CHAPTER 10 – COMPONENT DESIGN

Class Diagram

The class diagram of the new agentMom package is shown below. It consists of 16 classes, 9 abstract classes and 7 concrete classes. The abstract classes are the **MomObject, Agent, Component, AgentConversation, Conversation, MulticastConversation, BroadcastConversation, SecureUnicastConversation** and **SecureMulticastConversation**. The concrete classes are the **MessageHandler**, **MulticastHandler**, **BroadcastHandler**, **SecureUnicastHandler**, **SecureMulticastHandler**, **Message** and **Sorry**.

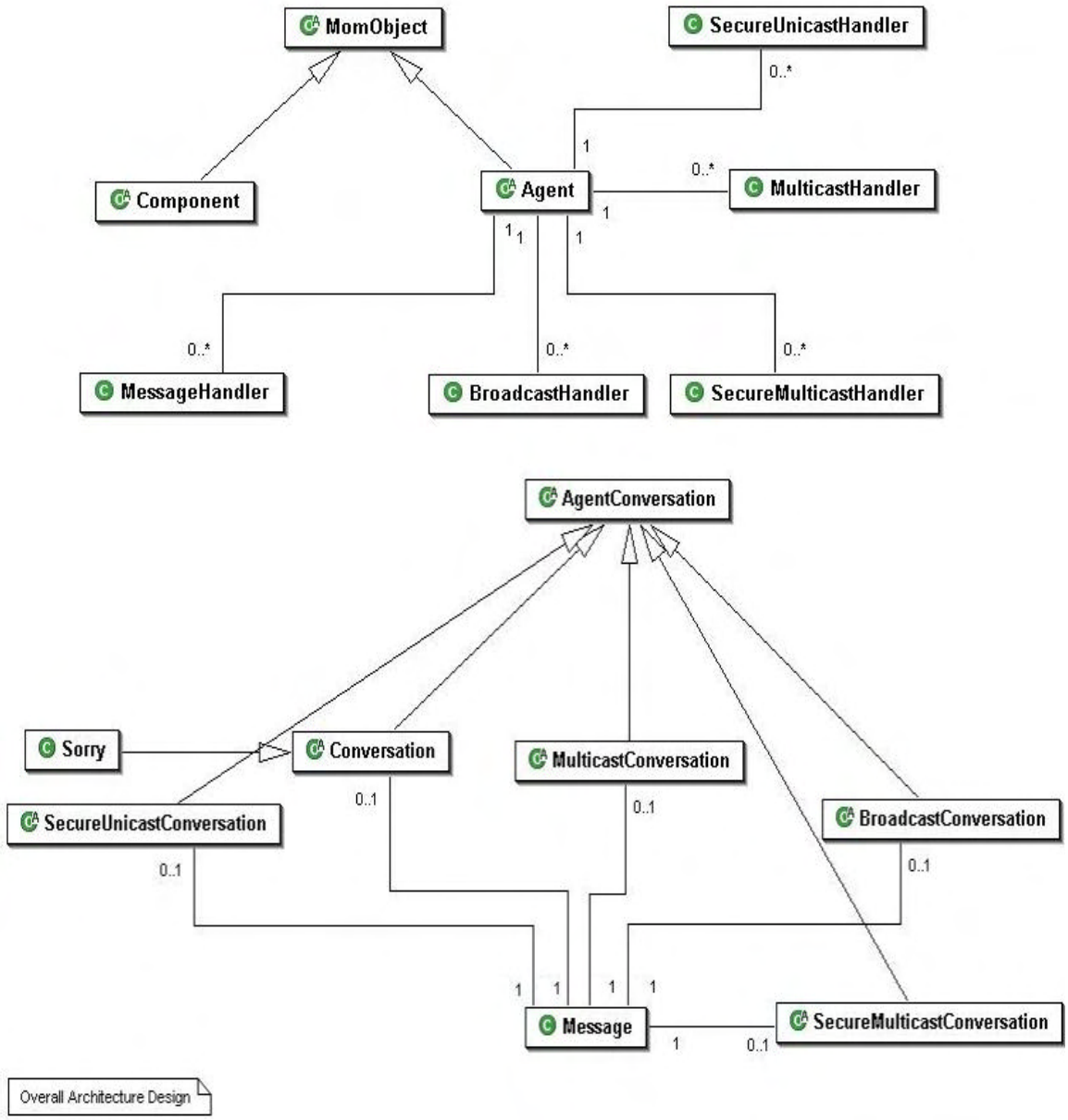

Figure 15 Class Diagram: new agentMom

Classes Descriptions

1. Agent Class

Inherits **MomObject**.

1.1 Detailed Description

The **Agent** class is an abstract class that defines the minimum requirements for an agent to use agentMom package. This class inherits from **MomObject** class. It must be runnable as a separate thread, which requires a run method. It also has two required parameters that must be set for each agent, the name of the agent and the port number on which its message handler will listen for incoming messages.

An agent needs to start the correct handler to be able to receive the start of conversation from other agents. Furthermore, each agent must implement/override receive message method for each type of conversation. For unicast conversation, the agent must implement the **receiveMessage** method. For the other conversation, the agent must override the receive method corresponding to each conversation. The receive method will be called by the corresponding message handler when the agent has received a connection starting a new conversation. For example, the corresponding handler of **receiveMulticastConversation** method is **MulticastHandler** class, and the corresponding handler of **receiveMessage** method **MessageHandler** class.

Public Member Functions

- **Agent** (String **name**, int **port**)
- **Agent** (String **name**, int unicast_port, int **multicast_port**[], int **broadcast_port**, int **secure_unicast_port**, int **secure_multicast_port**[])
- abstract void **receiveMessage** (Socket server, ObjectInputStream input, ObjectOutputStream output)
- void **receiveBroadcastConversation** (DatagramSocket bSocket, **Message** m, Vector **broadcast_queue**)
- void **receiveMulticastConversation** (MulticastSocket mSocket, **Message** m, Vector **multicast_queue**)
- void **receiveMulticastJoin** (**Message** m)
- void **receiveMulticastLeave** (**Message** m)
- void **receiveSecureMulticastConversation** (MulticastSocket mSocket, **Message** m, Vector **multicast_queue**, Key k, String algorithm)
- void **receiveSecureUnicastConversation** (SSLSocket server, ObjectInputStream input, ObjectOutputStream output)
- abstract void **run** ()
- void **sendInternal** (**Message** m)
- void **write** (String s)

Public Attributes

- Vector **broadcast_queue**
- Vector **multicast_queue**
- Vector **secure_multicast_queue**

Protected Member Functions

- **Agent** ()

Private Attributes

- Vector **components** = new Vector()

1.2 Constructor

Agent () [protected]

This is a default constructor. It is an empty constructor and does not require any argument.

Agent (String *name*, int *port*)

This is a constructor for using only unicast conversation. It takes two arguments, String of agent name and integer of unicast port.

Parameters:

name - String name of the agent
port - Integer port of unicast

Agent (String *name*, int *unicast_port*, int *multicast_port*[], int *broadcast_port*, int *secure_unicast_port*, int *secure_multicast_port*[])

This is a constructor for using any or all of five different conversations. It takes six arguments, String of agents name, integer of port used in each conversation. If any port is assigned to be less than one, then it indicates the conversation is not going to be used.

Note that the arguments multicast_port and secure_multicast_port are an array of integer type. It is because we allows agent to subscribe to multiple groups. Thus, one port is used for one group. Assigning the first element less than one indicates that the multicast will not be used.

Parameters:

name - String name of the agent
unicast_port - Integer port number used for unicast conversation
multicast_port - Array of integer port number used for multicast conversation
broadcast_port - Integer port number used for broadcast conversation
secure_unicast_port - Integer port number used for secured unicast conversation
secure_multicast_port - Array of integer port number used for secured multicast conversation

---

1.3 Member Function

void receiveBroadcastConversation (DatagramSocket *bSocket*, Message *m*, Vector *broadcast_queue*)

Receive message method for the broadcast conversation. An agent must override this method and defines all possible of broadcast conversations here.

This is the method that will be called by the **BroadcastHandler** when the agent has received connection starting a new broadcast conversation.

Parameters:

bSocket - datagram socket for handling broadcast conversation

m - message from other agents
broadcast_queue - message queue for broadcast conversation

abstract void receiveMessage (Socket *server*, ObjectInputStream *input*, ObjectOutputStream *output*)

Receive message method for the unicast conversation. An agent must implement this method and defines all possible of unicast conversations here.

The MessageHandler will call this method when the agent has received connection starting a new unicast conversation.

Parameters:

server - Socket for the unicast conversation
input - ObjectInputStream for the unicast conversation
output - ObjectOutputStream for the unicast conversation

void receiveMulticastConversation (MulticastSocket *mSocket*, Message *m*, Vector *multicast_queue*)

Receive message method for the multicast conversation. An agent must override this method and defines all possible of multicast conversations here.

This is the method that will be called by the **MulticastHandler** when the agent has received connection starting a new multicast conversation.

