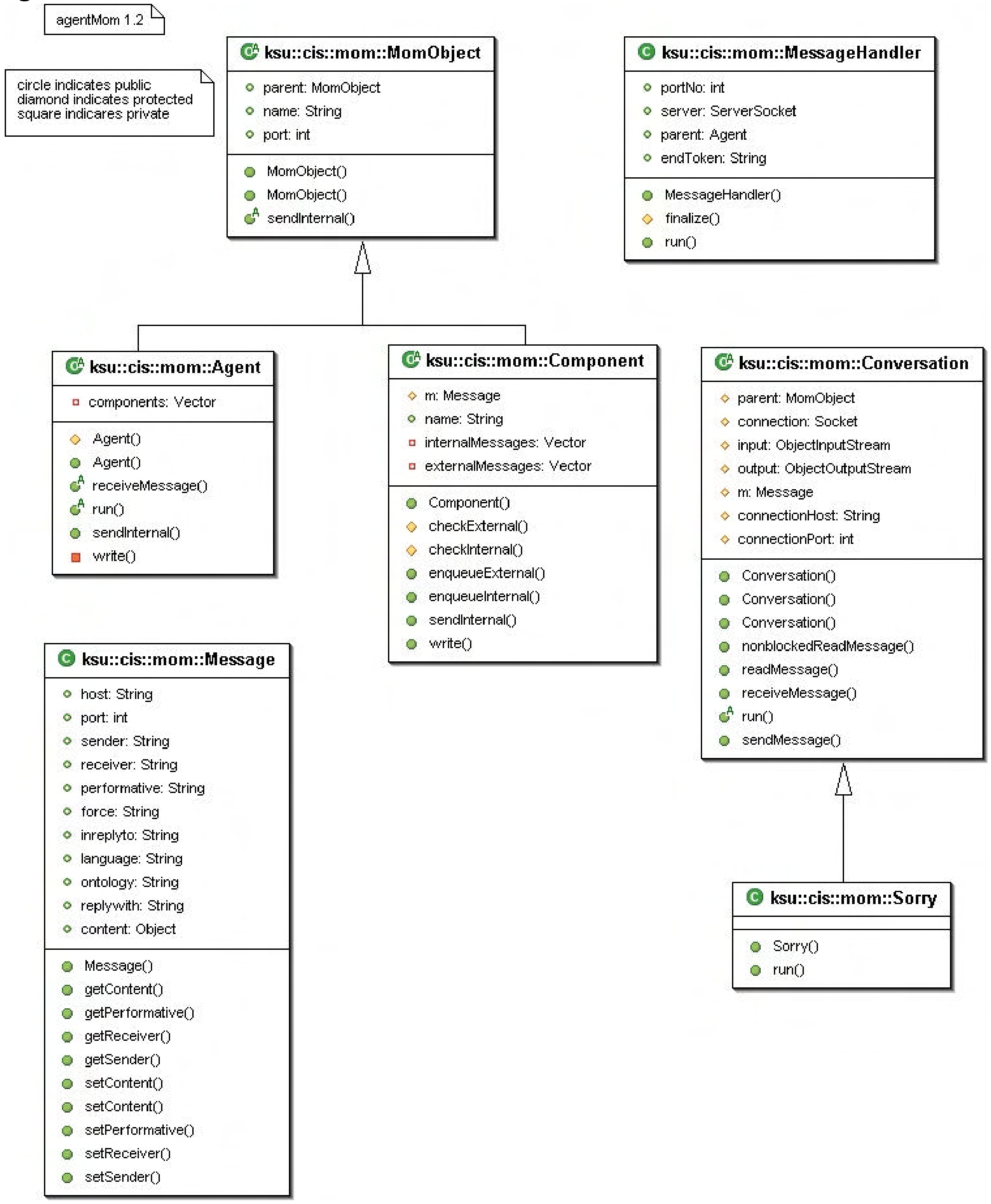


Figure 7 Class Diagram for agentMom1.2

Figure 1 shows the class diagram of agentMom version 1.2. This is the version that the project is based on. It consists of seven classes with four abstract classes, MomObject, Agent, Conversation and Component, and three concrete classes, Message, Sorry and MessageHandler.

2.2 New agentMom

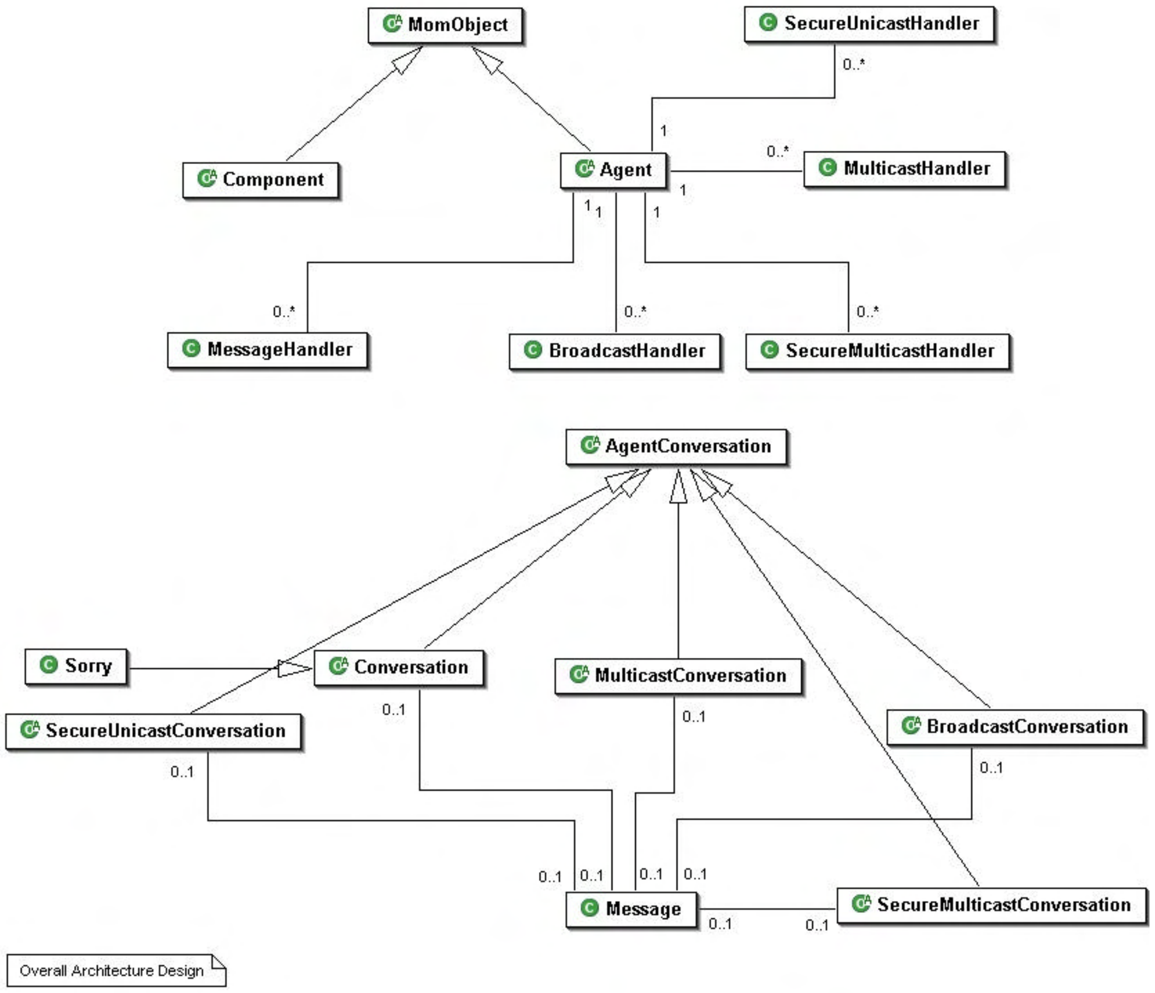
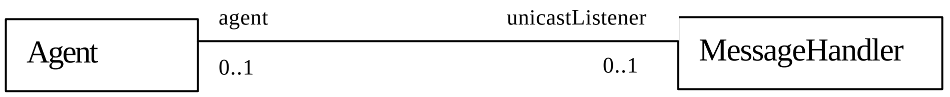


