CSS497 Summer 2005

Redesigning and Enhancing the UWAgent Execution Engine

Intermediate Report: Fri 08/19/2005

Summary

Dates: Mon 06/20/2005 – Fri 08/19/2005

Summary of work:

- Added **class SocketThread** to UWPlace. This class accepts a socket connection from remote resources that want to interact with the UWPlace (e.g., send agents and messages). It then calls the appropriate functions based on the information passed in the socket message.
- Replaced the RMI code in UWPlace, UWAgent, UWAgentMailbox, and UWInject with socket code.
- Added a new class, UWUtility, with functions used for logging, socket processing, etc.
- Removed code from UWPlace that was specific to AgentTeamwork, or was unused. UWPlace now contains only code that is actively used by the UWAgent execution engine.
- Integrated new UWPlace code being developed concurrently by Professor Fukuda.
- Modified test classes to verify that the new socket code is able to support sending/ receiving agents (TCP/IP-based agent navigation) and sending/receiving messages (TCP/IP-based agent communication).

Test 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 new 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
```

Source Code

All source code is in `/home/uwagent/MA/UWAgent.new`. This section shows some of the new source code.

SocketThread

```
// This thread is used to retrieve function call information from a socket
class SocketThread implements Runnable
{
    Thread thread;
    UWPlace uwP;
    ServerSocket srvr;

    SocketThread(UWPlace uwplace, ServerSocket srvrSock) {
        UWUtility.Log("SocketThread", "constructor", "", UWUtility.DEBUG_ENTER);

        // Instantiate thread
        thread = new Thread(this, "socketThread");

        // Initialize
        uwP = uwplace;
        srvr = srvrSock;

        UWUtility.Log("SocketThread", "constructor", "", UWUtility.DEBUG_EXIT);
        thread.start();
    }

    // Read len bytes from InputStream in into byte array buffer;
    // blocks until at least len bytes are available.
    private void readBytes(InputStream in, byte[] buffer, int len) {
        UWUtility.Log("SocketThread", "readBytes", "", UWUtility.DEBUG_ENTER);

        // Wait for bytes to become available
        try {
            do { Thread.yield(); } while (in.available() < len);
            in.read(buffer);
        } catch (IOException ioe) {}

        UWUtility.Log("SocketThread", "readBytes", "", UWUtility.DEBUG_EXIT);
    }

    // Returns the String representation (using UTF8 encoding) of the
    // first len bytes of ByteBuffer bb. String is trimmed of leading
    // and trailing whitespace.
    private String getString(ByteBuffer bb, int len) {
        byte [] buf = new byte[len];
        bb.get(buf);                            // in byte array format
        String s = null;
        try {
            s = new String(buf, "UTF8");        // Decode using UTF8
            s = s.trim();
        } catch (UnsupportedEncodingException uee) {}

        return s;
    }
```

```
public void run() {
    UWUtility.Log("SocketThread", "run", "", UWUtility.DEBUG_ENTER);

    try {
        // srvr.accept() blocks until client connects
        Socket skt = srvr.accept();

        InputStream in = skt.getInputStream();

        // Indicates this UWPlace is ready to accept UWAgent
        UWUtility.Log("SocketThread", "run", "UWPlace is ready at localhost:"
                   + uwP.getPortNumber() + "/"
                   + uwP.getPlaceName() + ".", 0);

        // Read message header
        byte[] header = new byte[UWPlace.HEADER_SIZE];
        readBytes(in, header, UWPlace.HEADER_SIZE);

        // Parse header
        // First, put header in a ByteBuffer
        ByteBuffer bb = ByteBuffer.allocate(header.length);
        bb.put(header);
        bb.rewind();
        // Now, decode each field
        int type = bb.getInt();             // Message type
        // Name of function to call
        String strFuncName = getString(bb, UWPlace.FUNC_NAME_SIZE);

        // These two parameters are used for different purposes depending on the
        // function being called.
        int headerParam1 = bb.getInt();
        int headerParam2 = bb.getInt();
        // Done parsing header