Parameters:

mSocket - multicast socket for handling multicast conversation
m - message from other agents
multicast_queue - message queue for multicast conversation

void receiveMulticastJoin (Message *m*)

Receive multicast join message method for the multicast conversation. This is the method that will be called by the **MulticastHandler** when the agent has received join message from other agents. An agent can override this method and performs some tasks. For example, agent can keep track of other agents who join the group after itself.

Parameters:

m - join message from other agents

void receiveMulticastLeave (Message *m*)

Receive multicast leave message method for the multicast conversation. This is the method that will be called by the **MulticastHandler** when the agent has received leave message from other agents. An agent can override this method and performs some tasks. For example, agent can keep track of other agents who leave.

Parameters:

m - leave message from other agents

void receiveSecureMulticastConversation (MulticastSocket *mSocket*, Message *m*, Vector *multicast_queue*, Key *k*, String *algorithm*)

Receive message method for the secured multicast conversation. An agent must override this method and defines all possible of secured multicast conversations here.

This is the method that will be called by the **SecureMulticastHandler** when the agent has received a connection starting a new secured multicast conversation.

Parameters:

mSocket - multicast socket for handling secured multicast conversation
m - message from other agents
multicast_queue - message queue for secured multicast conversation
k - symmetric private key.
algorithm - symmetric key algorithm name

void receiveSecureUnicastConversation (SSLSocket *server*, ObjectInputStream *input*, ObjectOutputStream *output*)

Receive message method for the secured unicast conversation. An agent must override this method and defines all possible of secured unicast conversations here.

This is the method that will be called by the **SecureUnicastHandler** when the agent has received a connection starting a new secured unicast conversation.

Parameters:

server - Secured socket for the secured unicast conversation
input - ObjectInputStream for the secured unicast conversation
output - ObjectOutputStream for the secured unicast conversation

abstract void run ()

An agent must be runnable as a separate thread, and each agent must implement this method.

void sendInternal (Message *m*)

This is a method for sending message between components within an agent. This method simply broadcast message to all active components of the agent.

Parameters:

m - message for internal conversation among the component.

void write (String *s*)

This is a method for printout String information on terminal screen.

Parameters:

s - String to print out on the terminal screen.

1.4 Member Data

Vector broadcast_queue

Message queue for broadcast conversation.

Vector components = new Vector()

Vector of internal components that are active.

Vector multicast_queue

Message queue for multicast conversation.

Vector secure_multicast_queue

Message queue for secured multicast conversation.

2. AgentConversation Class Reference

Detailed Description

This class is an abstract class that all types of conversation inherit from. It defines the minimum requirements for a conversation to be in agentMom package. It also allows user to easily implement a new type of conversation for agentMom package.

Public Member Functions

- **AgentConversation** (**MomObject** c)
- **AgentConversation** (**MomObject** c, String hostName, int portNum)
- **AgentConversation** (**MomObject** c, **Message m**)
- void **write** (String mesg)

Protected Attributes

- **MomObject parent**
- int **connectionPort**
- String **connectionHost**
- **Message m**

2.1 Constructor

AgentConversation (MomObject *c*)

Default **AgentConversation** constructor.

Parameters:

c - Reference to parent **MomObject** class.

AgentConversation (MomObject *c*, String *hostName*, int *portNum*)

Constructor for conversation initiator. It defines minimum requirement for conversation initiator.

Parameters:

c - Reference to parent **MomObject** class.
hostName - host name that this conversation is connected to.
portNum - port number of the host that this conversation is connected to.

AgentConversation (MomObject *c*, Message *m*)

Constructor for conversation respondent. It defines minimum requirement for conversation respondent.

Parameters:

c - Reference to parent **MomObject** class.
m - **ksu.cis.mom.Message**

2.2 Member Function

void write (String *mesg*)

Method for easily printout information on screen.

Parameters:

mesg java.lang.String

2.3 Member Data

String connectionHost `[protected]`

Specify the host address.

int connectionPort `[protected]`

Port number of the host that the conversation is connected to.

Message m `[protected]`

Message class is used for storing message sent from other agents.

MomObject parent `[protected]`

Reference to parent agent class.

---

3. BroadcastConversation Class

Inherits **AgentConversation**.

3.1 Detailed Description

The **BroadcastConversation** class is an abstract class that actually carries out the broadcast message to all agents under the same local network. This class provides two main services, read broadcast message and send broadcast message. It is responsible for passing the messages back and forth using datagram socket.

There are really two types of conversation classes that can be derived from the **BroadcastConversation** class, one for the conversation initiator and one for the conversation respondent. The basic difference lies in which constructor is used and the details in the abstract run method, which must be implemented in the concrete class derived from the **BroadcastConversation** class.

Public Member Functions

- **BroadcastConversation** (**MomObject** c)
- **BroadcastConversation** (**MomObject** c, InetAddress broadcast_address, int port, Vector **broadcast_queue**)
- **BroadcastConversation** (**MomObject** c, InetAddress broadcast_address, int port, Vector **broadcast_queue**, **Message m**)
- void **sendMessage** (**Message m**, String replywith, String inreplyto) throws IOException

- void **startConversation** (**Message m**, String replywith, String inreplyto) throws IOException
- **Message readMessage** (String **conversation_name**)
- **Message nonblockedReadMessage** (String **conversation_name**, long timeout)
- abstract void **run** ()

Static Public Attributes

- final int **START_CONVERSATION** = 0
- final int **CONTENT** = 1

Protected Attributes

- DatagramSocket **dSocket**
- Vector **broadcast_queue**
- String **conversation_name**

3.2 Constructor

BroadcastConversation (MomObject *c*)

Default **Conversation** constructor.

Parameters:

c - Reference to parent **MomObject** class.

BroadcastConversation (MomObject *c*, InetAddress *broadcast_address*, int *port*, Vector *broadcast_queue*)

Constructor for conversation initiator

Parameters:

c - reference to parent class.
broadcast_address - IP address to send broadcast message.
port - port for broadcast conversation.
broadcast_queue - message queue for broadcast conversation.

BroadcastConversation (MomObject *c*, InetAddress *broadcast_address*, int *port*, Vector *broadcast_queue*, Message *m*)

Constructor for conversation respondent.

Parameters:

c - reference to parent class.
broadcast_address - IP address to send broadcast message.
port - port for broadcast conversation.
broadcast_queue - message queue for broadcast conversation.
m - **ksu.cis.mom.Message**

3.3 Member Function

Message nonblockedReadMessage (String *conversation_name*, long *timeout*)

Method to fetch broadcast message from broadcast message queue destined for this conversation. This method can be considered a nonblocked read. This method search message destined for specified conversation name by comparing the String parameter with the inreplyto field in the message. If there is no message for specified conversation, it waits for the amount of specified timeout in miilisecond and then tries again. If there is still no message on the second try, it returns **Message** with the content field "timeout".

Parameters:

conversation_name - name of conversation to retrive message

Returns:

Message with content field "timeout" if there is no message destine for specified conversation.

Message readMessage (String *conversation_name*)

Method to fetch broadcast message from broadcast message queue destined for this conversation. This method can be considered a blocked read. This method search message destined for specified conversation name by comparing the String parameter with the inreplyto field in the message. If there is no message for specified conversation, it waits for 1000 miilisecond and try again. It continue to try and wait until the message is found.

Parameters:

conversation_name - name of conversation to retrive message

Returns:

ksu.cis.mom.Message

abstract void run ()

Run method for **BroadcastConversation** class. Each derived conversation must be run as separate thread and must implement this method regarding to the type of conversation, initiator or respondent conversation.

void sendMessage (Message *m*, String *replywith*, String *inreplyto*) throws IOException

This method is responsible for sending the **Message** m through the datagram socket. It also automatically fill the sender, host, port, replywith and inreplyto fields in the **Message** object m using the provided parameters, parent agent's name and port attributes and automatically gets the host name from the system.

It also converts **Message** type Object to byte array to be able to send through datagram socket.

Note that to send message request a start of new conversation the correct method is **startConversation**.

Parameters:

m - message to send
replywith - name of the originating conversation (this conversation's name).
inreplyto - name of the destined conversation on the other sides of agent.

void startConversation (Message *m*, String *replywith*, String *inreplyto*) throws IOException

This method is responsible for sending message request a start of new conversation. The **Message** m is sent through the datagram socket. It also automatically fill the sender, host, port, replywith and inreplyto fields in the **Message** object m using the provided parameters, parent agent's name and port attributes and automatically gets the host name from the system.

It also converts **Message** type Object to byte array to be able to send through datagram socket.

Parameters:

m - message to send
replywith - name of the originating conversation (this conversation's name).
inreplyto - name of the destined conversation on the other sides of agent.

3.4 Member Data

Vector broadcast_queue [protected]

Message queue for broadcast conversation. It is a Vector type that stores broadcast message sent to this agent.

final int CONTENT = 1 [static]

It is the header to indicate that the message is not a request to start conversation, but the message passing back and forth during the conversation.

String conversation_name [protected]

Name of the broadcast conversation. This name must be unique from others broadcast conversations within agent since it is used for identifying itself when fetching the message from message queue. It is recommended to use the class's name.

DatagramSocket dSocket [protected]

Datagram socket used to send and receive broadcast conversation.

final int START_CONVERSATION = 0 [static]

Header to indicate that the message is a request to start conversation.