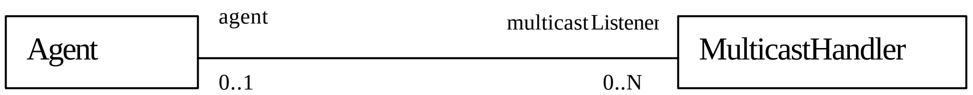
Figure 8 Overall Architecture Design

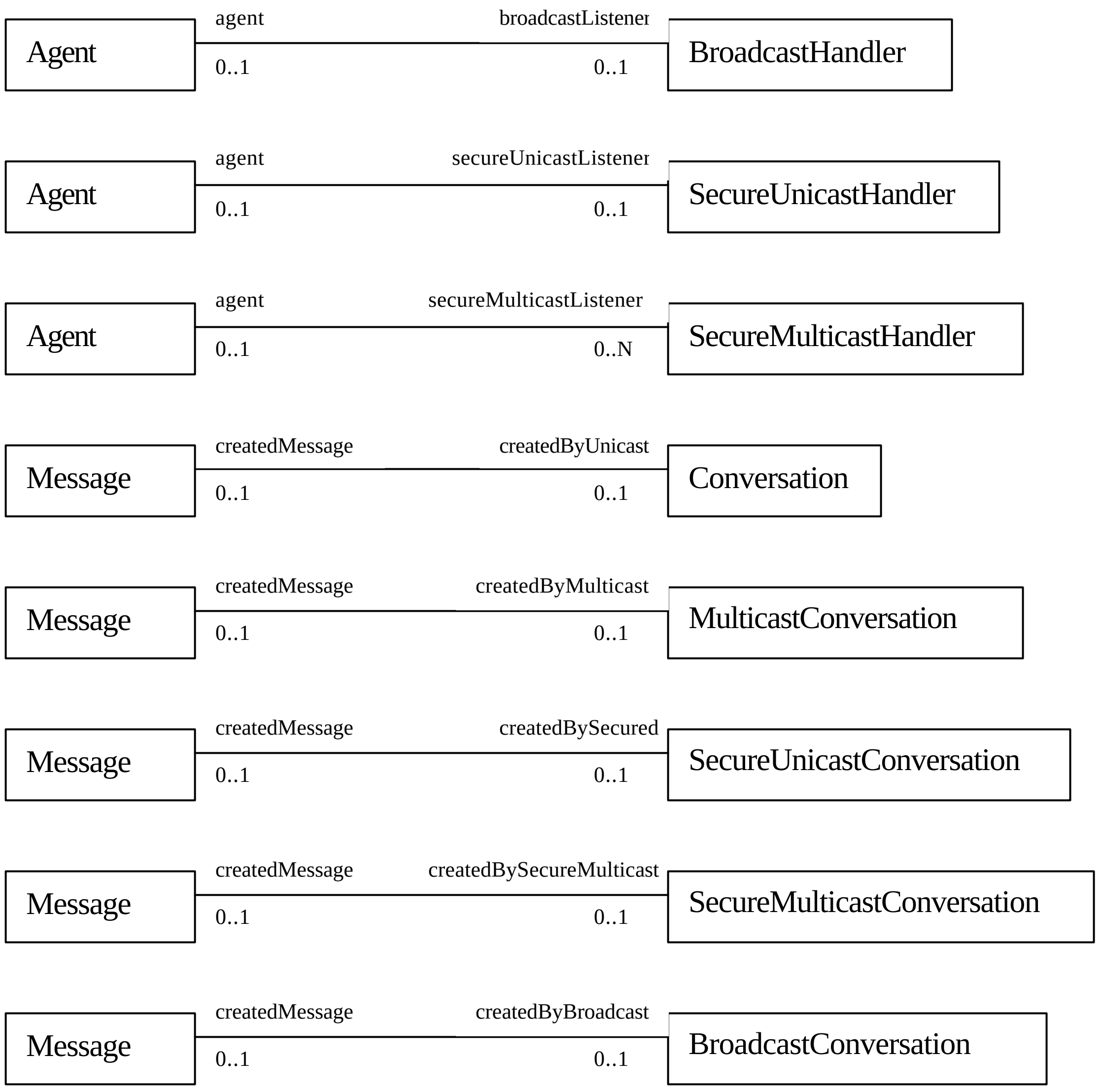
Figure 2 shows the overall design of new agentMom architecture with inheritance and association relationship. It consists of 16 classes with nine abstract classes, MomObject, Agent, Conversation, SecureUnicastConversation, MulticastConversation, SecureMulticastConversation, BroadcastConversation, AgentConversation and Component, and seven concrete classes, Message, Sorry, MessageHandler, MulticastHandler, SecureUnicastHandler, SecureMulticastHandler and BroadcastHandler.

2.3 Associations

From Figure 2, associations are shown with roles and multiplicities below:







2.4 New Classes

Figure 3, new classes added to agentMom are shown with attributes and method below:

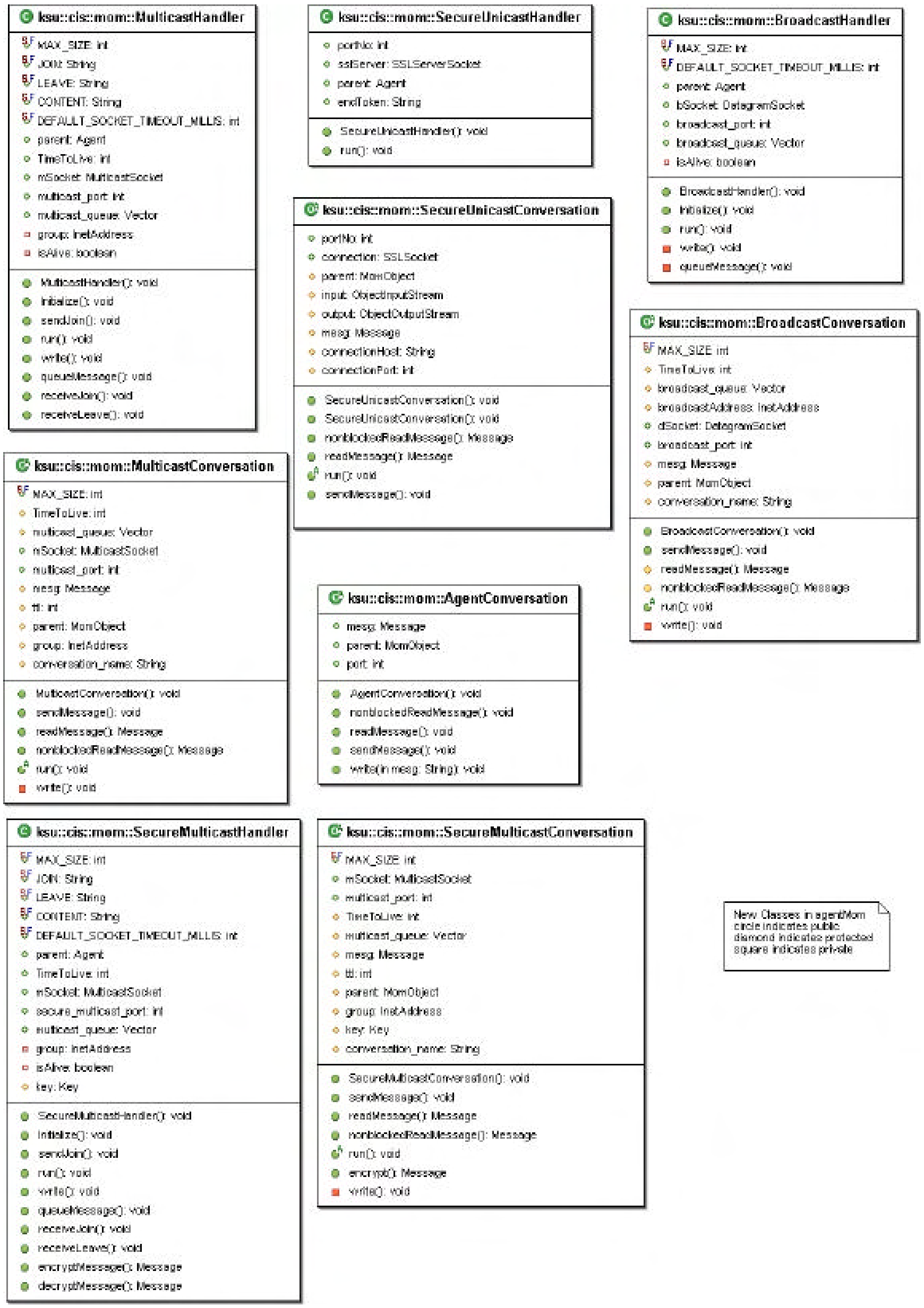


Figure 9 Nine New Classes in agentMom

Figure 3 shows the details of new nine classes added to agentMom 1.2. There are five new abstract classes, including AgentConversation, MulticastConversation, SecureUnicastConversation and BroadcastConversation and SecureMulticastConversation. Furthermore, there are four new concrete classes, including MulticastHandler, SecureUnicastHandler, BroadcastHandler, SecureMulticastHandler.

