Duncan Smith
CSS499 Summer Quarter 2004
Enhancing the UWAgent Execution Engine

Status Report 07/26/2004

Status Summary

In investigating the cascading termination and descendent termination functionality, my tests indicate that child agents are able to communicate with each other even after their parent has terminated. However, if the UWPlace on which the parent resided is terminated, then the communication fails. These results call into question the need for an agent to postpone its termination until all of its children have been terminated, or to terminate all of its descendents before terminating itself.

The system I am using for testing is located in **/home/uwagent/MA/UWAgent.new** (in the **uwagent** account's home directory).

Source Code

All of the source code is available in **/home/uwagent/MA/UWAgent.new**. The following is a listing of the agents used for testing communication between agents whose parents have terminated. CascadeTest.java is the root agent. CascadeTest2.java is child agent 0.0. CascadeTest3.java is child agent 0.1.

CascadeTest.java

```
import java.io.*;
import java.rmi.*;
import java.util.*;

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

        private void printMsg(String message) {
                if (message.equals("")) {
                        System.out.println();
                } else {
                        System.out.println(getAgentId() + ": " + message);
                }
        }

        public void sleepNow(int sleepTime) {
                try {
                        Thread.sleep(sleepTime*2);
                } catch (InterruptedException e) {}
        }

        public void init() {
                printMsg("");
                printMsg("Hello from agent " + getAgentId());
                printMsg("Spawning child 0.0");
                spawnChild("CascadeTest2");   // 0.0
                printMsg("Spawning child 0.1");
                spawnChild("CascadeTest3");   // 0.1
                printMsg("CascadeTest exiting");
                printMsg("");
        }
}
```

CascadeTest2.java

```
import java.io.*;
import java.rmi.*;
import java.util.*;

public class CascadeTest2 extends UWAgent implements Serializable {
	public CascadeTest2() {}

	private void printMsg(String message) {
		if (message.equals("")) {
			System.out.println();
		} else {
			System.out.println(getAgentId() + ": " + message);
		}
	}

	public void nextHop() {
		printMsg("");
		printMsg("Hello from agent " + getAgentId());
		printMsg("Waiting...");
		sleepNow(5000);
		if (getAgentId().equals("0.0")) {
			printMsg("Hopping to mnode3");
			hop("mnode3", "ContactAgent", null);
		} else {
			printMsg("Hopping to mnode4");
			hop("mnode4", "ContactAgent", null);
		}
	}

	public void sleepNow(int sleepTime) {
		try {
			Thread.sleep(sleepTime);
		} catch (InterruptedException e) {}
	}

	public void ContactAgent() {
		boolean success;

		printMsg("waiting...");
		sleepNow(5000);

		printMsg("Sending message");
		UWMessage message = new UWMessage(this, "Hello from agent " +
getAgentId());

		if (getAgentId().equals("0.1")) {
			success = talk("0.0", message);
		} else {
			success = talk("0.1", message);
		}

		if (success) {
			printMsg("Succeeded");
		} else {
			printMsg("Failed");
		}

		printMsg("waiting...");
		sleepNow(3000);

		printMsg("Checking for messages...");
		UWMessage receivedMessage = retrieveNextMessage();
		String senderAgentId = receivedMessage.getSendingAgentId();
		String [] header = receivedMessage.getMessageHeader();

		printMsg("senderAgentId = " + senderAgentId);
		printMsg("header[0] = " + header[0]);
	}

	public void init() {
```

```
                printMsg("");
                printMsg("Hello from agent " + getAgentId());
                printMsg("Hopping to mnode2");
                hop("mnode2", "nextHop", null);
                printMsg("CascadeTest2 exiting");
                printMsg("");
        }
}

import java.io.*;
import java.rmi.*;
import java.util.*;
```