---

4. BroadcastHandler Class

4.2 Detailed Description

BroadcastHandler is responsible for initializing and starting datagram socket for broadcast conversation. It uses the DatagramSocket class to send and receive broadcast conversation in form of datagram packet.

When an agent is created, it needs to create a new broadcast handler thread to be able to receive a start of broadcast conversation from the other agents. When **BroadcastHandler** is created, it creates the DatagramSocket class for broadcast conversation.

There are two constructors for this class. The first one does not require the maximum datagram received packet size and use the default size (1024 bytes). The second one allows user to specify the maximum datagram received packet size.

To create the **BroadcastHandler**, it requires three/four parameters, a reference to parent agent object, broadcast port number, broadcast address and/or maximum datagram received packet size. When this class is started, it starts a datagram socket on the indicated port and waits for messages from other agents. All of broadcast messages sent to this agent will be received by this class. When broadcast handler receives a message, it will check whether the message is a start of new conversation. If it is a start of conversation, the **BroadcastHandler** simply calls the parent agent's receiveBroadcastConversation method with the datagram socket, received message and broadcast message queue. The agent then verifies the received message and starts an appropriate conversation. If the received message is for any broadcast conversation class, it adds the message to the broadcast message queue. Then the broadcast conversation class can get the message from this queue later.

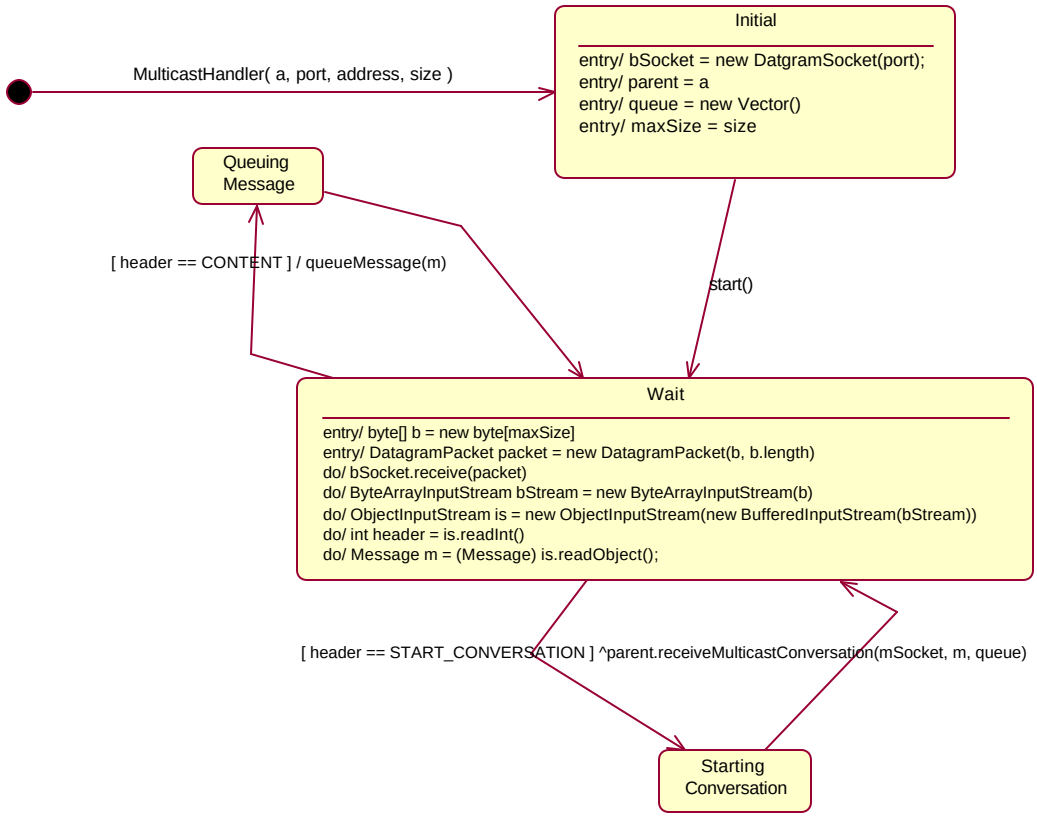

Figure 16 State chart: BroadcastHandler

From the state chart above, the BroadcastHandler initializes its attributes when the constructor is called. When the thread starts, it waits for the broadcast message to arrive. From the wait state, the state is changed according to the received message header. For example, if the header is CONTENT, the queueMessage method is called. If the header is

START_CONVERSATION is called, the parent agent's receiveBroadcastConversation method is called. This class does not have an end state. It keeps waiting for the broadcast message to arrive.

Public Member Functions

- **BroadcastHandler** (**Agent** p, int port, InetAddress b_address)
- **BroadcastHandler** (**Agent** p, int port, InetAddress b_address, int packetSize)
- void **run** ()

Public Attributes

- **Agent parent**
- DatagramSocket **bSocket**
- int **broadcast_port**
- Vector **broadcast_queue**

Static Public Attributes

- final int **START_CONVERSATION** = 0
- final int **CONTENT** = 1

Private Member Functions

- void **write** (String s)
- void **queueMessage** (**Message** m)

Private Attributes

- int **maxSize** = 1024

4.2 Constructor

BroadcastHandler (Agent *p*, int *port*, InetAddress *b_address*)

Constructor for **BroadcastHandler** class. It is responsible for initializing necessary variables and creates datagram socket.

Parameters:

p - reference to parent agent class.
port - port number for listening for broadcast message.
b_address - broadcast address. In general, it is in the form "xxx.xxx.xxx.255" for local broadcast.

BroadcastHandler (Agent *p*, int *port*, InetAddress *b_address*, int *packetSize*)

Constructor for **BroadcastHandler** class. It is responsible for initializing necessary variables and creates datagram socket. This constructor allows specifying the size of received broadcast packet.

Parameters:

p - reference to parent agent class.
port - port number for listening for broadcast message.
b_address - broadcast address. In general, it is in the form "xxx.xxx.xxx.255" for local broadcast.
packetSize - user defined maximum size in byte of received datagram packet (can be think of as a buffer).

4.3 Member Function

void queueMessage (Message *m*) [private]

Method for adding new message to the queue.

Parameters:

m - message to be retrived by broadcast conversation.

void run ()

Run method for **BroadcastHandler**. It simply waits for incoming broadcast message from other agents. Then, it checks the message whether it is a start of new broadcast conversation, or a message for broadcast conversation class.

void write (String *s*) [private]

Method for easily printout information on screen.

Parameters:

s - java.lang.String

4.4 Member Data

int broadcast_port

Broadcast port number.

Vector broadcast_queue

Message queue for broadcast conversation. It is a Vector type that stores broadcast message sent to this agent.

DatagramSocket bSocket

Datagram socket used to send and receive broadcast conversation.

final int CONTENT = 1 [static]

To indicate that the message is not a request to start conversation.

int maxSize = 1024 [private]

Default maximum size of received datagram packet.

Agent parent

Reference to parent agent class.

final int START_CONVERSATION = 0 [static]

Header to indicate that the message is a request to start conversation.

---

5. Component Class

Inherits **MomObject**.

5.1 Detailed Description

The **Component** class is an abstract class that defines the minimum requirements for a component. This class inherits from **`MomObject`** class. It implements the Runnable interface to be able to run as a thread. It requires only one parameter, **MomObject**. **MomObject** is used to be able to refer to the agent that uses this component.

The idea of **Component** class is to support agent architecture that component performs different tasks. Each component is responsible for particular tasks. Thus, the agents' role's tasks can be mapped to component. Also, components are responsible for starting the conversation with other agents, instead of agent itself. Therefore, agent starts the components, and component starts the conversations.

Public Member Functions

- **Component** (**MomObject** p)
- void **enqueueExternal** (**Message m**)
- void **enqueueInternal** (**Message m**)
- void **sendInternal** (**Message m**)
- void **write** (String s)

Public Attributes

- String **name**

Protected Member Functions

- **Message checkExternal** ()
- **Message checkInternal** ()

Protected Attributes

- **Message m**

Private Attributes

- Vector **internalMessages** = new Vector()
- Vector **externalMessages** = new Vector()

5.2 Constructor

Component (MomObject *p*)

Default constructor for component class.

Parameters:

p - Reference to MomObject class that this component class belong to.

5.3 Member Function

Message checkExternal () [protected]

To fetch message from other agents to this agent's component. It returns the first message(element) in the Vector externalMessage. If there is no message, null is return.

Returns:

the first message in the Vector externalMessage or null if there is no message.

Message checkInternal () [protected]

To fetch message for internal component communication. It returns the first message(element) in the Vector internalMessage. If there is no message, null is return.

Returns:

the first message in the Vector internalMessage or null if there is no message.

void enqueueExternal (Message *m*)

Add message for external communication (to other agents) to the Vector externalMessage. This method is used for passing message between component and conversation.

Parameters:

m - message for external communication

void enqueueInternal (Message *m*)

Add message for internal component to the Vector internalMessage.

