GPars - Groovy Parallel Systems

Jon Kerridge

Version 1.0, 2015-10-29

Table of Contents

Conclusion - Why Use *Groovy Parallel* and JCSP ?

At the end of this journey, we are able to reflect on the capabilities that have been described and considered.

We started with four very simple concepts:

- process
- channel
- alternative
- timer

and from these we have been able to construct a wide variety of systems with very different operational requirements and functionality.

CSP

In general, this has been achieved with one fundamental design pattern, the `client-server`, together with a small number of programming idioms that facilitate its use; such as the prompted buffer. The intellectual challenge is realised by understanding how to use this pattern and idioms in an effective manner.

Purpose

In this book, I have purposely avoided the use of any formal representation of the process and how networks of such processes can be analysed using formal tools. I believe that the engineering approach based upon the reuse of a single design pattern, which has its basis in the underlying formalism, is the best way to make software engineers appreciate what can be achieved, provided the capability we are using has a formal basis.

The real world is not populated with sufficient software engineers who have the mathematical skill to be able to undertake the formal analysis of their systems, even with the tools currently available.

More Cores, Please

The increasing availability of multi-core processor based systems is inevitable and the desire to make more effective use of this technology will be an increasing challenge. If the engineers use currently available models and methods then this technology will be increasingly difficult to use. Software engineers, therefore, require a better model with which they can program such systems. But why leave it to just multi-core systems? Why not better and more effective use of network based workstation systems? We might then be able to move to processing systems that makes more effective use of grid-computing because we have a viable model that allows us to interact over any size network.

Review

The content of this book started at a basic undergraduate level and ended with examples of mobile systems that are still the subject of intense research activity (**occam-pi, mobile**). All the examples presented are demonstrable and where necessary operate over a network and employ aspects of mobility. Yet this is achieved in manner that does not require a detailed understanding of the operation of networked systems and in which the definition of a process is not reliant upon whether it is to execute as one among many on a single processor or over a network.

Implementations

The underlying support is provided by **JCSP** and it has been made easier to assimilate by use of **Groovy** because it helps to reduce the amount of code that needs writing. These are of relatively little importance of themselves but it is important that they both utilise the underlying **JVM**.

What is really crucial is that **JCSP** provides an implementation of **CSP**. **CSP** provides the formal framework upon which **JCSP** is implemented and thereby the engineering support for programmers.

A programmer is no longer concerned with a poorly implemented underlying thread model

The Programmer’s Job

The programmers are not concerned with the inner workings of the underlying **JCSP** package because they can reason about their systems at a much higher level. The programmer is no longer concerned with the detailed workings of a poorly implemented underlying thread model, in effect, writing machine code. They can now concentrate on high-level design of the processes at the application layer; confident that, if they use the building blocks correctly and apply one pattern effectively then, the resulting system will operate as expected.

This does not remove the need for testing, which exposes the frailties of a system when exposed to a real operating environment. In this book, we have shown how systems can be tested, albeit in a cumbersome manner but which, with further research and the development of support tools, will make it easier to achieve.

Usage

The final chapters have shown how we can exploit the `process` concept in a more up-to-date setting and how it may address the problems that the software industry is starting to deal with in terms of how to exploit mobility, network connectivity and parallelism effectively. Previously, parallelism has been thought of as providing a solution to the needs of high performance computing, where ultimate speed was the only driving force.

Hopefully, with some of the later examples, in particular, the reader will have been convinced that approaching a solution to a problem from the parallel point of view actually makes it easier to achieve a working and effective solution.

Conclusion

The book ends at the point where the examples have to become real problems and which, of course, tend to be too large to explain within the confines of such a book. Hopefully, however, the book contains sufficient ideas, concepts and capabilities that the solution to larger problems can be broken down into sufficiently small processes that it becomes manageable.

As a final design consideration, I offer the advice that if you are having problems with a design and cannot get the design right then the solution is usually to add one or more processes. If a designer tries to restrict the number of processes then that is usually followed by problems. In the future perhaps, we will get to the situation where team leaders will ask why a serial solution has been adopted rather than one that relies on parallel design methods!

- Jon Kerridge