The Jpi interface

Paul Cockshott

October 27, 2003

The purpose of this is to describe a java interface loosely modeled on the primitives of the Pi calculus[1] to be used as a substratum for GRID based parallel computing. It has to allow the creation of processes and communications channels between processes. It also has to allow for the communications network between processes to be dynamically reconfigurable. The aim of the design is to achieve this with the minimum number of primitives and to integrate these primitives into the existing Java class framework.

1 Class Hierarchy

The overall class hierarchy is given below:

1. abstract class JPITask implements java.lang.Runnable
2. interface JPI
    (a) class UniprocessorJPI implements JPI
    (b) class GridJPI implements JPI
3. abstract class JPIInputChannel implements java.io.InputStream, java.io.serializable
4. abstract class JPIOutputChannel implements java.io.OutputStream, java.io.serializable
5. interface JPIPipe
    (a) class UniprocessorJPIPipe implements JPIPipe
    (b) class GridJPIPipe implements JPIPipe,, java.io.serializable;

Let us look at each of these in more detail.

1.1 abstract class JPITask

This repesents the unit of work to be done as a parallel process. It implements the java.lang.Runnable interface. This allows the construction of threads from JPITasks. It has additional methods that allow channels to be associated with JPITasks.

```
public abstract class JPITask implements Runnable{
public abstract void run();
public abstract void setInputs(JPIInputChannel [] chans);
public abstract void setOutputs(JPIOutputChannel [] chans);
public abstract JPIInputChannel[] getInputs();
public abstract JPIOutputChannel[] getOUtputs();
}
```

Before the run method is called the input and output channels must have been initialised. When the task runs, the input and output channels can be obtained by the running task using the get methods.

1.2 interface JPI

This is a factory interface whose job it to create processes, pipes and channels.

```
interface JPI{
boolean fork(JPITask job,double time, double memory);
JPIPipe createPipe();
JPIInputChannel createInput(InputStream stream);
JPIOutputChannel createOutput(OutputStream stream);


}
```

When fork is called, the job is run either locally or remotely depending on availablity of resources. The time and memory parameters specify the anticipated resource usage of the task. The time is the number of seconds that the job should take on the *current* cpu. The memory parameter gives the number of bytes of additional java virtual machine memory required to run the task.

The create input and create output channel methods convert streams in the local environment to JPI streams that can be shipped to remote machines. If one creates an input channel from an input stream, or an output channel from an output stream the mechanism for allowing local files to be read or written by remote tasks. Alternatively one can create a pipe, pass the input end to the remote process and use the output end of the pipe to write data to the remote task. The reverse configuration allows the spawning task to get results back from the remote task.

The create pipe method allows interprocess communication streams to be set up between two daughter tasks.

1.2.1 class UniprocessorJPI

This will implement all of the methods using local resources, and will run the the tasks in local threads.

1.2.2 class GridJPI

This will be built upon some form of Grid protocols and will transfer the class instance to be run to a remote machine to be run.

1.3 abstract class JPIInputChannel

This at the base level requires the implementation of a method to get a byte (read). The standard Java blocking protocols are followed.

1.4 abstract class JPIOutputChannel

This must provide a write method that writes a single byte to the output channel. It can block on write if the remote process has not done sufficient reads.

2 interface JPIPipe

This provides a bi-directional inter task communications channel. This has two methods

```
interface JPIPipe{
JPIInputChannel getInputEnd();
JPIOutputChannel getOutputEnd();

}
```

To set up a data flow graph one creates pipes and then passes the input and output channels to tasks which are then forked.

2.0.1 class UniprocessorJPIPipe

This will be implemented using the built in Java classes PipedInputStream and PipedOutputStream.

2.0.2 class GridJPIPipe

This will be implemented using some communications protocol. One has to take care here that the pipe is serialisable.

References

[1] "The Polyadic π-calculus, a tutorial", Milner, R., 1991.