Parameters:

m - message for internal component.

void sendInternal (Message *m*)

This method calls the agent's sendInternal method and passes the message that will be sent to internal component of the agent.

Parameters:

m - **Message** class that will be sent to internal component.

void write (String *s*)

Method for printout a String information on terminal screen.

Parameters:

s String to print out on the terminal screen.

5.4 Member Data

Vector externalMessages = new Vector() [private]

Message queue for external communication. It stores message that will be used to make conversation with other agents.

Vector internalMessages = new Vector() `[private]`

Message queue for internal communication. It stores message that will be used for communication with other components.

Message m `[protected]`

Message used for internal/external communication.

String name

Name of the component.

---

6. Conversation Class

Inherits **AgentConversation**.

6.1 Detailed Description

The **Conversation** class is an abstract class that actually carries out the message passing between agents. This class provides two main services, read message and send message. It is responsible for passing the messages back and forth over the TCP/IP socket connection.

There are really two types of conversation that can be derived from this **Conversation** class, one for the conversation initiator and one for the conversation respondent. The basic difference lies in which constructor is used and the details in the abstract run method, which must be implemented in the concrete class derived from the **Conversation** class.

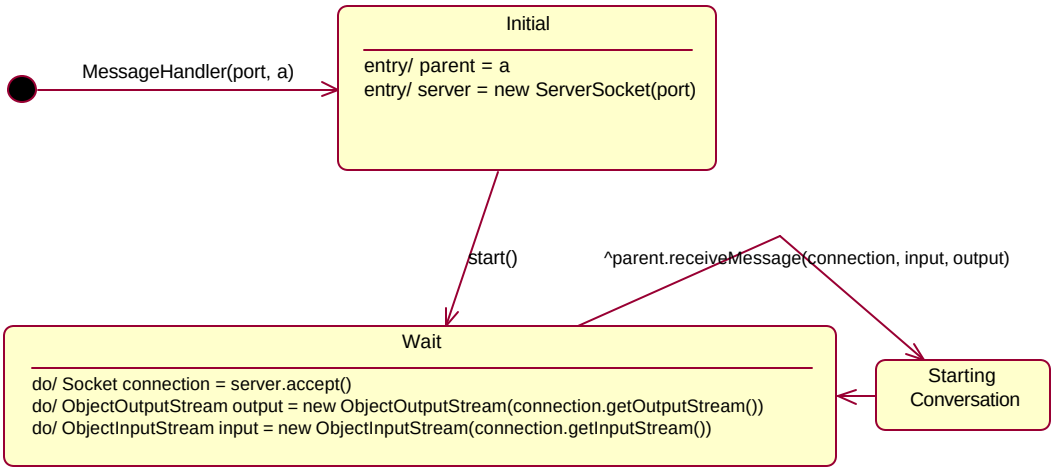

Figure 17 State chart: MessageHandler

From the state chart above, the MessageHandler initializes its attributes when the constructor is called. When the thread starts, it waits form unicast connection. From the wait state, the state is changed when it receives a connection from other agents. When it receives connection, it simply accepts and get the input and output stream. This class does not have an end state. It keeps waiting for new connection.

Public Member Functions

- **Conversation** (**MomObject** c)
- **Conversation** (**MomObject** c, String hostName, int portNum)
- **Conversation** (Socket s, ObjectInputStream i, ObjectOutputStream o, **MomObject** c, **Message m**)
- **Message nonblockedReadMessage** (ObjectInputStream **input**)
- **Message readMessage** (ObjectInputStream **input**)
- void **receiveMessage** (**Message m**)
- abstract void **run** ()
- void **sendMessage** (**Message m**, ObjectOutputStream **output**)

Protected Attributes

- Socket **connection**
- ObjectInputStream **input**
- ObjectOutputStream **output**

6.2 Constructor

Conversation (MomObject *c*)

Default **Conversation** constructor.

Parameters:

c - Reference to parent **MomObject** class.

Conversation (MomObject *c*, String *hostName*, int *portNum*)

Constructor for conversation initiator.

Parameters:

c - Reference to parent **MomObject** class.
hostName - host name that this conversation is connected to.
portNum - port number of the host that this conversation is connected to.

Conversation (Socket *s*, ObjectInputStream *i*, ObjectOutputStream *o*, MomObject *c*, Message *m*)

Constructor for conversation respondent.

Parameters:

s - connection socket
i - input stream
o - output stream
c - reference MomObject that this conversation belong to.
m - ksu.cis.mom.Message

6.3 Member Function

Message nonblockedReadMessage (ObjectInputStream *input*)

This read method allows timeout nonblocking read message. It allows the read message to "timeout" thus allowing the conversation to check to see if it has a message without waiting forever. The default value of timeout is 100 milliseconds.

Parameters:

input - input stream to read from.

Returns:

ksu.cis.mom.Message

Message readMessage (ObjectInputStream *input*)

This read method is blocking read message. It waits until the message is arrived.

Parameters:

input - input stream to read from.

Returns:

ksu.cis.mom.Message

abstract void run ()

Run method for conversation class. Each derived conversation must be run as separate thread and must implement this method regarding to the type of conversation, initiator or respondent conversation.

void sendMessage (Message *m*, ObjectOutputStream *output*)

This method is responsible for sending the **Message** m through the ObjectOutputStream output. It also automatically fills the sender, host, and port fields in the **Message** object m using the parent agent's name and port attributes and automatically gets the host name from the system.

Parameters:

m - **Message to send**
output - output stream for sending message

6.4 Member Data

Socket connection `[protected]`

Socket class is used for connecting to another agent.

ObjectInputStream input `[protected]`

ObjectOutputStream class is used for constructing output stream for sending out the message to other agents.

ObjectOutputStream output `[protected]`

ObjectInputStream class is used for constructing input stream for receiving the message from other agents.

•

7. Message Class

7.1 Detailed Description

Message class defines the field used in the message passed back and forth between agents. Note that these fields are derived from the fields in a KQML message, and some of them are automatically filled by the sendMessage method in each type of conversation classes. In agentMom, there is no restriction in using these fields. For more information about KQML please refer to http://www.fipa.org.

When a conversation calls the sendMessage method, it automatically fill the sender, host, and port fields using the parent agent's name and port attributes and automatically gets the host name from the system. The replywith and inreplyto fields are also automatically fill if the sendMessage is called from MulticastConversation, SecureMulticastConversation and BroadcastConversation. The other fields of interest in an agentMom message are the performative and content fields. The performative field describes the action that the message intends and is used in the agent and conversation classes to

(1) Determine the type of conversation being requested and

(2) To control the execution of a conversation in the run method.

Because agentMom does not have any specific performative types, users can define any performative they feel are necessary. The content of an agentMom message is also very general. Basically, the message passes any valid Java object type. This can be as simple as a string, or a more complex object that encapsulates a number of attribute types. These complex objects can be used to pass multiple parameters in a single message as shown in the class below.

```
public class ComplexObject implements Serializable
{
   String agent;
   String host;
   int port;
   String service;
   public ComplexObject(String a, String h, int p, String ser)
   {
     agent = a;
     host = h;
     port = p;
     service = ser;
   }
 }
```

This class encapsulates four parameters (three strings and an integer) that can be assigned to message content field.

Note that in order to pass an object across a socket connection, it must implement the interface Serializable.

Public Member Functions

- **Message** ()
- Object **getContent** ()
- String **getPerformative** ()
- String **getReceiver** ()
- String **getSender** ()
- void **setContent** (Object cont)
- void **setContent** (String cont)
- void **setPerformative** (String perf)

- void **setReceiver** (String name)
- void **setSender** (String name)

Public Attributes

- String **host** = null
- int **port** = 0
- String **sender** = null
- String **receiver** = null
- String **performative** = null
- String **force** = null
- String **inreplyto** = null
- String **language** = null
- String **ontology** = null
- String **replywith** = null
- Object **content** = null

7.2 Constructor

Message ()

Meesage class Constructor. It simply calls the super class object, the java.io.Serializable class

---

7.3 Member Function

Object getContent ()

Return content field in form of the Object.

Returns:

the content field in form of the Object.

String getPerformative ()

Return the performative field in form of the String.

Returns:

the performative field in form of the String.

String getReceiver ()

Return the receiver field in form of the String.

Returns:

the receiver field in form of the String.

String getSender ()

Return the sender field in form of the String.

Returns:

sender field in form of the String.

void setContent (String *cont*)

Set content field of the message in form of the String.

Parameters:

cont - content of the message in form of the String.

void setContent (Object *cont*)

Set content field of the message in form of the Object.

Parameters:

cont - content of the message in form of the Object.

void setPerformative (String *perf*)

Set performative of the message.

Parameters:

perf - performative of the message in form of the String.

void setReceiver (String *name*)

Set the receiver's name.

Parameters:

name - name of the receiver.

void setSender (String *name*)

Set the sender's name.

Parameters:

name - name of the sender.

7.4 Member Data

Object content = null

Support for complex object that encapsulates a number of attribute types. These complex objects can be used to pass multiple parameters in a single message. Note that in order to pass an object across a socket connection, it must implement the interface Serializable.

String force = null

Specify whether the sender will never deny the meaning of the performative.

