Applying Broadcasting/Multicasting/Secured Communication to agentMom in Multi-Agent Systems

Architecture Design

Version 1.1

This document is submitted in partial fulfillment of the requirements for the degree MSE.

Chairoj Mekprasertvit
CIS 895 – MSE Project
Kansas State University
Fall 2003

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

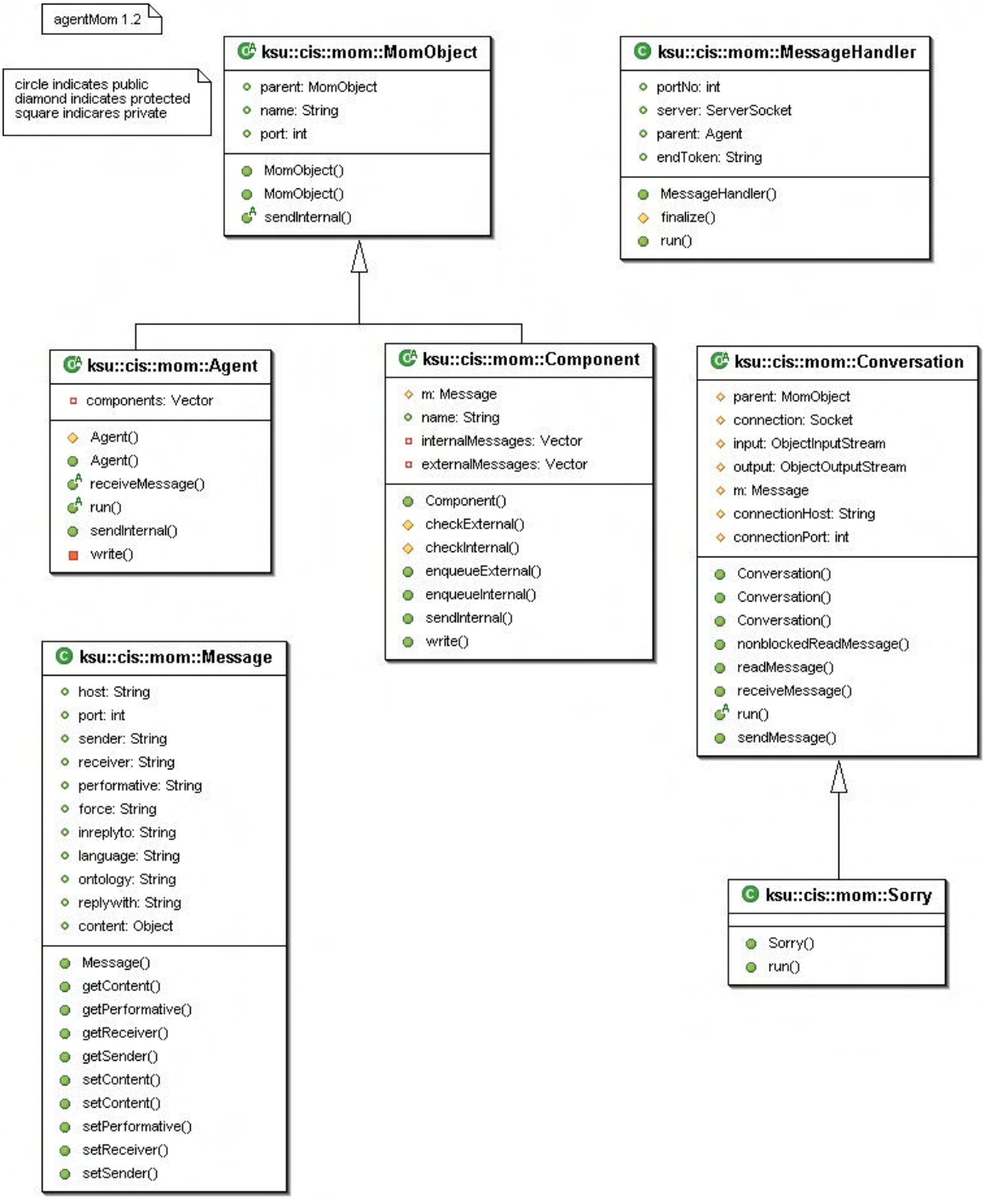


Figure 1 Class Diagram for agentMom1.2

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

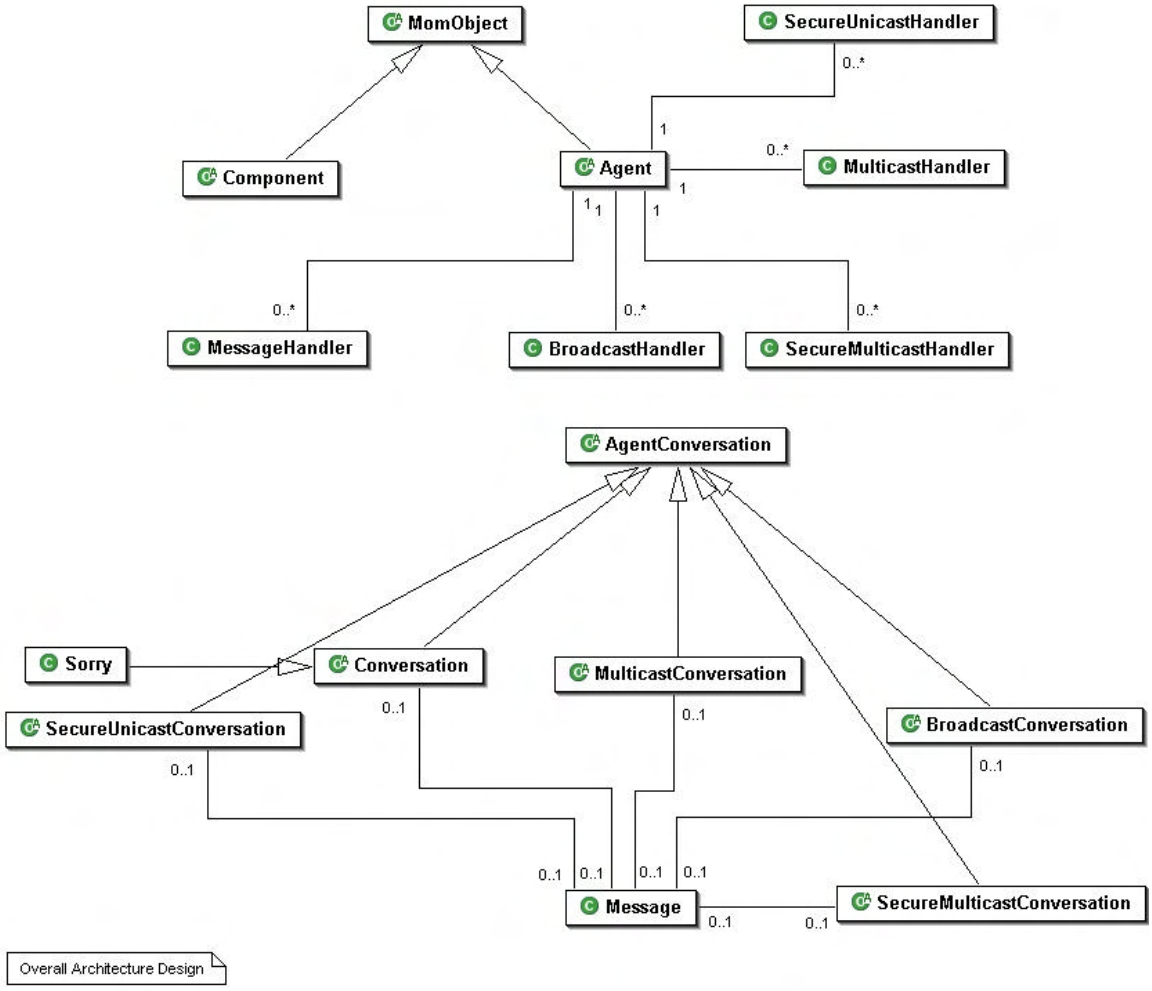
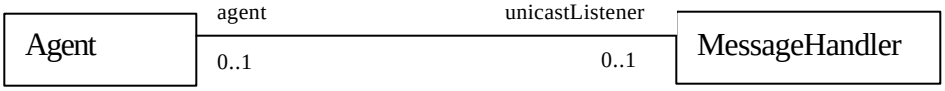


