GPars - Groovy Parallel Systems

Jon Kerridge

Version 1.0, 2015-10-29

Table of Contents

Mobile Agents and Processes: Process Discovery

In this chapter, we consider the difference between `mobile agents` and `mobile processes` by offering a defining example because this area is currently very confused as to their precise meaning. Previously, we have used these terms in a manner that reflects these definitions but up till now there has been no need to be specific about their differences because there has been no conflict.

Let's Dynamically Add Network Nodes

In this example, we are going to dynamically add nodes to a network. Nodes that are added may not have all the processes they require; even the initial nodes may not contain a full repertoire of processes. However, we shall assume that all the initial nodes do contain all the required data manipulation processes.

When a node receives a data input for which it does not have the required process, it will send an agent around the currently connected nodes. The goal of the agent is to locate a node with the required process, acquire the process, return to its original node and transfer the process into its home node. Upon receipt of such a process, the home node will dynamically connect it into its internal channel structure and thereby cause the process to execute. Thus the node will now be able to process any further data of the same type.

It can be seen from the above description that the `mobile agent` has a specific goal, or possibly goals that it seeks to achieve. This may require the agent to visit many nodes in order to find the solution to its goal. Once the goal has been achieved, including any return to its home node, it then ceases to exist. Other agents with similar goals may be created but each will have a predetermined life expectancy.

By contrast, a `mobile process` is one that can be moved from one node to another, plugged into the channel structure at the receiving node and then continues to run as part of the node until such time as the node itself is no longer required. In some cases, the `mobile process` may retain the ability to communicate with its original node, as was the case of the `Meeting Organiser` system described in `Chapter 19`.

System Architecture

The system architecture is shown in `Figure 21-1`. The **DataGenerator** process provides a named network input channel that can be connected to by any node, shown as a dotted arrow, thereby creating a networked `Any2One` channel. Similarly, the **Gatherer** process provides a named network input channel that can also be connected to by any node as indicated by the dotted arrow.

On creation, a **NodeProcess** simply needs to be told the names of these channels in order to be connected to both the **DataGenerator** and **Gatherer** processes. Once these connections have been made, the **NodeProcess** creates a number of net input channels as follows :

- The **From-Data-Generator** channel provides a means by which data can be received from the **DataGenerator** by the **NodeProcess**.
- The `Agent Visit Channel` is the channel upon which agents from other nodes will be input so they can interact with this node.
- The `Agent Return Channel` is the channel used by an agent to return to its originating node.

Once theses channels have been created, the **NodeProcess** outputs the location of the `From-Data-Generator` and the `Agent Visit Channel` to the **DataGenerator** using the `Nodes-to-Data-Generator` channel. On receiving these locations, the **DataGenerator** creates a `One2One` channel from it to the node using the `From-Data-Generator` channel location.

Role Of The DataGenerator

The **DataGenerator** maintains a list of all the `Agent Visit Channel` locations, which it outputs to all of the connected **NodeProcesses** whenever the list changes. The **NodeProcess** uses this information to update its **Agent** with the locations of the `Agent-Visit-Channels` that it can use when it searches for a data processing process. In addition, the **Node** also ensures that the **Agent** holds the location of the `Agent-Return-Channel` so that a returning **Agent** knows its home location.

Implementation

Once the system has been invoked, the **DataGenerator** randomly sends data object instances of any type to any of the nodes. If a **NodeProcess** already has an instance of the required data processing process, the data is sent to that process where it is manipulated and subsequently output to the **Gatherer** process. If the node does not have an instance of the required process then it informs the **Agent** of the process it requires and causes the **Agent** to be sent to the first location on its list of `Agent-Visit-Channel` locations. In due course, the **Agent** will return, the new process will be transferred to the **Node** and it will be connected into the *Node8 as described previously.

As soon as a **Node** sends an **Agent** to find a required process, it creates another instance of its **Agent** so that, should another data object arrive for which it does not have the data processing process, then an **Agent** can be sent to find it immediately.

The operation of a **Node** matches the interactions described above. On receipt of an input, it determines if it is a list of `Agent-Visit-Channel` locations and if so updates the **Agent** appropriately. If it is an instance of a data object, it determines its' type and, if it already has an instance of the required process, sends the data object to the required process. Otherwise, it sends the **Agent** the required process type information, which the **Agent** can use when visiting the other nodes. Each of the processes in a **NodeProcess** is invoked using the **ProcessManager** class.

When a process is received by a **NodeProcess**, it creates a channel by which the **NodeProcess** can send data objects to it. All such processes are connected to the `Nodes-To-Gatherer` channel.

Once a **NodeProcess** has received three such processes, its internal architecture would be as shown in `Figure 21-2`, ignoring its **Agent**.

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