

Putting the User at the Centre of the Grid: Simplifying Usability and Resource Selection for High Performance Computing

Stefan Joseph Zasada

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of
University College London.

Department of Computer Science
University College London
February 11, 2015

I, Stefan Joseph Zasada, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Specifically:

- I was lead developer of the Application Hosting Environment project, being solely responsible for the development of AHE client, and the design and development of AHE 2.0 and AHE 3.0 server, as well as the Application Interaction Model that underpins AHE.
- I designed and conducted the AHE usability tests presented in this thesis, and the AHE2/AHE3 performance tests.
- I was the sole designer and developer of the RAMP system presented in this thesis.
- In addition, I have been responsible for deploying and integrating AHE within several of the research projects discussed in this thesis, including the VPH Network of Excellence, ContraCancrum, GENIUS and MAPPER.

*“Depend upon it, Sir, when a man knows he is to be hanged in a fortnight,
it concentrates his mind wonderfully.”*

Samuel Johnson.

For Pam and Zygmunt

Abstract

Computer simulation is finding a role in an increasing number of scientific disciplines, concomitant with the rise in available computing power. Realizing this inevitably requires access to computational power beyond the desktop, making use of clusters, supercomputers, data repositories, networks and distributed aggregations of these resources. Accessing one such resource entails a number of usability and security problems; when multiple geographically distributed resources are involved, the difficulty is compounded. However, usability is an all too often neglected aspect of computing on e-infrastructures, although it is one of the principal factors militating against the widespread uptake of distributed computing.

The usability problems are twofold: the user needs to know how to execute the applications they need to use on a particular resource, and also to gain access to suitable resources to run their workloads as they need them. In this thesis we present our solutions to these two problems. Firstly we propose a new model of e-infrastructure resource interaction, which we call the *user–application interaction model*, designed to simplify executing application on high performance computing resources. We describe the implementation of this model in the Application Hosting Environment, which provides a Software as a Service layer on top of distributed e-infrastructure resources. We compare the usability of our system with commonly deployed middleware tools using five usability metrics. Our middleware and security solutions are judged to be more usable than other commonly deployed middleware tools.

We go on to describe the requirements for a resource trading platform that allows users to purchase access to resources within a distributed e-infrastructure. We present the implementation of this Resource Allocation Market Place as a distributed multi-agent system, and show how it provides a highly flexible, efficient tool to schedule workflows across high performance computing resources.

Acknowledgements

The work presented herein would not have been possible were it not for the inspiration, wisdom and support given to me by a great many people and I take pleasure in showing my appreciation to them.

Firstly, I am immensely grateful for the support, guidance and help of my supervisor, Peter Coveney. In addition to supervising this thesis, Peter has been a great mentor to and influence over me during the last eight years, and I am indebted to him for the opportunities that he has given me. I would also like to thank my secondary supervisor, Licia Capra, for useful pointers and conversations when embarking on this work.

This thesis would not have been completed without the unwavering love, support and patience of my wife Laura, daughter Ruby, father Zygmunt and sister Kathryn. Without the encouragement and succour given by my late mother Pam, my life would not have embarked on this path.

It is a great honour to have worked with members of the Centre for Computational Science at UCL, and I am also proud to have been part of such a distinguished group. My CCS colleagues past and present, too numerous to name, have given me a great deal of help, support and encouragement in this endeavour, and for that they have my thanks.

I would like to thank EPSRC for funding my PhD studentship under the RealityGrid project (GR/R67699). The developments of Application Hosting Environment reported in this thesis have also been supported by the EU FP7 *VPH-Share* (no 269978), *VPH-NoE* (no 223920), *MAPPER* (no 261507) and *ContraCancrum* (no 223979) projects, the EPSRC *Rapid Prototyping of Usable Grid Middleware* (GR/T27488/01) grant, and also by OMII under the Managed Programme *Robust Application Hosting in WSRF::Lite (RAHWL)* project. Finally, I would like to thank the many excellent collaborators I have been privileged to work with on these projects.

Contents

Declaration	2
Abstract	4
Acknowledgements	5
Table of Contents	6
List of Figures	13
List of Tables	17
List of Listings	19
List of Publications	20
1 Introduction	22
1.1 Context	22
1.2 Motivating Example	23
1.2.1 Traditional HPC Use	23
1.2.2 Harnessing the Power of HPC via a Distributed Computational e-Infrastructure	24
1.2.3 Gaining Access to HPC Resources	25
1.2.4 Uptake of Grid Techniques on HPC e-Infrastructures	26
1.3 Problem Statement	26
1.3.1 Research Premise	28
1.4 Contributions Made by This Work	29
1.5 Scope of this Thesis	30
1.6 Evaluation	32

1.6.1	Evaluation of Application Interaction Model	32
1.6.2	Experimental Validation of the Resource Allocation System	33
1.7	Overview of Remaining Chapters	33
2	High Performance Computing on Distributed e-Infrastructures	35
2.1	The Computational Ecosystem	35
2.1.1	Applications of High Performance Computing	37
2.1.2	Access Modalities & Resource Use	39
2.1.3	Classes of HPC Applications	40
2.2	Distributed Computing	42
2.2.1	Grid Computing	43
2.2.2	Web Services	44
2.2.3	Service Oriented Architectures	45
2.2.4	WSRF	46
2.2.5	Representational State Transfers	46
2.3	Middleware	47
2.3.1	Open Grid Services Architecture	47
2.3.2	Globus	49
2.3.3	UNICORE	49
2.3.4	gLite	50
2.3.5	QoS-CoS-Grid	50
2.3.6	Condor	50
2.3.7	SAGA	51
2.3.8	Data Transfer	51
2.3.9	Security Mechanisms	52
2.4	Grid Usability	52
2.4.1	Lightweight Grid Middleware	53
2.5	Scientific Workflow Tools	53
2.6	Cloud Computing	55
2.6.1	Virtualization	56
2.7	High Performance Computing and Distributed e-Infrastructures	57
2.7.1	Distributed Application Requirements	58

2.8	On Demand Access Mechanisms	59
2.9	Summary	61
3	Virtualizing Access to Scientific Applications	62
3.1	HPC Grid Use Cases	62
3.1.1	Calculating Drug Binding Affinities	62
3.1.2	Computational Investigations of Cranial Haemodynamics	65
3.1.3	Single Grid Application Launching	68
3.2	Analysing the Use Cases	69
3.3	Review of Current Middleware Approaches	70
3.4	Requirements	72
3.5	Meeting the Requirements	74
3.6	Service Oriented Computational Science	75
3.6.1	Design Constraints	76
3.7	The Application Interaction Model	78
3.8	The Application Hosting Environment	79
3.9	Modelling Applications as Services	81
3.10	Implementing the Application Interaction Model as WSRF Services	82
3.10.1	The Workflow of Launching an Application	85
3.10.2	Implementation	86
3.11	AHE 2.0 Deployment	88
3.12	Implementing the Application Interaction Model as RESTful Service	89
3.12.1	AHE 3.0 Application Life Cycle	94
3.12.2	Deployment of AHE 3.0	96
3.13	AHE 3.0: Comparison with AHE 2.0	97
3.14	AHE Client Tools	99
3.15	Common Core Foundations	100
3.15.1	Security	101
3.16	Building on Other Middleware Tools	103
3.16.1	Computational Steering	104
3.16.2	Geographically Distributed Parallel Computing	105
3.16.3	Advanced Co-reservation of Resources	106

3.16.4	Standards Compliant Submission	109
3.17	Summary	111
4	Usability Evaluation of the Application Hosting Environment	112
4.1	Evaluating Usability	112
4.2	Usability Study Objectives	113
4.3	Study Methodology	114
4.3.1	Participants	115
4.3.2	Tasks	116
4.3.3	Data Collection	117
4.3.4	Delivery	117
4.4	Results	118
4.5	Discussion of Results	120
4.6	Community Uptake of the AHE	123
4.6.1	Hosted Applications	123
4.7	Case Studies	125
4.7.1	RealityGrid and Materials Science	125
4.7.2	The Virtual Physiological Human Initiative	126
4.7.3	ImmunoGrid	129
4.7.4	ViroLab	129
4.7.5	MAPPER	131
4.8	VPH Share	133
4.9	Conclusions	134
4.10	Summary	135
5	Agents, Metascheduling and Mechanism Design	137
5.1	Resource Allocation	137
5.1.1	Condor	139
5.1.2	Nimrod/G	142
5.1.3	Gridbus	143
5.1.4	EMPEROR	144
5.1.5	XML-Based Policy Framework in EZGrid	145
5.1.6	Community Scheduler Framework	145

5.1.7	Kalman Filter Based Resource Scheduling	146
5.1.8	Conservative Scheduling using Predicted Variance	147
5.1.9	Comparisons of Metascheduling Approaches	148
5.2	Auctions	151
5.2.1	Take it or Leave it Auction	152
5.2.2	Sealed Bid Auction	152
5.2.3	English Auction	152
5.2.4	Dutch Auction	152
5.2.5	Vickrey Auction	152
5.3	Optimum Auction Design	153
5.3.1	Revenue Equivalence Theorem	153
5.4	Double Auction	154
5.5	Reverse Auction	154
5.6	Multi-attribute Auctions	155
5.7	Combinatorial Auctions	155
5.8	Software Agents	156
5.8.1	Multiagent Systems	157
5.9	Agents and Distributed e-Infrastructures	158
5.10	Computational Mechanism Design	159
5.11	Summary	160
6	Design and Implementation of a Resource Allocation Market Place	161
6.1	Designing a Resource Allocation System	161
6.1.1	Design Constraints	163
6.1.2	Functional Specification	163
6.2	The Architecture of the RAMP System	164
6.2.1	Developing the Negotiation Protocols	166
6.2.2	Specifying a Quotation Request	172
6.3	Implementation	175
6.3.1	User Agent	176
6.3.2	Resource Agent	178
6.3.3	Banking Agent	183

6.3.4	Inter-agent Communication	184
6.4	System Integration	186
6.5	Summary	188
7	Functional and Performance Testing of the RAMP System	190
7.1	Evaluating the System	190
7.2	Simulation Environment Setup	192
7.3	Using the RAMP System	193
7.3.1	Results	193
7.3.2	Discussion of Results	197
7.4	Investigating Job Pricing	198
7.4.1	Discussion of Results	201
7.5	Investigating RAMP Performance	202
7.5.1	Results	202
7.5.2	Discussion of Results	206
7.6	Summary	207
8	Overall Conclusions and Future Work	208
8.1	Contributions	208
8.2	Conclusions	208
8.3	Future Work	211
A	Request for Quotations XML Schema	213
B	Inter-agent Communication Ontology	217
C	AHE Usability Handouts	233
	Participant Instructions	233
D	AHE Usability User Manuals	249
	Globus Toolkit Manual	249
	Unicore Manual	252
	AHE Graphical Client Manual	255
	AHE Command Line Tools Manual	260

ACD Manual	263
Configuring AHE Security Guide	267
Glossary	269
Bibliography	274

List of Figures

2.1	The Branscomb Pyramid, showing the relative abundance of computational resources.	37
3.1	The Application Interaction Model underpinning AHE. The Application Instance is the central entity representing each instance of an application that a user launches. All user interaction is mediated via the Application Instance, which supports operations to launch, monitor and terminate the application, and to manage data sharing.	80
3.2	The architecture and message flow of the Application Hosting Environment. AHE 2.0 server builds a layer of middleware between the user and the grid to hide much of the complexity of launching grid applications.	85
3.3	The layered architecture of AHE 2.0. The AHE 2.0 client, running on a user's desktop machine, interacts with the AHE 2.0 services running in a Tomcat Web services container, and also stages data via GridFTP or WebDAV. The AHE 2.0 services interact with job submission components such as OGSA-BES or Globus GRAM in order to submit applications to distributed grid resources. AHE 2.0 also has the ability to interact with HARC and RealityGrid steering services.	87
3.4	The architecture of the AHE 3.0 server, showing the relationship between the different software modules.	90
3.5	The application life cycle. AHE 3.0 server manages the transition of an application instance through a number of states, in order to stage data, execute an application, and handle failures.	94
3.6	Comparison of the mean time required to submit using AHE 2.0 and AHE 3.0 for batches of 10 to 200 jobs. Bars show the standard deviation of the result mean.	99

- 3.7 The AHE graphical client has been extended to support advanced reservation, for example allowing users to (a) create new reservations, (b) launch an MPIg application into an existing reservation. 107
- 3.8 The AHE 2.0 graphical client gives the user the ability to (a) query the registry of launched applications and (b) monitor and steer a running application. 108
- 3.9 The use of standards compliant job submission protocols means AHE can act as a client to many different e-infrastructures and local computational resources, giving the user a single, consistent interface to a wide range of different resources. 110
- 4.1 (a) Mean time taken to complete a range of tasks with each tool; (b) a comparison of the percentage of users who were satisfied with a tool and the percentage who could successfully use that tool. 121
- 4.2 The architecture of the Individualized MEDicine Simulation Environment (IMENSE) system. Access to the different components is mediated through a web portal interface, and includes access to high performance computing (HPC) infrastructures such as DEISA to execute simulation codes, mediated by the AHE. 128
- 4.3 The role of AHE in the MAPPER architecture was to act as an interoperability layer, allowing simulation components to be submitted to resources running a range of back end middleware interfaces, in a way that was transparent to the user. 133
- 6.1 The sequence of FIPA messages between a user and a resource in an auction negotiation with $n=1$ rounds of bidding. 169
- 6.2 The sequence of FIPA messages between a resource and a banking agent required to notify the bank of a successful auction. 170
- 6.3 The sequence of FIPA messages between a user, a resource and a banking agent when a job is cancelled. 171
- 6.4 The hierarchy of behaviours implemented by the User agent. 177
- 6.5 The User agent graphical user interface. 179
- 6.6 The hierarchy of behaviours implemented by the Resource agent. 180