2.5 Class Diagram Description

MomObject: Abstract class that both Agent and Component inherit from. It has two required parameters that must be set for each agent to use agentMom package, name of the agent and port used for unicast conversation.

Agent: This abstract class defines the minimum requirements for an agent to use agentMom package.

MessageHandler: This concrete class is used for listening for initial message when other agents want to start a unicast conversation.

MulticastHandler: This concrete class is used for listening for initial message when other agents want to start a multicast conversation. It also performs joining multicast group to receive multicast message from the group. Multicast group is defined by Internet address class D. Furthermore, it performs actual receiving multicast messages, and then adds messages to the queue that will be fetched by MulticastConversation class. MulticastConversation then indirectly receives message.

BroadcastHandler: This concrete class is used for listening for initial message when other agents want to start a broadcast conversation. It also performs actual receiving broadcast messages, and then adds messages to the queue that will be fetched by BroadcastConversation class. MulticastConversation then indirectly receives message.

SecureUnicastHandler: This concrete class is used for listening for initial message when other agents want to start a secured unicast conversation. It uses the Secure Socket Layers (SSL) for secured communication.

SecureMulticastHandler: This concrete class is used for listening for initial message when other agents want to start a secured multicast conversation. It can also perform message encryption and decryption. This class works in the same way as MulticastHandler. The different is that it decrypts the received message before it adds message to the queue.

Component: This abstract class is used for internal communication of components within an agent.

AgentConversation: Abstract class that unicast, multicast, broadcast and secured conversation inherit from. Basically, all conversation classes are generalization of this class.

Conversation: This abstract class provides unicast communication among agents. It carries out the message passing between agents. Unicast conversation is controlled by the implementation class of this class.

Message: This class defines the field used in the message for agent communication.

MulticastConversation: This abstract class provides multicast communication. It is used for sending and receiving multicast message. This class indirectly receives message from the message queue controlled by MulticastHandler. Multicast conversation is controlled by the implementation class of this class.

BroadcastConversation: This abstract class provides broadcast communication. It is used for sending and receiving broadcast message. The message sent by this class is broadcasting to every host under the same local area network as the sender. This class indirectly receives message from the message queue controlled by BroadcastHandler. Broadcast conversation is controlled by the implementation class of this class.

Sorry: This class is a concrete class of conversation class. It is a default conversation class that sends message when an agent receives unknown conversation.

SecureUnicastConversation: This abstract class provides secured unicast communication among agents. It carries out the message passing between agents using SSL communication. SecureUnicastConversation is controlled by the implementation class of this class.

SecureMulticastConversation: This abstract class provides secured multicast communication. It is used for sending and receiving secured multicast message. This class indirectly receives message from the message queue controlled by SecureMulticastHandler. Secured multicast conversation is controlled by the implementation class of this class. This class works in the same way as MulticastConversation. The different is that it encrypts the message before sending out.

3. Sequence Diagram

This section shows the sequence diagrams of the basic scenarios of agent communication including unicast, multicast and broadcast conversation.

3.1 Unicast conversation

In Figure 11 shows how agent may exchange message using unicast conversation. On one side, the agent A2 creates the MessageHandler H1 that creates the ServerSocket class SS2 and then waits for a connection from other agents. When the agent A1 want to communicate with A2, A1 starts the unicast conversation with A2 by creating the Conversation object C1 that controls the unicast conversation between agents. Conversation class C1 then creates the Socket object for sending and receiving unicast message. First, Conversation C1 request for a connection with the ServerSocket SS2. The ServerSocket SS2 simply accepts it and then creates the Socket S2 that is connected to S1. After both Socket S1 and S2 are connected, the MessageHandler H1 calls the receiveMessage method in A2 with the created socket. Then, Agent A2 creates the

Conversation C2. At this point, conversation is controlled by two Conversation classes C1 and C2. Messages are passing back and forth until the conversation is completed as defined in the each Conversation class. Finally, each conversation closes the socket at each side.

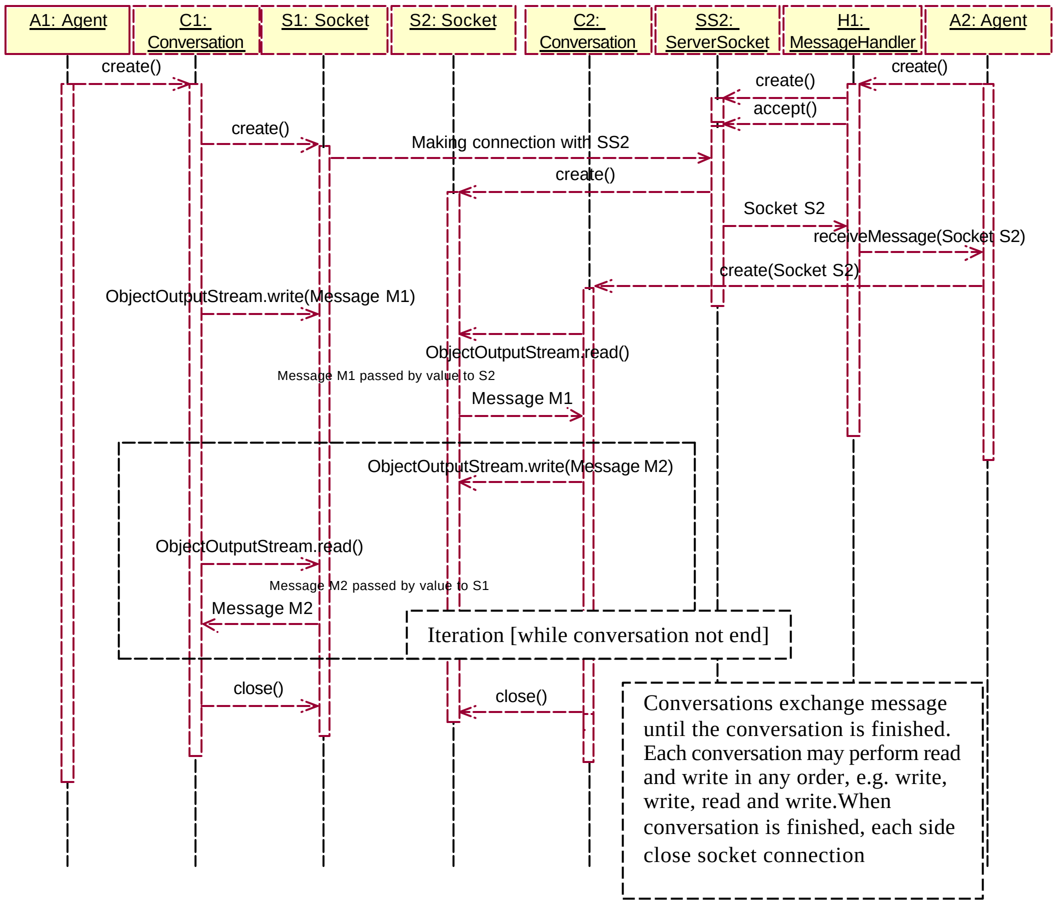

A1: Agent
C1: Conversation
S1: Socket
S2: Socket
C2: Conversation
SS2: ServerSocket
H1: MessageHandler
A2: Agent
create()
create()
accept()
create()
create()
Making connection with SS2
create()
Socket S2
receiveMessage(Socket S2)
create(Socket S2)
ObjectOutputStream.write(Message M1)
ObjectOutputStream.read()
Message M1 passed by value to S2
Message M1
ObjectOutputStream.write(Message M2)
ObjectOutputStream.read()
Message M2 passed by value to S1
Message M2
Iteration [while conversation not end]
close()
close()
Conversations exchange message until the conversation is finished. Each conversation may perform read and write in any order, e.g. write, write, read and write.When conversation is finished, each side close socket connection

Figure 10 Starting Unicast Conversation

3.2 Multicast conversation

Multicast conversation can be categorized into three scenarios, join group, leave group and conversation.

In Figure 12 shows how an agent may join the multicast group. To join the group, Agent A1 creates the MulticastHandler H1. The MulticastHandler H1 then creates the MulticastSocket S1 and calls joinGroup method in the MulticastSocket class to notify the router that this machine want to join the multicast group. The MulticastHandler H1 then sends a join message to all agents previously existing in the group. In this case, the MulticastHandler H2 belonging to Agent A2 receives the join message, and then calls in receiveMulticastJoin method in the Agent A2.