CascadeTest3.java

```
public class CascadeTest3 extends UWAgent implements Serializable {
        public CascadeTest3() {}

        private void printMsg(String message) {
                if (message.equals("")) {
                        System.out.println();
                } else {
                        System.out.println(getAgentId() + ": " + message);
                }
        }

        public void nextHop() {
                printMsg("");
                printMsg("Hello from agent " + getAgentId());
                printMsg("Waiting...");
                sleepNow(5000);
                if (getAgentId().equals("0.0")) {
                        printMsg("Hopping to mnode3");
                        hop("mnode3", "ContactAgent", null);
                } else {
                        printMsg("Hopping to mnode4");
                        hop("mnode4", "ContactAgent", null);
                }
        }

        public void sleepNow(int sleepTime) {
                try {
                        Thread.sleep(sleepTime*2);
                } catch (InterruptedException e) {}
        }

        public void ContactAgent() {
                boolean success;

                printMsg("waiting...");
                sleepNow(5000);

                printMsg("Sending message");
                UWMessage message = new UWMessage(this, "Hello from agent " +
getAgentId());

                if (getAgentId().equals("0.1")) {
                        success = talk("0.0", message);
                } else {
                        success = talk("0.1", message);
                }

                if (success) {
                        printMsg("Succeeded");
                } else {
                        printMsg("Failed");
                }

                printMsg("waiting...");
                sleepNow(3000);
```

```
                printMsg("Checking for messages");
                UWMessage receivedMessage = retrieveNextMessage();
                String senderAgentId = receivedMessage.getSendingAgentId();
                String [] header = receivedMessage.getMessageHeader();

                printMsg("senderAgentId = " + senderAgentId);
                printMsg("header[0] = " + header[0]);
        }

        public void init() {
                printMsg("");
                printMsg("Hello from agent " + getAgentId());
                printMsg("Hopping to mnode2");
                hop("mnode2", "nextHop", null);
                printMsg("CascadeTest3 exiting");
                printMsg("");
        }
}
```

Execution Output

Output from mnode1:

The output from mnode1 shows the root agent spawning two children (0.0 and 0.1), and each child hopping to mnode2. The "`0: CascadeTest exiting`" output shows that the root agent has terminated.

```
[uwagent@mnode1 UWAgent.new]$ java UWInject localhost CascadeTest -m 10
File : /home/uwagent/MA/UWAgent.new
URL : file:/home/uwagent/MA/UWAgent.new/
ip = mnode1/10.1.0.1, time = 1090819800795, ID = 0
file = ./CascadeTest.class
byteArrayClass.length = 1238

0: Hello from agent 0
0: Spawning child 0.0
end of UWInject (main)

0.0: Hello from agent 0.0
0.0: Hopping to mnode2
0: Spawning child 0.1
0: CascadeTest exiting


0.1: Hello from agent 0.1
0.1: Hopping to mnode2
0.1: CascadeTest3 exiting

0.0: CascadeTest2 exiting

[uwagent@mnode1 UWAgent.new]$
```

Output from mnode2:

The output from mnode2 shows each child agent waiting (to make sure the root has terminated). Child 0.0 then hops to mnode3, and child 0.1 hops to mnode4.

```
0.1: Hello from agent 0.1
0.1: Waiting...

0.0: Hello from agent 0.0
0.0: Waiting...
0.1: Hopping to mnode4
0.0: Hopping to mnode3
```

Output from mnode3:

The output from mnode3 shows agent 0.0 waiting (to ensure that agent 0.1 has finished migrating), then attempting to send a message to agent 0.1. The talk() function returns a success flag. This output also shows an invalid message log resulting from agent 0.1 attempting to send a message to agent 0.0. Although the message is sent successfully, it is considered invalid because it is not sent by an ancestor or descendent of 0.0. The code that enforces this rule is in UWAgent.receiveMessage.

```
0.0: waiting...
0.0: Sending message
talk to agentId: 0.1
INVALID MESSAGE LOG:
Sending Agent ID: 0.1
Receiving Agent ID: 0.0


0.0 sends message to 0.1 =)
0.0: Succeeded
0.0: waiting...
0.0: Checking for messages...
```

Output from mnode4:

The output from mnode4 is symmetric to the mnode3 output, but for agent 0.1 instead of agent 0.0. As with mnode3, the agent does not find any messages in its queue, because they are in the invalid message log.

```
0.1: waiting...
0.1: Sending message
talk to agentId: 0.0
0.1 sends message to 0.0 =)
0.1: Succeeded
0.1: waiting...
INVALID MESSAGE LOG:
Sending Agent ID: 0.0
Receiving Agent ID: 0.1


0.1: Checking for messages
```

Duncan Smith
CSS499 Summer Quarter 2004
Enhancing the UWAgent Execution Engine

Status Report 08/09/2004

Status Summary

This status report explains the following enhancements to the system:

- Automatically delete an agent's UWAgentMailbox (using deactivateMailbox()) before the agent terminates, to prevent orphan mailboxes from cluttering the UWPlace.
- Wait to terminate an agent until its descendants have terminated.
- Add a UWAgent method to allow agents to force their descendants to terminate immediately.