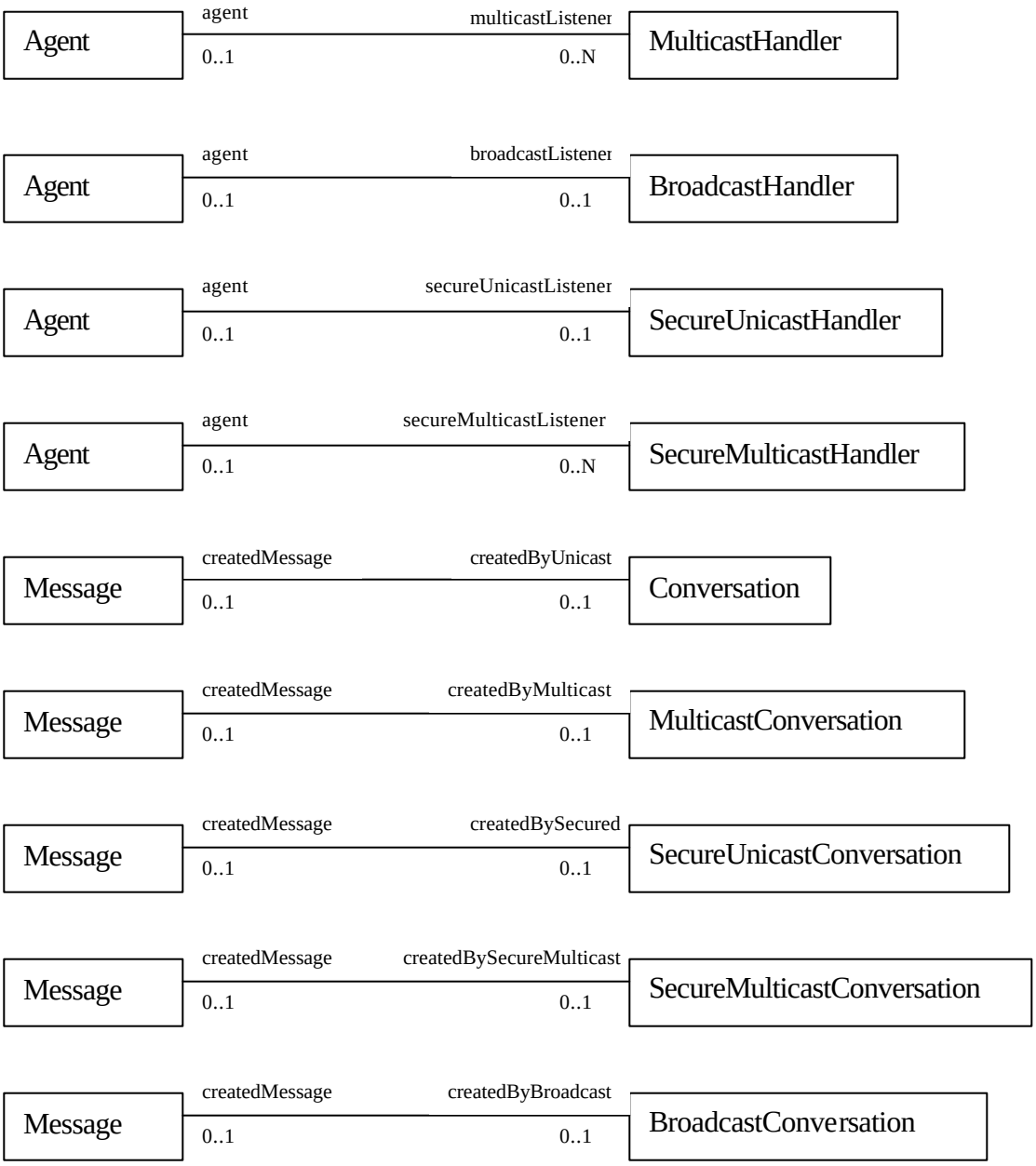
Figure 2 Overall Architecture Design

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:

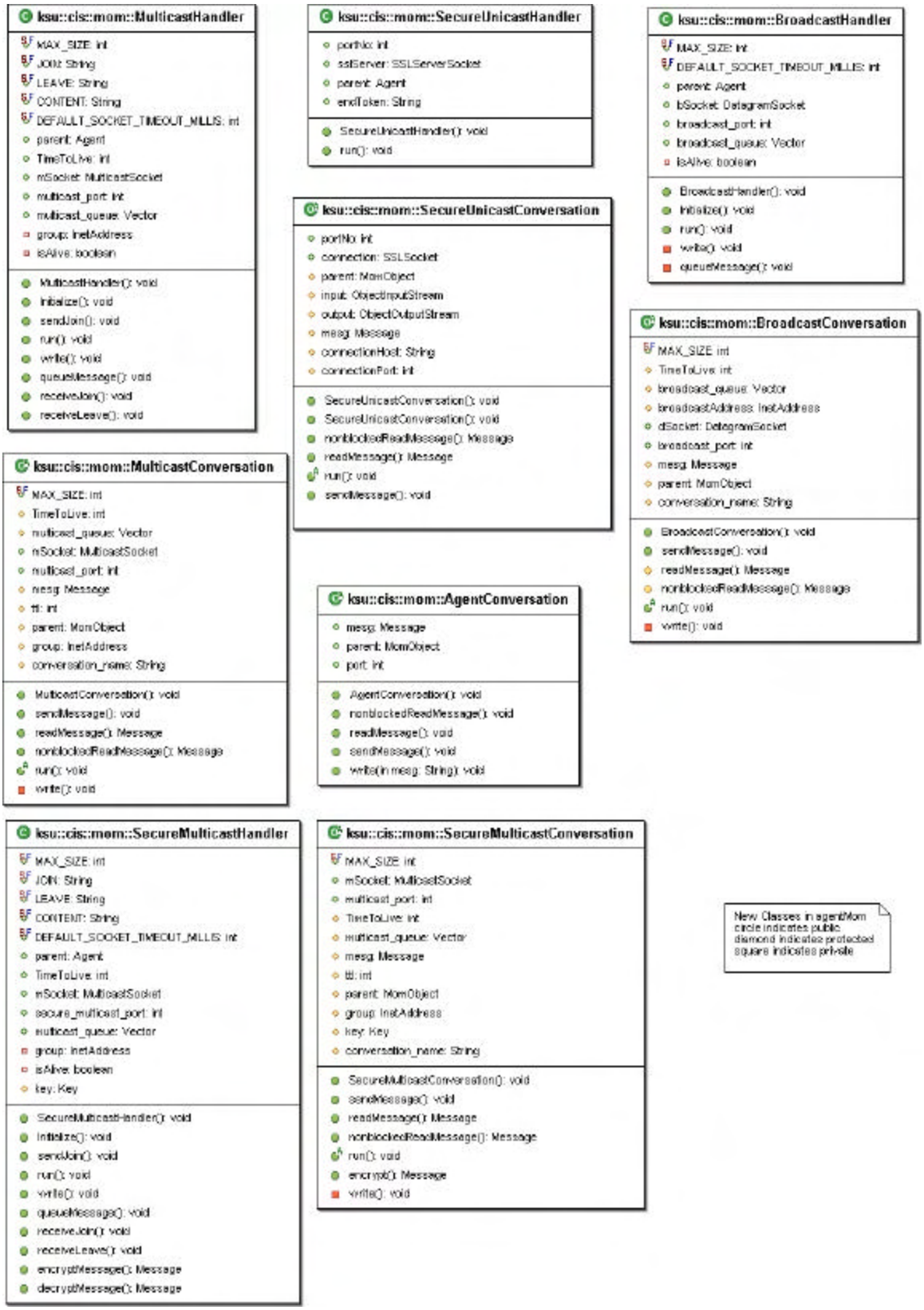


Figure 3 Nine New Classes in agentMom

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 4 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.

