Duncan Smith
CSS497 Summer/Autumn Quarter 2005
Redesigning and Enhancing the UWAgent Execution Engine
December 9, 2005

Duncan Smith
CSS497 Summer/Autumn Quarter 2005
Redesigning and Enhancing the UWAgent Execution Engine
December 9, 2005

Final Report

Summary

This report describes changes and enhancements to the UWAgent execution engine implemented during Summer and Autumn Quarters 2005 at UW Bothell. It includes information for users who want to implement systems using UWAgent, as well as technical details for developers making changes to UWAgent.

The following changes and enhancements were made from June-December 2005:

- **Socket redesign**: Replaced Java RMI (Remote Method Invocation) with Java sockets for agent navigation and communication.
- **Navigation and Communication over Gateways**: A new feature that allows agents to navigate and communicate back and forth between a public and a private network, using one or more gateway machines.
- **Monitor Commands**: Commands that allow management of running agents.
- **Secure Communication**: SSL support for all socket communication.
- **Class Name Collision Testing**: Testing of the condition in which two identically-named classes are used at the same UWPlace.
- **Logging**: Methods to centralize debugging and error-reporting code.
- **Code cleanup**: Refactoring of the existing UWAgent code for consistency and ease of maintenance. In general, code that is only used by AgentTeamwork should appear in that system's code base rather than the UWAgent system's code base.

The UWAgent source code with these enhancements is located in **/home/uwagent/MA/UWAgent.new** (in the **uwagent** account's home directory) on the medusa file system. This document is located in **/home/uwagent/doc/reports/FinalReportDS.doc**.

Socket Redesign

Description

The UWAgent system often requires that methods on one node be called by code running on another node. RMI (Remote Method Invocation) is a standard Java technology used for distributed systems, and it was used in the previous version of the UWAgent system. However, as a result of several issues with RMI, we decided to develop a custom remote method calling solution instead of continuing to use RMI. The problems with RMI are: 1) Users must run the rmiregistry command at every UWPlace, and kill it manually when they are done; 2) RMI introduces a communication layer that must be configured properly to avoid connection errors; 3) A problem can occur when an RMI client is on a gateway and its corresponding RMI server is on a machine in a private network. In cases where the server needs to establish a reverse connection to its client, the client may provide it with a public IP address. Since the server is on a private network, it cannot use the public address.

In order to implement remote method calls without using RMI, the UWAgent system uses a custom socket interface. The implementation of this socket interface is described next.

Implementation

Client Socket

When the UWAgent system needs to call a method on another node, it uses UWUtility.InitUWPSocket. For example, UWPlace.sendAgent is used to transfer a UWAgent from one UWPlace to another. It contains the following call:

```
OutputStream out =
        UWUtility.InitUWPSocket(UWUtility.MSG_TYPE_FUNC, hostName,
        getPortNumber(), "receiveAgent", byteArrayObj.length,
        classesHash.size(), getIsSSL());
```

Here is an explanation of this method call:

- **OutputStream out**: The OutputStream returned by InitUWPSocket is used to send additional data through the socket. The initial InitUWPSocket call sends a fixed-length header, but most remote method calls require that additional information be sent. For example, sendAgent needs to send the serialized agent being transferred to the remote host.
- **UWUtility.InitUWPSocket**: The UWUtility class contains static methods used by several classes in the UWAgent system.
- **UWUtility.MSG_TYPE_FUNC**: This indicates that the message being sent over the socket is of type “function call.” This is currently the only message type supported by InitUWPSocket, but this parameter could be used in the future to distinguish between function calls and other message types.
- **hostName**: This is the IP name of the host that InitUWPSocket will attempt to connect to. A server socket must be listening on the destination host.
- **getPortNumber()**: This UWPlace method returns the port number that the user has specified using the -p command-line switch, or the default port. This assumes that the remote host is using the same port: If the locally-specified port number is different from the port number that was specified when UWPlace was started on the remote host, then the socket connection will fail.
- **“receiveAgent”**: This string identifies the method to be called on the remote host.
- **byteArrayObj.length** and **classesHash.size()**: These two InitUWPSocket parameters are integers that are used for different purposes depending on the method being called. In this case, they store information about the amount of data that the destination host should expect to receive through the socket.
- **getIsSSL()**: This UWPlace method returns *true* if the user has requested secure communication (using the -e switch on the command line).

UWUtility.InitUWPSocket does the following:

- Instantiate a socket and connect to the specified port on the specified host.
- Get an output stream from the socket.