- 6.7 The hierarchy of behaviours implemented by the Bank agent. 183
- 6.8 The sequence of messages between a user agent and three resources required to implement a single round of bidding for a single unit auction. Inter-agent communication is performed using messages specified in the FIPA protocol. In this example, resource agents 1 and 2 respond to the RFQ with offers to accept the workload, while agent 3 refuses. The user agent accepts the offer from agent 2 using a two phase commit. . . 184
- 6.9 A schematic representation of the relationship between concepts in the inter-agent communication ontology. 187
- 6.10 The sequence of operations between the AHE client (containing the user agent), the AHE server, and a single resource, required to launch an application on that resource. 188
- 6.11 Interaction between RAMP system and AHE to schedule application execution on distributed e-infrastructure resources. 189
- 7.1 Request, offer and winning prices for the requests shown in table 7.3 for three rounds of bidding. Where bidding round values are not shown, no offers were made in that round. Experiment run 11, which failed, is not shown. 194
- 7.2 The winning resources for the auctions of workloads listed in table 7.3. Five resources were allocated all of the jobs submitted. 196
- 7.3 Resource *attractiveness* plotted over time. Attractiveness is analogous to the value by which a resource is willing to reduce its prices. 197
- 7.4 The change in resource offer prices over time for four resources in the system. 199
- 7.5 Comparison of the average offer price made by each resource to its minimum price and start price. 200
- 7.6 Plot showing how the increase in number of bidding rounds affects final cost price and auction completion time. 201

- 7.7 Plot showing how the increase in units per auction affect system performance. As the number of units increase, the time taken to complete the auction scales linearly. Bars show the standard deviation of the result mean. 203
- 7.8 Plot showing how the increase in competing agents affects mean system response time. The response time scales linearly with the number of concurrent User Agents. Bars show the standard deviation of the result mean. 204
- 7.9 Plot showing how the increase in competing agents affects mean auction round duration. The round duration rises sharply up to 10 simultaneous users, then tails off. Bars show the standard deviation of the result mean. 204
- 7.10 Plot showing how the increase in competing agents affects mean system response time with RAMP system deployed across a network of resources. Bars show the standard deviation of the result mean. 205
- 7.11 Plot showing how the increase in competing agents affects mean auction round duration with RAMP system deployed across a network of resources. Bars show the standard deviation of the result mean. 206

List of Tables

2.1	Decline in cost of CPU time on the UK HPCx machine over its lifetime	38
2.2	Decline in cost of CPU time on the UK HECToR machine over its lifetime	38
3.1	Average time taken to submit jobs using AHE 2.0 and AHE 3.0	98
4.1	Summary of statistics collected during usability trials for each tool under comparison.	118
4.2	The mean scores and <i>t</i> -test <i>P</i> -values for our five usability metrics, comparing the AHE and Globus command line clients, the AHE and UNICORE graphical clients, and the AHE with and without ACD. . . .	120
4.3	AHE downloads from RealityGrid.org and Sourceforge.org to January 2014. Note a standalone client release for AHE 3.0 has not been provided.	123
5.1	Comparison of features of different e-infrastructure metaschedulers . .	148
6.1	Resource requirement attributes from several job description languages	173
7.1	Parallel Workload Archive Project log files used in the simulation environment.	190
7.2	Simulation environment machine setup. The time point is the number of seconds within the log at which the test system started. All systems were started at a minimum of 50000 seconds into the machine log file in order to allow the load on the machine to reach a production level (some machine logs commence with the machine being turned on, meaning that initially the queue is empty).	192
7.3	Details of experimental workloads run on the RAMP simulation environment.	195
7.4	Mean round and winning prices for the auctions of the workloads listed in table 7.3.	195

7.5 Offer price and auction duration as the number of bidding rounds increases.	200
---	-----

Listings

3.1	JDD job description to run the HemeLB code	71
6.1	Sample Request For Quotation Document	175
A.1	Request For Quotation XML Schema	213
B.1	Inter-agent communication ontology in OWL format	217

List of Publications

The work presented in this thesis has been published in the following conference and journal articles:

- S. J. Zasada, D. C. W. Chang, A. N. Haidar, and P. V. Coveney, Flexible composition and execution of large scale applications on distributed e-infrastructures, *Journal of Computational Science*, 5(1):51–62, 2014
- S. J. Zasada and P. V. Coveney, Distributed biomedical computing, in P. V. Coveney, V. Daz-Zuccarini, P. Hunter, and M. Viceconti, editors, *Computational Biomedicine*, chapter 9, Oxford University Press, Oxford, 2014
- S. J. Zasada, D. Chang, A. N. Haidar, and P. V. Coveney, A lightweight platform for managing biomedical simulation, in *Proceedings of the 3rd international workshop on Emerging computational methods for the life sciences*, pages 75–78, ACM, 2012
- S. J. Zasada, M. Mamonski, D. Groen, J. Borgdorff, I. Saverchenko, T. Piontek, K. Kurowski, and P. V. Coveney, Distributed infrastructure for multiscale computing, in *Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, pages 65–74, IEEE Computer Society, 2012
- S. J. Zasada, T. Wang, A. Haidar, E. Liu, N. Graf, G. Clapworthy, S. Manos, and P. V. Coveney, IMENSE: An e-infrastructure environment for patient specific multiscale data integration, modelling and clinical treatment, *Journal of Computational Science*, 3(5):314–327, 2012
- S. J. Zasada, A. N. Haidar, and P. V. Coveney, On the usability of grid middleware and security mechanisms, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1949):3413–3428, 2011

- A. N. Haidar, S. J. Zasada, P. V. Coveney, A. E. Abdallah, B. Beckles, and M. A. S. Jones, Audited credential delegation: a usable security solution for the virtual physiological human toolkit, *Interface Focus*, 1(3):462–473, 2011
- S. J. Zasada and P. V. Coveney, From campus resources to federated international grids: bridging the gap with the application hosting environment, in *Proceedings of the 5th Grid Computing Environments Workshop*, GCE '09, pages 10:1–10:10, ACM, 2009
- S. J. Zasada and P. V. Coveney, Virtualizing access to scientific applications with the application hosting environment, *Computer Physics Communications*, 180(12):2513–2525, 2009
- S. Manos, S. J. Zasada, and P. V. Coveney, Life or death decision-making: The medical case for large-scale, on-demand grid computing, *CTWatch Quarterly Journal*, 4(2):35–45, 2008
- S. Manos, S. Zasada, M. Mazzeo, R. Haines, G. Doctors, S. Brew, R. Pinning, J. Brooke, and P. Coveney, Patient specific whole cerebral blood flow simulation: A future role in surgical treatment for neurovascular pathologies, in *Proceedings of the 3rd TeraGrid Conference*, 2008
- P. V. Coveney, R. S. Saksena, S. J. Zasada, M. M. Keown, and S. Pickles, The application hosting environment: Lightweight middleware for grid-based computational science, *Computer Physics Communications*, 176(6):406–418, 2007
- S. J. Zasada, R. S. Saksena, P. V. Coveney, M. M. Keown, and S. M. Pickles, Facilitating User Access to the Grid: A Lightweight Application Hosting Environment for Grid Enabled Computational Science, *Proceedings of the International Conference on e-Science and Grid Computing*, pages 50–58, 2006

Chapter 1

Introduction

In this chapter we define the overall scope of this thesis, introduce the work we will present and give an overview of the remaining chapters.

1.1 Context

Today's computational scientists face a growing number of challenges which affect their ability to fully exploit the computational resources available to them. Firstly, they have an unprecedented amount of computational power available to them, which will continue to grow in the future. A new generation of high performance computing (HPC) machines are now coming online with multi-petaflop performance, and machines operating at the exascale predicted to exist within the next 5-10 years; these present many challenges to an increasing number of scientific disciplines that rely on computer based modelling and simulation. While this may seem like a positive development, it brings with it a number of problems, such as the need to gracefully manage node failure in machines with tens to hundreds of thousands of cores.

Secondly, the architectures of these large scale HPC machines point to a growing trend; HPC machines made up of hybrids of scalar and vector processors, or multicore processors that include scalar and vector components on the same chip, are likely to be commonplace in the future [14, 15]. This challenges application scientists to ensure their code is optimized to take full advantage of the hybrid architecture of a specific machine.

Grid computing [16, 17] seeks to simplify end user access to and use of HPC resources, by establishing a software and policy infrastructure for distributed computing conducted transparently across multiple administrative domains. However, the middle-

ware tools developed to realize the computational grid concept have not always provided the transparency and ease of use envisaged [18]. Notwithstanding the problems of using middleware to interact with HPC resources, the challenges of using a single petascale machine described above are compounded when one attempts to use multiple resources, via a distributed computational e-infrastructure, as more than just a sum of their individual parts [19].

1.2 Motivating Example

In this section we look at how HPC resources have been traditionally used as ‘single islands’ of computational power, and how the proliferation of distributed computational e-infrastructures is changing these usage modalities.

1.2.1 Traditional HPC Use

Running a parallel scientific application on a HPC resource requires a user to perform a number of complex tasks. If the application is not already installed on the resource, then the user will have to download and compile the source code, which could require him to download and install specific required software libraries, or work with the machine’s administrators in order to have these installed centrally on the machine. Each resource that a user accesses will likely have its own set of compilers and libraries which the user will have to familiarize himself with before he builds the application; these often generate problems that an inexperienced user is not able to deal with. In addition the user may have to take extra steps to optimize the code (usually written by someone else) for the particular architecture of the machine being worked on.

Once the application is installed on the machine, the user then needs to generate a job launching script for the queuing system running on the resource they are using. The multitude of different queuing systems used to manage HPC resources, including LSF, PBS, LoadLeveler and SGE, mean that the scientist has to create a bespoke run script for each application that needs to run on each chosen machine. This can then be ‘cloned’ for each individual run of the application, by changing parameters such as the input and output files.

Finally, the user must stage all the data to the resource being used and submit the jobs to the queuing system. Depending on the load and policies of the machine, it could

take from minutes to several days for the job to run. Once the job is complete, the user will usually retrieve the output data to their local file system for further processing. When submitting a job a user must ensure the libraries required by the application have been installed and the correct scheduler options set.

1.2.2 Harnessing the Power of HPC via a Distributed Computational e-Infrastructure

The scenario described above is common to the vast majority of HPC users. While the launching of a single application on a single resource is straightforward, where a heterogeneous e-infrastructure of supercomputers is concerned, the complexity grows with the number of nodes on the e-infrastructure. The power of an e-infrastructure lies in the fact that it gives the user access to many resources, which they can use in combination to solve challenging problems. A single petascale machine does not make the concept of a distributed e-infrastructure obsolete. The power of a distributed e-infrastructure arises from its inhomogeneity, which includes resources such as visualization engines and storage servers, coupled by high performance networks, in addition to HPC clusters.

The software used to tie a distributed e-infrastructure together is referred to as the middleware; the majority of production e-infrastructures in existence today use one of three heavyweight middleware stacks, Globus [20], Unicore [21] or gLite [22], which are designed to provide a comprehensive set of services necessary to build a variety of different types of distributed e-infrastructure. By ‘heavyweight’ middleware we mean software stacks which require expert system administrator support to install and maintain, and are therefore difficult for end users to deploy and often contain far more functionality than the end user requires.

To be able to fully exploit the power of a distributed e-infrastructure, conventional MPI simulation code does not suffice; however, it can be used as a building block to create a distributed workflow, designed for example to perform a molecular dynamics (MD) equilibration protocol by running several different MD codes on different scale e-infrastructure resources [23], or to build a distributed or ‘grid enabled’ application which takes some intrinsic advantage of the fact that it is based on a distributed e-infrastructure, for example by utilizing some form of distributed MPI, or real time

visualization and computational steering.

Such applications deviate from the normal HPC usage modality described above, making them difficult to manage and deploy. The number of different steps needed to deploy and run such an application can be very onerous for the user. For example running a cross-site MPI code requires the user to build a version of the code on each different site that she wants to use (which could also have different architectures), stage the required data files in advance to each site, arrange with each machine's operator that the individual sites will all be available at the same time in order to run the application, and then launch the different application components at each site. Troubleshooting problems with such an application is often extremely difficult and time consuming.

The piecemeal fashion in which such applications are deployed [24] and executed means that they are very difficult to run in a routine, production way, in order to carry out large scale scientific studies. This difficulty denies the scientist the opportunity to explore new computational science at the boundaries of what is possible with today's HPC machines, and means the full power of distributed computational e-infrastructures is rarely exploited. What is required is appropriate middleware tools and interfaces designed to hide as much of the drudgery and potential for error as possible from the the user when running complex distributed applications and workflows.

1.2.3 Gaining Access to HPC Resources

The first step in running a simulation on any type of distributed e-infrastructure resource is simply to gain permission to access the machines required. The procedure for doing this will depend on the type of infrastructure involved. The biggest supercomputers usually require potential users to submit a formal proposal describing the science behind the simulation or other job to be run and estimate the amount of CPU and storage that is needed. This process often resembles the procedure used in applying for grants. Institutional level and other lower-level resources are often free to use and may only require the submission of a simple web form.

