Applying Functional Programming Theory to the Design of Workflow Engines

Peter M. Kelly

January 2011

A dissertation submitted to the School of Computer Science of
The University of Adelaide for the degree of Doctor of Philosophy

Supervisors:

Dr. Paul D. Coddington

Dr. Andrew L. Wendelborn

Contents

Abstract

Workflow languages are a form of high-level programming language designed for coordinating tasks implemented by different pieces of software, often executed across multiple computers using technologies such as web services. Advantages of workflow languages include automatic parallelisation, built-in support for accessing services, and simple programming models that abstract away many of the complexities associated with distributed and parallel programming. In this thesis, we focus on data-oriented workflow languages, in which all computation is free of side effects.

Despite their advantages, existing workflow languages sacrifice support for internal computation and data manipulation, in an attempt to provide programming models that are simple to understand and contain implicit parallelism. These limitations inconvenience users, who are forced to define additional components in separate scripting languages whenever they need to implement programming logic that cannot be expressed in the workflow language itself.

In this thesis, we propose the use of functional programming as a model for data-oriented workflow languages. Functional programming languages are both highly expressive and amenable to automatic parallelisation. Our approach combines the coordination facilities of workflow languages with the computation facilities of functional programming languages, allowing both aspects of a workflow to be expressed in the one language.

We have designed and implemented a small functional language called ELC, which extends lambda calculus with a minimal set of features necessary for practical implementation of workflows. ELC can either be used directly, or as a compilation target for other workflow languages. Using this approach, we developed a compiler for XQuery, extended with support for web services. XQuery's native support for XML processing makes it well-suited for manipulating the XML data produced and consumed by web services. Both languages make it easy to develop complex workflows involving arbitrary computation and data manipulation.

Our workflow engine, NReduce, uses parallel graph reduction to execute workflows. It supports both *orchestration*, where a central node coordinates all service invocation, and *choreography*, where coordination, scheduling, and data transfer are carried out in a decentralised manner across multiple nodes. The details of orchestration and choreography are abstracted away from the programmer by the workflow engine. In both cases, parallel invocation of services is managed in a completely automatic manner, without any explicit direction from the programmer.

Our study includes an in-depth analysis of performance issues of relevance to our approach. This includes a discussion of performance problems encountered during our implementation work, and an explanation of the solutions we have devised to these. Such issues are likely to be of relevance to others developing workflow engines based on a similar model. We also benchmark our system using a range of workflows, demonstrating high levels of performance and scalability.

Declaration

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution to Peter Kelly and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library catalogue, the Australasian Digital Theses Program (ADTP) and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Peter Kelly

Acknowledgments

Firstly, I would like to express my sincere gratitude to my supervisors, Paul and Andrew, for their invaluable guidance and assistance with my work. Both have been excellent supervisors, and provided a great deal of insightful advice and feedback that has guided my project. Their unique combination of skills — in particular, Andrew's expertise in programming language theory and parallel functional programming, and Paul's background in high performance computing and scientific applications — has played a crucial role in the development of the key ideas behind my project. I am indebted to them for the many ways in which they have contributed to my development as a researcher.

Secondly, to those who have accompanied me on this journey in various ways — Alison, Asilah, Brad, Daniel, Donglai, Jane, Joanna, Joseph, Miranda, Nick, Nicole, Peter, Wei, Yan, and Yuval — thanks for making this whole process a more enjoyable one. I value the many opportunities we've had to exchange thoughts, ideas, and experiences. You have all been a great help to me in many ways, both personally and professionally.

Thirdly, many thanks go to my parents, who have done so much over the years to support and encourage me in my studies, and teach me the value of working hard to pursue my goals. To fully describe the ways in which they have helped me get to where I am today would take another three hundred pages. It is to them that I dedicate this thesis.

Finally, I am very grateful to the School of Computer Science for giving me the opportunity to pursue a PhD, and for providing me with financial support and an excellent working environment in which to carry out my studies. I would also like to acknowledge eResearch SA for providing me with access to the high performance computing facilities necessary to run my experiments.

Related publications

1. Peter M. Kelly, Paul D. Coddington, and Andrew L. Wendelborn. Distributed, parallel web service orchestration using XSLT. In *1st IEEE International Conference on e-Science and Grid Computing*, Melbourne, Australia, December 2005.

2. Peter M. Kelly, Paul D. Coddington, and Andrew L. Wendelborn. A simplified approach to web service development. In *4th Australasian Symposium on Grid Computing and e-Research (AusGrid 2006)*, Hobart, Australia, January 2006.

3. Peter M. Kelly, Paul D. Coddington, and Andrew L. Wendelborn. Compilation of XSLT into dataflow graphs for web service composition. In *6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid06)*, Singapore, May 2006.

4. Peter M. Kelly, Paul D. Coddington, and Andrew L. Wendelborn. A distributed virtual machine for parallel graph reduction. In *8th International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT '07)*, Adelaide, Australia, December 2007.

5. Peter M. Kelly, Paul D. Coddington, and Andrew L. Wendelborn. Lambda calculus as a workflow model. *Concurrency and Computation: Practice and Experience*, 21(16):1999–2017, July 2009.

All of the software developed in this project is available from the following URL, under the terms of the GNU Lesser General Public License:

`http://nreduce.sourceforge.net/`

For Mum & Dad