- Write the fixed-length header to the output stream.
- Return the output stream so that the caller can write additional data.

Server Socket

The connection from InitUWPSocket is accepted by a server socket in the SocketThread class, which is defined in UWPlace.java. After the server and client sockets are connected, the thread does the following:

- Read and parse the fixed-size header.
- Based on the function name string, interpret the header data and read the specified amount of additional data from the socket.
- Close the socket connection.
- Call a local method using the data received on the socket.

Developer Note:

The readBytes method in SocketThread is a potential source of bugs, and should be changed carefully if at all. It currently works as follows:

- Accept an input stream and a length (len) in bytes.
- Attempt to read len bytes from the input stream, and store the number of bytes actually read.
- Continue to read from the input stream until the total number of bytes actually read matches the number of bytes expected.
- Return a byte array containing the data from the input stream.

This algorithm depends on two essential parts: knowing how many bytes are being sent from the client, and accurately keeping track of how many bytes were actually received over the socket. Bugs can occur as a result of problems with either of these parts. For example, if the SocketThread attempts to read too many bytes, it may block and never call the specified method. Since UWAgent is multithreaded, this blocking behavior may not be obvious, as other threads may continue to execute. If the thread reads too few bytes, it probably ends up with incomplete serialized objects. If the server thread gets out of sync with the client thread because of these problems, and the header is therefore not decoded correctly, SocketThread may log the "SocketThread: Unknown method name" error.

Navigation and Communication over Gateways

Description

The UWAgent system allows agents to travel between two separate networks if those networks are linked by one or more gateway machines running UWPlace. UWAgents on one network can then send messages to UWAgents on the other network. In addition to running UWPlace on the gateway machines, the user must specify the IP names of the machines in either of two ways: 1) On the UWPlace command line with the -g switch, in which case only one gateway machine may be specified, and 2) as a parameter to the hop method, which accepts a string array of machine names. For example, consider the following example of the command-line option:

```
[uwagent@mnode8 UWAgent.new]$ java UWPlace -g medusa &
[1] 7948
[uwagent@mnode8 UWAgent.new]$ java UWInject localhost GatewayMessageTest uw1-320-20
```

This pair of statements indicates that a gateway called *medusa* is available to allow agent GatewayMessageTest to travel from the local host, mnode8, to the destination, uw1-320-20.

Now consider this example, which uses the gateway version of the hop method:

```
String[] gateway = new String[1];
gateway[0] = "medusa";
hop("mnode8", gateway, "ArrivalMethod", null);
```

The agent running this code fragment will hop from its current location to a host called *mnode8*, using one gateway called *medusa*. When it arrives at mnode8, it will call a method called *ArrivalMethod*, with no arguments.

Implementation

The main idea of navigation and communication over gateways is that the UWAgent or UWMessage must take one or more *extra* hops between the starting and ending UWPlace. Instead of jumping directly from a private network to a public network or vice versa, the UWAgent travels through a gateway machine. The gateway machine is simply another machine running UWPlace. The only difference is that it is able to see both the public and private networks.

Here are the implementation details, by class:

UWAgent class

The UWAgent class stores gateway information in member variables. As the UWAgent hops from the source to the destination machine, this gateway information travels along with it. This means that information about the path from the source to the destination (and back) is always available.

- String[] destGateway and int destGatewayPos member variables store the gateway list and the position in the list. The message passing algorithm uses this information to transfer the UWMessage from the sender agent, across the gateway list, and to the recipient agent.
- Hashtable agentGatewayDirectory stores the gateways required to reach each agent. The hash key is the agent ID, and the hash value is the gateway array. This directory is used along with the agentIpDirectory, which stores the agent IP address, to allow UWMessages to find any agent given its agent ID.
- registerAgentIp()accepts String[] gateways and int gwPos: the gateway list and the current position in the list. The position is used while the gateway list is being traversed, to keep track of which gateway needs to be traversed next, and when the message has reached the end of the gateway list.
- notifyRelativesOfMyIp() accepts destGateway, which stores the list of gateways that allows the notification message to get back to the parent agent. When an agent hops to a new UWPlace across a gateway, it uses notifyRelativesOfMyIp() to tell its parent and

children both its current location (IP address), and the list of gateways required to reach this location.

SocketThread class (in UWPlace.java)

- registerAgentIp accepts the gateway array and position in the gateway list. This information is used when agents are contacted using talk().