In the best case scenario, an application will lead to an award of CPU time that can be used on all resources that comprise the distributed e-infrastructure. However, more often than not in the case of HPC e-infrastructures, an award will be made on either a small subset or a single resource on the e-infrastructure, meaning that the user

is constrained as to where they are able to perform their computation. Often an award of time will be made based on a user running a single application, and with the requirement it be used by a given deadline. This means that very often the user is under pressure to use time on a particular resource by a specific deadline, and is responding to demands imposed by the resource provider, rather than the demands of their scientific investigation.

1.2.4 Uptake of Grid Techniques on HPC e-Infrastructures

When considering the usability problems faced by users of HPC e-infrastructures, it is important to consider the uptake of grid middleware interfaces by users of HPC platforms. As noted, grid computing was intended to reduce the barrier to uptake of computational resources, by providing simplified, standardized interfaces and tools which allowed such resources to be accessed remotely by users, with the ultimate goal of making computation as easy to access as electrical power.

1.3 Problem Statement

As the previous section has illustrated, in the high performance distributed e-infrastructure space, all too often the user's time is spent investigating the availability of resources, marshalling data and nursing their applications. For many computational scientists using high performance computing, the grid concept has failed to deliver its promise of providing transparent, ubiquitous computational power on demand. This is due to both a lack of appropriate tools, and a lack of tools that present the right level of abstraction to the user [18], meaning that it is easier for them to carry on with their existing usage mechanisms, as if the grid were not there. When the researcher has access to more than one computational e-infrastructure, running different middleware stacks, the problem is compounded, with the user having to learn how to use different middleware client tools to interact with the resources available to them.

While the high performance computing community has been chasing ever increasing machine peak performance, with many petaflop machines available to researchers across the globe, the end user, the so called 'application scientist' is not generally interested in the peak performance of the machine they are using, but in the total time to solution of the scientific problem that they are working on [25]. Many strategies

have been employed to try to mitigate the total time to solution, but they are often dependent on the nature of the problem being solved. For example, Chakraborty *et al.* [25] propose decomposing large simulations into smaller jobs that can be distributed amongst several machines to reduce the total time to completion, but this approach doesn't apply to all classes of scientific application. A second consideration of the user of a distributed computational e-infrastructure is the cost of running their simulation, and the problem of choosing a computational resource (or multiple computational resources) in order to perform a simulation becomes a trade-off between the total cost of running the simulation and the total time to achieve a result.

Essential to realizing the vision of a computational e-infrastructure as ubiquitous, seamless to use and as transparent as the electrical power grid, as proposed by Foster *et al.* [17], is the broker. The broker is a component of the e-infrastructure responsible for efficiently distributing jobs between distributed resources, taking into account factors such as machine load and cost models. A broker provides a point of contact between the user and the e-infrastructure, placing application instances submitted by the user onto appropriate resources. The broker means that the user does not have to deal directly with each machine on the e-infrastructure, avoiding the need to log in to several resources when deciding where to run an application to find the one with the least load. The broker also means that expensive HPC resources are used as efficiently as possible, ensuring that one machine is not idle while another has a large queue of jobs.

The purpose of the broker is to match jobs to appropriate resources. As noted in Section 1.2.1, the majority of HPC resources on a distributed e-infrastructure will have their own local scheduling mechanism, responsible for allocating jobs submitted to the resource between the available processors. Unfortunately there is no ubiquitous meta-scheduling technology available that allows users to specify their requirements for running a job: the ability to specify trade-offs between when their job will run and how much it will cost. The allocation policies discussed in Section 1.2.3 also make this difficult in many cases.

The problems are therefore twofold: Firstly, the level of abstraction used in user/e-infrastructure interactions is not sufficiently powerful to allow distributed resources to be trivially used by anyone but the most dedicated user. Secondly, the user does not need to care which particular resource on the e-infrastructure they are using; they are

only concerned with getting results. This thesis will seek to address these two problems by i) developing a higher level of abstraction for user/e-infrastructure interaction and ii) developing a approach to resource allocation that allows the user to specify constraints on their workloads, such as the time to completion or the maximum cost.

We believe, with the right combination of tools and services, the initial concept of the grid/e-infrastructure as a provider of transparent and ubiquitous computing power can be realized, by:

1. Presenting an interface to access the e-infrastructure that provides an appropriate level of abstraction to allow the user to concentrate on running their applications without having to worry about the minutiae of dealing with every possible combination of compiler, architecture and queuing system, and
2. Developing a flexible decentralized workload allocation system that implements a controlled computational market place to enable the trading of time on HPC resources, allowing the user to control the aspects of the workload that they are interested in: the cost, and the time to solution.

We believe that this will lead to a decentralized system which is more scalable than currently available resource brokering technologies and which is able to more efficiently allocate work between a set of resources based on cost minimization and run time optimization. The decentralized nature will allow resources to easily join and leave the system, potentially creating dynamic virtual organizations based on aggregated resources from federated e-infrastructure resource providers.

1.3.1 Research Premise

Our research premise is that the interaction model currently prevalent in distributed e-infrastructures composed of high performance computing resources impedes usability, by forcing the end user to deal with too many low level considerations to run their applications. The problem is twofold. Firstly, the level of abstraction applied to user-resource interaction makes the middleware interfaces too difficult to use. We hypothesize that by moving to a user-application interaction model, that subsumes details of interacting with one or more resources, users will find distributed computational e-infrastructure easier to use. Secondly, current e-infrastructure meta-scheduling systems

do not allow the user of the system to be expressive enough in terms of setting cost and run time requirements on the jobs that they run. Most meta-schedulers, if they use any job wait time modelling at all, seek to minimize the time a job takes to run regardless of cost. However there are many cases where an e-infrastructure user will have low priority work to run which they do not mind waiting for, but want to minimize the cost of running. We hypothesize that a resource allocation system that allows resource providers to vary their costs based on the utilization of their system, and allows users to put cost and wait time constraints on the jobs that they submit, will allow users to optimally place their jobs on appropriate resources and allow providers to maximize their resource usage. We also hypothesize that scheduling based on predicted queue wait times will allow jobs to be allocated more optimally than naïve scheduling systems that make decisions based just on queue times, while not generating so great an overhead to negate the performance of the scheduler.

1.4 Contributions Made by This Work

The work proposed here seeks to develop a new way of e-infrastructure wide job scheduling, based on both resource availability predictions and resource cost, through use of an agent based auction model through which resource providers can vary their cost to maximize their resource usage. We believe that this will lead to a new paradigm of resource funding and provision, where users pay at the point of use. We also believe that this model will encourage commercial resource providers to join and form commercial e-infrastructures on which they can trade CPU power. This work makes the following specific contributions:

Contribution 1: The first contribution of this work is an application interaction model which promotes the ‘application’ as a high level concept with which the user interacts, instead of focusing on user-resource interaction. We believe this new level of abstraction will greatly aid user exploitation of computing on distributed e-infrastructures.

Contribution 2: The second contribution of this work is a software implementation of the application-interaction model. the Application Host Environment (AHE). This implementation will be built on existing middleware infrastructures, allowing users to

interact with applications rather than resources, while being as unobtrusive as possible to resource providers. The AHE will be used as the launching mechanism which initiates metascheduler requests, and will submit the job once an offer to run has been accepted.

Contribution 3: The third contribution of this work is to evaluation Web Services Resource Framework (WSRF) and Representational State Transfer (REST) approaches to implement the AHE.

Contribution 4: The fourth contribution of this work is an evaluation of the usability of AHE compared to other widely deployed e-infrastructure middleware tools.

Contribution 5: The fifth contribution of this work is an XML Schema for describing job requirements. The schema will allow users to specify the constraints on their job, such as the maximum wait time and the minimum cost.

Contribution 6: The sixth contribution of this work is a reverse auction market model of e-infrastructure resource provision. Based on a multi-agent auction, with buyer agents representing users and seller agents representing resources, the system will seek to match user specified job constraints, in terms of the price and wait time. This will employ a varying cost algorithm in the resource management agent, allowing it to vary its cost based on its predicted future usage.

Contribution 7: The seventh contribution of this work is an analysis of the performance of our market based resource allocation model.

1.5 Scope of this Thesis

We believe that distributed HPC e-infrastructures are a distinct class of computational e-infrastructures, and that the problems experienced by HPC e-infrastructure users are not necessarily applicable to users of any type of computational e-infrastructure. The systems discussed in this thesis are designed to assist the user of HPC e-infrastructures.

While the resulting systems and approaches may be applicable to other types of e-infrastructure (such as high throughput computing or HTC platforms and cloud systems), it is not the focus of the work presented here.

It is important to clarify what is in the scope of this work and what is not. The following is in the scope of this thesis:

- Development of a user-application interaction model.
- Development of the AHE job launching and management system.
- Analysis of implementation performance and usability of AHE.
- Design of an XML schema to allow users to specify a rich set of job constraints - a request for quotations (RFQ).
- Development of a decentralized, distributed resource allocation system.
- Implementation and extension of a reverse auction trading algorithm.
- Formulation and development of a resource queue wait time prediction algorithm based on historical usage data.
- Design and implementation of a resource seller agent cost adjustment algorithm.
- Experimental investigation of the resultant resource allocation system.
- Development of basic banking facilities to record transactions between user and resource agents.

The following is not in the scope of this thesis:

- Development of resource information services - where required we will make use of information sources currently made available by resource providers, such as Globus MDS.
- Development of new server side middleware - agents acting on behalf of a user should not require any changes to be made by resource providers in the middleware stack they support.

- The mechanism used by the scheduling agents to guarantee the availability of their resource after they have successfully bid for a job. We would recommend that the agents use the reservation mechanism within their resource manager, to reserve the time slot on the machine.
- Development of new security mechanisms - current e-infrastructure security mechanisms, such as GSI, will be used where required.
- Changes in resource provider policy - this research will seek to use production e-infrastructures as they are currently provided, without requiring providers to make major changes to their local scheduling policies etc.
- Development of distributed accounting services - robust accounting is essential for deriving billing information in a computational economy, however the provision of such services is outside the scope of this work. We envisage being able to use one of the accounting services currently being developed by the Globus Alliance or OMII UK, when they become available.
- Development of an explicit framework for constituting service level agreements (SLAs) - While we believe the combination of the RFQ and the offer to undertake the work made by the resource do constitute an SLA between the user and the resource, mechanisms to enforce the terms of the SLA are outside the scope of this thesis.

1.6 Evaluation

This work is constituted of two distinct but related components, the Application Hosting Environment and the Resource Allocation Market Place. By necessity, the success of these two different components must be evaluated separately.

1.6.1 Evaluation of Application Interaction Model

The purpose of application interaction model is to simplify experience of end user of computing on distributed e-infrastructure. We will evaluate the relative performance of RESTful and WSRF based implementations of the application interaction model, as realized in the Application Hosting Environment. However, this will not tell us much about how e-infrastructure usability has been improved. To test that the application

interaction model provides the right level of abstraction, we will conduct a usability study to compare AHE to existing middleware solutions such as Globus and Unicore, asking users to evaluate the software in terms of how easy it is to launch and monitor jobs, and install the client tools.

1.6.2 Experimental Validation of the Resource Allocation System

We will use the Dynamic Analysis method described by Zelkowitz and Wallace [26, 27] to validate the research premises above (*cf.* §1.3.1). We will investigate the capabilities and performance of our resource allocation system to ensure that it can successfully place workloads with appropriate resources in the system; that it can minimize the costs paid by the user; and that it can ensure good overall system utilization. Ideally we would like to use a production e-infrastructure environment, such as the EU PRACE system, to perform the experiments. However, due to the need to install and configure resource allocation components on machines under the control of others, we will base our investigations on a simulated HPC e-infrastructure. If this is the case we will set up a lab based simulation system on which to experiment. The experiments we perform will use sample workloads provided by computational scientists at the Centre for Computational Science at UCL, to ensure that comparisons are made using real world applications.

1.7 Overview of Remaining Chapters

The remainder of this thesis sets out to examine current usage practice of HPC e-infrastructure computing and propose mechanisms by which such practice can be improved, through the introduction of services to provide an extra level of abstraction on top of existing e=infrastructures, thus easing the users interaction with this infrastructure, and to allow jobs to be automatically allocated to appropriate resources within the e-infrastructure in order to satisfy the users requirements. To that end, in Chapter 2 we look at the current state of play of e-infrastructure/user interaction. In Chapter 3 we go on to state the requirements of the system under consideration, look at the steps necessary to satisfy those requirements and describe our implementation of the system, the Application Hosting Environment. In Chapter 4 we present our analysis of the implementation of the AHE as RESTful and WSRF services, and additionally present the

results of our usability studies into the application interaction model. In Chapter 5 we review the elements of computational mechanism design that can assist in the development of a decentralized e-infrastructure allocation mechanisms and consider similar systems. In Chapter 6 we present the development of a reverse auction based system, the Resource Allocation Market Place (RAMP), designed to satisfy our resource allocation requirements. We discuss the experimentation validation of this system in Chapter 7. Finally, in Chapter 8, we draw conclusions from the work presented here and consider the future work we could undertake.

Chapter 2

High Performance Computing on Distributed e-Infrastructures

This chapter will to give a background overview of high performance computing (HPC) conducted across distributed e-infrastructures, and review the relevant literature in the field. Firstly we will look at what is meant by high performance computing, and why the idea of grids of HPC resources was conceived and realized in order to facilitate access and optimize usage of these resources. Within this text the term ‘resource’ is used to describe a high performance computer. In this section we will also examine the development of cloud computing and consider the middleware tools that have been developed to help realize the e-infrastructure concept and look at the needs of users when interacting with distributed, heterogeneous computational resources.

