Creating programs to run under the Simulator

Version 1.0 (07/15/05)

Jonathan Misurda (jmisurda@cs.pitt.edu)

Contents:

1. Creating a Simple Program

Creating programs to run on a single PIM in the simulator begins by implementing the common interface as shown below:

```
public interface SimRunnable
{
      public int SimRunnableMain(SimPIM argPIM);
}
```

To create a specific program, create a class that implements `SimRunnable` and fill out the body to `SimRunnableMain`, the equivalent of a main method for the program.

`SimRunnableMain` returns an integer, which is hijacked by Kaffe to return the length of time that the program ran (cycle count.) Whatever value you return will be lost, so the `return 0` is just a formality to satisfy the Java Compiler.

`SimRunnableMain` has one parameter, a reference to the PIM that this program is executing on. Through this object, the program can send and receive messages.

SimPIM currently exposes three methods for programs to use.

```
public void Sim_send(SimEvent argEvent, long argTime)
public SimPacket Sim_receive()
public void Sim_Exit(long argTime)
```

`Sim_receive()` and `Sim_Exit()` are meant only for defining PIM "operating system" (resident) programs. The ability to define custom OS programs is forthcoming.

`Sim_send()` allows a program to post a message.

Below is an example program:

```
public class SimHelloWorld implements SimRunnable {

	public int SimRunnableMain(SimPIM argPIM)
	{
		//Your program goes here
		int useless_math = 0;
		useless_math++;
		PIM_instrs.hello();
		useless_math--;

		//Create an output packet
		SimPacket lclOutputPacket = new SimPacket();
		lclOutputPacket.setInstruction(SimPacket.EXECUTE);
		lclOutputPacket.setCodePayload(new SimHelloWorld());
		argPIM.Sim_send(
			new SimPacketArriveEvent(
				lclOutputPacket,
				1,
				argPIM ),
			1
		);

		//Hijacked by Kaffe to return the time this program took
		return 0;
	}
}
```

This program does some native Java code, calls a PIM instruction (See Section 2) and then creates a simple output packet which it sends via `Sim_send()`.

To avoid many CLASSPATH problems, make your custom program part of the `com.cogent` package.

2. Defining new PIM instructions

All PIM instructions must be defined in the `PIM_instrs` class found in the `com.cogent.instructions` package.

```
public class PIM_instrs {

	public static void hello() {
		System.out.println("Hello, World!");
	}
}
```

Add any new instructions as other public static methods to this class. They can return anything and take any parameters; there is no method signature requirement.