String host = null

Host name that this message is sent to.

String inreplyto = null

The expected label in a reply.

String language = null

Name of representation language of the content.

String ontology = null

Name of the ontoloty used in the content

String performative = null

Describe the action that the message intends. The user can define any performative they feel are necessary.

int port = 0

Port number used for the message.

String receiver = null

Name of the receiver

String replywith = null

Whether the sender expects a reply, and if so, a label for the reply.

String sender = null

Name of the sender (agent's name).

---

8. MessageHandler Class

8.1 Detailed Description

The **MessageHandler** class is used to handle unicast connection from other agents. It uses the Socket and ServerSocket class to establish a TCP/IP connection stream.

When an agent is created, it also needs to create a new message handler thread to be able to receive a start of unicast conversation from the other agents.

To create the **MessageHandler**, it requires two parameters, port number and a reference to parent agent object. When it is started, the message handler starts a socket server on the indicated port and waits for a connection from another agent. When a connection is received, the **MessageHandler** simply calls the parent agent's receiveMessage method with the connection socket and the input and output streams. The agent then verifies the received message and starts an appropriate conversation.

Public Member Functions

- **MessageHandler** (int port, **Agent** p)
- void **run** ()

Public Attributes

- int **portNo**
- ServerSocket **server**
- **Agent parent**

Protected Member Functions

- void **finalize** () throws Throwable

8.2 Constructor

MessageHandler (int *port*, Agent *p*)

MessageHandler Constructor.

Parameters:

port - Integer port number.
p - Reference to parent agent object that use this **MessageHandler** class.

8.3 Member Function

void run ()

Run method for **MessageHandler** class. It starts by accepting the connection from other agens, and then initialize input and output streams, and pass the connection socket, input and output stream to the parent agent's receiveMessage method. receiveMessage method then verify the input and start an appropriate conversation.

8.4 Member Data

Agent parent

Reference to parent agent object that use this **MessageHandler** class.

int portNo

Port number that the **MessageHandler** listens for unicast conversation.

ServerSocket server

ServerSocket class used for accepting/constructing unicast conversation.

•

9. MomObject Class

Inherits **java.lang.Object**.

9.1 Detailed Description

It is an abstract class that both Agents and Components inherit from. It allows conversations to work with either agents or components as their parents. Class that inherits from this class must implement the sendInternal method.

Public Member Functions

- **MomObject** ()
- **MomObject** (**MomObject** c)
- abstract void **sendInternal** (**Message** m)

Public Attributes

- **MomObject parent**
- String **name**
- int **port**
- int **multicast_port** []
- int **broadcast_port**
- int **secure_unicast_port**
- int **secure_multicast_port** []
- InetAddress **group** []
- InetAddress **broadcast_address**

9.2 Constructor

MomObject ()

Default **MomObject** constructor. This constructor does not require any argument. It simply calls the super class constructor, the java.lang.Object.

MomObject (MomObject *c*)

MomObject constructor. This constructor is used by the Component class so that the conversation classes can work with either agents or components as their parents.

Parameters:

c ksu.cis.MomObject

9.3 Member Function Documentation

abstract void sendInternal (Message *m*) [pure virtual]

Class that inherits from the **MomObject** class must implement the sendInternal method for sending message among component within agent.

Parameters:

m - message for internal conversation.

9.4 Member Data Documentation

InetAddress broadcast_address

Internet address of the broadcast address. Normally, local broadcast address is end with 255 (xxx.xxx.xxx.255). However, broadcast address is not available in all networks, and can be different from the form state previously. Many networks do not permit the broadcast. Please consult the admin about the availability, and the exact address.

int broadcast_port

Port used for broadcast conversation.

InetAddress group[]

Array of InetAddress type used for storing the address of multicast group. Note that array is used to allow subscribing to multiple groups.

int multicast_port[]

Array of port number used for multicast conversation. Note that each port is used for each subscribed multicast group.

String name

Name of the agent.

Reimplemented in **Component** (*p.69*).

int port

Port used for unicast conversation.

int secure_multicast_port[]

Array of port number used for secured multicast conversation. Note that each port is used for each subscribed secured multicast group.

int secure_unicast_port

Port used for secured unicast conversation.

---

10. MulticastConversation Class

Inherits **AgentConversation**.

10.1 Detailed Description

The **MulticastConversation** class is an abstract class that actually carries out the multicastcast message to all agents subscribed to the same group as the sender.

This class provides two main services, read multicast message and send multicast message. It is responsible for passing the messages back and forth using multicast socket.

There are really two types of conversation classes that can be derived from the **MulticastConversation** class, one for the conversation initiator and one for the conversation respondent. The basic difference lies in which constructor is used and the details in the abstract run method, which must be implemented in the concrete class derived from the **MulticastConversation** class.

Public Member Functions

- **MulticastConversation** (**MomObject** c)
- **MulticastConversation** (**MomObject** c, InetAddress group, int port, Vector **multicast_queue**)

- **MulticastConversation** (**MomObject** c, InetAddress group, int port, Vector **multicast_queue**, **Message m**)
- void **sendMessage** (**Message m**, String replywith, String inreplyto) throws IOException
- void **startConversation** (**Message m**, String replywith, String inreplyto) throws IOException
- **Message readMessage** (String **conversation_name**)
- **Message nonblockedReadMessage** (String **conversation_name**, int timeout)
- abstract void **run** ()

Public Attributes

- int **maxSize** = 1024
- MulticastSocket **mSocket**

Static Public Attributes

- final int **START_CONVERSATION** = 0
- final int **CONTENT** = 1

Protected Attributes

- int **TimeToLive**
- Vector **multicast_queue**

Private Attributes

- String **conversation_name**

10.2 Constructor

MulticastConversation (MomObject *c*)

Default **Conversation** constructor. It simply calls the super class constructor, the AgentConversation class

Parameters:

c - Reference to parent **MomObject** class.

MulticastConversation (MomObject *c*, InetAddress *group*, int *port*, Vector *multicast_queue*)

Constructor for conversation initiator.

Parameters:

c - reference to parent class.
group - multicastt address (Class D IP address).
port - multicast port for multicast conversation.
multicast_queue - message queue for mu lticast conversation.

MulticastConversation (MomObject *c*, InetAddress *group*, int *port*, Vector *multicast_queue*, Message *m*)

Constructor for conversation respondent.

Parameters:

c - reference to parent class.
group - multicastt address (Class D IP address) .
port - multicast port for multicast conversation.

multicast_queue - message queue for multicast conversation.
m - ksu.cis.mom.Message

10.3 Member Function Documentation

Message nonblockedReadMessage (String *conversation_name*, int *timeout*)

Method to fetch multicast message from multicast message queue destined for this conversation. This method can be considered a nonblocked read. This method search message destined for specified conversation name by comparing the String parameter with the inreplyto field in the message. If there is no message for specified conversation, it waits for the amount of specified timeout in miilisecond and then try again. If there is still no message on the second try, it returns **Message** with content field "timeout".

Parameters:

conversation_name - name of conversation to retrive message
timeout - read timeout in miilisecond

Returns:

Message with content field "timeout" if there is no message destine for specified conversation.

Message readMessage (String *conversation_name*)

Method to fetch multicast message from multicast message queue destined for this conversation. This method can be considered a blocked read. This method search message destined for specified conversation name by comparing the String parameter with the inreplyto field in the message. If there is no message for specified conversation, it waits for 1000 miilisecond and try again until the message destined for this conversation is found.

Parameters:

conversation_name - name of conversation to retrive message.

Returns:

ksu.cis.mom.Message

abstract void run ()

Run method for **MulticastConversation** class. Each derived conversation must be run as separate thread and must implement this method regarding to the type of conversation, initiator or respondent conversation.

void sendMessage (Message *m*, String *replywith*, String *inreplyto*) throws IOException

This method is responsible for sending the **Message** m through the multicast socket. It also automatically fill the sender, host, port, replywith and inreplyto fields in the **Message** object m using the provided parameters, parent agent's name and port attributes and automatically gets the host name from the system.

It also converts **Message** type Object to byte array to be able to send through datagram socket.

Note that to send message request a start of new conversation the correct method is startConversation.

Parameters:

m - message to send
replywith - name of the originating conversation (this conversation name).
inreplyto - name of the destined conversation on the other sides of agent.

void startConversation (Message *m*, String *replywith*, String *inreplyto*) throws IOException

This method is responsible for sending message request a start of new conversation. The **Message** m is sent through the multicast socket. It also automatically fill the sender, host, port, replywith and inreplyto fields in the **Message** object m using the provided parameters, parent agent's name and port attributes and automatically gets the host name from the system.

It also converts **Message** type Object to byte array to be able to send through multicast socket.

Parameters:

m - message to send
replywith - name of the originating conversation (this conversation name).
inreplyto - name of the destined conversation on the other sides of agent.

10.4 Member Data Documentation

final int CONTENT = 1 [static]

Header to indicate that the message is not a request to start conversation, but a message passing back and forth during the conversation.

String conversation_name [private]

Name of this conversation. This name must be unique from others multicast conversations within agent since it is used for identifying the destined conversation when multicast message is received.

int maxSize = 1024

Default maximum size of received multicast packet (1024 bytes).