        UWUtility.Log("SocketThread", "run", "Received a header.", 0);
        UWUtility.Log("SocketThread", "run", "Type field is: " + type, 0);
        UWUtility.Log("SocketThread", "run", "Function name is: " + strFuncName, 0);
        UWUtility.Log("SocketThread", "run", "Header parameter 1: " + headerParam1, 0);
        UWUtility.Log("SocketThread", "run", "Header parameter 2: " + headerParam2, 0);

        // TODO: handle remaining functions
        if (strFuncName.equals("receiveAgent")) {
            // Size of byte array representation of object
            int byteArraySize = headerParam1;

            int numClasses = headerParam2;      // Number of classes in ClassesHash

            // Read byte array representation of object
            byte[] byteArrayObj = new byte[byteArraySize];
            readBytes(in, byteArrayObj, byteArraySize);

            // Read class hash map
            int hashSize = 0;
            byte[] bytHashSize;
            byte [] bytClassHash;
            HashMap classesHash = new HashMap();
            for (int i=0; i<numClasses; i++) {
                // Get size of next classesHash
                bytHashSize = new byte[UWPlace.INT_SIZE];
                readBytes(in, bytHashSize, UWPlace.INT_SIZE);
                ByteBuffer bbh = ByteBuffer.allocate(bytHashSize.length);
                bbh.put(bytHashSize);
                bbh.rewind();
                int intHashSize = bbh.getInt();

                // Get serialized classesHash
                bytClassHash = new byte[intHashSize];
```

```
            readBytes(in, bytClassHash, intHashSize);
            ByteBuffer bbch = ByteBuffer.allocate(bytClassHash.length);
            bbch.put(bytClassHash);
            bbch.rewind();

            // Get string and byte array and insert into classesHash
            int stringLength = bbch.getInt();
            int byteArrayLength = bbch.getInt();
            String className = getString(bbch, stringLength);
            byte [] bytClassFile = new byte[byteArrayLength];
            bbch.get(bytClassFile);
            classesHash.put(className, bytClassFile);
        }
        // Close InputStream and client socket
        in.close();
        skt.close();

        // This will push the agent onto the stack
        uwP.receiveAgent(byteArrayObj, classesHash, null);
    } else if (strFuncName.equals("registerAgentLocation")) {
        int ts = headerParam1;  // agent ID
        int rawAddressLength = headerParam2;

        // Read byte[] representation of InetAddress
        byte[] rawAddress = new byte[rawAddressLength];
        readBytes(in, rawAddress, rawAddressLength);

        byte[] bytFs = new byte[UWPlace.INT_SIZE];
        readBytes(in, bytFs, UWPlace.INT_SIZE);
        ByteBuffer fsBuff = ByteBuffer.allocate(UWPlace.INT_SIZE);
        fsBuff.put(bytFs);
        fsBuff.rewind();
        int fs = fsBuff.getInt();

        UWAgent ag = uwP.getAgentById(fs);
        if (ag != null) {
            InetAddress ti = InetAddress.getByAddress(rawAddress);
            UWUtility.Log("SocketThread", "run",
                "Calling ag.registerAgentLocation with ts = " +
                ts + ", ti = " + ti, 0);
            ag.registerAgentLocation(ts, ti);
        } else {
            UWUtility.Log("SocketThread", "run", "Agent " + fs + " not found", 0,
            UWUtility.DEBUG_EXCEPTION);
        }

        in.close();
        skt.close();
    } else if (strFuncName.equals("verifyAgentLocation")) {
        int agentId = headerParam1;
        // (headerParam2 is not used here)

        boolean success = uwP.verifyAgentLocation(agentId);

        OutputStream out = skt.getOutputStream();
        out.write(success?1:0);

        out.close();
        in.close();
        skt.close();
    } else if (strFuncName.equals("receiveMessage")) {
        int agentID = headerParam1;
        int messageLength = headerParam2;

        // Read byte[] representation of UWMessage
        byte[] byteArrayObj = new byte[messageLength];
        readBytes(in, byteArrayObj, messageLength);
```