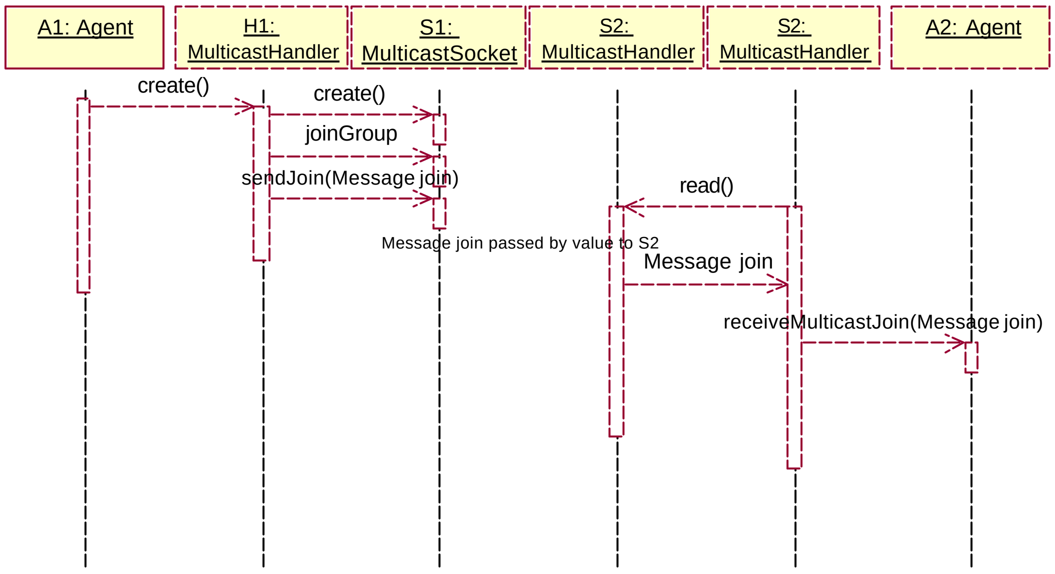


Figure 11 Join Multicast Group

In Figure 13 shows an agent may starts the multicast conversation with the group. Agent A1 creates the MulticastConversation C1. MulticastConversation C1 then send the start conversation message to the group. In this case Agent A2's MulticastHandler receive a request to start a new conversation. It calls the receiveMulticastConversation method in Agent A2. Then Agent A2 starts a new MulticastConversation corresponding to the request. At this point, conversation is controlled by the MulticastConversation class at each side of Agent. Messages are passing back and forth within the group until the conversation is completed as defined in the each MulticastConversation class.

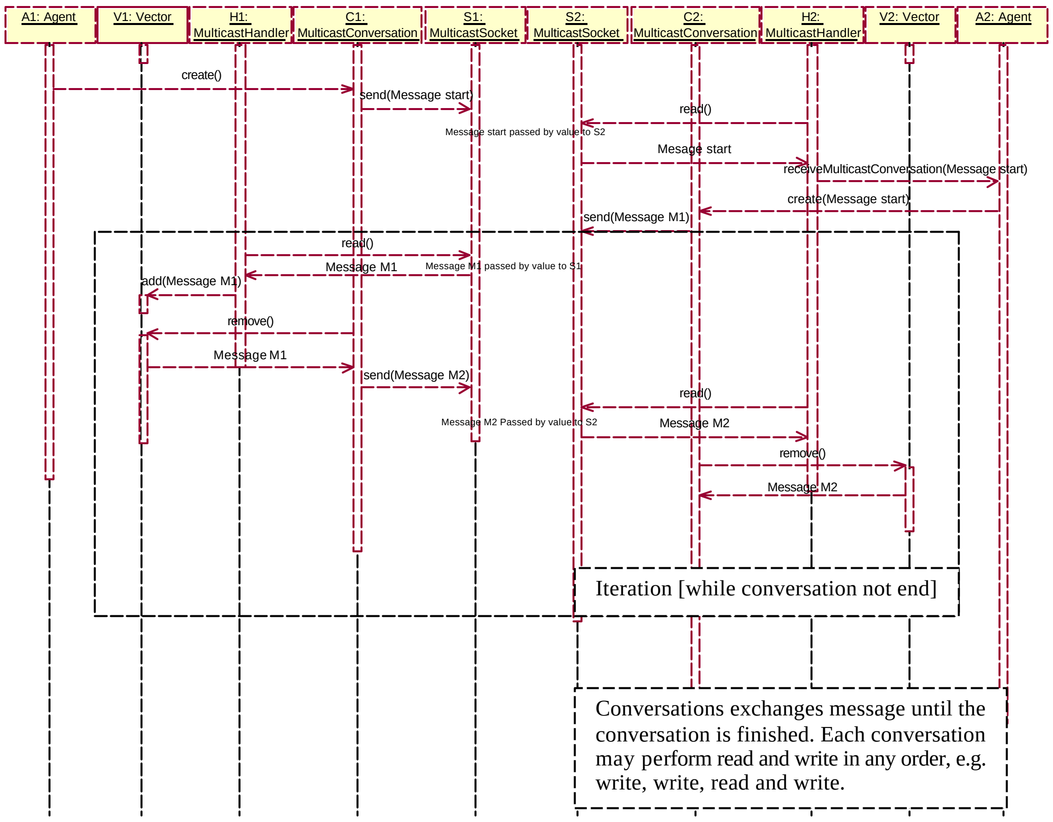

A1: Agent
V1: Vector
H1: MulticastHandler
C1: MulticastConversation
S1: MulticastSocket
S2: MulticastSocket
C2: MulticastConversation
H2: MulticastHandler
V2: Vector
A2: Agent
create()
send(Message start)
read()
Message start passed by value to S2
Mesage start
receiveMulticastConversation(Message start)
create(Message start)
send(Message M1)
read()
Message M1
Message M1 passed by value to S1
add(Message M1)
remove()
Message M1
send(Message M2)
read()
Message M2 Passed by value to S2
Message M2
remove()
Message M2
Iteration [while conversation not end]
Conversations exchanges message until the conversation is finished. Each conversation may perform read and write in any order, e.g. write, write, read and write.

Figure 12 Multicast Conversation

In Figure 14 shows how an agent may leave the multicast group. To leave the group, Agent A1 calls setLeave method in MulticastHandler H1 and passes the value true. The MulticastHandler H1 then send a leave message to the all agents in the group by calling the send method in MulticastSocket. In this case, the Agent A2's MulticaastHandler receives the leave message and then calls the receiveMulticastLeave method in the Agent A2. Finally, the MessageHandler H1 calls leaveGroup method in the MulticastSocket class to notify the router that this machine wants to leave the multicast group, and then close the multicast socket.