2.1 The Computational Ecosystem

The computational landscape is constantly shifting, with desktop machines becoming more and more powerful, incorporating multicore central processing units (CPUs) and general purpose graphics processing units (GPGPUs), and multiple processors. However, even the most high specification workstations available today do not provide enough computational power for many models to be simulated in a tractable amount of time (and this becomes even more important if the results of a simulation are time sensitive and required to, for example, influence a clinical decision, since they need to be calculated within a timeframe that allows them to be incorporated into the decision making process).

This means that simulations have to run on resources beyond the desktop. Such

resources range in scale from networks of high performance workstations (e.g. Condor pools, *cf.* §2.3.6) to loosely coupled compute clusters (also called Beowulf clusters [28]) to more tightly integrated supercomputers, which use high performance interconnects to couple compute nodes together, or make use of shared memory architectures, all of which is loosely grouped together under the term e-infrastructure.

The boundaries between these different classes of compute resource are somewhat blurred. However, most computational resources used by computational scientists fall into the following broad categories:

- **Multicore computing:** A multicore processor is a processor that includes multiple execution units on the same chip. These processors differ from super-scalar processors, which can issue multiple instructions per cycle from one instruction stream. In contrast, a multicore processor can issue multiple instructions per cycle from multiple instruction streams. A single workstation may contain one or more multicore chips.
- **Symmetric multiprocessing:** A symmetric multiprocessor (SMP) is a computer system with multiple identical processors that share memory and connect via a bus.
- **General-purpose computing on graphics processing units (GPGPU):** General-purpose computing on graphics processing units is the technique of using a GPU, which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the CPU. Often such machines are used to run data parallel applications.
- **Heterogeneous multicore:** many newer processor architectures are adopting a heterogeneous multicore architecture, designed for parallel processing. Chips such as the Cell or Intel MIC combine traditional compute cores with specialized processing cores, such as GPUs or other coprocessors, to improve performance and power efficiency while maintaining the versatility of a traditional CPU.
- **Cluster computing:** A Cluster is a group of loosely coupled computers that work together closely, so that in some respects they can be regarded as a single

computer. Clusters are composed of multiple standalone machines connected by a network.

- **Massively parallel processing:** A massively parallel processor (MPP) is a single computer with many networked processors. MPPs have many of the same characteristics as clusters, but MPPs have specialized interconnect networks (whereas clusters use commodity hardware for networking).

Resources range in cost from the desktop to the supercomputer. Hence, there are many more desktop workstation class resources at the researchers disposal than there are supercomputers. This fact was noted by Lewis Branscomb in a 1993 report [29], which introduced the idea of a Branscomb Pyramid, showing the relative abundance of different classes of computational resources. Figure 2.1 shows the Branscomb Pyramid of HPC class grid resources considered in this thesis.

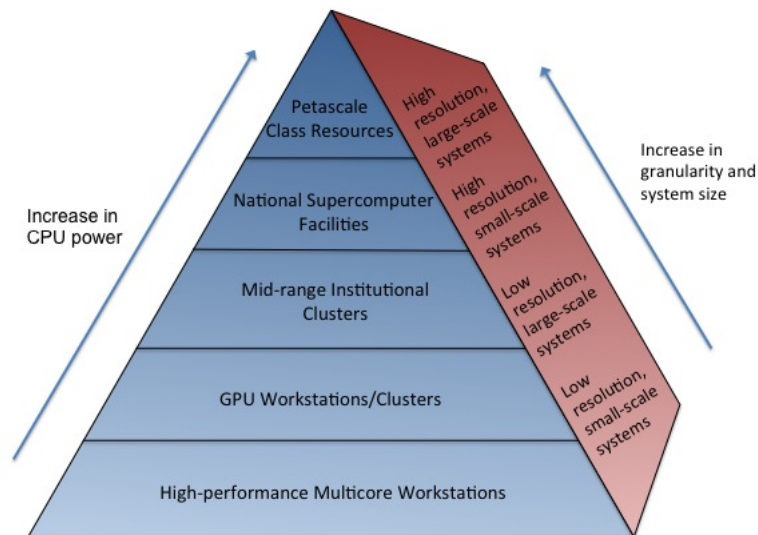


Figure 2.1: The Branscomb Pyramid, showing the relative abundance of computational resources.

2.1.1 Applications of High Performance Computing

High Performance Computing has been employed by many different scientific communities to help the furtherance of their scientific investigations. Traditional communities include physics, chemistry and engineering, which have been joined more recently by fields such as medical research, where researchers are seeking to use high performance computing and simulation to augment the clinical decision making process [30, 2].

Phase	Start date	AUs per hour	Cost per AU
1	December 2002	3241	0.38759
2	June 2004	6188	0.203
2A	November 2005	7395	0.17417
3	November 2006	12290	.010480

Table 2.1: Decline in cost of CPU time on the UK HPCx machine over its lifetime

Phase	Scalar cost/AU	Vector cost/AU
Phase 1 (Oct 2007 - Sep 2009)	0.06354	0.50836
Phase 2 (Oct 2009 - Sep 2011)	0.01371	0.10969
Phase 3 (Oct 2011 - Sep 2013)	0.00578	0.04628

Table 2.2: Decline in cost of CPU time on the UK HECToR machine over its lifetime

Traditionally based on tightly integrated parallel computing systems such as those made by Cray, IBM and Silicon Graphics, HPC resources increasingly consist of ‘clusters’ of commodity hardware with a fast interconnect linking the compute nodes together, and running software such as Beowulf [28]. The rise of the distributed memory supercomputer has necessitated the development of suitable programming frameworks in order to harness their power. The *de facto* standard for communication between parallel processes on a distributed memory cluster machine is Message Passing Interface (MPI) [31], although other similar technologies do exist such as Parallel Virtual Machine (PVM) [32].

The policies governing allowing users to access high performance compute resources, and charging them for the access, vary amongst resource providers. Most resource providers will however assign a notional cost related to the work done on their resource, typically setting a price for an abstract unit of processing work, which is related to the use of an hours use of a single CPU on their resource. Table 2.1 shows the changes in cost of an abstract unit (AU) of processing work over the period that the UK’s HPCx machine was available for research use, while table 2.2 shows a similar decline in cost for the UK’s HECToR machine. Data for the current UK flagship machine, Archer, is unavailable at the time of writing. These machines have been expanded in phases over their lifetime, with more and faster processors added, and as the processing power of the machine has grown (number of AUs available per hour) the cost per AU has fallen.

2.1.2 Access Modalities & Resource Use

The batch processing model is still the predominant usage model in the world of HPC, with queuing systems being used to ensure fair access to the resource for all users while at the same time maximizing resource usage. Many such queuing systems exist including PBS [33], SGE [34] and LSF [35]; they are typically configured by the administrators of the resource to support local queuing policies and queue sizes. In most systems all users are treated with equal priority, with jobs being run on a first come first serve basis.

In order to access a machine a user will typically use protocol such as SSH to establish an interactive login session on the machine, or could use some form of grid middleware to launch their job, as described later in this section. In the former case running a parallel scientific application on an HPC resource requires a user to perform a number of complex tasks. If the application is not already installed on the resource, then the user will have to download and build the source code, which could require them to download and install extra required software libraries, or work with the machines administrators in order to have them installed centrally on the machine.

Each resource that a user uses will likely have its own set of compilers and libraries which the user will have to familiarize himself with before he builds the application, and which could generate problems that an inexperienced user is not able to deal with. In addition they may have to take extra steps to optimize (usually written by someone else) the code for the particular architecture of the machine they are working on.

Once the application is installed on the machine, the user then needs to generate a job launching script for the queuing system running on the resource they are using. The multitude of different queuing systems used to manage HPC resources, including LSF, PBS, LoadLeveler and SGE, mean that the scientist has to create a bespoke run script for each application that they want to run on each machine they want to run it on. This can then be cloned for each individual run of the application, by changing parameters such as the input and output files.

Finally, the user needs to stage all of their data to the resource they are using and submit their job to the queuing system. Depending on the load of the machine it could take from minutes to days for the job to run. Once the job is complete, the user will usually retrieve their data to their local file system for further processing.

2.1.3 Classes of HPC Applications

The key benefit of using resource beyond the desktop is the ability to exploit large-scale parallelism, using multiple computer processors in concert. Where workflows of operations are being carried out applications are likely to be composed of multiple application codes [36]. Different degrees of computational parallelism are likely to be required by different sections of a workflow, with some sections running on single core workstations, some requiring cluster computing and some requiring access to the biggest supercomputers available. Other application scenarios may only involve a single code, which might be serial, *embarrassingly parallel* (also called *data parallel*) or *inherently parallel* (also called *task parallel*), and run on an appropriate scale resource.

No one degree of parallelism is appropriate for all application scenarios, and often different components of a particular application workflow cover all possible types of parallelism, and require a range of computational resources to match. The two main parallelization schemes mentioned above involve the following:

- **Data parallelism:** this form of parallelism splits a dataset into small packets distributed across multiple compute nodes, with the same algorithm run over each data packet. Typically, there is little to no communication between the different data processing nodes while the computation is taking place, with the output from each of the separate computations assembled once all computations have completed.
- **Task parallelism:** task parallelism is achieved when execution processes are distributed across a set of compute nodes, and operate on a single dataset. This usually requires a significant amount of synchronization between execution processes, mandating a high performance communication infrastructure between compute nodes. Task parallelism leads to more tightly coupled applications, with varying levels of communication between computing nodes as a simulation progresses.

The form of parallelism appropriate to a particular simulation is inherently coupled to the nature of the simulation algorithm and the data operated on. The growth in the scale of resources available today means that often schemes that incorporate both

data parallelism and task parallelism are appropriate, with, for example, multiple task parallel codes being executed simultaneously on different sections of a dataset.

2.1.3.1 Achieving parallelism

Typically task parallelism relies on specific libraries and frameworks in order to implement a degree of parallelism. Message Passing Interface [31] (MPI) is an application programming interface (API) specification that allows processes to communicate with one another by sending and receiving messages. It is typically used for parallel programs running on computer clusters and supercomputers.

The Message Passing Interface was developed by Gropp *et al.* [31] as a platform independent protocol for communication between processes on a parallel computer and sits at level 5 of the Open Systems Interconnection (OSI) network model [37]. Typically a high performance computing (HPC) resource will have an implementation of MPI installed that is configured to work with the particular interconnect technology used to connect the compute nodes, to provide optimal application performance. MPI applications use multiple tightly coupled processors to run tasks that communicate via MPI API calls. Often the nodes of a cluster will be connected via a fast network interface such as InfiniBand [38] or Myrinet [39]. A user then compiles their application against the system's MPI libraries, and launches their application using a tool supplied with the MPI system, which takes care of starting instances of the application on the appropriate set of compute nodes, and configures the environment to allow the processes to establish communication with each other.

The MPI interface provides virtual topology, synchronization, and communication functionality between a set of processes (that have been mapped to nodes, servers or computer instances) in a language-independent way, with language-specific bindings. MPI library functions include point-to-point rendezvous-type send/receive operations, choosing between a Cartesian or graph-like logical process topology, exchanging data between process pairs (send/receive operations), combining partial results of computations (gather and reduce operations), synchronizing nodes (barrier operation) as well as obtaining network-related information such as the number of processes in the computing session and current processor identity that a process is mapped to. It is the de facto standard model for parallel software development on most HPC machines.

Task parallelization can also be achieved using shared memory multiprocessor programming environments such as OpenMP [40]. OpenMP implements thread-based parallel processing, whereby a master thread initiates multiple ‘forked’ threads to execute the parallel parts of a code, which are then joined back to the main master thread of execution when the forked, parallel sections of the code execution have completed. Hybrid schemes that combine both OpenMP and MPI style techniques can achieve performance improvements by optimizing inter and intra-node communications.

Data parallelism requires some scheme to split input data into chunks which can be operated on independently, which is often application specific. Once the data is partitioned, the execution of the distributed computations can be managed by a workflow engine, or some dedicated job submission manager such as Condor (*cf.* §2.3.6).

MapReduce [41] is a framework that has been designed to automate the process of distributing data-parallel tasks between compute nodes. A MapReduce program is comprised of two parts. The first is a `map()` function, where a master node takes a set of inputs, divides it into smaller sub-problems, and then distributes them between compute nodes. Each of the compute nodes then processes its sub-problem and passes the results back to the master node.

The second part of a MapReduce program is the `reduce()` function, which assimilates the results from the computation of each of the sub-problems and produces an answer to the problems being computed. Since MapReduce applications typically do not have any interdependence between sub-problems, limited network bandwidth is required between compute nodes. This means that problems can be run across a single cluster, or even a set of separate clusters.

MapReduce manages fault tolerance within a distributed system; if one or more sub-problems fail, due to a fault on a particular cluster node for example, MapReduce can rerun the failed tasks on another compute node. MapReduce programs can also implement multiple levels of hierarchy, with sub-problems that consist of their own `map()` and `reduce()` functions.

2.2 Distributed Computing

Isolated computational resources are of limited use. Deploying workflows onto high performance computers involves choosing an appropriate resource for each section of

the workflow. Manually executing a single simulation on an HPC resource is a laborious process; executing each stage of a workflow manually in anything approaching a routine way is extremely arduous. Automating the workflow process is therefore extremely desirable, and is made possible through middleware interfaces that allow simulations to be executed in a more automated fashion. These interfaces allows resources to be linked together and accessed using common mechanisms, in order to constitute distributed e-infrastructures or grids.

