CSS497 Summer 2005

Redesigning and Enhancing the UWAgent Execution Engine

Status Report: Fri 07/15/2005

Summary

Dates: Sat 07/02/2005 – Fri 07/15/2005

This work period: Extended the socket code to call receiveAgent().

Next work period: Generalize the socket code to support more method calls.

Code Changes

The new socket code is in UWPlace.main() and UWPlace.sendAgent().

```
   public static void main( String[] args ) {
             // ...

                // Listen on the UWPlace port number
                byte[] header = new byte[40];
                int portNum = Integer.parseInt(uwplace.getPortNumber());
                ServerSocket srvr = new ServerSocket(portNum);
                Socket skt = srvr.accept();
                System.out.println("Server socket connected.\n");
              //PrintWriter out = new PrintWriter(skt.getOutputStream(), true);
                //OutputStream out = skt.getOutputStream();
                InputStream in = skt.getInputStream();

              //in.close();
              //skt.close();
              //srvr.close();

                // Indicates this UWPlace is ready to accept UWAgent
                System.out.println("UWPlace is ready at localhost:"
                                + uwplace.getPortNumber() + "/"
                                + uwplace.getPlaceName() + ".");

                // Busy waiting for UWAgent to come
                while (true) {
                       in.read(header);
                       ByteBuffer bb = ByteBuffer.allocate(header.length);
                       bb.put(header);
                       bb.rewind();
                       int type = bb.getInt();
                       byte [] b = new byte[28];
                       bb.get(b);
                       String t = new String(b, "UTF8");
                       t = t.trim();
                       int byteArraySize = bb.getInt();    // size of byte array
representation of object
                       int numClasses = bb.getInt();
```

```
                    System.out.println("UWPlace main() received a header.");
                    System.out.print("Type field is: ");
                System.out.println(type);
                System.out.print("Function name is: ");
                System.out.println(t);
                System.out.print("Size of byte array: ");
                System.out.println(byteArraySize);
                System.out.print("Number of classes: ");
                System.out.println(numClasses);

                byte[] byteArrayObj = new byte[byteArraySize];
                in.read(byteArrayObj);

                int hashSize = 0;
                   byte[] bytHashSize;
                      HashMap classesHash = new HashMap();
                for (int i=0; i<numClasses; i++) {
                      bytHashSize = new byte[4];
                      in.read(bytHashSize);
                      ByteBuffer bbh = ByteBuffer.allocate(bytHashSize.length);
                      bbh.put(bytHashSize);
                      bbh.rewind();
                      int intHashSize = bbh.getInt();
                      byte [] bytClassHash = new byte[intHashSize];
                      in.read(bytClassHash);
                      ByteBuffer bbch = ByteBuffer.allocate(bytClassHash.length);
                      bbch.put(bytClassHash);
                      bbch.rewind();
                      int stringLength = bbch.getInt();
                      int byteArrayLength = bbch.getInt();
                      byte [] bytClassName = new byte[stringLength];
                      bbch.get(bytClassName);
                      String className = new String(bytClassName, "UTF8");
                      byte [] bytClassFile = new byte[byteArrayLength];
                      bbch.get(bytClassFile);

                      classesHash.put(className, bytClassFile);

                      System.out.print("Class name: ");
                      System.out.println(className);
                }

                uwplace.receiveAgent(byteArrayObj, classesHash, null);

               uwplace.engine(uwplace);
            }


public void sendAgent(UWAgent ag, HashMap classesHash, String hostName) {
               // ...

            // Allow no messages during network transfer.
            unregisterAgentAtPlace(ag.getAgentId());

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

```
                  // Set up a socket to send the message header and body
                  int portNum = Integer.parseInt(getPortNumber());
               Socket skt = new Socket(hostName, portNum);
                  OutputStream out = skt.getOutputStream();

                  // TODO: replace magic numbers with constants
                  // Set up a 40-byte header with the following structure
                  //       message type (int) | function name (string) |
                     // byteArrayObj size (bytes)
                  //              | number of classes in classesHash
                  ByteBuffer headerBuff = ByteBuffer.allocate(40); // header
                  headerBuff.putInt(1);        // message type (1 = function)