The code is located in **/home/uwagent/MA/UWAgent.new** (in the **uwagent** account's home directory). The following files have been modified since the previous status report: UWPlace.java, UWAgent.java, UWAgentMailbox.java, UWMessagingSystem.java, and CascadeTest.java.

Implementation Details

Delete UWAgentMailbox

The only change required to add this functionality was to call uwA.deactivateMailbox(); at the end of the run() method of AgentThread. Previously, the AgentThread would terminate at the end of the run() method, but the UWAgentMailbox would continue to run at the UWPlace. The deactivateMailbox() call removes the mailbox.

Delayed Termination

One of the side effects of terminating the UWAgentMailbox when the AgentThread exits is that descendants may not be able to communicate with each other if they rely on the parent for addressing information. Consequently, an agent should postpone its termination until all of its children have terminated. UWAgent's getChildrenNum() function returns the total number of children spawned by an agent. However, this value does not decrease when children finish executing. To find out which children are still executing, we can attempt to send each of them a notification, and count the number of successful notifications. When the count reaches zero, then it is safe for the parent to exit.

Cascading Termination

A parent agent may not want to wait for its children to terminate. To provide for cases when the parent must terminate immediately, there is now a setTerminationRequest() method in UWAgent. This method sets a termination request flag in UWAgent. AgentThread.run() in UWPlace checks this flag for the current agent when its next function is ready to be executed. If the flag is set, then the next function is not executed.

Note that cascading termination does not interrupt a user program in the middle of a function, so the parent agent may still have to wait for the current function to complete before its descendants terminate themselves.

In addition to setting the terminationRequest flag for the current agent, the setTerminationRequest method also sends a notification to all of the agent's children instructing them to set their terminationRequest flags. In this way, all of the original parent agent's descendants receive the termination request in a cascading fashion.

Source Code

All of the source code is available in **/home/uwagent/MA/UWAgent.new**. The CascadeTest, CascadeTest2, and CascadeTest3 agents described in the 7/26/04 status report are used again here, with one change: the CascadeTest agent includes a call to the new setTerminationRequest() method in the second test to demonstrate that functionality. This is shown in the second test below.

Execution Output

No Termination Request

In this test, the CascadeTest agent does not call setTerminationRequest. Therefore, its children (CascadeTest2 and CascadeTest3) complete their execution by sending messages to each other. They are able to find each other because the parent, CascadeTest, waits for its children to finish before ending itself and its UWAgentMailbox.

Output from mnode1:
The output from mnode1 shows the root agent spawning two children (0.0 and 0.1), and each child hopping to mnode2. The "`0: CascadeTest exiting`" output indicates the end of the parent's user code. However, notice that "`AgentThread for CascadeTest ending`" is the last output to be shown. This indicates that the parent has waited for its children to finish.

```
File : /home/uwagent/MA/UWAgent.new
URL : file:/home/uwagent/MA/UWAgent.new/
ip = mnode1/10.1.0.1, time = 1092030776708, ID = 0
file = ./CascadeTest.class
byteArrayClass.length = 1236
end of UWInject (main)

0: Hello from agent 0
0: Spawning child 0.0
0: Spawning child 0.1

0.0: Hello from agent 0.0
0.0: Hopping to mnode2
0: CascadeTest exiting


0.1: Hello from agent 0.1
0.1: Hopping to mnode2
0.0: CascadeTest2 exiting

AgentThread for CascadeTest2 ending
0.1: CascadeTest3 exiting
```

```
AgentThread for CascadeTest3 ending
AgentThread for CascadeTest ending
```

Output from mnode2:
The output from mnode2 shows each child agent waiting. Child 0.0 then hops to mnode3, and child 0.1 hops to mnode4. The AgentThreads for mnode2's UWPlace then end.

```
0.0: Hello from agent 0.0
0.0: Waiting...

0.1: Hello from agent 0.1
0.1: Waiting...
0.0: Hopping to mnode3
0.1: Hopping to mnode4
AgentThread for CascadeTest2 ending
AgentThread for CascadeTest3 ending
```

Output from mnode3:
The output from mnode3 shows agent 0.0 waiting (to ensure that agent 0.1 has finished migrating), then attempting to send a message to agent 0.1. The talk() function returns a success flag. The invalid message log issue has been discussed in the 07/26/04 status report and in e-mail.

```
0.0: waiting...
0.0: Sending message
talk to agentId: 0.1
0.0 sends message to 0.1 =)
0.0: Succeeded
0.0: CascadeTest2 waiting...
INVALID MESSAGE LOG:
Sending Agent ID: 0.1
Receiving Agent ID: 0.0


AgentThread for CascadeTest2 ending
```