2.2.1 Grid Computing

The term grid computing [16, 17] originated in the early 1990s as a paradigm for accessing datasets and running jobs automatically on high-performance resources. Since then, cloud computing has taken over this role to some extent, but grid computing is still valuable and widespread. We define grid computing as distributed computing conducted transparently by disparate organizations across multiple administrative domains. The goal of grid computing is to give the end user transparent, uniform access to resources owned and operated by disparate organizations, which may have different security and access policies at an institutional level, whilst at the same time giving reasonable security assurances to the institutions participating in the grid. A good place to start when trying to understand grid computing is Foster *et al.* [17]. They defined the underlying problem of grid computing as:

“co-ordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations”

A *virtual organization* can be thought of as a group of institutions or companies that have come together to share their resources, with some kind of software system used to manage the resources and sharing policies. The above quote neatly summarizes the key concept of grid computing, namely that resources are shared between different institutions with varying security and access policies. Examples of resources a grid participant may want to share are high-performance super computers, scientific instruments such as radio telescopes and microscopes, visualization servers and databases. As these resources are very often geographically dispersed, seamless access to them is provided via high performance computer networks such as the Internet, or dedicated high-speed point-to-point networks [42].

The term grid is used to describe this arrangement of inter-institutional resource sharing by analogy with the electrical power grid [43]. That is, it describes the infrastructure for dependable, consistent, pervasive and inexpensive access to high-end computational resources in much the same way as the electrical power grid provides dependable, consistent, pervasive and inexpensive access to electrical power.

The different types of scientific resource that make up grids, including scientific instruments, data repositories, and computers that range in size from desktop machines to large-scale parallel supercomputers, are made available at national and international levels. The grid model is particularly appropriate for, and has been very widely implemented in, the sharing of HPC resources, since the grid model is a good way to get the most from machines that are expensive to purchase and run. The power of a grid arises largely from its inhomogeneity, as it links visualization engines, data storage, and instruments as well as HPC machines [44, 24, 11].

2.2.2 Web Services

Web services provide a mechanism to execute remote procedures using standard interface mechanisms and technology. A Web service can be thought of as a loosely coupled software component that communicates via a standardized interface [45, page 593]. The fact that Web services are loosely coupled means that they can be changed independently of each other, and also that they are platform agnostic. A Web service's functionality is only exposed via its interface, so no details need be known about its implementation or the environment on which it is hosted. These features of Web services are also things that are very desirable when constructing a grid of heterogeneous resources.

Although not the only mechanism, the *de facto* standard for message exchange in Web services is SOAP [46]. SOAP defines an XML schema that can be used for the packaging, encoding and exchange of structured data. The document consists of two required parts; an envelope that packages the data and a body which is the data. The message can optionally contain a header with additional descriptions about the message and extra processing instructions. In a Web services environment the SOAP message can be transported over a number of different transportation protocols such as FTP, POP3 or SMTP, but most commonly HTTP is used.

The nature of Web services means that they can be used to build highly flexible distributed systems, such as the myGrid system [47]. For this reason Web services have greatly interested researches in grid computing and have become a widely adopted programming methodology in grid computing systems (see Section 2.3.1).

2.2.3 Service Oriented Architectures

The advent of Web services has led to new software architectures being proposed and adopted that address the issues faced when integrating distributed, heterogeneous and possibly legacy systems within an organization. One of these is the *Service Oriented Architecture* (SOA) [48, 49]. SOA represents the big picture of what can be done with Web services. It is an approach to building distributed systems that deliver functionality to end-user clients in the form of services, or allow new services to be built by combining existing ones. SOA allows Web services to be used in systems that transcend traditional client-server models to become systems of arbitrary complexity. Indeed there is much interest in transforming many types of vertically integrated software systems to horizontally integrated, service-oriented systems [50], to facilitate the building of highly flexible, component based systems.

SOA and Web services provide an implementation and platform agnostic way of describing software components. Put another way, a client application needs to know nothing about the programming language or machine architecture of the piece of software it wants to access. It just needs to understand the interface that the software component provides to allow interaction, which is based on standard technologies, such as eXtensible Markup Language (XML) and described using Web Services Description Language (WSDL), the standard Web services interface description mechanism. Thus, by understanding a software component's (service's) interface, a client application does not need to be concerned with where or how a service is implemented.

Much of the interest in Service Oriented Architectures has come from business organizations wishing to integrate their legacy software systems with newer technologies such as web based interfaces, and to simplify application development and deployment [51], but interest has also been shown by the grid computing community. Indeed, as Web services and grid computing have matured alongside each other, it has become clear that many of their goals are similar and thus the technologies have started to con-

verge [52].

2.2.4 WSRF

WSRF (Web Services Resource Framework) [53] is an OASIS standard to provide mechanisms to allow state to be associated with stateless Web services, an essential property when using Web services to build grid middleware systems. The WSRF specification has evolved from the earlier Open Grid Services Infrastructure (OGSI) specification [54], splitting OGSI's functionality into a family of separate specifications that can be combined to produce the desired functionality for a given task. This has resulted in four WSRF specifications (WS-ResourceProperties, WS-ResourceLifetime, WS-ServiceGroup, WS-BaseFault) plus the WS-Notification family of specifications.

Although WSRF differs in its approach to modelling a stateful resource (a WS-Resource as opposed to a grid service), the functionality provided is essentially the same as OGSI. As Web services have evolved since the creation of OGSI, some of the extensions to WSDL that OGSI promoted have been included as parts of standard WSDL. WSRF can be thought of as the maturing of the OGSI specification.

2.2.5 Representational State Transfers

Somewhat parallel to the growth in popularity of SOAP based Web services has been the development of Representational State Transfer (REST), or RESTful Web services [55]. REST builds on the architecture of the web to define a set of architectural principles with which to design Web services that focus on a system's resources, including how resource states are addressed and transferred using the stateless HTTP protocol, by clients written in a wide range of different languages.

REST has been designed as a way to construct simple, high performance distributed infrastructures from the outset. According to Fielding [55, §5.3.1]:

“REST's clientserver separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries (proxies, gateways, and firewalls) to be introduced at various points in the communication without changing the interfaces between components, thus

allowing them to assist in communication translation or improve performance via large-scale, shared caching.”

REST makes available limited number of operations (verbs) with which to interact with a resource. Each resources or service (nouns) is assigned its own unique universal resource indicator (URIs). A subset of HTTP operations (GET, POST, PUT and DELETE) constitute the verb that can act on the noun, each with its own specific meaning.

2.3 Middleware

Fundamental to allowing the inter-institutional sharing of resources in a grid is the grid middleware, that is the software that allows the institution to share their resources in a seamless and uniform way. The most widely used grid middleware systems today include Globus [20], gLite [22] and UNICORE [21]. In this section we review the most widely deployed middleware solutions in order to discern their strengths and weaknesses, and how they can be improved on by the AHE system presented later in this thesis.

2.3.1 Open Grid Services Architecture

The effort to build grids using a service oriented paradigm has led the Open Grid Forum (formerly the Global Grid Forum) to propose the Open Grid Services Architecture (OGSA) [56]. OGSA aims to address some of the deficiencies perceived in earlier grid middlewares by building upon the practical experienced gained from using those systems. OGSA has been put forward as an architecture for producing interoperable grid middlewares using open standards and industry standard technologies such as Web services.

Using a service oriented paradigm means that emphasis is put on the services offered by the resources being shared rather than the physical resources themselves. Storage resources, computational resources, networks and databases are all represented as services. All of the entities in the grid are described in terms of their interface and behaviour.

OGSA is built on stateful WSRF Web services (see §2.2.4) that provide a set of well-defined interfaces and follows a set of specific conventions. OGSA places grid

services in the second tier of an architecture that is comprised of four layers:

- Physical and logical resources layer - including servers, storage and file systems.
- Web/grid services layer - All resources are modelled as Web services.
- OGSA architected grid services layer - provides facilities such as authorization and service management.
- Grid applications layer - the applications that run on the grid.

2.3.1.1 OGSA-BES

The OGSA Basic Execution Service (BES) [57] defines a standard service oriented interface for submitting jobs to resources on a distributed e-infrastructure. An OGSA-BES compliant Web service can create, monitor, and control computational entities such as UNIX or Windows processes, other Web services, or parallel applications. These entities are called activities and they are described by the Job Submission Description Language (JSDL) [58] documents. In production e-infrastructures, each resource provides one or more BES interface which users can use to create activities on that resource. OGSA-BES is a ratified standard of the Open Grid Forum, meaning that implementations that support OGSA-BES should be mutually interoperable.

2.3.1.2 HPC Basic Profile

In order to help facilitate interoperation between HPC resources running OGSA-BES interfaces, the Open Grid Forum have created a further specification, HPC Basic Profile (HPC-BP) [59]. The HPC Basic Profile standard describes how the OGSA-BES and JSDL specifications are composed in order to support basic job submission to (HPC) systems in a way that manages differences between the different queuing systems and software environments the systems may be running.

The profile consists of references to existing specifications, along with any clarifications of the contents of those specifications, restrictions on the use of those specifications, and references to extensions to those specifications. By supporting the basic set of capabilities outlined by HPC-BP, systems can reasonably be assumed to interoperate at the job submission level.

2.3.2 Globus

Globus [20] is widely deployed by many international grid projects. From Globus Toolkit version 3 (GT3), Globus followed a service oriented approach, tracking to a certain extent the development of the Open Grid Services Architecture (OGSA) [60], described more fully in Section 2.3.1. The most widely deployed version (GT4) comprises a set of Web services (see Section 2.2.2) and a Web services container, as well as a set of non-Web services related tools. The services are used to manage tasks such as job launching and data transfer. Recently, Globus has moved away from Web services architecture, with GT5 reverting to the clientserver system previously used in GT2. The client tools supplied with all versions of Globus are command line based. All versions of Globus feature a security infrastructure based on X.509 [61] public key cryptography.

2.3.3 UNICORE

UNICORE (UNiform Interface to COmputing REsources) [21] is a grid middleware system that has been adopted for use by several international e-Infrastructure initiatives, for example the EU funded PRACE platform [62]. It implements a three-tier architecture in the form of client-gateway-server. Jobs are passed around the grid as Abstract Job Objects (AJOs), which are serialized Java objects. The first tier of the architecture consists of a client with which the user prepares and submits jobs and also to receive output back from the job. The middle tier consists of a gateway which controls authentication to the target resource. This tier also contains the Network Job Supervisor (NJS), UNICORE User Database (UADB) and the Incarnation Database (IDB) [21]. The NJS manages jobs and performs authorization of a user on a target resource. The UADB maps user certificates onto logins on the resource and the IDB translates AJOs to platform specific commands that the resource understands. The authenticated job is then submitted to the server tier (Target System Interface or TSI) which takes care of running the job on the target resource. With the exception of the TSI, the UNICORE system is implemented as a Java API and set of related programs. Version 6 of the UNICORE middleware presents a Web services interface based on the Open Grid Services Architecture [60] framework. A description of OGSA is given in Section 2.3.1. Security is maintained in the UNICORE system through the use of X.509 certificates.

2.3.4 gLite

gLite [22] is the product of the Enabling Grids for E-Science (EGEE) project [63], and uses components from a number of different sources to produce a middleware stack with a wide range of basic grid services. gLite is the middleware underpinning the EGEE grid, designed to process the 15 petabytes of data produced annually by the Large Hadron Collider instrument based at CERN.

2.3.5 QoS-CoS-Grid

QoS-Cos-Grid is a full middleware stack, with the QCG-Computing service at its heart. The QCG-Computing service (QCG BES/AR) is the domain-level component of the QoSCosGrid middleware stack, typically deployed on the head node of a compute cluster. QCG-Computing is a highly efficient implementation of the Open Grid Services Architecture Basic Execution Service (OGSA BES) Web service interface, designed to facilitate remote, multi-user access and policy-based job control routines provided by various queuing and batch systems [64, 65]. In contrast to many existing middleware services, the service uses Distributed Resource Management Application API (DRMAA) [66] to communicate with underlying queuing systems and has been successfully used with many of the most widely deployed cluster queuing systems, including Sun Grid Engine, Load Sharing Facility, Torque/Maui, PBS Professional, Condor, Apple XGrid, SLURM and LoadLeveler. QCG-Computing also supports basic, built-in, file transfer mechanisms.

Additionally, QCG-Computing is compliant with the OGF HPC Basic Profile specification. Moreover, it offers innovative remote interfaces to queuing systems for advance reservation management. This unique advance reservation capability facilitates cross-site co-allocation of computing resources, in order to execute distributed multi-scale applications. QCG-Computing has been designed to support a variety of plugins and modules for external communication as well as to handle a large number of concurrent requests from external clients and services.

2.3.6 Condor

Condor [67] is a software system for the management of high throughput computational environments, that is computing environments that can deliver large amounts of processing capacity over long periods of time, made up of collections of distributed

computing resources. It can be used to manage workload on a dedicated cluster of computers, or to distribute work to pools of idle desktop computers on a cycle scavenging basis, and is often suitable for running embarrassingly parallel type jobs.

For Condor systems made up of pools of user desktop machines, the user is given full control over the use of their machine, for example by allowing any Condor run processes to be killed when a user resumes work on the machine. This ensures that Condors use of the machine doesn't interfere with its owner/users use of it, and cause them to withdraw if from use in the pool. Originally Condor pools consisted of computational resources contained within a single administrative domain.

2.3.7 SAGA

SAGA (the Simple API for Grid Applications) [68] is an Open Grid Forum effort to produce a simple application programming interface to allow application developers to take advantage of the possibilities opened up by the advent of distributed e-infrastructures.