While registerAgentIp is used for both gateway and non-gateway transfers, the following two methods are used only when gateways are involved:

- registerAgentIpGateway is used to pass the agent location information through the list of gateways. This method calls itself until it reaches the end of the gateway list, at which point it calls registerAgentIp.
- enqueueMessageGateway is used to pass a message through the list of gateways. In a similar way to registerAgentIpGateway, it calls itself until it reaches the end of the gateway list, at which point it calls enqueueMessage.

UWAgentMailbox

- notifyAgentLocation() accepts String[] gateway, which it uses to call registerAgentIpGateway() at the remote UWPlace, using the socket interface.
- sendMessage() calls enqueueMessageGateway() to pass the message through a gateway list to the recipient.

Monitor Commands

Description

UWAgent supports the following commands that allow management of running agents:

- as (Agent Status): report the status (Ready, Running, or Suspended) of each agent at the local place
- kill: terminate an agent
- suspend: pause an agent's execution
- resume: cause an agent that has been suspended to continue executing

Implementation

The monitor commands need to operate on or retrieve information about agents running on the local UWPlace. In order to affect and query agents at a UWPlace, the monitor needs to access the private agentsList ArrayList in UWPlace. This is done using two classes:

- The UWMonitor class accepts command-line instructions, in a similar way to UWInject.
- The UWMonitorAgent class is a UWAgent that is automatically injected to the local UWPlace by UWMontior.

Because UWMonitorAgent is a UWAgent, it can call methods provided by the local UWPlace. The following UWPlace methods are used:

  - agentStatus: For each agent in the local agentsList, print getAgentId(), getName(), and a status based on the values of the m_isSuspended and m_isRunning boolean member variables. These variables are set in the AgentThread class.

- killAgentNoGroup: Call Thread.interrupt() followed by Thread.stop() on the specified agent's thread. Another method, killAgent, is available if a program needs to kill an entire thread group.
- suspendAgent: Call Thread.suspend() on the specified agent's thread.
- resumeAgent: Call Thread.resume() on the specified agent's thread.

Secure Communication

Description

The UWAgent system implements secure communication by replacing Socket and ServerSocket with SSLSocket and SSLServerSocket. Since all network communication in the UWAgent system takes place over its custom socket interface, using secure sockets ensures that both UWAgent communication (passing of UWMessages) as well as information in commands such as hop() is protected. Note that these security features only protect messages sent over the UWAgent socket interface. Other aspects of the UWAgent system rely on external security providers. For example, Secure Shell (SSH) should be used to protect the shell commands used to interact with the system.

Both UWPlace and UWInject accept a command-line parameter, -e (for "encrypted"), indicating that secure sockets should be used. If encryption is specified at a UWPlace, it must be specified in the UWInject command used to inject agents to that place. In addition, UWPlaces that interact with a secure UWPlace (e.g., by sending or receiving agents) must all be started with the -e flag.

Before the UWAgent security features can be used, a certificate must be generated using the Java `keytool` command. Here is an example:

```
[uwagent@mnode8 SSL]$ keytool -genkey -keystore UWAgentKeystore -keyalg RSA
Enter keystore password:  <enter a strong password>
What is your first and last name?
  [Unknown]:  Duncan Smith
What is the name of your organizational unit?
  [Unknown]:  CSS
What is the name of your organization?
  [Unknown]:  UW Bothell
What is the name of your City or Locality?
  [Unknown]:  Bothell
What is the name of your State or Province?
  [Unknown]:  WA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Duncan Smith, OU=CSS, O=UW Bothell, L=Bothell, ST=WA, C=US correct?
  [no]:  y

Enter key password for <mykey>
        (RETURN if same as keystore password):
```

When the -e switch is used, the keystore file must be in the working directory, and the file name and password must be included on the command-line, as follows. The -D switch to the Java VM is used to set the system properties that are required for SSL.

```
java -Djavax.net.ssl.keyStore=UWAgentKeystore -
Djavax.net.ssl.keyStorePassword=<password> -Djavax.net.ssl.trustStore=UWAgentKeystore
-Djavax.net.ssl.trustStorePassword=<password> UWPlace -e
```

```
java -Djavax.net.ssl.keyStore=UWAgentKeystore -
Djavax.net.ssl.keyStorePassword=<password> -Djavax.net.ssl.trustStore=UWAgentKeystore
-Djavax.net.ssl.trustStorePassword=<password> UWInject localhost <AgentName> -e
```

Implementation

- UWInject
    - Accept a new command-line parameter, -e (for "encrypted"), indicating that secure sockets should be used.
- UWPlace
    - Accept a new command-line parameter, -e (for "encrypted"), indicating that secure sockets should be used.
    - Create either a ServerSocket or an SSLServerSocket:

      ```
      ServerSocket srvr = null;
      if (uwplace.getIsSSL()) {
          SSLServerSocketFactory sslserversocketfactory =
              (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
          srvr = sslserversocketfactory.createServerSocket(portNum);
      } else {
          srvr = new ServerSocket(portNum);
      }
      ```

    - Create either a Socket or an SSLSocket. Since SSLSocket is derived from Socket, we can use a variable of type Socket to store both secure and insecure sockets. Therefore, the rest of the code doesn't need to know which type of socket is being used.