Output from mnode4:
The output from mnode4 is symmetric to the mnode3 output, but for agent 0.1 instead of agent 0.0.

```
0.1: waiting...
0.1: Sending message
talk to agentId: 0.0
INVALID MESSAGE LOG:
Sending Agent ID: 0.0
Receiving Agent ID: 0.1

0.1 sends message to 0.0 =)
0.1: Succeeded
0.1: CascadeTest3 waiting...
AgentThread for CascadeTest3 ending
```

Termination Request

In this test, the CascadeTest agent calls setTerminationRequest when it reaches the end of its user code. Therefore, its children (CascadeTest2 and CascadeTest3) terminate instead of executing their next functions (sending messages to each other).

Output from mnode1:
The output from mnode1 is similar to the output in the first test, with the addition of the setTerminationRequest call.

```
File : /home/uwagent/MA/UWAgent.new
URL : file:/home/uwagent/MA/UWAgent.new/
ip = mnode1/10.1.0.1, time = 1092031476129, ID = 0
file = ./CascadeTest.class
byteArrayClass.length = 1344

0: Hello from agent 0
0: Spawning child 0.0
0: Spawning child 0.1
end of UWInject (main)

0.0: Hello from agent 0.0
0.0: Hopping to mnode2

0.1: Hello from agent 0.1
0.1: Hopping to mnode2
0.1: CascadeTest3 exiting

AgentThread for CascadeTest3 ending
0.0: CascadeTest2 exiting

AgentThread for CascadeTest2 ending
0: calling setTerminationRequest
0: CascadeTest exiting

AgentThread for CascadeTest ending
```

Output from mnode2:
The output from mnode2 is the same as for the first test, since the termination request does not interrupt the function that is in progress (nextHop()).

```
0.1: Hello from agent 0.1
0.1: Waiting...

0.0: Hello from agent 0.0
0.0: Waiting...
0.1: Hopping to mnode4
0.0: Hopping to mnode3
AgentThread for CascadeTest3 ending
AgentThread for CascadeTest2 ending
```

Output from mnode3:
The output from mnode3 shows that CascadeTest2 immediately exits, since its terminationRequest flag was set by its parent, CascadeTest.

```
AgentThread for CascadeTest2 ending
```

Output from mnode4:
The output from mnode3 shows that CascadeTest3 immediately exits, since its terminationRequest flag was set by its parent, CascadeTest.

```
AgentThread for CascadeTest3 ending
```

Duncan Smith
CSS499 Summer Quarter 2004
Enhancing the UWAgent Execution Engine

Status Report 08/16/2004

Status Summary

This status report explains the Agent Scheduling enhancement to the UWAgent system. Each UWPlace now has a Scheduler, which the AgentTeamwork Sentinel (or other UWAgents programs) may use to run jobs in a scheduled environment. As part of the scheduling mechanism, the Scheduler provides a resource status reporting function. This allows the Sentinel to make decisions about where to run user processes based on conditions at the current UWPlace (CPU and memory load, number of agents currently residing at the UWPlace, and number of user jobs currently executing).

The code is located in **/home/uwagent/MA/UWAgent.new** (in the **uwagent** account's home directory). The following files have been modified or added since the previous status report: UWPlace.java, ScheduleTest.java, ScheduleTestUser.java, and Status.java.

Implementation Details

The Sentinel agent (here represented by an agent called ScheduleTest) runs a user process by passing it (as a Thread) to the UWPlace.submit() function. Each UWPlace has a Scheduler associated with it. The Scheduler runs the thread according to a scheduling policy. Currently, only FIFO scheduling is implemented. This scheduling mechanism works as follows: The first thread to be submitted is allowed to run for a given time quantum. Control is then returned to the caller, which may decide to continue running the thread (user process), submit another user process, or migrate to another UWPlace based on conditions at the current UWPlace. If the caller submits another user process, it is placed at the end of the job queue, and does not execute until the first process is finished.

Thread.suspend() is used to suspend a process in order to return control to the caller. If the caller decides to continue running the process, then Thread.resume() is used to resume it. Calling UWPlace.submit(null) instructs the scheduler to run processes that are already in the job queue, without adding any new ones.

