GPars - Groovy Parallel Systems

Jon Kerridge

Version 1.0, 2015-10-29

Table of Contents

Redirecting Channels: A Self Monitoring Process Ring

In `Chapter 18`, it was shown how `Mobile Agents` can be constructed using serializable **CSProcesses**. In `Chapter 10`, a solution was developed to the problem of a ring of processes that circulated messages around themselves. No consideration was given to the problems that might happen if messages were not taken from the ring immediately.

The Solution

In this chapter, we explore a solution to the problem that utilises two mobile agents that dynamically manage the ring connections in response to a node of the ring detecting that incoming messages are not being processed sufficiently quickly.

The solution, as presented, does not require any form of central control to initiate the corrective action. The agents are invoked by the node when it is determined that the processing of incoming messages has stopped. The solution essentially builds an `Active Network` at the application layer, rather than the usual normal network layer. The solution also utilises the `Queue and Prompter` processes developed in `Chapter 5`. These provide a means of providing a finite buffer between the ring node process and the process receiving the messages. Additionally, a console interface has been added to the message receiver process, so that users can manipulate its behaviour and more easily observer the effect that the agents have on the overall system operation.

Architectural Overview

`Figure 20-1` shows the process structure of one node and also its relationship to its adjoining nodes. It is presumed that there are other nodes on the ring all with the same structure. It shows the state of the system once it has been detected that `RingElement -n` has stopped receiving messages. The net channel connections joining `RingElement –n` to the ring have been removed and replaced by the connection that goes between `RingElement n-1` and `RingElement n+1`.

The figure also shows the additional processes used to provide the required management. The **RingElement** outputs messages into the **Queue** process, instead of directly into the **Receiver** process. The **Prompt** process requests messages from the **Queue** which it passes on to the **Receiver** process. Whenever the **Queue** process is accessed, either for putting a new message or getting a message in response to a **Prompt8 request, the number of messages in the *Queue** is output to the `StateManager` process. The `StateManager` process is able to determine how full the **Queue** is and depending on pre-defined limits will inform the `RingElement` that the **Receiver** has stopped inputting messages or has resumed. This will be the trigger to send either the **StopAgent** or the **RestartAgent** around the network.

The **Queue** contains sufficient message slots to hold two messages from each node. The signal to indicate that the receiver has stopped inputting messages is generated by the **StateManager** when the

Queue is half full.

A naive solution would just create an infinite queue to deal with the problem and not worry about the fact that the messages were no longer being processed by the **Receiver**. However, this is not sensible because were the situation to be sustained over a long period the processor would run out of memory and would fail in a disastrous manner.

It is thus much better to deal with the situation rather than ignore it. The **Sender** process has been modified in as much that a delay has been introduced so that there is a pause between the sending of a message and the next. This was done so that the operation of the revised system could be more easily observed.