      ```
      InputStream in = null;
      Socket skt = null;
      if (uwP.getIsSSL()) {
          skt = (SSLSocket) srvr.accept();
      } else {
          skt = srvr.accept();
      }
      in = skt.getInputStream();
      ```

    - Since InputStream.available() always returns 0 when using SSL sockets, the readBytes method must use another algorithm to determine when data is available to read. For more information, see the Developer Note in the Socket Redesign section above.

- UWUtility
    - Accept an isSSL boolean parameter to InitUWPSocket. This parameter indicates whether regular or SSL sockets should be used.
    - Create either a Socket or an SSLSocket:

      ```
      InputStream in = null;
      Socket skt = null;
      if (uwP.getIsSSL()) {
          skt = (SSLSocket) srvr.accept();
      } else {
          skt = srvr.accept();
      }
      in = skt.getInputStream();
      ```

- UWPlace, UWAgent, and UWMailbox
    - Pass the appropriate boolean parameter to UWUtility.InitUWPSocket, depending on the type of socket required.

Class Name Collision Testing

Description

When an agent hops to another node, the UWAgent system allows it to carry along classes that it needs at the destination. This raises the possibility of a name collision if another agent running on the same node instantiates a class with the same name. Testing indicated that UWAgent is not currently affected by this problem. For test output, see the Class Name Collision Testing part of the Testing section below. Another mobile agent system, IBM Aglets, is affected because it supports caching. It solves the problem by using multiple class loaders, effectively isolating the identically-named classes from each other.

Logging

Description

The UWUtility class contains the following methods for logging:

- **Log**: Log an exception, or another message that should always be shown.
- **LogInfo**: Log an informational message.
- **LogEnter and LogExit**: Log entry to or exist from a method.

The CURRENT_DEBUG_LEVEL constant in UWUtility determines which of these logging methods actually produce output. Normally, the debug level should be set to DEBUG_EXCEPTION, meaning that only exception messages are shown and output from LogInfo, LogEnter, and LogExit is suppressed. When the level is set to DEBUG_INFO, informational messages (displayed by LogInfo) are shown. LogInfo calls can be left in the code permanently, and turned on only when necessary. When the debug level is set to DEBUG_METHOD, then the output from LogEnter and LogExit is shown. LogEnter and LogExit are used to log entry and exit from a method. This can be useful for detailed debugging of multithreaded program activity.

A note on performance: since the logging methods are always called, even when the debug level is set to DEBUG_EXCEPTION, they could affect performance testing. This could be solved by using a preprocessor that stripped out these calls before generating a “production” version of the UWAgent code. Alternatively, they could be removed manually by searching for the substring “UWUtility.Log.”

Testing

A number of agents were created to test the functionality described in this document. This section shows how to run these agents, as well as the expected output.

SpawnTest

SpawnTest spawns several children, each of which spawns more children. The execution engine limits the number of children based on user specification. SpawnTest shows child agent spawning functionality using the custom socket code.

mnode1

```
[uwagent@mnode1 UWAgent.new]$ java UWInject mnode0 SpawnTest -m 3 6 5
File : /home/uwagent/MA/UWAgent.new
URL : file:/home/uwagent/MA/UWAgent.new/
URL : null
ip = mnode1/10.1.0.1, time = 1124472917466, ID = 0
end of UWInject (main)
[uwagent@mnode1 UWAgent.new]$
```

