UWAgents User's Manual

(version 2.0)

Munehiro Fukuda*

Miriam Wallace*

* Computing and Software Systems, University of Washington, Bothell

UWAgents User's Manual

Table of Contents

Table of Figures

UWAgents User's Manual

About UWAgent

UWAgent is a java-based execution platform optimized for grid computing which uses antonymous mobile **agents** to perform necessary work. Agents can coordinate the completion of distributed computation independently of each other, over a network of **host** computers. In order to host an agent, each computer in the grid is initialized as a UWPlace, a potential **location** agents can migrate (**hop**) to and perform tasks.

Injecting a UWAgent into the grid of host computers initiates the process of spawning child agents, dynamically distributing tasks across the network of host computers. Each UWAgent adapts in parallel as the network or number of available hosts changes.

UWAgents and the UWPlace daemon it runs on form a robust distributed computing platform made for networks which already serve other purposes. This allows UWAgents to move autonomously through the network of UWPlace nodes where computing power may be found, allowing individual host machines to be used for purposes other than running an agent.

This movement, or **migration**, is accomplished through the UWAgent hop function which triggers an agent to pack up and move to another **place** in the network. The transition from one place to another is made possible by serialization which allows a UWAgent to save its current state, transmit that data to a new UWPlace location and then be re-instantiated to perform its tasks at the new location.

Agents can also **talk** to one another as necessary in order to communicate information or trigger tasks as necessary for the current agent's task.

Please note that words in bold are defined in the *Glossary* section for future reference.

System Invocation and Agent Injection

To set up a mobile agent network to run distributed programs on, first run UWPlace on all potential host machines. The UWPlace daemon runs in the background on host computers flagging them as sites where UWAgents can migrate and perform tasks.

Run UWPlace locally, from the machines the host machines with the following command:

UWPlace Command:

```
java –cp UWAgent.jar:. UWAgent.UWPlace –p <port#>
```

Example, UWPlace Command:

```
java –cp UWAgent.jar:. UWAgent.UWPlace –p <port#>
```

An SSH tunnel establishes a port-to-port connection between two computers, channeling the information from one computer to another, and back. Information from one port on the first computer is funneled to the second computer at another port. Similarly, information can go from the second computer to the first via another sets of ports, as illustrated in *Figure 1: An SSH Tunnel*.