The SAGA API is intended to be simple, and as close to the programming paradigms and interfaces used by application developers to perform common operations they need to perform such as remote job submission and file transfer. The API specifically targets scientific applications, which aim to take advantage of some of the features that distributed e-infrastructures offer. The API targets developers of scientific applications who wish to grid enable their applications whilst spending as little time as possible learning new paradigms.

2.3.8 Data Transfer

A key requirement for operating on a distributed e-infrastructure is the need move data. In the most basic case, a user needs to move input files from their local workstation to the target resource they intend to use to run a simulation, and then move the output data back when the simulation is finished. GridFTP [69], an extension of FTP designed to use grid security credentials, has grown out of the Globus project and the Open Grid Forum, to established itself as the default mechanism for high performance data transfer between HPC resources.

GridFTP aims to provide reliable and high performance file transfer, through support for parallel and striped transfer, and fault tolerance. GridFTP's acceptance as a

de facto standard for file transfer means that it answers the problem of incompatibility between HPC file systems and storage devices, through the provision of a common interface (as used by the EUDAT collaborative data infrastructure [70] for example, where GridFTP provides a common user interface to storage resources running iRODS and OpenStack Swift).

2.3.9 Security Mechanisms

Access to grid resources is usually secured through authentication and authorization mechanisms based on X.509 certificates, a security credential used to authenticate the user when accessing a grid. To access the resources on a particular distributed e-infrastructure, the user needs a certificate recognized by that infrastructure. Certificates are generally issued on a national basis, by a national research certificate authority (CA). The Interoperable Grid Trust Federation [71] exists to ensure mutual trust between different national certificate issuing bodies. This means that certificates issued in one country will be accepted by e-infrastructure resources based in a different country.

2.4 Grid Usability

One of the key purposes of developing grid computing technologies has been to provide an infrastructure for the facilitation of large-scale scientific “experiments” [72, 44]. Conversely, one of the central problems facing users engaging with grid technologies has been lack of ease of use of existing tool kits [18, 73, 74, 75]. One of the reasons stated for this is that current tool kits are too cumbersome for a user to rapidly build prototype applications. Existing tool kits may also be resource intensive and require the user to install additional software packages that are not appropriate to the task they want to perform.

In addition to this, some toolkits have previously required users to install custom patched libraries for the middleware to work. This leads to system administrators having to maintain multiple copies of libraries on machines using the middleware. Another criticism levelled at many grid middleware systems is the difficulty in using their security infrastructure [76], particularly with reference to user management of X.509 certificates.

2.4.1 Lightweight Grid Middleware

The problem of monolithic software systems has been successfully addressed in other areas of computer science by producing lightweight, component-based designs. For example the UNIX operating system [77] is built from a small kernel with all extra functionality provided by separate programs that run on top of the kernel. The longevity of UNIX is testament to the power of this software development model.

WSRF::Lite [78] is one project that aims to address the problems of unwieldy and monolithic grid middlewares by developing a lightweight component-based architecture. WSRF::Lite implements the WSRF standard (discussed in Section 2.2.4) in Perl to produce a toolkit from which lightweight grid middleware components can be built. It is based on SOAP::Lite [79], a Perl Web service hosting environment. It aims to address some of the shortcomings of more heavyweight middlewares, such as Globus, by just addressing the specific problem of exposing program functionality as WSRF services. It also uses standard Perl libraries and is built using standard technologies such as SOAP, while still supporting a rich set of Web services standards, such as WS-Security [80]. WSRF::Lite provides a framework with which lightweight grid middleware tools can be quickly developed, such as the Application Hosting Environment described in the next section.

GROWL [81] is another example of a lightweight middleware system designed to address some of the short comings of other grid middlewares, using both a combination of Web service components and wrapper scripts to automate tasks performed by Globus grid middleware client tools.

2.5 Scientific Workflow Tools

Much research has been carried out into the design and implementation [82] of workflow management systems designed to automate common time consuming tasks that the scientist carries out when performing *in silico* experiments. These can range from the mundane (such as transferring a job's input files to a grid resource and then executing a job), to the more complicated (such as executing a series of interdependent tasks represented by a directed acyclic graph). User interaction with workflow management systems ranges from command line clients using textual descriptions of the workflow to graphical web portals [83] to rich desktop clients, or Problem Solving En-

vironments (PSEs). The advantage of the latter examples is that they allow workflows to be composed visually, using drag and drop interfaces.

There follows a brief summary of the key features of a selection of grid workflow management systems.

GridSpace2 [84] provides a user interface which allows users to orchestrate the atomic operations needed to execute distributed simulations, from moving data to initiating and managing application execution. GridSpace2 is based on the idea of exploratory programming, where each workflow application can be decomposed into a number of so-called snippets. Each snippet may be written in a different programming language; moreover, the Workbench enables users to execute entire experiments or just selected snippets. In this way, time-consuming submodels do not have to be started from scratch each time a modification is made during development. The Experiment Workbench web portal helps its users to iteratively develop virtual experiments with the use of scripting languages, including Ruby, Python and Perl. Underneath the Experiment Workbench the Experiment Execution Environment exists to evaluate snippets.

GridANT [85] is based on the Apache Ant build tool (a Java tool similar to UNIX make), but extended to execute grid workflows. The motivation behind using Ant is that it is portable (written in Java) and can easily express dependency between tasks in XML. The composition of workflows is done via an XML workflow specification and submitted via the command line, but a graphical tool is also provided to visualize and monitor the workflow. GridANT is currently able to orchestrate GT 2 and 3 grids, and is being extended to support others.

GridWorm [86] is a grid workflow orchestration system that is part of the UGanDA project. It is targeted at enterprise level workflows, rather than high performance computing users. It represents workflows using a language called GWLang which is based on Business Process Execution Language (BPEL) and Grid Service Flow Language (GSFL), and allows the orchestration of both standalone applications and Web services. It can submit jobs to the UGanDA MAGI system, and to Condor middleware.

Triana [87, 88] was originally designed as an analysis tool for gravitational wave data, but in practice is independent of any particular domain. Developed in Java, it comprises a toolkit of over 500 tools, and has been extended to orchestrate Web service workflows via its graphical workbench. It uses the Grid Application Prototype (GAP)

Interface to bind to arbitrary service oriented architectures, and provides a graphical interface to allow users to develop workflows.

Taverna [89] was designed to orchestrate workflows in bioinformatics applications, and was created to meet the specific workflow requirements of the myGrid project before any open source Web service workflow tools existed. Taverna uses the bespoke Simple Conceptual Unified Flow (SCUFL) language, an XML based notation for describing workflows using conceptual atomic tasks. Taverna also features the SCUFL workbench to compose workflows graphically. Taverna uses the Freefluo Web service orchestration tool, which is based on Web Services Flow Language (WSFL), a predecessor of BPEL.

BPEL [90] is an OASIS standard for Web service orchestration within industry. Although originally designed to meet the needs of business, several scientific projects have successfully used BPEL to orchestrate workflows [91, 92]. BPEL workflows are described in XML files. Since BPEL is a specification rather than a single tool, there are a number of graphical tools available to create these documents, or they can be created by hand.

The OMII-BPEL [92] project uses the open source ActiveBPEL engine from ActiveEndpoints. In common with other BPEL engines it both interacts with other Web services and is itself interacted with as a Web service. It is hosted in a Tomcat container, which has the advantage over some of the systems discussed above (e.g. GridANT) that the user does not need to keep their client running until the workflow has finished. They simply start the engine running and can come back to monitor it at any point. The OMII-BPEL graphical workflow composition tool, SEDNA, is based the free graphical composition tool ActiveBPEL Designer developed by ActiveEndpoints, which runs inside the Eclipse integrated development environment to provide a rich client platform interface.

2.6 Cloud Computing

Subsequent to the development of grid computing has been the rise of cloud computing. Cloud computing adopts many different forms, but a unifying idea behind cloud computing is that a business model is used to monetize access to compute cycles in some way, and provide access to various resources such as CPU, memory and storage

(known as Infrastructure as a Service clouds) and applications (Software as a Service or SaaS clouds). For example, with so called Infrastructure as a Service (IaaS) clouds, a user can gain access to a virtualized sever, and have complete control over that server as if it were his own machine, even though it is running in an administratively distinct domain.

Cloud computing is a rapidly growing area due to major strategic investments from global software players such as Microsoft, Amazon, Google and IBM. Cloud storage today is thriving, particularly due to its shared data at low cost capabilities however there are many security and legal issues in cloud computing that are yet to be resolved. Typically, access to cloud resources is metered, and users must pay for the amount of CPU time or number of megabytes of storage that they use, with cloud computing users entrusting their data and software to third-party providers. Cloud providers may be commercial companies selling access for profit, or academic institutions, providing access under a research funding model.

2.6.1 Virtualization

Virtualization is a broad term used in computer science to describe the abstraction of resources. It has found specific realization in many areas of computer science and information technology. The key benefit of virtualization is that it abstracts the details of an underlying hardware or software system from the user. The benefits of this are manifold: developers can code to a single virtualized interface or system rather than for a specific hardware implementation; multiple virtual instances of a system can often be run side by side on a single physical system (in machine virtualization for example); physical resources can be protected,

The growth of virtualization technologies, along with service oriented architectures (SOA), has also powered the development of cloud computing. In an IaaS cloud, the use of virtualized interfaces and systems means that the specific details of a cloud's architecture are hidden from consumers of the cloud resources. Several cloud computing models exist that take the form of virtualized servers running on hardware platforms managed by a cloud hosting company, where each user is given access to one or more virtual servers, solely under their control, which has the effect of separating users of the system from each other (since they each have access to their own virtualized system).

This also provides a degree of elasticity, as the number of virtual machines in a cloud environment can be greater than the number of physical servers available to the hosting entities.

The interfaces used to access cloud resources differ from platform to platform, and an extensive discussion of such interfaces is beyond the scope of this thesis. Examples of cloud software stacks include Eucalyptus [93], OpenStack [94] and OpenNebula [95]. To employ an IaaS in a scientific workflow, a user would have to install or upload their scientific code onto a cloud virtual machine instance along with the input data needed to run the application, then launch the VM instance on their target cloud platform, and retrieve the results once the execution of the application had completed.

2.7 High Performance Computing and Distributed e-Infrastructures

Distributed e-infrastructures can be made up from many different types of resource, including scientific instruments, data repositories, and computers, ranging in size from desktop machines to large scale parallel machines, made available at national and international levels. One area where grids have been widely implemented is in the sharing of high performance compute (HPC) resources, the distributed e-infrastructure model being a good way to get the most utility from machines which are expensive to purchase and run (examples include TeraGrid [96] and XSEDE [97] in the US and DEISA [98] and PRACE [62] in Europe).

Typical a grid of HPC resources will consist of multiple compute clusters made up of tightly coupled nodes containing single or multiple processor cores. A cluster will consist of mostly of worker nodes (that is, nodes that solely carry out computation), with a lesser number of log in/submit nodes, which act as log in, data staging, job preparation and job submission. As mentioned, the cluster will be managed by a local resource manager (LRM) such as PBS [33] from Altair Computing, LSF [35] from Platform Computing, or Load Leveler [99] from IBM. It is the job of the LRM to allocate jobs submitted to the cluster amongst the worker nodes, and queue jobs as they arrive, until appropriate resources are available. Typically users will submit jobs via a job submission interface such as Globus GRAM [100], or, depending on the policies of

the resource provider, by logging in to a job submission node and submitting their job directly to the queue.

The various different processor queues on a HPC machine have associated maximum wall times, that is, maximum times that the jobs in the particular queue are allowed to run for. When users submit jobs, they are able to specify the maximum amount of time that their job will run for, or, if not specified the scheduler will assign the job a default run time configured by the resource administrator. The LRM uses a scheduling algorithm, or combinations of scheduling algorithms, to order jobs in a queue. Examples of algorithms commonly used are *first come first serve* or *back fill*. In the first case jobs are run on resources in the order that they are submitted. In the latter, the scheduler attempts to fill holes in the queue with jobs that will fit, even if jobs are not run in the order that they arrive.

Typically all nodes of the cluster will be within a single administrative domain; often the back end worker nodes will be on their own private network, with only the log in nodes network reachable. The queuing systems of such clusters are usually configured to run parallel applications, such as those written using the Message Passing Interface (MPI) (*cf.* 2.1.3.1).

2.7.1 Distributed Application Requirements

While the MPI specification primarily concerns itself with intra-machine process communication, the prevalence of grid middleware tools such as Globus on HPC resources has facilitated the development of inter-machine versions of MPI which use grid middleware to create secure communication channels between resources that are part of the grid. One of the most widely used implementations of MPI over grid middleware is MPIg [101], formerly called MPICH-G2, developed at Argonne National Laboratory.

The benefits of doing this are twofold. Firstly, it allows problems that can not be addressed by a single HPC resource (because it requires more memory or CPU power than the resource is able to provide for example) to be run on a 'meta-computer' made up of multiple actual compute resources. Secondly, it allows large problems which will potentially take a long time to run on a single machine (due to the load on the machine, meaning that the job will have to wait in the queue for a long time) to harvest smaller numbers of processors from multiple machines, meaning that they could in principle run

faster, by avoiding long queue wait times [11].

2.8 On Demand Access Mechanisms

A growing number of projects seek to exploit high performance compute grids for which the batch processing model is not sufficient to support the scientific investigations that they want to perform. Such projects frequently require interactive access to one or more resources, or simultaneous access to a number of resources. For example, the SPICE project [44] used steered simulations conducted on a grid to understand DNA translocation across lipid membranes embedded in protein nanopores. In compute terms this requires simultaneous, interactive access to both compute resources, where the simulation code runs, and also to high performance visualization resources to display the results. The fact that the simulations are steered [102] by the investigating scientist, that is, parameters of the simulation are altered while it is running, means that interactive access to resources is required - it is not acceptable for the scientist to have to wait until his or her job has reached the top of the batch queue, potentially having to wait long in to the night to start their experiment. The need for the ability to reserve resources is recognized by McGough *et al.* [103] in their discussion of the need to predict when jobs in a computational workflow will run.