mnode0

```
[uwagent@mnode0 UWAgent.new]$ java UWPlace

1: Hello from SpawnTest
1: My ID = 0
1: My parent's ID = -1
1: My class name = SpawnTest
1: My parent's class name = null
1: SpawnTest spawning SpawnTest2
1: Number of children I have spawned so far = 1
1: SpawnTest spawning SpawnTest2
1: Number of children I have spawned so far = 2
1: SpawnTest spawning SpawnTest2
UWAgent#spawnChild: Cannot spawn more than 2 children
1: Number of children I have spawned so far = 2
1: SpawnTest spawning SpawnTest2
UWAgent#spawnChild: Cannot spawn more than 2 children
1: Number of children I have spawned so far = 2
1: SpawnTest spawning SpawnTest2
UWAgent#spawnChild: Cannot spawn more than 2 children
1: Number of children I have spawned so far = 2
1: SpawnTest spawning SpawnTest2
UWAgent#spawnChild: Cannot spawn more than 2 children
1: Number of children I have spawned so far = 2
1: SpawnTest exiting


2: Hello from SpawnTest2
2: My ID = 2
2: My parent's ID = 0
2: My class name = SpawnTest2
2: My parent's class name = SpawnTest
2: SpawnTest2 spawning SpawnTest3

2: Hello from SpawnTest2
2: My ID = 1
2: My parent's ID = 0
2: My class name = SpawnTest2
2: My parent's class name = SpawnTest
2: SpawnTest2 spawning SpawnTest3
2: Number of children I have spawned so far = 1
2: SpawnTest2 spawning SpawnTest3
```

```
2: Number of children I have spawned so far = 2
2: SpawnTest2 spawning SpawnTest3
2: Number of children I have spawned so far = 3
2: SpawnTest2 spawning SpawnTest3
UWAgent#spawnChild: Cannot spawn more than 3 children
2: Number of children I have spawned so far = 3
2: SpawnTest2 spawning SpawnTest3
UWAgent#spawnChild: Cannot spawn more than 3 children
2: Number of children I have spawned so far = 3
2: SpawnTest2 exiting

2: Number of children I have spawned so far = 1
2: SpawnTest2 spawning SpawnTest3
2: Number of children I have spawned so far = 2
2: SpawnTest2 spawning SpawnTest3
2: Number of children I have spawned so far = 3
2: SpawnTest2 spawning SpawnTest3
UWAgent#spawnChild: Cannot spawn more than 3 children
2: Number of children I have spawned so far = 3
2: SpawnTest2 spawning SpawnTest3
UWAgent#spawnChild: Cannot spawn more than 3 children
2: Number of children I have spawned so far = 3
2: SpawnTest2 exiting


3: Hello from SpawnTest3
3: My ID = 3
3: My parent's ID = 1
3: My class name = SpawnTest3
3: My parent's class name = SpawnTest2
3: SpawnTest3 exiting


3: Hello from SpawnTest3
3: My ID = 4
3: My parent's ID = 1
3: My class name = SpawnTest3
3: My parent's class name = SpawnTest2
3: SpawnTest3 exiting


3: Hello from SpawnTest3
3: My ID = 5
3: My parent's ID = 1
3: My class name = SpawnTest3
3: My parent's class name = SpawnTest2
3: SpawnTest3 exiting


3: Hello from SpawnTest3
3: My ID = 6
3: My parent's ID = 2
3: My class name = SpawnTest3
3: My parent's class name = SpawnTest2
3: SpawnTest3 exiting


3: Hello from SpawnTest3
3: My ID = 7
3: My parent's ID = 2
3: My class name = SpawnTest3
3: My parent's class name = SpawnTest2
3: SpawnTest3 exiting


3: Hello from SpawnTest3
3: My ID = 8
```

```
3: My parent's ID = 2
3: My class name = SpawnTest3
3: My parent's class name = SpawnTest2
3: SpawnTest3 exiting
```

MessageTest

MessageTest spawns a child agent which hops to another node. The parent then sends a message to the child, and the child displays it. MessageTest shows hopping and message passing functionality using the new socket code.

mnode1 (parent)

```
[uwagent@mnode1 UWAgent.new]$ java UWPlace &
[1] 30970
[uwagent@mnode1 UWAgent.new]$ java UWInject localhost MessageTest -m 10
File : /home/uwagent/MA/UWAgent.new
URL : file:/home/uwagent/MA/UWAgent.new/
URL : null
ip = mnode1/10.1.0.1, time = 1124469527924, ID = 0
end of UWInject (main)

0: Hello from MessageTest
0: My parent's ID = -1
0: waiting before sending message...

1: Hello from MessageTest
1: My parent's ID = 0
1: Hopping to mnode2
1: MessageTest exiting

[uwagent@mnode1 UWAgent.new]$ 0: Sending message
0: Succeeded
0: MessageTest exiting
```

mnode2 (child)

```
[uwagent@mnode2 UWAgent.new]$ java UWPlace
1: waiting before receiving message...
1: Received the following message: Hello from agent 0
```

GatewayMessageTest

These three test cases test message passing and traversal between two networks separated by a gateway. An agent and a message cross the gateway and also traverse one more node on the destination network. In this test, the machines on the public network do not share a file system with the machines on the private network. Therefore, the Java source code must by synchronized between two files systems, and compiled separately on each one.