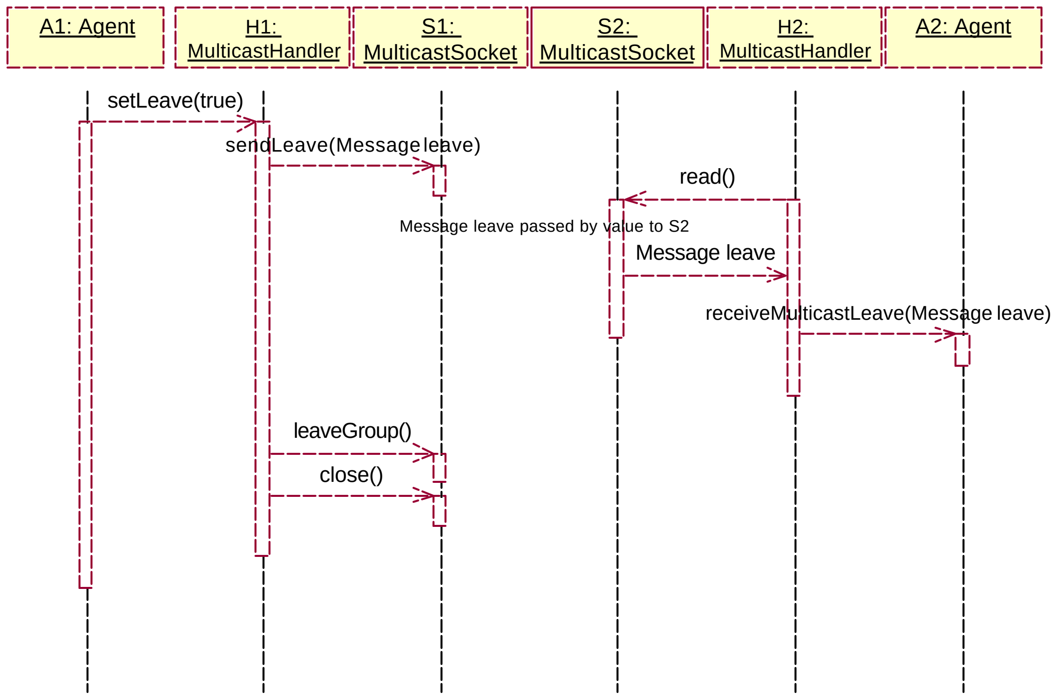


Figure 13 Leave Multicast Group

3.3 Broadcast conversation

In Figure 15 shows how an agent may start the broadcast conversation with other agents on the same local network. Agent A1 creates the BroadcastConversation C1. The BroadcastConversation C1 then send the start conversation message to all agent under the same local network. In this case Agent A2's BroadcastHandler receive a request to start a new conversation. It calls the receiveBroadcastConversation method in Agent A2. Then Agent A2 starts a new BroadcastConversation corresponding to the request. At this point, conversation is controlled by the BroadcastConversation class at each side of Agent. Messages are passing back and forth within the agents in the same local network until the conversation is completed as defined in the each BroadcastConversation class.

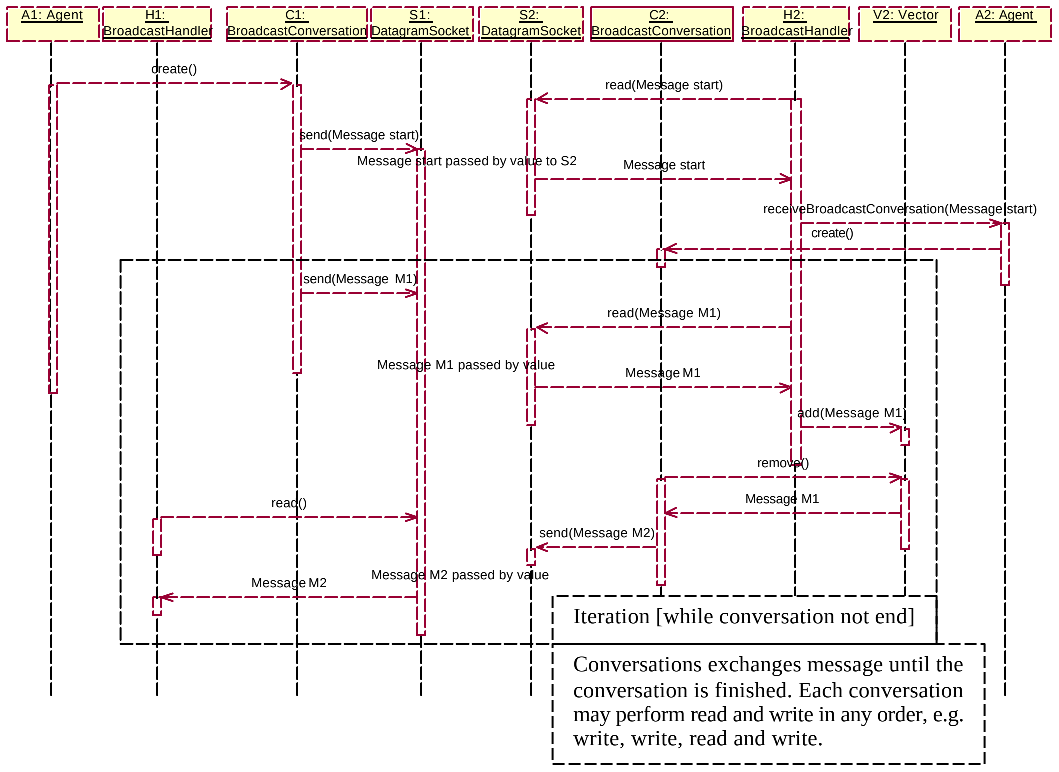

A1: Agent
H1: BroadcastHandler
C1: BroadcastConversation
S1: DatagramSocket
S2: DatagramSocket
C2: BroadcastConversation
H2: BroadcastHandler
V2: Vector
A2: Agent
create()
read(Message start)
send(Message start)
Message start passed by value to S2
Message start
receiveBroadcastConversation(Message start)
create()
send(Message M1)
read(Message M1)
Message M1 passed by value
Message M1
add(Message M1)
remove()
Message M1
read()
send(Message M2)
Message M2 passed by value
Message M2
Iteration [while conversation not end]
Conversations exchanges message until the conversation is finished. Each conversation may perform read and write in any order, e.g. write, write, read and write.

Figure 14 Broadcast Conversation

CHAPTER 6 – FORMAL REQUIREMENT SPECIFICATION

1. Introduction

1.1 Purpose

The purpose of this document is to provide the formal requirement specification of the project "Applying Broadcast/Multicast/Secured communication to agentMom in Multi-Agent Systems". This specification uses the UML/OCL methodology as specified in the UML specification version 1.5. The Object Constraint Language (OCL) is a formal language used to express constraint and specify invariant for the system being model. It provides a precise and unambiguous specification of the system.

1.2 Scope

In the specification, we specify the pre and post condition of the interest properties to ensure that these properties are hold in our system model. These properties are:

1.) Unicast conversation
- 1.1) Only the specified address receives the unicast message.
- 1.2) Sent message is the same as received message

2.) Multicast conversation
- 2.1) Only the specified group receives the multicast message for that group
- 2.2) Sent message is the same as received message

3.) Broadcast conversation
- 3.1) Only the conversations holding the same broadcast address receive the broadcast message.
- 3.2) Sent message is the same as received message

4.) Secured unicast conversation
- 4.1) Only the specified address receives the unicast message
- 4.2) Sent message is the same as received message

The properties are based on the driving requirement as stated in the Software Requirement Specification version 1.0. Furthermore, we use the UML- based Specification Environment (USE) tool to check the type and syntax to ensure correctness of the specification. Please refer to the file "agentMom_ocl.doc" in the included CD for a full specification of the model.

2 Formal Requirement Specification Descriptions

This section explains the unicast conversation and multicast conversation specification in detail. Because unicast conversation and secured unicast conversation specifications are almost identical, only the unicast conversation specification is covered. Also, it is the same as multicast conversation and broadcast conversation specifications.

2.1 Unicast conversation

The unicast conversation is named "Conversation". The attributes of this class are: m, localhost and connectionHost. The attribute m is a Message type, and it is used for storing a message sent to another agent. The attribute Localhost is a String type of Internet address of the agent. The attribute connectionHost is a String type of Internet address of the connecting agent (receiver agent). Furthermore, the class and association related to unicast conversation is shown below:

```
class Conversation
attributes
m: Message;
Localhost: String;
connectionHost: String;
connectionPort: Integer;
operations
sendMessage(m: Message)
receiveMessage(): Message
end

association Agent-Conversation between
      Agent[1] role agent
      Conversation[0..*] role unicastConversation
End

association ConstructUnicast between
      Conversation[0..1] role createdByUnicast;
      Message[0..1] role createdMessage;
end

association ReceiveUnicast between
      Conversation[0..1] role receivedByUnicast;
      Message[0..1] role receivedMessage;
end
```

Finally, the pre and post condition related to this class is described below:

2.1.1 Only the specified address receives the unicast message.

```
context Conversation::sendMessage(m: Message)
-- unicast conversation associates with the Message parameter
      pre cond_1: self.createdMessage = m
-- Message must be well defined before sending
      pre cond_2: m.isDefined
-- Only the destined address and port receive the message.
      post cond_3: Conversation.allInstances->
                              exists(c: Conversation|
                                    ((c.Localhost = self.connectionHost
                                    and
                                    c.agent.port = self.connectionPort)
                                    implies
                                    c.receivedMessage = m)
                                    and
                                    (c.receivedMessage = m
                                    implies
                                    (c.Localhost = self.connectionHost
                                    and
                                    c.agent.port =
                                    self.connectionPort)))
```

This part of specification defines pre and post condition of the operation sendMessage of the class Conversation. There are two pre-conditions and one post-condition. The pre-condition “cond_1” states that the Message object m must be created. The pre-condition “cond_2” states that the attributes of Message object m must be defined. Finally, the post-condition “cond_3” states that there exist a Conversation object that receives the Message object m, and the Internet address and port number of the receiver must be the same as the address that sender connects to. Therefore, only the specified address and port number receives the unicast message.