```
                UWUtility.Log("UWPlace", "run", "Attempting to retrieve agent ID " +
                    agentID, 0);
                UWAgent ag = null;
                ag = uwP.getAgentById(agentID);
                if (ag != null) {
                    // Convert the byte array to an instance of UWMessage.
                    ByteArrayInputStream bais = new ByteArrayInputStream(byteArrayObj);
                    ObjectInputStream ois = new ObjectInputStream(bais);
                    Object agobj = ois.readObject();
                    UWMessage message = (UWMessage)agobj;   // Create the UWMessage
                    ois.close();
                    bais.close();

                    ag.receiveMessage(message);
                } else {
                    UWUtility.Log("SocketThread", "run", "Agent " + agentID + " not found",
                        0, UWUtility.DEBUG_EXCEPTION);
                }

                in.close();
                skt.close();
            } else {
                UWUtility.Log("SocketThread", "run", "SocketThread : Unknown function
name");
            }
            uwP.notifyAgents();

        } catch (java.net.BindException be) {
            UWUtility.Log("SocketThread", "run", "Bind exception in SocketThread");
        } catch (Exception e) {
            UWUtility.Log("SocketThread", "run", "" + e);
            UWUtility.Log("SocketThread", "run", "Cause : " + e.getCause());
        }
        UWUtility.Log("SocketThread", "run", "", UWUtility.DEBUG_EXIT);
    }
}
```

UWUtility.InitUWPSocket

```
// Initialize a Socket to send a command to a UWPlace
// Sends a header through the socket, and returns an OutputStream for sending
// more data.
public static OutputStream InitUWPSocket(int messageType, String hostName,
        String portNum, String funcName, int headerParam1, int headerParam2)
{
    try {
        // Set up a socket to send the message header and body
        // The server socket should be set up to listen on this port in main()
        Socket skt = new Socket(hostName, Integer.parseInt(portNum));
        OutputStream out = skt.getOutputStream();

        // Set up a 40-byte header with the following structure
        //  message type (int) | function name (string) | header parameter 1 (int)
        //      | header parameter 2 (int)
        ByteBuffer headerBuff = ByteBuffer.allocate(HEADER_SIZE);   // header
        headerBuff.putInt(messageType);     // message type

        // Function name
        byte[] bytFuncName = funcName.getBytes("UTF8");
        headerBuff.put(bytFuncName);
        for (int i=bytFuncName.length; i<FUNC_NAME_SIZE; i++)
        {
            headerBuff.put((byte)' ');  // pad with spaces to end of func name field
        }

        headerBuff.putInt(headerParam1);
```

```
        headerBuff.putInt(headerParam2);

        out.write(headerBuff.array());      // write the header
        return out;
    } catch (IOException ioe) {
        System.err.println("UWUtility#InitUWPSocket: IOException : " + ioe);
        System.err.println("Cause : " + ioe.getCause());
        return null;
    }
}
```

UWAgentMailbox.sendMessage

```
// Send the specified UWMessage to the UWAgentMailbox specified within the UWMessage.
public boolean sendMessage(UWMessage message, InetAddress ip, long time) {
    UWUtility.Log("UWAgentMailbox", "sendMessage", "", UWUtility.DEBUG_ENTER);

    boolean success = false;
    try {
        InetAddress receivingIpAddr = message.getReceivingIp();

        // Write the byte representation of the message to the output
        // stream and send to the new place.
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(message);
        oos.close();
        byte[] byteArrayObj =  baos.toByteArray();
        baos.close();

        OutputStream out =
            UWUtility.InitUWPSocket(UWPlace.MSG_TYPE_FUNC,
                receivingIpAddr.getHostName(),
                getPortNumber(), "receiveMessage", message.getReceivingAgentId(),
                byteArrayObj.length);
        out.write(byteArrayObj);

        // Indicates message went to new UWAgentMailbox successfully
        UWUtility.Log("UWAgentMailbox", "sendMessage",
                message.getSendingAgentId() + " sends message to " +
                message.getReceivingAgentId() + " =) ", 0);
    }
    catch (Exception e) {
        System.out.println("UWAgentMailbox#sendMessage (via messaging): " + e);
        System.out.println("Cause : " + e.getCause());
    }

    UWUtility.Log("UWAgentMailbox", "sendMessage", "", UWUtility.DEBUG_EXIT);
    return success;
}
```