Test #1: Private to public network with **no** gateway specified. The operation fails (as expected) because the public node is not reachable unless a gateway is used.

mnode8

```
[uwagent@mnode8 UWAgent.new]$ java UWPlace &
[1] 7908
[uwagent@mnode8 UWAgent.new]$ java UWInject localhost GatewayMessageTest uw1-320-20
[uwagent@mnode8 UWAgent.new]$
0: Hello from GatewayMessageTest
0: My parent's ID = -1
0: waiting before sending message...
```

```
1: Hello from GatewayMessageTest
1: My parent's ID = 0
1: Hopping to uw1-320-20
UWPlace#sendAgent: java.net.SocketTimeoutException: connect timed out
UWPlace#sendAgent: Cause: null
1: GatewayMessageTest exiting

0: Sending message
0: GatewayMessageTest exiting

SocketThread#run: java.net.SocketTimeoutException: connect timed out
SocketThread#run: Cause: null

[uwagent@mnode8 UWAgent.new]$
```

medusa

```
[uwagent@medusa UWAgent.new]$ java UWPlace
```

uw1-320-20

```
[duncans@uw1-320-20 UWAgent.new]$ java UWPlace
```

uw1-320-21

```
[duncans@uw1-320-20 UWAgent.new]$ java UWPlace
```

Test #2: Private to public network **with** gateway specified as a parameter to UWPlace. The operation succeeds by using the gateway machine as an intermediate hop.

mnode8

```
[uwagent@mnode8 UWAgent.new]$ java UWPlace -g medusa &
[1] 7948
[uwagent@mnode8 UWAgent.new]$ java UWInject localhost GatewayMessageTest uw1-320-20
[uwagent@mnode8 UWAgent.new]$
0: Hello from GatewayMessageTest
0: My parent's ID = -1
0: waiting before sending message...

1: Hello from GatewayMessageTest
1: My parent's ID = 0
1: Hopping to uw1-320-20
1: GatewayMessageTest exiting

0: Sending message
0: GatewayMessageTest exiting


[uwagent@mnode8 UWAgent.new]$
```

medusa

```
[uwagent@medusa UWAgent.new]$ java UWPlace
```

uw1-320-20

```
[duncans@uw1-320-20 UWAgent.new]$ java UWPlace
1: Hopping to uw1-320-21
```

uw1-320-21

```
[duncans@uw1-320-21 UWAgent.new]$ java UWPlace
1: waiting before receiving message...
```

```
1: Received the following message: Hello from agent 0
```

Test #3: Public to private network **with** gateway specified in the hop() method. The operation succeeds by using the gateway machine as an intermediate hop.

uw1-320-20
```
[duncans@uw1-320-20 UWAgent.new]$ java UWPlace &
[1] 3048
[duncans@uw1-320-20 UWAgent.new]$ java UWInject localhost GatewayMessageTest mnode8
[duncans@uw1-320-20 UWAgent.new]$
0: Hello from GatewayMessageTest
0: My parent's ID = -1
0: waiting before sending message...

1: Hello from GatewayMessageTest
1: My parent's ID = 0
1: Hopping to mnode8
1: GatewayMessageTest exiting

0: Sending message
0: GatewayMessageTest exiting
```

medusa
```
[uwagent@medusa UWAgent.new]$ java UWPlace
```

mnode8
```
[uwagent@mnode8 UWAgent.new]$ java UWPlace
1: Hopping to mnode9
```

mnode9
```
[uwagent@mnode9 UWAgent.new]$ java UWPlace
1: waiting before receiving message...
1: Received the following message: Hello from agent 0
```

Monitor Commands

The monitor commands tests use an agent called MonitorTest. This agent spawns a child agent every 0.5 seconds. The child agent starts, does some work, and ends. This test runs in one mnode8 window. The other mnode8 window accepts UWMonitor commands. Using the AgentStatus command, we can see the effect of the Suspend, Resume, and Kill commands on the parent and children agents.

