GPars - Groovy Parallel Systems

Jon Kerridge

Version 1.0, 2015-10-29

Table of Contents

More Testing: Non-terminating Process Networks

Chapter 6 showed it is possible to use the **GroovyTestCase** capability to test networks of processes, provided each of the processes in the network terminates. Most of the processes used in this book do not terminate and so a means of testing such non-terminating process networks has to be developed.

A PAR Process

First, however, we need to reflect on the operation of PAR. A PAR only terminates when all the process in the list of processes passed to it terminate. Thus, if only one of the processes does not terminate, then the PAR will never terminate. However, if the assertion testing commonly used in **JUnit** and **GroovyTestCase** is to be undertaken then at least some of the test environment has to terminate. Figure 10-1 shows a generic architecture that allows a process network under test (**PNUT**) to run without terminating, while the Test-Network does terminate, which then allows the assertion testing to take place in the normal manner.

Process Separation

The separation of the **PNUT** from the Test-Network by means of a **TCP/IP** communications network, means that the two process networks run independently of each other and it does not matter if the **PNUT** Process-Network-Under-Test does not terminate, provided the Test-Network does. We can assume that the **PNUT** requires input and also that it outputs results in some format. This data is communicated by means of the network channels shown.

TCP/IP allows process networks to run independently

Both the Input-Generator and Output-Gatherer processes must run as a **PAR** within the process Test-Network, then terminate; after which their internal data structures can be tested within Assertion-Testing. This demonstrates the generic nature of the architecture in that the only part that has to be specifically written is the processes that implement the Input-Generator and Output-Gatherer respectively. The architecture will now be demonstrated using the Scaling Device example described previously in Chapter 6. The Scaling Device takes a stream of input numbers and outputs an equivalent stream of scaled numbers, while monitoring the operation of a **Scale** process by modifying the applied scaling factor.

Want to read more of this chapter? Download this chapter's PDF here.