2.1.2 Received message is the same as sent message

```
-- Receive unicast pre-post condition
-- Received message is the same as sent message
context Conversation::receiveMessage(): Message
-- New received message is created
      post cond_1: self.receivedMessage.oclIsNew = true
-- New created received message is the same as sent Message
      post cond_2: Conversation.allInstances->
                              exists(c: Conversation|
                                    ((c.connectionHost = self.Localhost
                                    and
                                    c.connectionPort = self.agent.port)
                                    implies
                                    c.createdMessage =
                                    self.receivedMessage)
                                    and
                                    (c.createdMessage =
                                    self.receivedMessage
                                    implies
                                    (c.connectionHost = self.Localhost
                                    and
                                    c.connectionPort =
                                    self.agent.port)))
-- Result of receiveMessage()
      post cond_3: result = self.receivedMessage
```

This part of specification defines pre and post condition of the operation recevieMessage of the class Conversation. There are three post-conditions. The post-condition “cond_1” states that the received Message object is created during the operation receiveMessage (Message is received from another agent). The post-condition “cond_2” states that the new received Message object must be the same as the Message object that is sent by the sender. The post-condition “cond_3” specifies the return result of this operation. In this case, it is the received message is returned. Therefore, the sent message is the same as received message.

2.2 Multicast conversation

The multicast conversation is named “MulticastConversation”. The attributes of this class are: multicastPort, m, join and multicastAddress. The attribute m is a Message type, and it is used for storing a message sent to another agent. The attribute multicastAddress is a String type of multicast address of the group that agent subscribes to. The attribute multicastPort is a Integer type of the port that multicast listening. Furthermore, the association related to multicast conversation is shown below:

```
class MulticastConversation
attributes
multicastPort: Integer;
m: Message;
join: Boolean;
multicastAddress: String;
operations
sendMessage(m: Message)
sendJoin()
sendLeave()
receiveMessage(): Message
end

association Agent-MulticastConversation between
      Agent[1] role agent
      MulticastConversation[0..*] role multicastConversation
end

association ConstructMulticast between
      MulticastConversation[0..1] role createdByMulticast;
      Message[0..1] role createdMessage;
end

association ReceiveMulticast between
      MulticastConversation[0..1] role receivedByMulticast;
      Message[0..1] role receivedMessage;
end
```

2.2.1 Only the specified group receives the multicast message for that group

```
-- Send multicast pre-post condition
context MulticastConversation::sendMessage(m: Message)
-- Multicast conversation associates with the Message parameter
      pre cond_1: self.createdMessage = m
-- Message must be well defined before sending
      pre cond_2: m.isDefined
-- Need to subscribe to the multicast group first
      pre cond_3: self.join = true
-- All conversations that have the same multicast address and port
receives the
-- message, including itself.
      post cond_4: MulticastConversation.allInstances->
                              forAll(c: MulticastConversation|
                                    ((c.multicastAddress =
                                    self.multicastAddress
                                    and
                                    c.multicastPort =
                                    self.multicastPort)
                                    implies
                                    c.receivedMessage = m)
                                    and
                                    (c.receivedMessage = m
                                    implies
                                    (c.multicastAddress =
                                    self.multicastAddress
                                    and
                                    c.multicastPort =
                                    self.multicastPort)))
```

This part of specification defines pre and post condition of the operation sendMessage of the class MulticastConversation. There are three pre-conditions and one post-condition. The pre-condition “cond_1” states that the Message object m must be associated with the sender. The pre-condition “cond_2” states that the attributes of Message object m must be defined. The pre-condition “cond_3” states that the join attribute of the agent must be true (agent must join in the group first). Finally, the post-condition “cond_4” states that all MulticastConversation objects that subscribes to the same multicast address and listening to the same port as the sender receive the Message object m. Therefore, all subscribers receive multicast message.

2.2.2 Multicast sent message is the same as received message

```
-- Receive multicast pre-post condition
context MulticastConversation::receiveMessage(): Message
      pre cond_1: self.join = true
-- New received message is created
      post cond_2: self.receivedMessage.oclIsNew = true
-- New created received message is the same as sent
      post cond_3: MulticastConversation.allInstances->
                              exists(c: MulticastConversation|
                                    ((c.multicastAddress =
                                    self.multicastAddress
                                    and
                                    c.multicastPort =
                                    self.multicastPort)
                                    implies
                                    c.createdMessage =
                                    self.receivedMessage)
                                    and
                                    (c.createdMessage =
                                    self.receivedMessage
                                    implies
                                    (c.multicastAddress =
                                    self.multicastAddress
                                    and
                                    c.multicastPort =
                                    self.multicastPort)))
-- Result of receiveMessage()
      post cond_4: result = self.receivedMessage
```

This part of specification defines pre and post condition of the operation receiveMessage of the class MulticastConversation. There are one pre-condition and three post-conditions. The pre-condition “cond_1” states that the join attribute must be true. The post-condition “cond_2” states that the received Message object is created during the operation receiveMessage (Message is received from another agent). The post-condition “cond_3” states that the received Message object must be the same as the sent

Message object. That is there exists a sending conversation that subscribe to same group and listen to the same port as the receiving conversation, then sent message is the same as received message. The post-condition “cond_4” specifies the return result of this operation. In this case, it is the received message is returned. Therefore, the sent message is the same as received message.

CHAPTER 7 – IMPLEMENTATION PLAN

1 Introduction

This section provides an overview of implementation plan.

1.1 Purpose

The purpose of this document is to provide work breakdown structure of the implementation phase and implementation plan for the project "Applying Broadcasting/Multicasting/Secured Communication to agentMom in Multi-Agent Systems". This document is intended to be viewed only by project advisor and committee members.

2. Work Breakdown Structure for Implementation Phase

Deliverable	Task	Completion Criteria	Time	Cost (day)
Revised Architecture Design	1.) Update class diagram from architecture phase	Class diagram is approved by committee members.	22 October – 30 October	1
	2.) Update sequence diagram from architecture phase	All sequence diagrams are approved by committee members.	22 October – 30 October	1
Revised Test Plan	1.) Update test plan from architecture phase	Test Plan is approved by committee members.	22 October – 30 October	1
Revised Formal Requirement Specification	1.) Update Formal Requirement Specification from architecture phase	Formal Requirement Specification is approved by committee members.	22 October – 30 October	1
Component Design	1.) Complete class diagram	Internal design of each class is defined.	December 01 – December 07	3
	2.) StateChart Diagram	All conversation handlers is defined.	December 01 – December 07	4
Coding	1.) Produce multicast conversation module	Executable codes are produced.	December 08 – December 31	1

	2.) Produce broadcast conversation module	Executable codes are produced.	December 08 – December 31	1
	3.) Produce secured multicast conversation module	Executable codes are produced.	December 08 – December 31	3
	4.) Produce secured unicast module	Executable codes are produced.	December 08 – December 31	1
	5.) Produce multicast handler module	Executable codes are produced.	December 08 – December 31	1
	6.) Revise unicast conversation module	Executable codes are produced.	December 08 – December 31	1
	7.) Revise unicast handler module	Executable codes are produced.	December 08 – December 31	1
	8.) Revise component module	Executable codes are produced.	December 08 – December 31	1
	9.) Revise agent module	Executable codes are produced.	December 08 – December 31	1
Unit testing	1.) Produce stub modules for unit testing	Stub module for unit testing is produced as defined in test plan.	January 01 – January 18	2
	2.) Perform unit testing	All unit testing are passed.	January 01 – January 18	2
Integration testing	1.) Produce stub modules for integration testing	Stub module for integration testing is produced as defined in test plan.	January 01 – January 18	2

	2.) Perform unit testing and integration testing	All unit testing and integration testing are passed	January 01 – January 18	2
System testing	1.) Produce simple multi-agent systems	Stub module for System testing is produced as defined in test plan.	January 01 – January 18	2
	2.) Perform system testing	System testing is passed.	January 01 – January 18	2
JavaDoc	1.) Produce JavaDoc from all source codes.	JavaDoc is approved by project advisor.	December 08 – December 31	1
Assessment Evaluation	1.) Document the testing results	Unit, integration and system testing is summarized, including failure rate.	December 08 – December 31	1
Project Evaluation Document	1.) Evaluate the usefulness of the methodologies used and accuracy of the estimations	Document is approved by project advisor.	January 19 – January 25	1
	2.) Evaluate the final product whether it meet all the requirements stated in the SRS	Document is approved by project advisor.	January 19 – January 25	1
References	1.) Compile references from all documents	References from all documents are compiled.	January 26 – January 31	0.5
Formal Technical Inspection Letters	1.) Collect Formal Technical Inspection Letter	Two inspection letters are collected.	January 26 – January 31	0.5
User Manual	1.) Produce user manual	User manual is approved by project advisor.	January 19 – January 25	2
Total cost				41

As described by Boehm, there are 152 working hours in a month, or 7 hours per day if there are 22 working days per month. Therefore, implementation phase will need

41*7 = 287 hours. Furthermore, using COCOMO model gives 715 hours to complete this project, and so far less than 400 hours are spent on this project by counting from the timelog. Thus, it is likely that this project will complete early.