MPI-g/MPICH-G2 applications [101] provide another use case where simultaneous access to multiple resources is required. MPI-g applications are parallel codes distributed across a number of grid resources, often because no single machine has enough CPU cores/memory to run the application on its own. Examples of MPI-g/MPICH-G2 applications include Nektar [24], a fluid dynamics application to simulate human arterial blood flow, and Vortronics [24], an application used to investigate computationally-intensive problems in fluid dynamics.

In order to support the computationally intensive projects discussed above, a computational grid with appropriate advanced reservation policies is required. While it is often technically possible to make an advanced reservation of time on a single machine using the machine's queuing system (for example PBSPro [33]), often this facility is either not offered at all, or only available to system administrators, meaning that an end user has to jump through a number of administrative hoops in order to make a reservation.

QCG-Computing (see §2.3.5) is a job submission and advanced reservation interface designed to allow advanced a user to reserve a block of processing cores or nodes on a cluster for a defined duration, starting at a time specified by the user. By making multiple reservations at different sites, the user can co-reserve access to numerous machines, in order to run a workflow for example. Typically a user will only be able to plan a future interactive or cross-site session - they will have to wait until all of the resources that they want to access are available. The ability to make cross-site advanced reservations does not imply that a user will be given immediate interactive access to a resource in order to carry out his or her work.

The Highly Available Robust Co-scheduler (HARC) [104] is a similar system designed not only to allow advanced reservations of time to be made on a single machine, but to allow cross-site reservations of time to be made on a number compute resources and the switched light path networks that connect them. HARC works by deploying software components, called resource managers, on to the resources that it is managing reservations for, and provides an interface which users interact with to make, query and cancel reservations. HARC uses the Paxos Commit protocol [105, 106] to ensure that reservation requests between a number of machines are dealt with atomistically. Paxos Commit removes the problem of Transaction Manager failure experienced by classic two phased commit protocols by replicating the Transaction Managers, so that as long as a majority of Transaction Managers are running (called Acceptors in Paxos) the system will still be able to function.

Related in aims to advanced resource reservation is the concept of job priority and pre-emption. Based on the idea that urgent jobs (for example simulations of approaching hurricanes) should take priority over regular jobs, job pre-emption requires resource providers to put in place policies and mechanisms to allow urgent jobs to be run immediately, rather than waiting in a queuing system. The US XSEDE infrastructure uses the Special PRiority and Urgent Computing Environment (SPRUCE) [107]. SPRUCE allows certain users, deemed to be running high priority jobs, to have a computational resource put at their disposal whenever they need it. With SPRUCE a user can gain immediate interactive access to a machine without having to make an advanced reservation.

2.9 Summary

This chapter has sought to provide a brief background review of the current state of play in the world of high performance computing and grids built from HPC machines. As we have seen, computational e-infrastructures are typically made up of a great many heterogeneous resources. As well as differing architectures and operating systems, the resources will typically use a wide range of queue management and scheduling software, each of which uses its own proprietary protocols and with no common interface between them. Grid middleware builds a layer of abstraction on top of the management systems of the individual resources on the e-infrastructure to provide a consistent interface with which users can interact. The Open Grid Services Architecture defines a standard set of service abstractions on which higher level tools can be built.

All middleware solutions described in Section 2.3, such as Globus and UNICORE, present the end user with similar levels of abstraction: they are based around a model whereby a user interacts with a single machine in order to run her job. While this model has been extended, for example in the submission of MPI-g jobs where a user interacts with a single service that then interacts with the individual compute resources on the users behalf, this does not really provide a high enough level of abstraction for easy, scalable user interaction across a grid. Workflow engines, especially ones that use a visual programming interface, provide a higher level of abstraction for user-grid interaction, but still require the user to perform many tasks, such as resource selection, manually.

HPC resources shared on a grid are typically made up of multiple tightly coupled processing units, with a local resource management system tasked with distributing work between the processors. With the middleware solutions discussed above, the onus is on the user to decide where to run their job. Resource providers will typically assign a notional cost to a unit of work on their resource, but usually the end user will be unaware of this cost, as resources are provided on a per project basis.

Advance reservation of resources is necessary in some cases, such as where access to resource is at a specific time to run a cross site parallel application, but leads to under utilization of the resource, as the local resource manager has to drain the job queues prior to the reservation, to ensure that the reserved queue is available at the specified time.

Chapter 3

Virtualizing Access to Scientific Applications

In this chapter we examine several distributed e-infrastructure application case studies. From these case studies we derive the requirements users have of such systems, and consider the levels of abstraction required to simplify the use of distributed e-infrastructure systems. We go on to discuss the design of software systems that can help mitigate the problems of distributed e-infrastructure use.

3.1 HPC Grid Use Cases

In order to understand the needs of distributed e-infrastructure users, it is useful to consider the different scientific studies that HPC grids are routinely used to support. Here we consider three different use cases: a distributed grid application designed to allow clinicians to calculate the efficacy of a range of drugs for the treatment of HIV, a grid based bloodflow simulation tool developed to assist in the surgical planning of neurosurgical procedures, and the simplest case of launching a parallel application on a supercomputer class resources. These use cases were chosen as they represent ambitious cases where non-typical people (that is, people who are not professional computational scientists with backgrounds in the physical sciences) are trying to make use of HPC grids as part of their scientific endeavours.

3.1.1 Calculating Drug Binding Affinities

A major problem in the treatment of AIDS is the development of drug resistance by the human immuno-deficiency virus (HIV). HIV-1 protease is the enzyme which is

crucial to the role of the maturation of the virus, and is therefore an attractive target for HIV/AIDS therapy. Although several effective treatment regimes have been devised which involve inhibitors that target several viral proteins [108], the emergence of drug resistant mutations in these proteins is a contributing factor to the eventual failure of treatment.

Doctors have limited ways of matching a drug to the unique profile of the HIV virus as it mutates in each patient. A drug treatment regimen is prescribed using knowledge-based clinical decision support software, which attempts to determine optimal inhibitors using existing clinical records of treatment response to various mutational strains. The patient's immune response is used as a gauge of the drug's effectiveness and is periodically monitored so that ineffective treatment can be minimized through an appropriate change in the regimen. The efficacy of such clinical decision support systems can be enhanced through a unification of existing databases, as well as integration with means of assessing drug resistance at the molecular level [109].

At the molecular level it is the biochemical binding affinity (free energy) with which an inhibitor binds to a protein target that determines its efficacy. Experimental methods for determining biomolecular binding affinities are well established and have been implemented to study the in-vitro resistance conferred by particular mutations. These in turn add invaluable information to any decision support system, but are limited as studies are performed usually on key characteristic mutations and not with respect to the unique viral sequence of a patient. An exhaustive experimental determination of drug binding affinities in a patient-specific approach is far too costly and time-consuming to perform in any clinically relevant way.

Computational methods also exist for determining biomolecular binding affinities. A recent study has shown that, using molecular dynamics (MD) simulation techniques, a patient specific model can be built based on the patient's genomic structure, and then used to test the binding of a range of drugs *in silico*, by implementing a protocol that makes such simulations accurate and repeatable [110]. Ultimately, these binding affinities allow available drugs to be ranked based on how well they will bind with a specific patient's mutations, and hence inform the clinical decision making process. The study made use of a tool, the Binding Affinity Calculator (BAC) [111], for the rapid and automated construction, deployment, implementation and post processing stages of

the molecular simulations across multiple supercomputing grid-based resources.

The generic technique can be applied to a range of other disease cases such as cancer [112], and also to new drug discovery. In the HIV case, BAC automates binding affinity calculations for all nine drugs currently available to inhibit HIV-1 protease and for an arbitrary number of mutations away from a given wild-type sequence. In order to achieve repeatable, accurate results, ensembles of simulations must be run, to generate sufficient statistics on which to base a prediction. This means, that for each drug that is to be tested, 50 separate simulations must be run, each requiring 64-128 CPU cores. The turn around time required to model all patient-drug interactions using BAC for such studies is 12-18 hours, with optimal computational resources (taking advantage of the thousands of cores available on today's petascale resources); this is more than suitable for the time-scales required for effective clinical decision support. Given enough computational power such that binding affinity calculations can be routinely applied, the potential to achieve patient-specific HIV decision support may then become realistic.

The BAC constitutes a complex distributed e-infrastructure workflow that depends on a range of computational resources at different scales and a number of computational codes. For a particular drug/mutation combination, BAC initially runs a set of job setup scripts on a local cluster to create the necessary input files for set of simulations by taking a default model and customising with the mutations and drugs under consideration. Next this model data is staged to a petascale supercomputer class resource typically on the US XSEDE or EU PRACE e-infrastructures, and the NAMD molecular dynamics code is used to run a number of equilibration steps to prepare the model, followed by a number of nano seconds of simulation, which in reality take several hours of wall-clock time using between 32 and 64 processors per simulation. To increase the sampling effectiveness, often several such simulations are run in parallel. The resource used is typically chosen by the researcher based on precursory glance at the queue lengths on several large XSEDE [97] and PRACE [62] resources where the NAMD application is deployed, Once the simulation chain is complete, the output data (measured in terabytes) is staged to a smaller scale cluster for post processing, using the Amber molecular dynamics package to calculate a drug binding affinity. Typically this workflow might need to be run in its entirety nine times (once for each of the nine

FDA approved HIV-1 protease inhibitors) in order to generate a ranking of the drugs for an individual's set of mutations.

3.1.2 Computational Investigations of Cranial Haemodynamics

Cardiovascular disease is the cause of a large number of deaths in the developed world [113]. Cerebral blood flow behaviour plays a crucial role in the understanding, diagnosis and treatment of the disease; problems are often due to anomalous blood flow behaviour in the neighbourhood of bifurcations and aneurysms within the brain, leading to strokes for example, although the details are not well understood [114]. Experimental studies are often impractical owing to the difficulty of measuring behaviour in humans; however, computer tomography (CT) and three-dimensional rotational angiography (3DRA) enable static and dynamical data acquisition [115].

Notwithstanding these advances in measurement methods, modelling and simulation have a crucial role to play in haemodynamics. There are evident limitations to the experimental methods which can be complemented by simulation. The GENIUS project [11] is concerned with performing entire brain neurovasculature blood flow simulations in support of clinical neurosurgery. Simulation offers the clinician the possibility of performing non-invasive, patient specific, virtual experiments to plan and study the effects of certain courses of surgical treatment with no danger to the patient, including support for diagnosis, therapy and planning of vascular treatment [116]. Simulation techniques also offer the prospect of modelling the poorly understood flow patterns in the normal brain and in neurovascular pathologies such as aneurysms and arterio-venous malformations (AVMs).

Simulation input data is provided by CT 3DRA scans, from which a patient specific models of the vascular structure of the brain is built. Conventional continuum solvers, based on finite difference, finite volume and finite element codes, are beset with problems in three spatial dimensions due to the computational costs of mesh generation, the need to solve the auxiliary Poisson equation for the pressure field, and various approximations associated with the calculation of the shear stress from the flow velocity field [117]. For large systems, such as those the GENIUS project is concerned with in addressing cerebral blood flow, it is essential to develop and utilize scalable, high performance parallel codes, which further complicates the use of continuum models.

To overcome these limitations the project uses a highly optimized flow model based on the lattice-Boltzmann (LB) method, called HemeLB [118].

The HemeLB application is the basis of an efficient brain blood flow simulator that can act in an advisory capacity to the surgeon before (and eventually also during) surgery. HemeLB is ideally suited to studying flow in sparse complex geometries such as the neurovasculature. The algorithm is based on a data layout that automatically handles lattices with large unstructured regions covered by void (i.e. non-fluid) nodes. In the parallel code, spatial domain decomposition is handled by a simple and fast cluster algorithm; partial overlapping of communication between different processor domains and on-processor computation is also exploited to enhance performance. In addition to the lattice-Boltzmann solver, HemeLB includes an in-built ray-tracing engine, giving it real time visualization capabilities, with frames of visualization rendered on the same processor cores as the simulation. A simple desktop display and computational steering client connects to the application at run time to display the visualization and allow the user to interact with it.

3.1.2.1 GENIUS e-Infrastructure Requirements

The requirements of the medical computing scenarios that underpin the GENIUS project are not met by the traditional ‘batch’ model supported by the majority of HPC resource providers [10, 30]. Simulations, used to support clinical procedures, require large numbers of processor cores, and have to be run when required by the clinician. Surgical procedures cannot be postponed until a simulation reaches the top of a batch queue; emergency medical simulations must either pre-empt all other jobs running on a machine, or be able to be scheduled into the clinical workflow.

Clinicians are not concerned with where simulations are running, nor the details of reservations; thus features such as advanced reservations and emergency computing capabilities, managed application launching and data marshalling are all done behind the scenes. Of key concern are the time scales involved in the clinical decision making process in the treatment of arterio-venous malformations and aneurysms. From the acquisition of a 3D dataset (which is typically 2 to 4 GB in size), to the surgical treatment procedure, a time scale of 15 to 20 minutes is typical, and for such computational approaches to be clinically relevant, the entire workflow has to fit into this time scale.