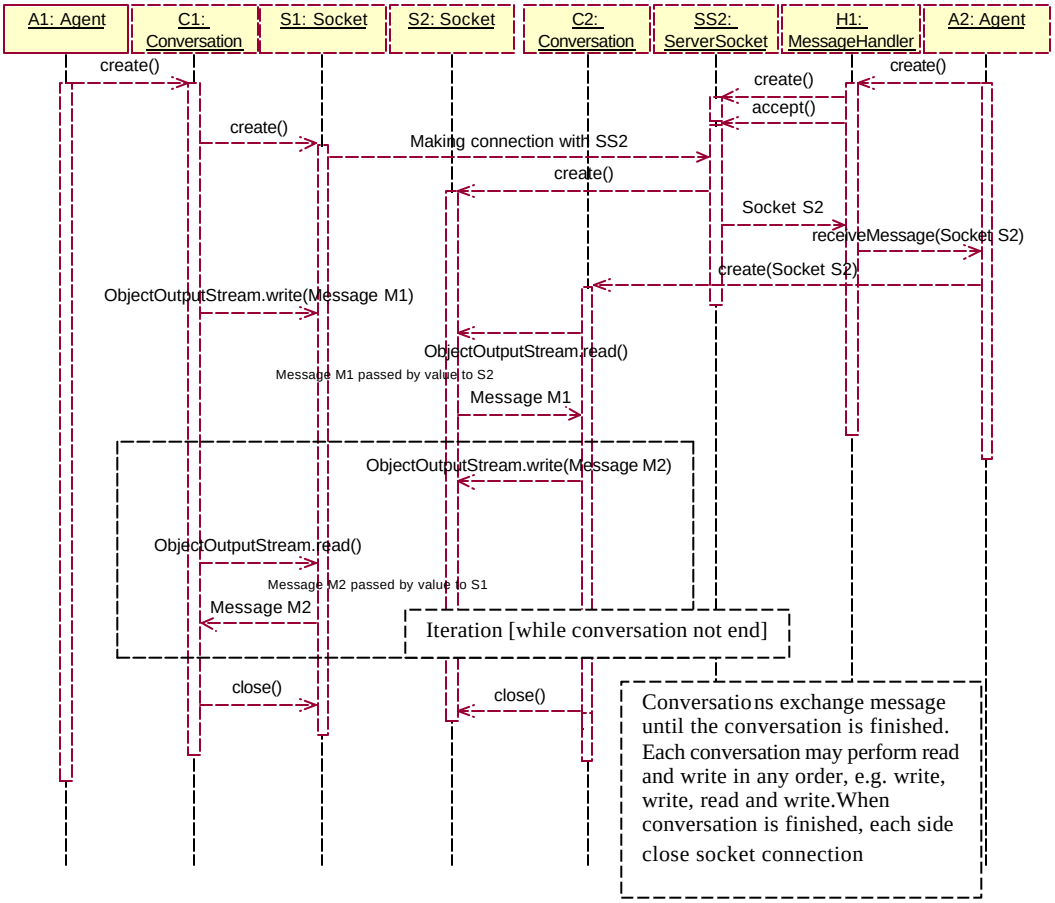


Figure 4 Starting Unicast Conversation

3.2 Multicast conversation

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

In Figure 5 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.

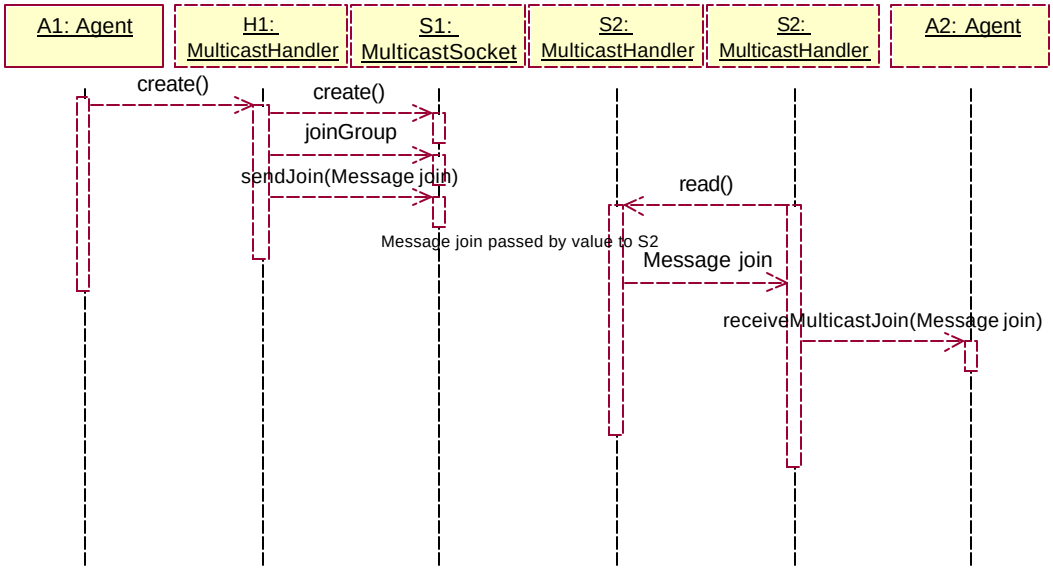


Figure 5 Join Multicast Group

In Figure 6 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.