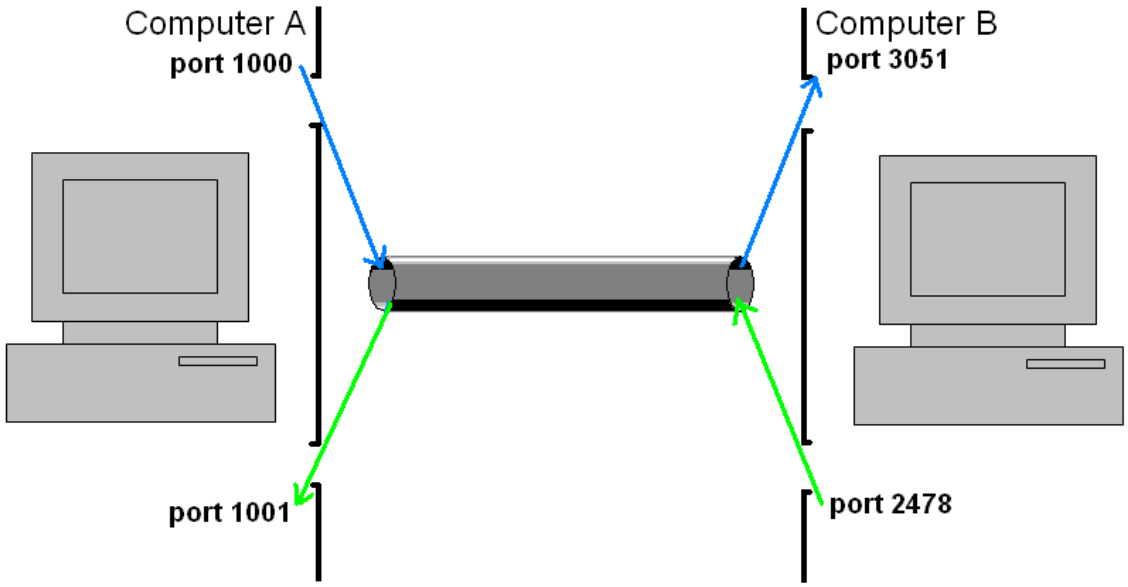

Figure 1: An SSH Tunnel. Data flows through the tunnel from one computer's out-port to the other computer's in-port: from Computer A to B and vice versa.

Connect your current computer to the target remote system (host) and run the following command, to connect the two computers through an SSH tunnel:

SSH Tunnel Command:

```
shh –l <accountName> -L <localOutPort#>:localhost:<targetInPort#> -R
    <targetOutPort#>:localhost:<localInPort#> <targetHostIP>
```

Example, SSH Tunnel Command:

```
shh –l myaccount -L 1000:localhost:2000 -R   2500:localhost:1500
    remoteHost
```

The phrase "localhost" in the command indicates the current, local computer; changing that keyword allows you set up SSH tunnels between two remote systems.

Once an SSH tunnel is established, run the UWPlace command via the SHH tunnel on the remote computer (repeat this process as necessary for multiple remote systems):

UWPlace Command for Remote Systems:

```
java –cp UWAgent.jar:. UWAgent.UWPlace –p <localInPort#>
    -<localOutPort#> <targetHostIP>
```

Example, UWPlace Command for Remote Systems:

```
java –cp UWAgent.jar:. UWAgent.UWPlace –p 1500 -1000 remoteHost
```

Once a system has the UWPlace daemon running it is available for use as a place for UWAgents to migrate and perform tasks.

To introduce an agent into your network of host computers (or nodes) inject the agent into the network via the inject command. This command sends an instance of the agent into the UWPlace network where the agent can then run and potentially roam across the network of

nodes independently. Inject can be run locally from a computer in the network or remotely via an SSH tunnel.

To inject an agent into local machine, run the following command:

UWInject Command:

```
java –cp UWAgent.jar:. UWAgent.UWInject –p <port#> <hostIP>
    <AgentClassName> <@argumentList>
```

Example, UWInject Command:

```
java –cp UWAgent.jar:. UWAgent.UWInject –p 1234 localhost MyAgent
    uw1-320-21 uw1-320-22
```

To inject an agent into a remote system connect to the remote place via an SSH tunnel and run the UWPlace command on the target remote place. Then inject the agent into the system using the following command:

UWInject Command for Remote Systems:

```
java –cp UWAgent.jar:. UWAgent.UWInject –p <localOutPort#> <hostIP>
    <AgentClassName> <@argumentList>
```

Example, UWInject Command for Remote Systems:

```
java –cp UWAgent.jar:. UWAgent.UWInject –p 1234 localhost MyAgent
    uw1-320-21 uw1-320-22
```