                  // Function name
                  String funcName = "receiveAgent";
                  byte[] bytFuncName = funcName.getBytes("UTF8");
                  headerBuff.put(bytFuncName);
                  for (int i=bytFuncName.length; i<28; i++)
                  {
                     headerBuff.put((byte)' '); // pad with spaces to end of func name
                  }
                  headerBuff.putInt(byteArrayObj.length);
                     // bytes in byte representation of the agent

                  headerBuff.putInt(classesHash.size());    // number of classes

                     // classesHash stores one or more of the following pair
                     // of objects:
                  //              (class name, class file)
                     // Class name is a string; class file is a byte array.
                  // We will serialize classesHash(String, byte[]) as follows for
                     // each pair:
                  //       string length|byte array length|string|byte array
                  ByteBuffer byteBuff;   // one serialized pair
                  ArrayList classesHashArray = new ArrayList();
                     // Array to store all pairs

                  // Iterate through set of object pairs, serialize each one,
                     //and add each one to array
                     for (Iterator iter = classesHash.keySet().iterator();
iter.hasNext(); )
                     {
                         String className = (String)iter.next();
                         byte[] bytClassName = className.getBytes("UTF8");
                         byte[] bytClass = (byte[])classesHash.get(className);

                         // Allocate space for 2 integers + class name +
                            // class byte array
                         byteBuff = ByteBuffer.allocate(8 + bytClassName.length +
bytClass.length);
                         byteBuff.putInt(bytClassName.length);      // string length
                         byteBuff.putInt(bytClass.length);   // byte array length
                         // Insert string (as byte array)
                         byteBuff.put(bytClassName);
                         // Insert class (as byte array)
                     byteBuff.put(bytClass);
                     // Add to array
                         classesHashArray.add(byteBuff);
                     }

                  // Send classes to destination for function call
               out.write(headerBuff.array());  // send header
```

```
                out.write(byteArrayObj);
                   // send byte representation of the agent

             ByteBuffer hashSize;     // size (bytes) of a classesHash pair
                   for (Iterator iter = classesHashArray.iterator(); iter.hasNext();
)
                   {
                          ByteBuffer bb = (ByteBuffer)iter.next();
                          hashSize = ByteBuffer.allocate(4);
                          hashSize.putInt(bb.array().length);
                          out.write(hashSize.array());     // send size (bytes)
                          out.write(bb.array());
                                 // send serialized classesHash pair
                   }
             }
```

Output

Here is the output on mnode0/mnode1 (note: rmiregistry is not used).
The source code is in `/home/uwagent/MA/UWAgent.new`.

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 = 1121448604131, ID = 0
file = ./SpawnTest.class
byteArrayClass.length = 1612
end of UWInject (main)
[uwagent@mnode1 UWAgent.new]$
```

mnode0

```
[uwagent@mnode0 UWAgent.new]$ java UWPlace &
[1] 21332
[uwagent@mnode0 UWAgent.new]$ All backup files are deleted.
Server socket connected.

UWPlace is ready at localhost:35355/UWPlace.
UWPlace main() received a header.
Type field is: 1
Function name is: receiveAgent
Size of byte array: 1224
Number of classes: 1
Class name: SpawnTest
UWPlace#receiveAgent: before activateMailbox()
UWPlace#receiveAgent: registerd id= 0
Agentthread#Agentthread: before activateMailbox(), myId = 0

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

UWPlace main() received a header.
Type field is: 1
Function name is: receiveAgent
Size of byte array: 1224
Number of classes: 1
Exception in thread "main" java.nio.BufferUnderflowException
        at java.nio.Buffer.nextGetIndex(Buffer.java:404)
        at java.nio.HeapByteBuffer.getInt(HeapByteBuffer.java:336)
        at UWPlace.main(UWPlace.java:884)

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

Notice that the first part of the SpawnTest test is executed. The second call to receiveAgent fails due to problems with the socket receive code.