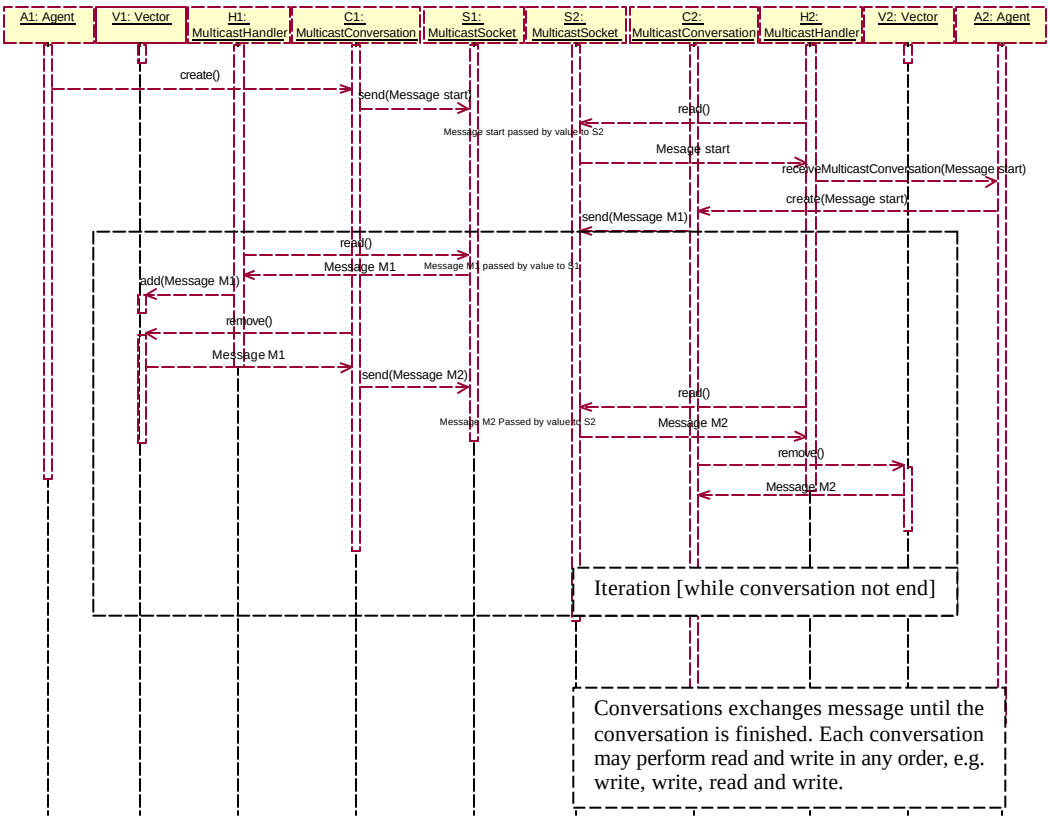


Figure 6 Multicast Conversation

In Figure 7 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.