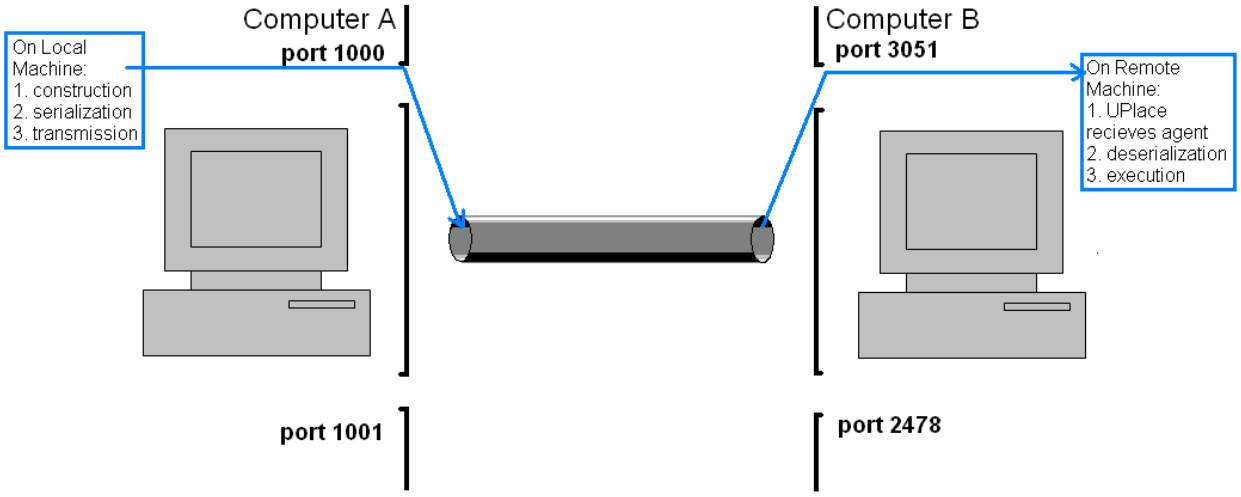

Figure 2: Agent Injection via SSH tunnel. When an agent is injected into a remote system via an SSH tunnel, the agent is first constructed on the local machine and then transmitted to the remote machine for execution.

Injection instantiates and initializes the agent, introducing it into the network of potential hosts running UWPlace daemons. The –p option allows you to specify the port as above.

Another option, -m, allows you to specify the *maximum number* of child agents a given agent can spawn. This allows the parent agent to use a simple formula to calculate the agent IDs of its children. Agents calculate the IDs of their children by multiplying their ID with the *max* and then adding 0 thru *max* minus 1 ($ChidID = ParentID * m + [0..m-1]$). Note that when –m is not used the default *max* is 10.

The first agent created is given the ID 0 and is a special case and can only spawn *max* minus 1 children. For example, if you inject with the –m option and specify and maximum of 4, agent 0 will spawn a maximum of 3 children with IDs 1 thru 3 while those children can spawn a total of 4 children, as seen in *Figure 3: Parent and Child IDs* below.

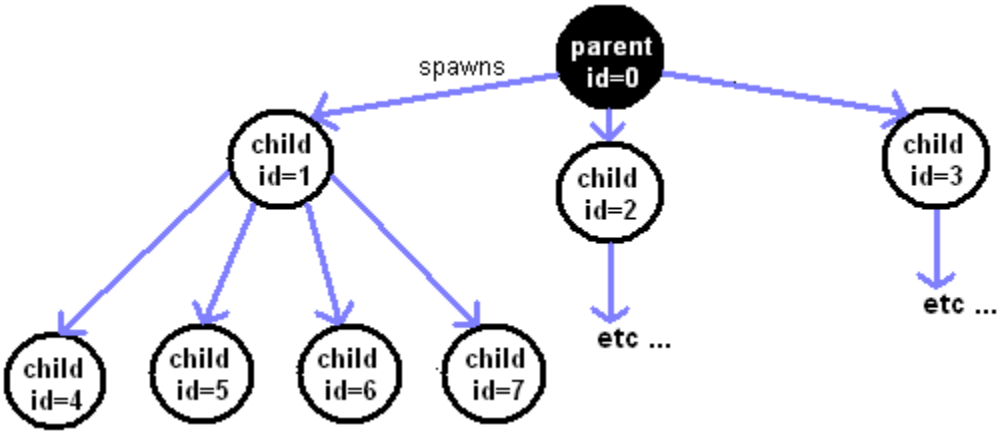

Figure 3: Parent and child IDs with a max of 4 children

To use the –m option, inject the agent using the following variation on the existing command:

UWInject Command with -m option:

```
java -cp UWAgent.jar:. UWAgent.UWInject -m <max> -p <port#> <hostIP>
    <AgentClassName> <@argumentList>
```

Example, UWInject Command with -m option:

```
java -cp UWAgent.jar:. UWAgent.UWInject -m 4 -p 1234 localhost MyAgent
    uw1-320-21 uw1-320-22
```

Agent Programming

Programming UWAgent allows control over how the agent class should perform across the network and what it should be used for. Agents can be tailored to specific purposes and problems.