MulticastSocket mSocket

Multicast socket used to send and receive multicast message.

Vector multicast_queue [protected]

Message queue for multicast conversation. It is a Vector type that stores multicast message sent to this agent.

final int START_CONVERSATION = 0 [static]

Header to indicate that the message is a request to start conversation.

---

11. MulticastHandler Class

11.1 Detailed Description

MulticastHandler is responsible for initializing and starting multicast socket, including joining/leaving multicast group. In generally, it handles multicast connection with other agents subscribed to the group. It uses the MulticastSocket class to subscribe to multicast group.

When an agent is created, it needs to create a new multicast handler thread to be able to receive multicast messages from other agents. When **MulticastHandler** class is created, it creates the multicast socket and joins to specified multicast group. Then, it automatically sends a multicast message to the group indicating that this agent has join the group. When a **MulticastHandler** receives the message indicating the join, it calls the receiveMulticastJoin method in agent class.

To create the **MessageHandler**, it requires four/five parameters, port number, a reference to parent agent object, time to live of multicast packet, multicast address and/or maximum datagram received packet size.

Note that the multicast address is actually a class D IP addresses that is in the range 224.0.0.0 to 239.255.255.255.

When this class is started, the multicast handler starts a multicast socket on the indicated port and waits for messages from other agents. When multicast handler receives a message, it will check whether the message is a start of new conversation/join/leave/conversation message. If it is a start of conversation, the **MulticastHandler** simply calls the parent agent's receiveMulticastcastConversation method with the multicast socket, received message and multicast message queue. The agent then verifies the received message and starts an appropriate conversation. If the received message is for any multicast conversation class, it adds the message to the multicast message queue. Then the multicast conversation class can get the message from this queue later.

Note that the multicast communication has loopback effect(message also send to itself), but the **MulticastHandler** will ignore this message.

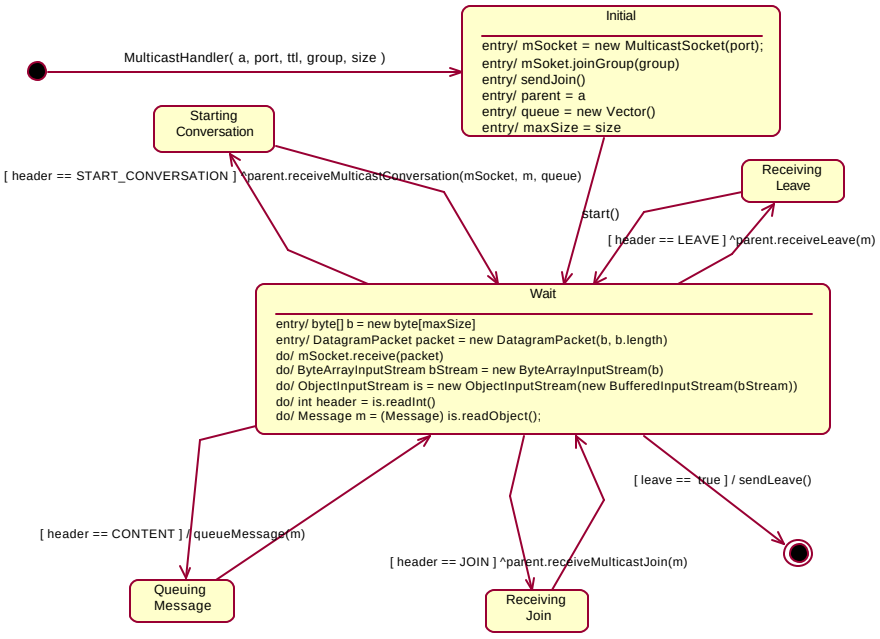

Figure 18 State chart: MulticastHandler

From the state chart above, the MulticastHandler initializes its attributes, subscribes to the multicast group and send the join message to the group when the constructor is called. When the thread starts, it waits for the multicast message to arrive. From the wait state, the state is changed according to the received message header. For example, if the header is CONTENT, the queueMessage method is called. If the header is JOIN, the parent agent's receiveMulticastJoin method is called. Finally, this class goes to the end state when the agent explicitly set the leave to true. It calls the sendLeave method and then goes to exit state.

Public Member Functions

- **MulticastHandler** (**Agent** p, int port, int ttl, InetAddress **group**)
- **MulticastHandler** (**Agent** p, int port, int ttl, InetAddress **group**, int packetSize)
- void **sendJoin** () throws IOException
- void **sendLeave** () throws IOException
- void **run** ()
- void **write** (String s)
- void **queueMessage** (**Message** m)
- void **setLeave** (**boolean** leave)

Public Attributes

- int **maxSize** = 1024
- **Agent parent**
- int **TimeToLive**
- MulticastSocket **mSocket**
- int **multicast_port**
- Vector **multicast_queue**

Static Public Attributes

- final int **START_CONVERSATION** = 0
- final int **CONTENT** = 1
- final int **JOIN** = 2
- final int **LEAVE** = 3

Protected Attributes

- InetAddress **group**

11.2 Constructor & Destructor Documentation

MulticastHandler (Agent *p*, int *port*, int *ttl*, InetAddress *group*)

Default constructor for **MulticastHandler**. It is responsible for initializing necessary variables and creates multicast socket.

Parameters:

p - reference to parent agent class.
port - port number for listening for multicast message.
ttl - time to live of datagram packet to control the scope of multicast packet.
group - multicast address, in the range 224.0.0.0-239.255.255.255

MulticastHandler (Agent *p*, int *port*, int *ttl*, InetAddress *group*, int *packetSize*)

Default constructor for **MulticastHandler**. It is responsible for initializing necessary variables and creates multicast socket. This constructor allows specifying the size of received multicast packet.

Parameters:

p - reference to parent agent class.
port - port number for listening for multicast message.
ttl - time to live of datagram packet to control the scope of multicast packet.
group - multicast address, in the range 224.0.0.0-239.255.255.255
packetSize - user defined maximum size in byte of received multicast packet.

11.3 Member Function

void queueMessage (Message *m*)

Method for adding new message to the queue that the multicast conversation will fetch the message later .

Parameters:

m - message to be retrived by multicast conversation.

void run ()

Run thread method for MulticastHandler. It simply waits for incoming multicast message from other agents. Then, it checks the message whether it is a start of new multicast conversation, a join group message, a leave group message, or a message for multicast conversation class.

Note that this is a blocking read message.

void sendJoin () throws IOException

Sends a multicast JOIN message to notify other agents in the same multicast group. We need to convert **Message** object to Byte type to be able to send it using multicast socket.

void sendLeave () throws IOException

Sends a multicast LEAVE message to notify other agents in the same multicast group. We need to convert **Message** object to Byte type to be able to send it using multicast socket.

void write (String *s*)

Method for easily printout information on screen.

Parameters:

s java.lang.String

11.4 Member Data Documentation

final int CONTENT = 1 [static]

Header to indicate that the message is not a request to start conversation, but a message passing back and forth during the multicast conversation.

InetAddress group [protected]

multicast group address.

final int JOIN = 2 [static]

Identifies a JOIN multicast message.

final int LEAVE = 3 [static]

Identifies a LEAVE multicast message.

int maxSize = 1024

Default maximum size of received multicast packet (1024 bytes).

MulticastSocket mSocket

Multicast socket used to send and receive multicast message

int multicast_port

Multicast port

Vector multicast_queue

Message queue for multicast conversation. It is a Vector type that stores multicast message sent to this agent.

Agent parent

Reference to parent agent class.

final int START_CONVERSATION = 0 [static]

Header to indicate that the message is a request to start conversation.

int TimeToLive

Time-to-live for multicast packets sent out on this MulticastSocket. It is used to control the scope of the multicasts message.

•

12. SecureMulticastConversation Class

Inherits **AgentConversation**.

12.1 Detailed Description

The **SecureMulticastConversation** class is an abstract class that actually carries out the secured multicast message to all agents subscribed to the same group as the sender.

This class provides two main services, read secured multicast message and send secured multicast message. It is responsible for passing the messages back and forth using multicast socket.

This class uses the symmetric key algorithm to encrypt the message. Thus, the algorithm and private key must be the same at the sender and receiver side.

There are really two types of conversation classes that can be derived from the **SecureMulticastConversation** class, one for the conversation initiator and one for the conversation respondent. The basic difference lies in which constructor is used and the details in the abstract run method, which must be implemented in the concrete class derived from the **SecureMulticastConversation** class.

Public Member Functions

- **SecureMulticastConversation** (**MomObject** c)
- **SecureMulticastConversation** (**MomObject** c, InetAddress group, int port, Vector **secure_multicast_queue**, Key **key**, String **algorithm**)
- **SecureMulticastConversation** (**MomObject** c, InetAddress group, int port, Vector **secure_multicast_queue**, **Message m**, Key **key**, String **algorithm**)
- void **sendMessage** (**Message m**, String replywith, String inreplyto) throws IOException
- void **startConversation** (**Message m**, String replywith, String inreplyto) throws IOException
- **Message readMessage** (String **conversation_name**)
- **Message nonblockedReadMessage** (String **conversation_name**, long timeout)
- abstract void **run** ()

Public Attributes

- int **maxSize** = 1024
- MulticastSocket **mSocket**

Static Public Attributes

- final int **START_CONVERSATION** = 0
- final int **CONTENT** = 1

Protected Attributes

- Vector **secure_multicast_queue**
- Key **key** = null
- String **algorithm** = null
- Cipher **cipher** = null

Private Attributes

- String **conversation_name**

12.2 Constructor

SecureMulticastConversation (MomObject *c*)

Default **SecureMulticastConversation** constructor. It simply calls the super class constructor.