Source Code

All of the source code is available in **/home/uwagent/MA/UWAgent.new**. ScheduleTest.java is used to test the Scheduler functionality. ScheduleTestUser.java is the user program that is run in the scheduler environment. These two programs are shown below:

```
// ScheduleTest
import java.io.*;
import java.rmi.*;
import java.util.*;
import java.text.*;
```

```
public class ScheduleTest extends UWAgent implements Serializable {
	public ScheduleTest() {}

	private void printMsg(String message) {
		if (message.equals("")) {
			System.out.println();
		} else {
			System.out.println(getAgentId() + ": " + message);
		}
	}

	public void sleepNow(int sleepTime) {
		try {
			Thread.sleep(sleepTime);
		} catch (InterruptedException e) {}
	}

	public void init() {
		UWPlace place = getPlace();
		Status status;

		// Submit a user job using the scheduler
		ScheduleTestUser userJob = new ScheduleTestUser();
		ScheduleTestUser userJob2 = new ScheduleTestUser();

		printMsg("Submitting job 1");
		place.submit(userJob);
		printMsg("Submitting job 2");
		place.submit(userJob2);

		printMsg("");
		printMsg("Machine status:");
		status = place.getStatus();
		DecimalFormat pctFormat = new DecimalFormat("###.#%");
		String memUsage = pctFormat.format(status.memUsage);

		System.out.println("Memory Usage = " + memUsage);
		System.out.println("Number of agents = " + status.numAgents);
		System.out.println("Number of user jobs = " + status.numUserJobs);

		printMsg("Running job 1 to completion");
		do {
			status = place.submit(null);
		} while (!status.jobCompleted);


		printMsg("");
		printMsg("Running job 2 to completion");
		do {
			status = place.submit(null);
		} while (!status.jobCompleted);

	}
}


// ScheduleTestUser
import java.io.*;
import java.rmi.*;
import java.util.*;
import java.math.*;

public class ScheduleTestUser extends Thread {
	public ScheduleTestUser() {}

	public void doWork() {
		double num1, num2, result;

		Random generator = new Random();
		int i=0,j=0;
		for (i=0; i<10; i++) {
```

```
                        System.out.println("i = " + i);
                        for (j=0; j<700000; j++) {
                                num1 = generator.nextDouble();
                                num2 = generator.nextDouble();
                                result = num1/num2;
                        }
                }
        }

        public void run() {
                System.out.println("Schedule Test User Program starting");
                doWork();
                System.out.println("Schedule Test User Program exiting");
        }
}
```

Execution Output

The output below shows ScheduleTest running on a single node. It submits the first instance of ScheduleTestUser (first bolded line, "Submitting job 1"). This program simply performs 10 sets of floating point operations, printing the number of each set. After the fourth set (i = 4), the scheduler returns control to the caller (ScheduleTest). At this point, ScheduleTest submits a second instance of ScheduleTestUser ("Submitting job 2"). However, notice that the numbers continue at i = 5, rather than starting at i = 0. This is because the FIFO scheduling policy requires that the first ScheduleTestUser instance complete. The second instance is placed at the end of the queue.

After ScheduleTestUser instance #1 executes for another time slice (i = 4 to 8), ScheduleTest prints out some statistics about the current UWPlace. It then finishes executing instance #1 (i = 9). After this, instance #2 is allowed to run from start to finish, since it is the only user job left.

```
[uwagent@mnode1 UWAgent.new]$ java UWInject localhost ScheduleTest -m 10
File : /home/uwagent/MA/UWAgent.new
URL : file:/home/uwagent/MA/UWAgent.new/
ip = mnode1/10.1.0.1, time = 1092635122693, ID = 0
file = ./ScheduleTest.class
byteArrayClass.length = 1856
0: Submitting job 1
Schedule Test User Program starting
i = 0
end of UWInject (main)
i = 1
i = 2
i = 3
[uwagent@mnode1 UWAgent.new]$ i = 4
0: Submitting job 2
i = 5
i = 6
i = 7
i = 8

0: Machine status:
Memory Usage = 40.516%
Number of agents = 1
Number of user jobs = 2
0: Running job 1 to completion
i = 9
Schedule Test User Program exiting

0: Running job 2 to completion
Schedule Test User Program starting
```

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
Schedule Test User Program exiting
AgentThread for ScheduleTest ending

[uwagent@mnode1 UWAgent.new]$
```