When creating a custom UWAgent class, ensure the UWAgent class is accessible from the current agent class project and then inherit from the UWAgent class using the keyword “extends”.

All UWAgent extensions must also implement the Serializable interface which allows an agent to travel from one UWPlace to another. The current state of a Serializable class can be saved to a stream from which they can be restored later. UWAgents are transmitted from one place in the network to another using this method. Add the keywords “implements Serializable” to the class declaration as illustrated in the *UWAgent Code Sample 0* below.

UWAgent Code Sample 0: class declaration

```
import java.io.Serializable;

public class MyAgentSample extends UWAgent implements Serializable {
     …
}
```

After the UWAgent class declaration the agent's constructor, init method, and a function to execute when it moves from one location to another should be implemented. Sample agent implementations with and without hops are provided below in *UWAgent MyAgentHello Sample 1*.

UWAgent MyAgentHello Sample 1: without hop():

```
import java.io.Serializable;

public class MyAgentHello extends UWAgent implements Serializable {
    public MyAgentHello() { }

    public void init() {
        // Execute function helloWorld
        helloWorld();
    }

    // An agent function that performs a task or series of tasks and
    // may be called by other UWAgent functions such as hop
    public void helloWorld() {
        System.out.println("Hello World");
    }
}
```

When injected into a remote host the *UWAgent MyAgentHello Sample 1* code would be executed in the following order: the constructor would be called on the local machine, creating an instance of the MyAgentHello class. This instance would then be transmitted to the remote host named in your inject command, where the init method would be executed. Init would call the helloWorld method which would print "Hello World" to be printed to the console of the remote host. See *Figure 2* for a generalized example.

UWAgent MyAgentHello Sample 2: with hop()

```
import java.io.Serializable;

public class MyAgentHello extends UWAgent implements Serializable {
    public MyAgentHello() { }

    public void init() {
        // Go to host1 and execute "helloWorld" function
        hop(host1, "helloWorld");
    }

    // An agent function that performs a task or series of tasks and
    // may be called by other UWAgent functions such as hop
    public void helloWorld() {
        System.out.println("Hello World");
    }
}
```