Parameters:

c - Reference to parent **MomObject** class.

SecureMulticastConversation (MomObject *c*, InetAddress *group*, int *port*, Vector *secure_multicast_queue*, Key *key*, String *algorithm*)

Constructor for conversation initiator.

Parameters:

c - reference to parent class.
group - multicast address (Class D IP address) .
port - secured multicast port.
secure_multicast_queue - message queue for secured multicast conversation.
key - symmetric private key
algorithm - Symmetric key algorithm.

SecureMulticastConversation (MomObject *c*, InetAddress *group*, int *port*, Vector *secure_multicast_queue*, Message *m*, Key *key*, String *algorithm*)

Constructor for conversation respondent.

Parameters:

c - reference to parent class.
group - multicast address (Class D IP address).
port - secured multicast port.
secure_multicast_queue - message queue for secured multicast conversation.
m - **ksu.cis.mom.Message**
key - symmetric private key.
algorithm - symmetric key algorithm.

12.3 Member Function

Message nonblockedReadMessage (String *conversation_name*, long *timeout*)

Method to fetch multicast message from secured multicast message queue destined for this conversation. This method can be considered a nonblocked read. This method search message destined for specified conversation name by comparing the String parameter with the inreplyto field in the message. If there is no message for specified conversation, it wait for the amount of specified timeout in millisecond and then try again. If there is still no message on the second try, it returns **Message** with content field "timeout".

Parameters:

conversation_name - name of conversation to retrieve message.
timeout - read timeout in millisecond.

Returns:

Message with content field "timeout" if there is no message destine for specified conversation.

Message readMessage (String *conversation_name*)

Method to fetch secured multicast message from secured multicast message queue destined for this conversation. This method can be considered a blocked read. This method search message destined for specified conversation name by comparing the String parameter with the inreplyto field in the message. If there is no message for specified conversation, it waits for 1000 millisecond and try again until the message for this conversation is found.

Parameters:

conversation_name - name of conversation to get the message

Returns:

ksu.cis.mom.Message

abstract void run ()

Run method for **SecureMulticastConversation** class. Each derived conversation must be run as separate thread and must implement this method regarding to the type of conversation, initiator or respondent conversation.

void sendMessage (Message *m*, String *replywith*, String *inreplyto*) throws IOException

This method is responsible for sending the **Message** m through the multicast socket. It also automatically fill the sender, host, port, replywith and inreplyto fields in the **Message** object m using the provided parameters, parent agent's name and port attributes and automatically gets the host name from the system. Furthermore, the message will be encrypted using the specified symmetric key algorithm.

We need to convert **Message** object to Byte type, and then encrypts the byte information to be able to send encrypted message via multicast socket.

Note that to send message request a start of new conversation the correct method is startConversation.

Parameters:

m - message to send
replywith - name of the originating conversation (this conversation).
inreplyto - name of the destined conversation on the other sides of agent.

void startConversation (Message *m*, String *replywith*, String *inreplyto*) throws IOException

This method is responsible for sending message request a start of new conversation. The **Message** m is encrypted and then sent through the multicast socket.

It also automatically fill the sender, host, port, replywith and inreplyto fields in the **Message** object m using the provided parameters, parent agent's name and port attributes and automatically gets the host name from the system. Furthermore, the message will be encrypted using the specified symmetric key algorithm.

We need to convert **Message** object to Byte type, and then encrypts the byte information to be able to send encrypted message via multicast socket.

Parameters:

m - message to send
replywith - name of the originating conversation (this conversation).
inreplyto - name of the destined conversation on the other sides of agent.

12.4 Member Data Documentation

String algorithm = null `[protected]`

Key class to store the secret key for encryption and decryption.

Cipher cipher = null `[protected]`

Cipher class to perform encryption and decryption.

final int CONTENT = 1 [static]

Header to indicate that the message is not a request to start conversation, but a message passing back and forth during the multicast conversation.

String conversation_name [private]

Name of the conversation. This name must be unique from others secured multicast conversations within agent since it is used for identifying itself when sending message and fetching message from the queue.

Key key = null [protected]

Key class to store the secret key for encryption and decryption.

int maxSize = 1024

Default maximum size of received multicast packet (1024 bytes).

MulticastSocket mSocket

Multicast socket used to send and receive secured multicast message.

Vector secure_multicast_queue [protected]

Message queue for multicast conversation. It is a Vector type that stores secured multicast message sent to this agent.

final int START_CONVERSATION = 0 [static]

Header to indicate that the message is a request to start conversation.

---

13 SecureMulticastHandler Class

13.1 Detailed Description

SecureMulticastHandler is responsible for initializing and starting multicast socket, including joining/leaving multicast group. It also performs message encryption and decryption. In generally, it handles secured multicast connection with other agents. It uses the MulticastSocket class to subscribe to multicast group, and uses symmetric key algorithm to perform encryption and decryption.

When an agent is created, it needs to create a new secured multicast handler thread to be able to receive secured multicast messages from other agents. When **SecureMulticastHandler** class is started, it creates the multicast socket and join to specified multicast group. Then, it automatically sends a multicast message to the group indicating that this agent has join the group. When a **MulticastHandler** receives the message indicating the join, it calls the receiveMulticastJoin method in agent class.

To create the **SecureMulticastHandler**, it requires six/seven parameters, port number, a reference to parent agent object, time to live of multicast packet, multicast address, private key for encryption and decryption, an algorithm to perform encryption and decryption and/or maximum datagram received packet size.

Note that the multicast address is actually a class D IP addresses that is in the range 224.0.0.0 to 239.255.255.255.

When this class receives a message from other agents, it will decrypt the message and check whether the message is a start of new conversation/join message/leave message/conversation message. If it is a start of new conversation, the **SecureMulticastHandler** simply calls the parent agent's receiveSecureMulticastcastConversation method with the multicast socket, received message, multicast message queue, private key and the algorithm for encryption and decryption. The agent then verifies the received message and starts an appropriate conversation. If the received message is for any secured multicast conversation class, it adds the message to the message queue. Then the multicast conversation class can get the message from this queue later. If the message is a join or a leave message, it calls the parent agent's receiveMulticastJoin or receiveMulticastLeave.

Note that we use symmetric key algorithm, so it is necessary to keep the key and algorithm consistent among agent in the group.

Public Member Functions

- **SecureMulticastHandler** (**Agent** p, int port, int ttl, InetAddress **group**, Key k, String algorithm)
- **SecureMulticastHandler** (**Agent** p, int port, int ttl, InetAddress **group**, int packetSize, Key k, String algorithm)
- void **sendJoin** () throws IOException
- void **sendLeave** () throws IOException
- void **run** ()
- void **write** (String s)
- void **queueMessage** (**Message** m)
- void **setLeave** (**boolean** leave)

Public Attributes

- int **maxSize** = 1024
- **Agent parent**
- int **TimeToLive**
- MulticastSocket **mSocket**
- int **secure_multicast_port**
- Vector **secure_multicast_queue**
- boolean **leave**

Static Public Attributes

- final int **START_CONVERSATION** = 0
- final int **CONTENT** = 1
- final int **JOIN** = 2
- final int **LEAVE** = 3

Protected Attributes

- Key **key** = null
- Cipher **cipher** = null

Private Attributes

- InetAddress **group**

13.2 Constructor

SecureMulticastHandler (Agent *p*, int *port*, int *ttl*, InetAddress *group*, Key *k*, String *algorithm*)

Constructor for **SecureMulticastHandler**. It is responsible for initializing necessary variables, creates multicast socket.

Parameters:

p - reference to parent agent class.
port - port number for listening for secured multicast message.
ttl - time to live of datagram packet.
group - multicast address, in the range 224.0.0.0-239.255.255.255
k - secret key.
algorithm - Symmetric key algorithm.

SecureMulticastHandler (Agent *p*, int *port*, int *ttl*, InetAddress *group*, int *packetSize*, Key *k*, String *algorithm*)

Default constructor for **SecureMulticastHandler**. It is responsible for initializing necessary variables, creates multicast socket. This constructor allows to specify the maximum size of received secured multicast packet.

Parameters:

p - reference to parent agent class.
port - port number for listening for secured multicast message.
ttl - time to live of datagram packet.
group - multicast address, in the range 224.0.0.0-239.255.255.255
k - secret key.
algorithm - Symmetric key algorithm.

13.3 Member Function Documentation

void queueMessage (Message *m*)

Method for adding new message to the queue.