3. Implementation Plan

3.1. Documents Update

After the second presentation, suggestions provided by the committee will be used to update the documents produced during the architecture phase. Also, the results from formal technical inspection will be used to update the documents. The revised document will be approved by the major professor and committee members.

3.2 User Manual

The user manual will be produced based on the previous agentMom's User Manual. The manual will describe how to use the agentMom framework. The manual also includes new agentMom's source code and an example on how to use the framework.

3.4 Component Design

The internal design of each component will be produced. The completed class diagram and sequence diagram will be produced and well document.

3.5 Source Code

agentMom source code will be document using JavaDoc standard. Also, JavaDoc API document for the new agentMom will be produced.

3.6 Assessment Evaluation

Test summary will be produced including testing results, error rate diagrams and description.

3.7 Project Evaluation

The developer will review the project. The process will be reviewed, including the usefulness of the methodologies used, the accuracy of the estimations, and the usefulness of the reviews. Furthermore, the product will be reviewed and evaluated for whether it accomplishes the ideas presented in the initial overview and for the quality of the product. Summarized of the evaluation will be document.

3.8 References

Annotated bibliography with cited references for all notations used in the project portfolio will be document.

3.9 Alpha version

All classes defined in the completed class diagram will be implemented, and unit testing will be performed before the release of alpha version.

3.10 Beta version

Integration testing must be passed as defined in the test plan before the release of beta version.

3.11 Final product

System testing must be passed as defined in the test plan before the release of final product.

3.12 Final Product Demonstration

A simple multi-agent system will be produced to demonstrate the software requirements and features.

CHAPTER 8 – FORMAL INSPECTION CHECKLIST

1. Introduction

The purpose of this document is to provide a formal checklist for the architecture design documents of the project “Applying Broadcast/Multicast/Secured Communication to agentMom in Multi-Agent Systems. Formal technical inspection process will ensure the quality of the software design. Two independent MSE students will perform the inspection and provide the formal report on the result of their inspection.

2. Item to be inspected

Architecture design documents of the project “Applying Broadcasting/Multicasting/ Secured Communication to agentMom in Multi-Agent Systems” including use cases diagram, class diagram and sequence diagram will be inspected.

The following documents will be supplied to each inspector for inspection and references:

1.) Software Requirements Specification version 1.0 *
2.) Project Overview version 1.0 *
3.) agentMom User’s Manual *
4.) Class Diagram
5.) Sequence Diagram
6.) Use Case Diagram

Note: The star (*) indicates that the document is available only for references, not for inspection.

3. Organization

Supervisory Committee consisted of:

Dr. Scott A. DeLoach
Dr. David Gustafson
Dr. William Hankley

Major Professor:

Dr. Scott A. DeLoach

Developer:

Chairoj Mekprasertvit

Formal Technical Inspector consisted of

Madhukar Kumar
Acharaporn Pattaravanichanon

4. Formal Technical Inspection Checklist

Item	Pass/Fail/Partial	Comment
1. All the symbol used in the use case diagram conforms to the UML standard.		
2. All the symbol used in class diagram conforms to UML standard.		
3. All the symbol used in Sequence diagram conforms to UML standard		
4. If there is a message passing between		

objects in sequence diagram, association relationship in class diagram is defined.		
5. Each message in sequence diagram is a method in class diagram.		
6. Use case scenarios and description are clear. Example: use case scenarios are clearly explained.		
7. Class diagram and description are clear. Example: role and responsibility of each class are clearly explained.		
8. Sequence diagram and description are clear.		
9. Names used in class diagram indicated their meaning. Example: class MulticastConversation indicates that it is used for sending and receiving multicast message.		
10. The defined public attributes should be accessible to the outside class.		
11. The defined private attributes should be accessible only within the class.		
12. The defined protected attributes should be accessible by subclass or other classes in the agentMom package.		

Formal Technical Inspection

<u>Completed for Dr DeLoach and Chairoj Mekprasertvit</u>

I have completed the Formal Technical Inspection for Chairoj Mekprasertvit's MSE project and found no obvious or serious errors in the documentation provided to me. According to my opinion, the checklist and the documentation appear to be in the desired order.

Acharaporn Pattaravanichanon

Formal Technical Inspection Checklist

Item	Pass/Fail/Partial	Comment
1. All the symbol such used in the use case diagram conforms to the UML standard.	Pass	
2. All the symbol used in class diagram conforms to UML standard.	Pass	
3. All the symbol used in Sequence diagram conforms to UML standard	Pass	
4. If there is a message passing between objects in sequence diagram, association relationship in class diagram is defined.	Pass	
5. Each message in sequence diagram is a method in class diagram.	Pass	
6. Use case scenarios and description are clear. Example: use case scenarios are clearly explained.	Pass	
7. Class diagram and description are clear. Example: role and responsibility of each class are clearly explained.	Pass	
8. Sequence diagram and description are clear.	Pass	
9. Names used in class diagram indicated their meaning. Example: class MulticastConversation indicates that it is used for sending and receiving multicast message.	Pass	
10. The defined public attributes should be accessible to the outside class.	Pass	
11. The defined private attributes should be accessible only within the class.	Pass	
12. The defined protected attributes should be accessible by subclass or other classes in the agentMom package.	Pass	

Formal Technical Inspection

Completed for Dr DeLoach and Chairoj Mekprasertvit

I have completed the Formal Technical Inspection for Chairoj Mekprasertvit's MSE project and found no obvious or serious errors in the documentation provided to me. According to my opinion, the checklist and the documentation appear to be in the desired order.

Madhukar Kumar

Formal Technical Inspection Checklist

Item	Pass/Fail/Partial	Comment
1. All the symbol such used in the use case diagram conforms to the UML standard.	Pass	
2. All the symbol used in class diagram conforms to UML standard.	Pass	
3. All the symbol used in Sequence diagram conforms to UML standard	Pass	
4. If there is a message passing between objects in sequence diagram, association relationship in class diagram is defined.	Pass	
5. Each message in sequence diagram is a method in class diagram.	Pass	
6. Use case scenarios and description are clear. Example: use case scenarios are clearly explained.	Pass	
7. Class diagram and description are clear. Example: role and responsibility of each class are clearly explained.	Pass	
8. Sequence diagram and description are clear.	Pass	
9. Names used in class diagram indicated their meaning. Example: class MulticastConversation indicates that it is used for sending and receiving multicast message.	Pass	
10. The defined public attributes should be accessible to the outside class.	Pass	
11. The defined private attributes should be accessible only within the class.	Pass	
12. The defined protected attributes should be accessible by subclass or other classes in the agentMom package.	Pass	

CHAPTER 9 – TEST PLAN

1. Test Plan Identifier

TestPlan-agentMom-001

2. Introduction

This test plan is used to address the requires tests to show that the agentMom framework after the integration of broadcast, multicast and security features satisfies the requirements stated in the Software Requirements Specification version 1.0

2.1 Objectives

a.) To detail the activities required to prepare for and conduct the test
b.) To define the test cases needed to be performed
c.) To define the types of tests that will be used for each test cases
d.) To define the environment needed to perform the test

3. Test Items

The executable java classes to be tested are identified below:

a.) Conversation class
b.) MulticastConversation class
c.) BroadcastConversation class
d.) SecureUnicastConversation class
e.) SecureMulticastConversation class
f.) MulticastHandler class
g.) BroadcastHandler class
h.) SecureMulticastHandler class
i.) MessageHandler class
j.) SecureUnicastHandler class

4. Features to be tested

The following list describes the features that will be tested:

Specification Number	Description
T-001	Sending unicast message
T-002	Sending multicast message
T-003	Sending Broadcast Message
T-004	Sending secured unicast message
T-005	Sending secured multicast message
T-006	Subscribe to multiple multicast group
T-007	Receiving unicast message
T-008	Receiving multicast message
T-009	Receiving broadcast message
T-010	Receiving secured unicast message
T-011	Receiving secured multicast message
T-012	Encrypting message
T-013	Decrypting message

5. Features not to be tested

The test cases will not cover all possible size and value of sent message. Only possible size and value that are known to be required by project committee will be tested. Also, The test cases will not cover all possible combined features. Only classes that are related will be performed integration testing.