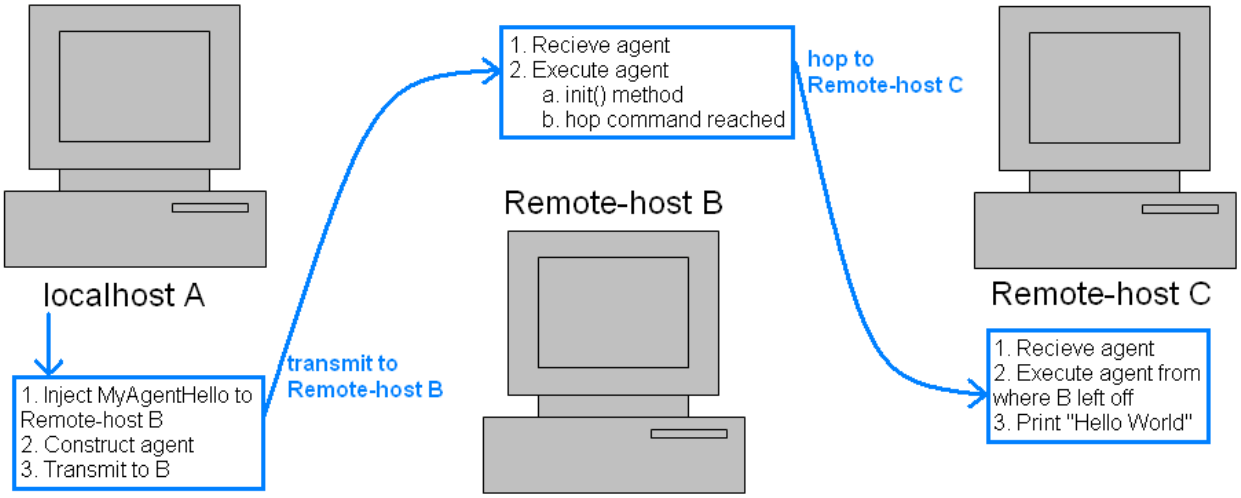

Figure 4: *UWAgent MyAgentHello Sample 2* **execution illustration.**

UWAgent MyAgentHello Sample 3: with Spawn and Hop

```
import java.io.Serializable;

public class MyAgentHello extends UWAgent implements Serializable {
    public MyAgentHello() { }

    public void init() {
        // When the current agent is the parent spawn a child
        if (getMyAgentId() == 0) {
            // Create a child agent at the target host node
            // This acts like an inject command to the host
            UWAgent agentRef = spawnChild(host1, MyAgentHello);
        }
        // When the current agent is not the parent
        else {
            // Go to another host and execute “helloWorld” function
            // at the new location
            hop(host2, “helloWorld”);
        }
    }

    // An agent function that performs a task or series of tasks and
    // may be called by other UWAgent functions such as hop
    public void helloWorld() {
        System.out.println("Hello World");
    }
}
```

As indicated in the code sample above, `spawnChild(…)`returns a UWAgent reference. This UWAgent instance is the child created by the spawn command. This reference, however, should not be used to call methods from the child agent because the child may already be in transit to another **place**. The one method that may be called with this reference is the `getAgentId()` method which allows the parent agent to to retrieve the child's ID.

Agent Communication with UWMessage

In programming agents to distribute work across a network they may need to communicate with each other to allow greater coordination. The UWAgent **talk** method provides this ability.

The UWAgent talk method takes two arguments, the ID of the agent to communicate with and the message to be sent. Invoking talk with these arguments will cause an agent to send a message to the recipient agent specified. Talk's Boolean return value indicates when messages are successfully received.

The agent ID of the recipient allows the sending (or talking) agent to locate the corresponding agent's IP address and gateways en route to the recipient. Once the agent ID, IP address and gateways have been identified, the message is then sent to the target agent even if that agent is currently in transit from one **node** to another.

The message is sent as an instance of the UWMessage class. The UWMessage class is a robust container which holds a message header and body but also tracks sending and receiving information. All of these components of the UWMessage class are accessible via get and set methods after UWMessage construction.

When one agent talks to another, the message is sent to the recipient agent's mailbox, an instance of UWAgentMailbox. This works remarkably like a real mailbox except that agents read from their mailbox as from a FIFO queue, the first message in is the first message read.

Messages can be sent and received successfully even when the receiving agent is in transit from one place to another. When an agent hops to a new node, a stub for the agent is created at the old location and the new location is broadcast to other agents. This assures messages already in transit to the hopping agent are received by the agent stub at the old location and forwarded to the recipient at its new location and that new messages are sent to the new location.

If the sender agent does not know its receivers IP address, the message can be sent to a parent or child of the recipient which then relays the message, allowing the message to ultimately arrive at the correct recipient.