Parameters:

m - message to be retrived by multicast conversation.

void run ()

Run thread method for SecureMulticastHandler. It simply waits for incoming encrypted multicast message from other agents. Then, it decrypts message and checks the message whether it is a start of new multicast conversation, join message, leave message or a message for multicast conversation class.

void sendJoin () throws IOException

Sends a multicast JOIN message to notify other agents. We need to convert **Message** object to Byte type, and then encrypts the byte information to be able to send encrypted message via multicast socket.

void sendLeave () throws IOException

Sends a multicast LEAVE message to notify other agents. We need to convert **Message** object to Byte type, and then encrypts the byte information to be able to send encrypted message via multicast socket.

void write (String *s*)

Method for easily printout information on screen.

Parameters:

s java.lang.String

13.4 Member Data Documentation

Cipher cipher = null [protected]

Cipher class to perform encryption and decryption.

final int CONTENT = 1 [static]

Header to indicate that the message is not a request to start conversation.

InetAddress group [private]

Multicast address

final int JOIN = 2 [static]

Identifies a JOIN multicast message

Key key = null [protected]

Key class to store the secret key for encryption and decryption.

final int LEAVE = 3 [static]

Identifies a LEAVE multicast message

int maxSize = 1024

Default maximum size of received multicast packet (1024 bytes).

MulticastSocket mSocket

Multicast socket used to send and receive secured multicast message

Agent parent

Reference to parent agent class.

int secure_multicast_port

Secured multicast port

Vector secure_multicast_queue

Message queue for secured multicast conversation. It is a Vector type that stores multicast message sent to this agent.

final int START_CONVERSATION = 0 [static]

Header to indicate that the message is a request to start conversation.

int TimeToLive

Time-to-live for multicast packets sent out on this MulticastSocket in order to control the scope of the multicasts.

---

14. SecureUnicastConversation Class Reference

Inherits **AgentConversation**.

14.1 Detailed Description

The **SecureUnicastConversation** class is an abstract class that actually carries out the secured unicast message passing between agents. This class provides two main services, read secured message and send secured message using Secure Socket Layers (SSL) technology provided in java 1.4. It is responsible for passing the messages back and forth over the secure socket layers connection.

SSL uses many cryptography technologies together such as public key, private key, session key, authentication, digital signature, etc. These are transparent to the user of SSL technology. Basically, SSLSocket and SSLServerSocket can be used almost the same way as Socket and ServerSocket class. However, the "keystore", "trustore" and "certificate" must be generated on both sides of communications. Also, each side of communication must have "certificate" of the other side installed. For example, the tool "keytool", provided in java version 1.4 packages, can be used to generate these requirements.

There are really two types of conversation classes that can be derived from the SecuredUnicastConversation class, one for the conversation initiator and one for the conversation respondent. The basic difference lies in which constructor is used and the details in the abstract run method, which must be implemented in the concrete class derived from the **SecureUnicastConversation** class.

Public Member Functions

- **SecureUnicastConversation** (**MomObject** c)
- **SecureUnicastConversation** (**MomObject** c, String hostName, int su_port)
- **SecureUnicastConversation** (SSLSocket sslSocket, ObjectInputStream **input**, ObjectOutputStream **output**, **MomObject** c, **Message m**)
- **Message nonblockedReadMessage** (ObjectInputStream **input**)
- **Message readMessage** (ObjectInputStream **input**)
- abstract void **run** ()
- void **sendMessage** (**Message m**, ObjectOutputStream **output**)

Public Attributes

- SSLSocket **connection**

Protected Attributes

- ObjectInputStream **input**
- ObjectOutputStream **output**

14.2 Constructor

SecureUnicastConversation (MomObject *c*)

Default **Conversation** constructor. It simply call a super class constructor.

Parameters:

c - Reference to parent **MomObject** class.

SecureUnicastConversation (MomObject *c*, String *hostName*, int *su_port*)

Constructor for conversation initiator.

Parameters:

c - Reference to parent **MomObject** class.
hostName - host name that this conversation is connected to.
su_port - port number of the host that this conversation is connected to.

SecureUnicastConversation (SSLSocket *sslSocket*, ObjectInputStream *input*, ObjectOutputStream *output*, MomObject *c*, Message *m*)

Constructor for conversation respondent.

Parameters:

sslSocket - javax.net.ssl.SSLSocket
input - java.io.ObjectInputStream
output - java.io.ObjectOutputStream
c - **ksu.cis.mom.MomObject**
m - ksu.cis.mom.Message

14.3 Member Function Documentation

Message nonblockedReadMessage (ObjectInputStream *input*)

This read method allows timeout nonblocking read message. It allows the read message to "timeout" thus allowing the conversation to check to see if it has a message without waiting forever. The default value of timeout is 100 milliseconds.

Parameters:

input java.io.ObjectInputStream

Returns:

ksu.cis.mom.Message

Message readMessage (ObjectInputStream *input*)

This read method is blocking read message. It waits until the message is arrived.

Parameters:

input java.io.ObjectInputStream

Returns:

ksu.cis.mom.Message

abstract void run ()

run method for conversation class. Each derived conversation must be run as separate thread and must implement this method regarding to the type of conversation, initiator or respondent conversation.

void sendMessage (Message *m*, ObjectOutputStream *output*)

This method is responsible for sending the **Message** m through the ObjectOutputStream output using SSLSocket class. It also automatically fills the sender, host, and port fields in the **Message** object m using the parent agent's name and port attributes and automatically gets the host name from the system.

Parameters:

m **ksu.cis.mom.Message**
output java.io.ObjectOutputStream

14.4 Member Data Documentation

SSLSocket connection

SSLSocket class is used for connecting secured communication with other agents.

ObjectInputStream input `[protected]`

ObjectOutputStream class is used for constructing output stream for sending out the message to other agents.

ObjectOutputStream output `[protected]`

ObjectInputStream class is used for constructing input stream for receiving the message from other agents.

---

15 SecureUnicastHandler Class

15.1 Detailed Description

The **SecureUnicastHandler** class is used to handle secured unicast connection from other agents. It uses the Secure Socket Layers (SSLSocket class provided in java version 1.4) and Secure Socket Layers Server Socket (SSLServerSocket class provided in java version 1.4) to establish a secured connection over TCP/IP stream.

SSL uses many cryptography technologies together such as public key, private key, session key, authentication, digital signature, etc. These are transparent to the user of SSL technology. Basically, SSLSocket and SSLServerSocket can be used almost the same way as Socket and ServerSocket class. However, the "keystore", "trustore" and "certificate" must be generated on both sides of communications. Also, each side of communication must have "certificate" of the other side installed. For example, the tool "keytool", provided in java version 1.4 packages, can be used to generate these requirements.

When an agent is created, it can create a new secured unicast handler thread to be able to receive a start of secured unicast conversation from the other agents.

To create the **SecureUnicastHandler**, it requires two parameters, port number and a reference to parent agent object. When it is started, the secured unicast handler starts a ssl socket server on the indicated port and waits for a connection from another agent. When a connection is received, the **SecureUnicastHandler** simply accepts and calls the parent agent's receiveMessage method with the connection ssl socket and the input and output streams. The agent then verifies the received message and starts an appropriate conversation.

Public Member Functions

- **SecureUnicastHandler** (int port, **Agent** p)
- void **run** ()
- void **setAlive** (boolean **alive**)

Public Attributes

- int **portNo**
- SSLServerSocket **sslServer**
- **Agent parent**

15.2 Constructor

SecureUnicastHandler (int *port*, Agent *p*)

SecureUnicastHandler Constructor

Parameters:

port - Integer port number
p - Reference to parent agent object that use this class.

15.3 Member Function Documentation

void run ()

Run method for **SecureUnicastHandler**. This method waits for a connection starting a new secured unicast conversation from other agents. SSL server simply accepts the connection. Then, initialize input and output streams, and call the parent agent's receiveSecureUnicastConversation method. Finally, the connection, input stream and output stream is passed to the parent agent's receiveMessage message.

15.4 Member Data Documentation

Agent parent

Reference to parent agent object that use this **SecureUnicastHandler** class.

int portNo

Port number that the **SecureUnicastHandler** listens for secured unicast conversation.

SSLServerSocket sslServer

SSLServerSocket class used for accepting/constructing secured unicast conversation.

---

16. Sorry Class

Inherits **Conversation**.

16.1 Detailed Description

The **Sorry** class defines a general purpose conversation to reply "Sorry" to any unknown/unexpected type of unicast conversation. It is a simply concrete class of Conversation class, so there is no implementation required in this class. Automatically, performative field is set to "sorry" and content field is set to "unknown conversation request" when using this class.

Public Member Functions

- **Sorry** (Socket s, ObjectInputStream i, ObjectOutputStream o, **MomObject** c, **Message** m1)
- void **run** ()

16.2 Constructor

Sorry (Socket *s*, ObjectInputStream *i*, ObjectOutputStream *o*, MomObject *c*, Message *m1*)

Sorry class constructor.

Parameters:

s socket
i ObjectInputStream
o ObjectOutputStream
c **MomObject**
m1 Message

16.3 Member Function Documentation

void run ()

Run method for the **Sorry** class. It simply reply message with performative "sorry" and content "unknown conversation request".