6. Approach

Unit testing – each executable java class identified in section 3 will be tested. One or more stub modules will be created to test functionality of each class. Junit 3.8 will be the tool to perform testing. Unit testing will be performed before alpha release.

Integration testing – Several related classes will be tested together to ensure sufficient execution and compliance with the requirements after integration. One or more stub modules will be created to test functionality of combined classes. Integration testing will be performed before beta release. In this test, two architectures, component-based and agent-based, are to be considered.

System testing – The whole system will be used for system testing to ensure all requirements is satisfied, and reliability will be included in the testing to measure successful rate of message delivery. Simple multi-agent systems will be created to perform system testing. System testing will be performed before final release.

7. Environmental needs

7.1 Hardware

The testing will be done on the CIS computer lab at Kansas State University. Furthermore, the testing will be done on the Sun Sparc machine and Intel-based machine available in the computer lab.

7.2 Software

j2sdk version 1.4.2 is used to compile and execute the program.

7.3 Operating Systems

1.) Windows XP professional

2.) Linux Debian

3.) Unix Solaris

8. Test Cases

Unit testing:

8.1 Sending and receiving unicast message

Input: Message Object

Test Item: Conversation class

Method: Create sender agent and receiver agent. Sender agent sends Message object to receiver agent through unicast conversation.

Valid:

Received Message object is the same as sending Message object.

Invalid:

Received Message object is not the same as sending Message object.

8.2 Sending and receiving multicast message

Input: Message Object

Test Item: MulticastConversation class

Method: Create sender agent and two receiver agents. All of agents subscribe to the same multicast address. Sender agent sends Message object to the receiver agents through multicast conversation.

Valid:

All receiver agents receive Message object

All received Message objects are the same as sending Message object.

Invalid:

One or more agents do not receive Message object

One or more received Message objects are not the same as sending Message object.

8.3 Sending and receiving Broadcast Message

Input: Message Object

Test Item: BroadcastConversation class

Method: Create sender agent and two receiver agents. All of agents are under the same local network. Sender agent sends Message object to the receiver agents through broadcast conversation.

Valid:

All receiver agents receive Message object

All received Message objects are the same as sending Message object.

Invalid:

One or more agents do not receive Message object

One or more received Message objects are not the same as sending Message object.

8.4 Sending and receiving secured unicast message

Input: Message Object

Test Item: SecureUnicastConversation class

Method: Create sender agent and receiver agent. Sender agent sends Message object to receiver agent through secured unicast conversation.

Valid:

Received Message object is the same as sending Message object after decrypting.

Message object is encrypted before sending.

Invalid:

Received Message object is not the same as sending Message object after decrypting.

Message object is not encrypted

8.5 Sending and receiving secured multicast message

Input: Message Object

Test Item: SecureMulticastConversation class

Method: Create sender agent and two receiver agents. All of agents subscribe to the same multicast address. Encryption and decryption key are pre-defined. Each agent has the same encryption and decryption key. Sender agent sends Message object to the receiver agents through secured multicast conversation.

Valid:

All receiver agents receive Message object

All received Message objects are the same as sending Message object.

Invalid:

One or more agents do not receive Message object

One or more received Message objects are not the same as sending Message object.

8.6 Encrypting and decrypting message

Input: Message Object

Test Item: SecurityManager class

Method: Create agent to read Message object. Input the Message object to SecurityManager class.

Valid:

Message object is unreadable after it is encrypted.

Message object is readable after it is decrypted.

Invalid:

Message object is readable after it is encrypted.

Message object is unreadable after it is decrypted.

<u>**Integration testing:**</u>

8.7 Subscribe to multiple multicast group plus agent-based architecture

Input: Message Object

Test Item: MulticastHandler and MulticastConversation class

Method: Create sender agent and two receiver agents. All of agents subscribe to three different multicast addresses. Sender agent sends Message object to the three multicast addresses through multicast conversation.

Valid:

All receiver agents receive all Message object.

All received Message objects are the same as Sending Message objects.

Invalid:

One or more agents do not receive one or more of Message objects.

One or more received Message objects are not the same as sending Message objects.

8.8 Subscribe to multiple multicast group with multicast security plus agent-based architecture

Input: Message Object

Test Item: SecureMulticastConversation, MulticastConversation and MulticastHandler class

Method: Create sender agent and two receiver agents. Encryption and decryption key are pre-defined. Each agent has the same encryption and decryption key. All of agents subscribe to two multicast addresses, one for multicast conversation and another one for secured multicast conversation. Sender agent sends Message object to the receiver agents through multicast conversation and secured multicast conversation.

Valid:

Message object is encrypted before sending.

Message object is decrypted after receiving.

All receiver agents receive all Message objects

All received Message objects are the same as Sending Message object.

Invalid:

Message object is not encrypted before sending.
Message object is not decrypted after receiving.
One or more agents do not receive Message object.
One or more received Message objects are not the same as sending Message object.

8.9 Subscribe to multiple multicast group plus component component-based architecture

Input: Message Object

Test Item: MulticastHandler and MulticastConversation class

Method: Create sender agent and two receiver agents. Each agent has two components. All of agents subscribe to three different multicast addresses. Sender agent sends Message object to the three multicast addresses through multicast conversation.

Valid:
All receiver agents receive all Message object.
All received Message objects are the same as Sending Message objects.

Invalid:
One or more agents do not receive one or more of Message objects.
One or more received Message objects are not the same as sending Message objects.

8.10 Subscribe to multiple multicast group with multicast security plus component-based architecture

Input: Message Object

Test Item: SecureMulticastConversation, MulticastConversation and MulticastHandler class

Method: Create sender agent and two receiver agents. Each agent has two components. Encryption and decryption key are pre-defined. Each agent has the same encryption and decryption key. All of agents subscribe to two multicast addresses, one for multicast conversation and another one for secured multicast conversation. Sender agent sends Message object to the receiver agents through multicast conversation and secured multicast conversation.

Valid:
Message object is encrypted before sending.
Message object is decrypted after receiving.
All receiver agents receive all Message objects
All received Message objects are the same as Sending Message object.

Invalid:
Message object is not encrypted before sending.
Message object is not decrypted after receiving.
One or more agents do not receive Message object.
One or more received Message objects are not the same as sending Message object.

System testing:

8.11 Test all features using agent-based architecture

Input: Message Object

Test Item: all items identified in section 3

Method: Create sender agent and two receiver agents. One agent performs encryption and decryption key distribution. Each agent requests the key from the specified agent. All of agents subscribe to two multicast address, one for multicast conversation and another one for secure multicast communication. Each agent requests the key from specified agent. Sender agent sends Message object to the receiver agents through unicast conversation, secured unicast conversation, multicast conversation, secured multicast conversation and broadcast conversation.

Valid:

All receiver agents receive all Message objects.

All received Message object is the same as sending Message object.

Invalid:

Some receiver agents do not receive all Message objects.

Some received Message object is not the same as sending Message object.

8.12 Test all features using component-based architecture

Input: Message Object

Test Item: all items identified in section 3

Method: Create sender agent and two receiver agents. Each agent has components. One agent performs encryption and decryption key distribution. Each agent requests the key from the specified agent. All of agents subscribe to two multicast address, one for multicast conversation and another one for secure multicast communication. Each agent requests the key from specified agent. Sender agent sends Message object to the receiver agents through unicast conversation, secured unicast conversation, multicast conversation, secured multicast conversation and broadcast conversation.

Valid:

All receiver agents receive all Message objects.

All received Message object is the same as sending Message object.

Invalid:

Some receiver agents do not receive all Message objects.

Some received Message object is not the same as sending Message object.

Compatibility Testing:

8.13 Test backward compatibility of new agentMom and agentMom 1.2

Test Item: new agentMom

Method: Multi-agent systems that can be run on agentMom 1.2 should be able to run on new agentMom. The source of multi-agent systems will be supplied by Dr. DeLoach, and will test on new agentMom.

Valid:

Supplied systems must be able to run as same as under agentMom 1.2 without modifying the source code.

Invalid:

Supplied systems fails to run under the new agentMom.

9. Schedule

The testing will be performed during January 1, 2003 – January 18, 2003.