UWMessage Sample

```
import java.io.Serializable;

public class MyAgentHello extends UWAgent implements Serializable {
    public MyAgentHello() { }

    public void init() {
        // Activate mailbox for all agents when they are initialized
        activateMailbox();

        // When the current agent is the parent spawn a child
        if (getMyAgentId() == 0) {
            // Create a child agent at the target host node
            // This acts like an inject command to the host
            spawnChild(host1, MyAgentHello);
            // Msg receiving code
            UWMessage msg = retrieveNextMessage();
            int senderId = msg.getSendingAgentId();
            String[] msgHeader = msg.getMessageHeader();
            // Do something with the msg
            System.out.println(senderId + ": " + msgHeader);
```

```
        }
        // When the current agent is not the parent
        else {
            // Go to another host and execute “helloWorld” function
            // at the new location
            hop(host2, “helloWorld”);

            // Send the an empty message to the this agent’s parent
            // (Get the parent agent’s id for the current agent)
            talk( getParentId( getAgentId() ),
                  new UWMessage(this,”hello”) );
        }
    }

    // An agent function that performs a task or series of tasks and
    // may be called by other UWAgent functions such as hop
    public void helloWorld() {
        System.out.println("Hello World");
    }
```

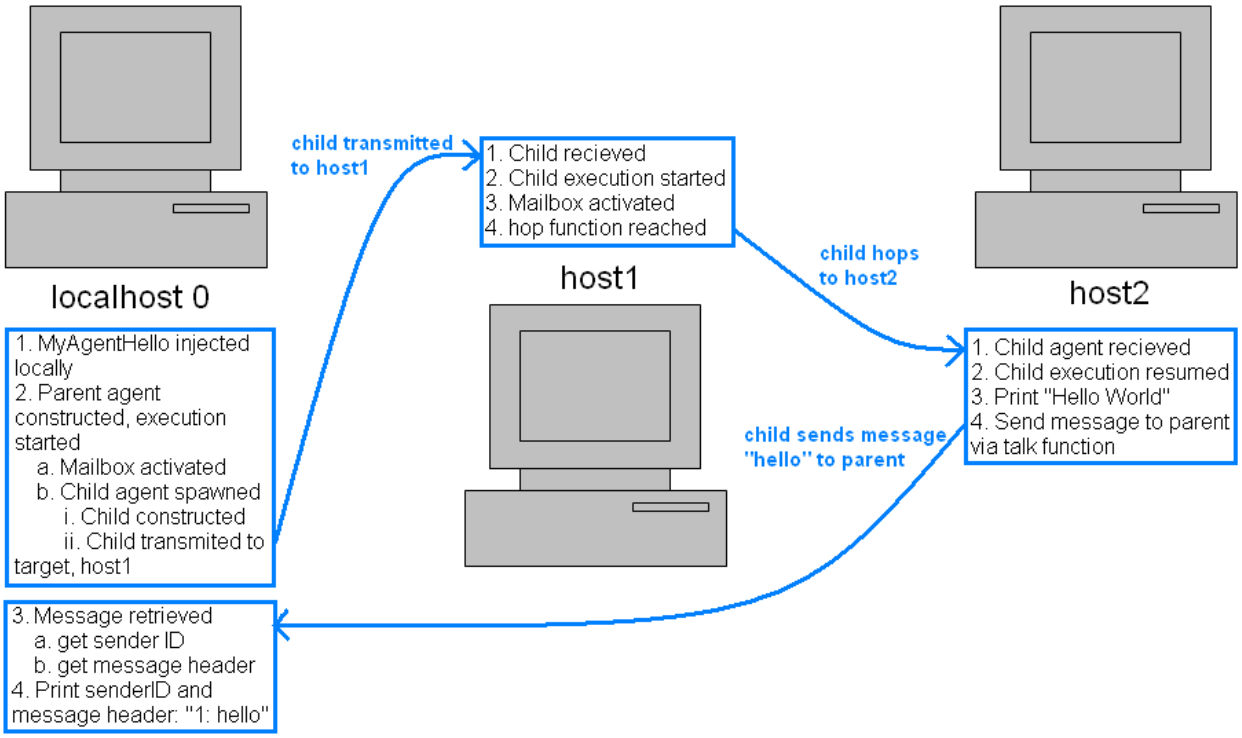

Figure 5: Talk is used to pass a message from a child agent to a parent.