mnode8 (UWMonitor)
```
[uwagent@mnode8 UWAgent.new]$ java UWMonitor as
[uwagent@mnode8 UWAgent.new]$ java UWMonitor suspend 0
[uwagent@mnode8 UWAgent.new]$ java UWMonitor as
[uwagent@mnode8 UWAgent.new]$ java UWMonitor resume 0
[uwagent@mnode8 UWAgent.new]$ java UWMonitor kill 0
[uwagent@mnode8 UWAgent.new]$ java UWMonitor as
[uwagent@mnode8 UWAgent.new]$ java UWMonitor as
[uwagent@mnode8 UWAgent.new]$
```

mnode8 (MonitorTest)
```
[uwagent@mnode8 UWAgent.new]$ java UWPlace &
[1] 16117
[uwagent@mnode8 UWAgent.new]$ java UWInject localhost MonitorTest -m 9999
```

```
0: Hello from MonitorTest
1: Working
2: Working
1: Done
3: Working
2: Done
4: Working
3: Done
5: Working
4: Done
6: Working
5: Done
7: Working
6: Done
```

```
-- Agent status --
Number of agents: 3
ID      Name            Status
--      ----            ------
0       MonitorTest     Ready
7       MonitorTest     Running
0       UWMonitorAgent  Ready


7: Done
8: Working
9: Working
10: Working
8: Done
9: Done
11: Working
10: Done
12: Working
Agent 0 (MonitorTest) suspended
11: Done
13: Working
12: Done
13: Done

-- Agent status --
Number of agents: 2
ID      Name            Status
--      ----            ------
0       MonitorTest     Suspended
0       UWMonitorAgent  Ready

Agent 0 (MonitorTest) resumed
14: Working
15: Working
14: Done
16: Working
15: Done
17: Working
18: Working
16: Done
17: Done
19: Working
20: Working
18: Done
```

```
21: Working
19: Done
20: Done
22: Working
24: Working
21: Done
23: Working
22: Done
Agent 0 (UWMonitorAgent) killed
25: Working
Agent 0 (MonitorTest) killed
24: Done

-- Agent status --
Number of agents: 3
ID      Name            Status
--      ----            ------
23      MonitorTest     Ready
25      MonitorTest     Running
0       UWMonitorAgent  Ready

23: Done
26: Working
25: Done
27: Working
27: Done
26: Done

-- Agent status --
Number of agents: 1
ID      Name            Status
--      ----            ------
0       UWMonitorAgent  Ready


[uwagent@mnode8 UWAgent.new]$
```

Class Name Collision Testing

To test a potential class name collision scenario this, three test classes are used: a UWAgent called UnloadTest, and two regular classes that are both called UnloadTestClass. UnloadTestClass version 1 contains a method called PrintMessage that prints out its version number, waits 10 seconds, and then exits. UnloadTestClass version 2 contains a method of the same name with the same behavior. UnloadTest hops to another node (to test that the classes are carried properly through a hop operation) instantiates UnloadTestClass, and calls PrintMessage. Both classes are compressed into jar files with different names. As the test results show, the correct version of PrintMessage is called, even though Version 1 is still loaded and running when Version 2 is called. The test is run several times, so two copies of UnloadTestClass v1 and two copies of UnloadTestClass v2 are in memory (on mnode9) simultaneously.

mnode8

```
[uwagent@mnode8 UWAgent.new]$ java UWPlace &
[1] 19451
[uwagent@mnode8 UWAgent.new]$ java UWInject localhost UnloadTest -j
UnloadTestClass1.jar
file = UnloadTestClass.class
byteArrayClass.length = 614
[uwagent@mnode8 UWAgent.new]$
0: Hello from UnloadTest init()
```

```
0: Hopping to mnode9

[uwagent@mnode8 UWAgent.new]$ java UWInject localhost UnloadTest -j
UnloadTestClass2.jar
file = UnloadTestClass.class
byteArrayClass.length = 614

0: Hello from UnloadTest init()
0: Hopping to mnode9
[uwagent@mnode8 UWAgent.new]$
[uwagent@mnode8 UWAgent.new]$ java UWInject localhost UnloadTest -j
UnloadTestClass1.jar
file = UnloadTestClass.class
byteArrayClass.length = 614

0: Hello from UnloadTest init()
0: Hopping to mnode9
[uwagent@mnode8 UWAgent.new]$
[uwagent@mnode8 UWAgent.new]$ java UWInject localhost UnloadTest -j
UnloadTestClass2.jar
file = UnloadTestClass.class
byteArrayClass.length = 614

0: Hello from UnloadTest init()
0: Hopping to mnode9
[uwagent@mnode8 UWAgent.new]$
```

mnode9

```
[uwagent@mnode9 UWAgent.new]$ java UWPlace
0: Creating test class
Hello from UnloadTestClass Version ** 1 **
0: Creating test class
Hello from UnloadTestClass Version ** 2 **
0: Creating test class
Hello from UnloadTestClass Version ** 1 **
0: Creating test class
Hello from UnloadTestClass Version ** 2 **
UnloadTestClass Version 1 exiting
UnloadTestClass Version 2 exiting
UnloadTestClass Version 1 exiting
UnloadTestClass Version 2 exiting
```