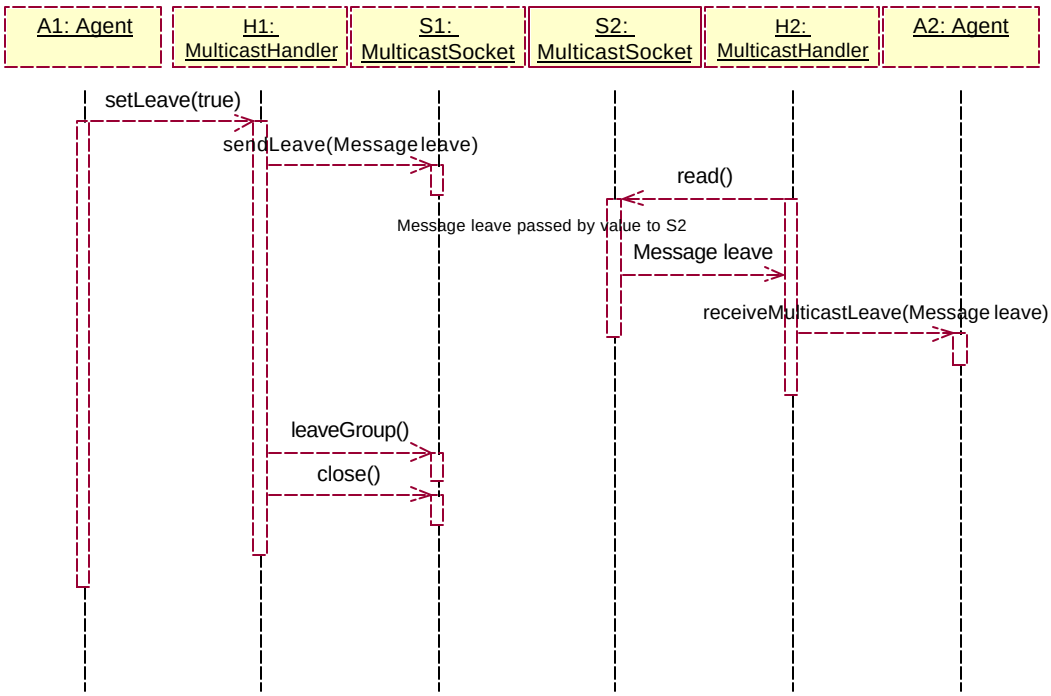


Figure 7 Leave Multicast Group

3.3 Broadcast conversation

In Figure 8 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.

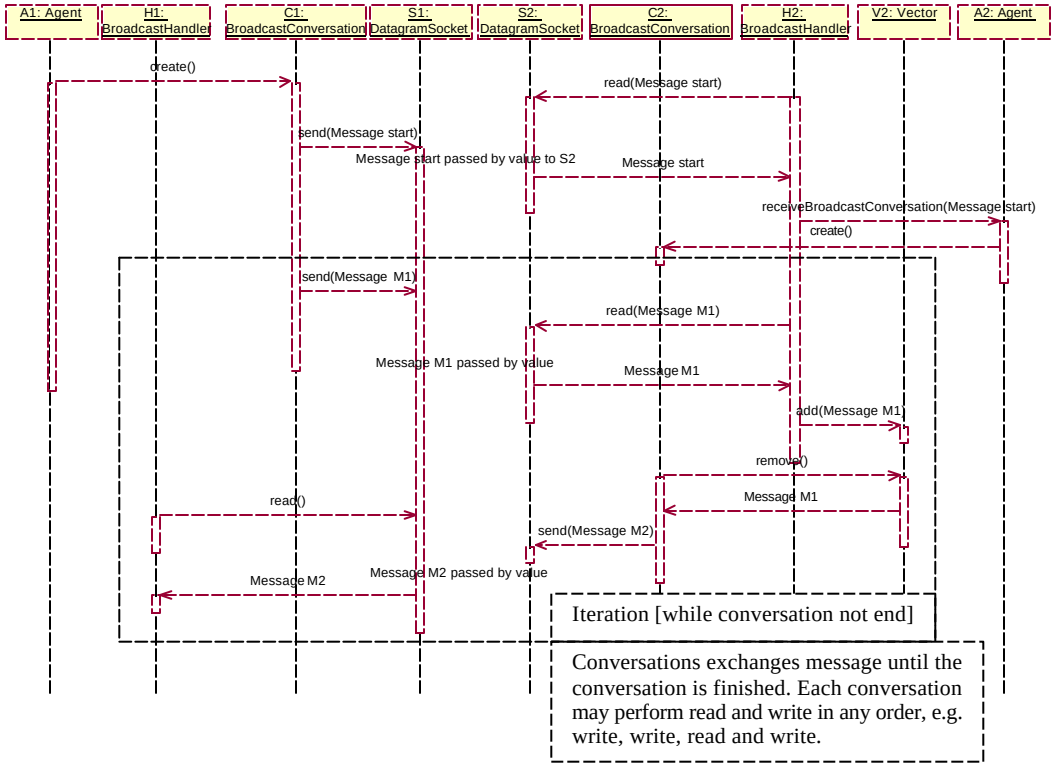


Figure 8 Broadcast Conversation