Messages are funneled to an agent by the UWPlace daemon running on the host machine. When a message is received, after a connection has been established, the message is processed in an independent thread allowing the host machine to continue accepting incoming communications while a message is processed. When a message for an agent is received by the daemon, it is added to the agent’s mailbox: a queue of received messages which are read on a first-in-first-out basis.

UWMonitor

UWMonitor is a specialized UWPlace class created to provide status information about agents running at a UWPlace and provides utilities for controlling them. UWMonitor uses a specialized UWAgent class to carry out utility commands. Once an instance of the UWMonitor class is injected at a place you can view the status of agents running at that place as well as suspend, resume and kill those agent processes.

To invoke a specific command after the monitor has been injected use the following command:

UWMonitor Command:

```
java -cp UWAgent.jar UWAgent.UWMonitor <port#> <commands>
```

Example, UWMonitor Command:

```
java -cp UWAgent.jar UWAgent.UWMonitor 12345 kill <agentID>
```

UWPlace Internal Design

UWPlace internal design takes advantage of multi-processor systems by strategically separating processes to run on independent threads.

The main function of the UWPlace class creates a server socket to listen for client requests and then spawns a child accept thread.

The accept thread continually triggers the server accept function allowing UWPlace to accept incoming communication from clients. When a client-server connection is established the accept thread spawns a new thread to handle the communication and continues listening for new communication requests. Once this communication thread is established, messages such as UWAgents in transit from another UWPlace can be sent to the server and added to the queue of agents to run at the current UWPlace.

While the accept thread handles accepting and establishing client-server communications, the UWAgent main function calls the engine function in an infinite loop. The engine function pops incoming agents off the agent queue creating a thread for the new agent and adding it the ready queue to be run at the current UWPlace.

Final Comments

Glossary

agent – an active instance of the UWAgent class

inject – to pass a program into the UWAgent/UWPlace system to be executed on the nodes in a network

hop – when an agent moves from one host location to another, or the method which causes the agent to move

hop

```
public final void hop(java.lang.String hostname,
                      java.lang.String funcName)
```

Provides mobility for the UWAgent to move to another UWPlace.

Parameters:

`hostname` - a new destination host IP name

`funcName` - a function to be called at the destination

---

hop

```
public final void hop(java.lang.String hostName,
                      java.lang.String funcName,
                      java.lang.String[] funcArgs)
```

Provides mobility for the UWAgent to move to another UWPlace.

Parameters:

`hostName` - a new destination host IP name

`funcName` - a function to be called at the destination

`funcArgs` - arguments passed to the function

---

hop

```
public final void hop(java.lang.String hostName,
                      java.lang.String[] gateway,
                      java.lang.String funcName)
```

Hops through gateway list to destination host as accepting no arguments.

Parameters:

`hostName` - a new destination host IP name

`gateway` - an array of gateways all the way to the hostName

`funcName` - a function to be called at the destination

---

hop

```
public final void hop(java.lang.String hostName,
                      java.lang.String[] gateway,
                      java.lang.String funcName,
                      java.lang.String[] funcArgs)
```

Hops through gateway list to destination host as accepting string[].

Parameters:

`hostName` - a new destination host IP name
`gateway` - an array of gateways all the way to the hostName
`funcName` - a function to be called at the destination
`funcArgs` - arguments passed to the function

---

host – see node

location – see node

migrate – see hop

node – a computer in the network which has been initialized as a UWPlace and can therefore run an agent

place – see node

talk – the method which passes a message from one agent to another

talk

```
public final boolean talk(int recipId,
                          UWMessage message)
```

Provides communication for the UWAgent to send a UWMessage to another UWAgent though its "personal" AgentMailbox.

Parameters:

`recipId` - the ID of the intended recipient
`message` - the message to be sent

Returns:

true if the message has been sent or forwarded successfully.