Secure Communication

These tests show message passing using SSL sockets, using the MessageTest and GatewayMessageTest agents. These are the same test agents that were used for testing non-SSL hopping and message passing.

Test #1: Private to private (SSL, no gateway).

mnode8

```
[uwagent@mnode8 UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore -Dja
vax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgentKeys
tore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWPlace -e &
[1] 3273
[uwagent@mnode8 UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore -Dja
vax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgentKeys
tore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWInject localhost MessageT
est -e
```

```
[uwagent@mnode8 UWAgent.new]$
0: Hello from MessageTest
0: My parent's ID = -1
0: waiting before sending message...

1: Hello from MessageTest
1: My parent's ID = 0
1: Hopping to mnode9
1: MessageTest exiting

0: Sending message
0: Succeeded
0: MessageTest exiting


[uwagent@mnode8 UWAgent.new]$
```

mnode9

```
[uwagent@mnode9 UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore -Dja
vax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgentKeys
tore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWPlace -e
1: waiting before receiving message...
1: Received the following message: Hello from agent 0
```

Test #2: Private to public (SSL) with gateway specified as a parameter to UWPlace. Operation succeeds by using the gateway machine as an intermediate hop.

mnode8

```
[uwagent@mnode8 UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore -Dja
vax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgentKeys
tore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWPlace -e -g medusa &
[1] 3522
[uwagent@mnode8 UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore -Dja
vax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgentKeys
tore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWInject localhost GatewayM
essageTest uw1-320-20 -e
[uwagent@mnode8 UWAgent.new]$
0: Hello from GatewayMessageTest
0: My parent's ID = -1
0: waiting before sending message...

1: Hello from GatewayMessageTest
1: My parent's ID = 0
1: Hopping to uw1-320-20
1: GatewayMessageTest exiting

0: Sending message
0: GatewayMessageTest exiting

[uwagent@mnode8 UWAgent.new]$
```

medusa

```
[uwagent@medusa UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore -Dja
vax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgentKeys
tore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWPlace -e
```

uw1-320-20

```
[duncans@uw1-320-20 UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore
-Djavax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgent
Keystore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWPlace -e
```

```
1: Hopping to uw1-320-21
```

uw1-320-21

```
[duncans@uw1-320-21 UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore
-Djavax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgent
Keystore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWPlace -e
1: waiting before receiving message...
1: Received the following message: Hello from agent 0
```

Test #3: Public to private (SSL) **with** gateway specified in the hop() method. Operation succeeds by using the gateway machine as an intermediate hop.

uw1-320-20

```
[duncans@uw1-320-20 UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore
-Djavax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgent
Keystore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWPlace -e &
[1] 14294
[duncans@uw1-320-20 UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore
-Djavax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgent
Keystore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWInject localhost Gate
wayMessageTest mnode8 -e
[duncans@uw1-320-20 UWAgent.new]$
0: Hello from GatewayMessageTest
0: My parent's ID = -1
0: waiting before sending message...

1: Hello from GatewayMessageTest
1: My parent's ID = 0
1: Hopping to mnode8
1: GatewayMessageTest exiting

0: Sending message
0: GatewayMessageTest exiting


[duncans@uw1-320-20 UWAgent.new]$
```

medusa

```
[uwagent@medusa UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore -Dja
vax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgentKeys
tore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWPlace -e
```

mnode8

```
[uwagent@mnode8 UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore -Dja
vax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgentKeys
tore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWPlace -e
1: Hopping to mnode9
```

mnode9

```
[uwagent@mnode9 UWAgent.new]$ java -Djavax.net.ssl.keyStore=UWAgentKeystore -Dja
vax.net.ssl.keyStorePassword=m0bile@gents -Djavax.net.ssl.trustStore=UWAgentKeys
tore -Djavax.net.ssl.trustStorePassword=m0bile@gents UWPlace -e
1: waiting before receiving message...
1: Received the following message: Hello from agent 0
```