These requirements lead to a demand on resource providers to implement policies and tools that allow computational access to be gained as and when required, so that such methodologies can be incorporated into a clinician's day to day clinical activities, or a researcher's normal activities, rather than just providing such facilities on an *ad hoc* basis [10]. GENIUS therefore make use of enabling technologies such as HARC [104], which allow HPC resources to be used in a more interactive manner.

The HemeLB bloodflow simulator at the heart of the GENIUS project is capable of running a single simulation distributed across multiple resources, using the MPIg middleware [101]. Tying together multiple resources in this fashion can decrease the turnaround time substantially but, to effectively run HemeLB across multiple resources, CPU time on disparate resources needs to be (co)-reserved in advance. Since the resources provided by a single e-infrastructure may not always be sufficiently powerful or appropriate to run large-scale distributed models, resources provided by multiple e-infrastructures may need to be federated in order for a particular investigation to be conducted [119].

In order to deliver the sought after flexible access to computational resources, the GENIUS project also has advanced networking needs, typically requiring resources to be connected with dedicated lightpath links to perform inter-machine simulations and facilitate real-time steering and visualization. In addition to these hardware requirements, suitable middleware tools are needed to hide the components of the grid from researchers and clinicians.

3.1.2.2 GENIUS Workflow

The software environment deployed by the GENIUS project aims to bring to the forefront details and processes clinicians need to be aware of, such as (i) the process of image segmentation to obtain a 3D neurovascular model, (ii) the specification of pressure and velocity boundary conditions, and (iii) the 'real-time' rendered images.

GENIUS researchers use a rich client platform tool to automate the entire simulation building, launching and interaction workflow. This client, run by the clinician on a desktop workstation, automates the whole process of retrieving the imaging data from a data repository, running a segmentation tool to create a HemeLB model, staging the model to a GridFTP server, launching the application on the grid, and allowing the

clinician to steer and visualize the running simulation.

Security and anonymization are of primary concern when dealing with medical data and, as such, all data sets used by clinicians and researchers in the GENIUS project are anonymized, adhering to a protocol specified by clinical partners in the UK National Health Service (NHS) before they leave the confines of the NHS secure network. This protocol ensures that the datasets used to run simulations on grid resources contain no data that could be used to identify the individual patient concerned.

3.1.3 Single Grid Application Launching

Typically in supercomputer/HPC usage scenarios, an expert user (or system administrator) installs and tunes an application on an HPC system, and makes it available to users within a research community. This draws a parallel with many different communities that use parallel applications on high performance compute resources, such as the UK Collaborative Computational Projects (CCPs) [120], where a group of expert users/developers develop a code, which they then share with the end user community. Users of a particular application will use the resources on the e-infrastructure where they know their application is installed.

The following simple use case illustrates the process a user goes through to launch one such application, the NAMD [121] molecular dynamics code, on a remote grid resource. Typically the user will go through the steps of selecting a computational resource on which to run the application, where she know it to be pre-installed. She may log in to a set of resources to check the load on each machine and select the one she believes will deliver the fastest turn around time. She then uploads her NAMD input files from her local machine to the target resource, writes an appropriate job submission script that runs the NAMD application with her chosen input files, and then submits it to the scheduler on the target resource, either directly, or via a middleware interface such as Globus GRAM. Once the application is launched, the use will monitor it until it completes, then download the NAMD output files back to her local machine for analysis, or leave them on a remote file system to perform further computation.

3.2 Analysing the Use Cases

The use case scenarios considered above have many features in common, even though they differ in the level to which they seek to exploit a distributed e-infrastructure as a whole (instead of just individual resources that constitute the e-infrastructure). The commonalities include:

- For supercomputer class applications, the user generally has to install the application himself; it is not possible simply to stage an executable to the target resource as it requires too much bespoke tailoring to the particular hardware setup of the resource (although application binary staging is commonly occurs on systems such as Condor).
- Generally a group of researchers will want to use the same application on a given resource. However, many users will not know where a particular application is installed on a target system - they will also not know the best way to run the application on a particular system. Often with supercomputer class systems, applications have to be run in specific ways to achieve the best performance. The communities expert user typically has to spend time educating other users on the vagaries of different queuing systems and machines.
- Typically, the user will need to stage data to the supercomputer before she is able to execute her application. Therefore, the supercomputer must provide accessible interfaces over which data can be staged.
- In order to launch an application, the user has to prepare a description of the job that they want to run to submit to the queue management system on their target resource, in a format that the queue management system understands and which is potentially incompatible with other instances of the same queue management system running on other resources.
- Once the job has been submitted, the user monitors the progress of their job through the queuing system, using interfaces provided by the resources.
- Applications can consist of multiple computational codes launched on multiple distributed resources, as well as single codes launched on single resources. Some

applications may require access to resources that cannot be provided by a single e-infrastructure, and may therefore need to be federated across multiple e-infrastructures.

- Applications can get their data from multiple sources, such as online data repositories and databases, and store their output data in similar resources.
- Typically, users will be given allocations of time on individual resources, or the e-infrastructure as a whole, through awards made to their project's principal investigator. These allocations will have a notional associated cost, the cost per CPU hour, derived by the resource operator from their running costs and a projected resource utilization.

The distributed e-infrastructure user's primary concern is running their application in a timely fashion, in order to obtain results that further their scientific objectives. All the services and facilities provided by an e-infrastructure are subservient to this end. Typically, the user does not care which machine on the e-infrastructure their application is run on, as long as results are delivered within a time frame that makes them useful, whether that is the time to publish a scientific paper, or the time to conduct a potentially life-saving medical simulation.

3.3 Review of Current Middleware Approaches

As we have shown above, the user of a distributed e-infrastructure is most interested in running applications, be they single codes or multiple distributed codes linked together in some way. However, the predominant concept in current HPC e-infrastructures is the 'job' or the submission of an application to a resource queue management system. We believe the concept of a job is too low level for the average grid user. By way of illustration, consider the following scenario of launching the aforementioned HemeLB application on the Stampede resource on the XSEDE e-infrastructure using the Globus Toolkit middleware.

Firstly, the user has to generate a proxy certificate, a short lived credential generated from his own full grid certificate, which allows the middleware tools to perform actions on his behalf. To do this, he uses the command:

```
grid-proxy-init
```

Once the proxy is generated, the user can stage his input data (the model used to run the simulation) to his target resource as follows:

```
globus-url-copy file:///home/username/hemelb/input/* \
gsiftp://stampede.tacc.xsede.org/username/*
```

Next, he needs to generate a job description file which specifies the number of processors to use, the location of the executable, the type of job, redirects for the standard out and standard error, and arguments to the application. This is written using the Globus JDD XML syntax, and saved in a file called `ex.jdd`, shown in listing 3.1.

Listing 3.1: JDD job description to run the HemeLB code

```
<?xml version="1.0"?>
<job>
  <factoryEndpoint xmlns:gram="http://www.globus.org/namespaces/2004/10/gram/job"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <wsa:Address>
      https://tg-login.ncsa.teragrid.org:8443/wsrf/services/ManagedJobFactoryService
    </wsa:Address>
    <wsa:ReferenceProperties>
      <gram:ResourceID>PBS</gram:ResourceID>
    </wsa:ReferenceProperties>
  </factoryEndpoint>
  <executable>/home/ac/smanos/pub/bin/hemelb</executable>
  <directory>${GLOBUS_USER_HOME}/tmp</directory>
  <argument>${GLOBUS_USER_HOME}/tmp</argument>
  <argument>1000</argument>
  <argument>100000</argument>
  <argument>0.01</argument>
  <argument>0</argument>
  <argument>0</argument>
  <argument>107112</argument>
  <environment>
    <name>MP_BUFFER_MEM</name>
    <value>32m</value>
  </environment>
  <environment>
    <name>LD_LIBRARY_PATH</name>
    <value>/usr/lib:/usr/X11R6/lib:/usr/local/lib</value>
  </environment>
  <stdout>${GLOBUS_USER_HOME}/tmp/stdout.txt</stdout>
  <stderr>${GLOBUS_USER_HOME}/tmp/stderr.txt</stderr>
  <project>TG-ASC090009</project>
  <maxTime>20</maxTime>
  <jobType>mpi</jobType>
  <extensions>
    <resourceAllocationGroup>
      <hostType>ia64-compute</hostType>
      <hostCount>2</hostCount>
      <cpusPerHost>2</cpusPerHost>
    </resourceAllocationGroup>
  </extensions>
</job>
```

The user submits the job description file to the GRAM service running on the NCSA machine with the following command:

```
globusrun-ws -submit -batch -J -S -f ex.jdd 2>\&1 1> epr_file
```

This is then translated by the GRAM service into a job description understood by the underlying queue management system on the machine (in this case PBS). The submission command generates a handle to the job, which is saved in the file `epr_file`. The user can monitor the status of his job with the following command:

```
globusrun-ws -status -job-epr -file epr_file
```

Once the job is complete, the user can stage back the output data from the HemeLB code as follows:

```
globus-url-copy gsiftp://stampede.tacc.xsede.org/username/* \  
file:///home/username/hemelb/input/*
```

The above scenario assumes that the user has already decided on which resource to use (for example, through logging in to a set of resources via GSISSH and examining the lengths of the queues on the machine). It also assumes that the application executes without any problems, which is often not the case when running supercomputer class applications.

3.4 Requirements

By looking at the common distributed e-infrastructure use cases in (*cf.* §3.1) and the current practice of running applications on e-infrastructure based supercomputer class resources (*cf.* §3.3), we can begin to derive a set of requirements for a system designed to make the use of distributed e-infrastructure based applications more transparent to the end user. We enumerate these requirements in the list below:

1. Current distributed e-infrastructure systems focus on submitting jobs to batch schedulers on computational resources, meaning the user has to interact at both job level and resource level. Since users predominant interest is running their application within a useful time frame, our system should promote applications as a first class resource concept. All user interactions should be with the application, rather than the machine, scheduler and job.
2. Distributed e-infrastructure user communities consist of individuals with a range of different skills and experiences. Since building and launching an application

on a supercomputer class resource is in itself a non-trivial task, many users look to an ‘expert’ within their community to build and maintain their applications, and show them how to best use the application on each individual resource. Our system should provide a mechanism to capture the knowledge of the expert users within a community, and facilitate the sharing of knowledge between users.

3. Current distributed e-infrastructure job submission mechanisms put the onus on the user to manage and curate their application’s output data. Our system should preserve the full state of each instance of an application, including all parameters and data used to launch the application, and all simulation output. This will assist with tracing the provenance of simulation results, and is key to simulation reproducibility.
4. Current distributed e-infrastructure middleware tools require the user to perform a number of steps in order to launch their code. Our system should reduce the number of steps required to the minimum number possible in order to successfully run an application.
5. The process of submitting an application should be initiated by the user and done at the user’s convenience, rather than at a time specified by the computational resource provider.
6. With current systems the onus is on the user to choose the resource on which they want to run, meaning that they often choose the one they think will be able to run their job fastest, or the one they are most comfortable using. Instead of requiring users to choose resources, our system should allow the user to specify requirements for their application run, such as the time they need the results to be produced by, or the maximum cost they are willing to pay in order to run the application.
7. Current systems require users to manually stage their data to their target resource before launching their job. Our system should take care of automatically staging data on behalf of the user.
8. Current systems require the user to generate complicated job description documents in order to submit their job. Our system should allow the user to launch

their application using the simplest set of requirements possible, and take care of generating whatever underlying job descriptions the middleware on the underlying resource requires.

9. In current systems, users are required to manage their security credentials, and generate proxies from those credentials, manually. Our system should manage credential delegation on behalf of the user.
10. Users of supercomputer class resources often have access to a number of such resources via different distributed computational e-infrastructure, running different middleware stacks, requiring them to learn how to use different middleware tools to submit their jobs. Our system should present a uniform interface to the user to access resources running different middleware stacks, allowing the user to transparently access federated resources from multiple distributed e-infrastructure.
11. Computational e-infrastructure resource providers often invest much time and effort in selecting and maintaining their back end middleware stacks. Our system should be as minimally invasive as possible, requiring resource administrators to make as few modifications to their system as possible, and interfacing with the back end middleware provided by the resource.
12. Users may require access to multiple resources in order to run their application. For example, for an application that consists of a simulation code and a coupled visualization engine, the user would need access to a compute resource and a visualization resource.

3.5 Meeting the Requirements

In order to meet the requirements outlined above, we have developed a system that consists of two independent but linked systems. These systems can be used as standalone middleware tools, but work best when coupled together. They are designed to satisfy the requirements outlined above by abstracting from the user much of the information that they currently must possess in order to interact with the distributed e-infrastructure: the names of the machines they want to run on, the location of application binaries on those machines, and the mechanisms needed to stage data to and

from machines.

The first system is the *Application Hosting Environment (AHE)*, described in the remainder of this chapter. AHE is a middleware system designed to ease user interaction with the grid by promoting the application to the status of first class resource, by a process we term *grid application virtualization* [9, 6, 1].

The second system is the *Resource Allocation Market Place (RAMP)*, described in Chapters 6 and 7. RAMP is a multi-agent distributed system that implements a multi-attribute, combinatorial reverse auction market place, to allow users to ask resources to provide quotes to run jobs, which satisfy a set of requirements specified by the user.

We go on to describe these systems in detail in the remainder of this chapter.

3.6 Service Oriented Computational Science

Our approach to simplifying the use of grid computing and addressing the requirements described in Section 3.4 is a processes that we call *grid application virtualization*. The key aim of virtualization [122] is to abstract away all the details of an underlying hardware or software system from the concern of the user. The benefits are manifold: developers can code to a single virtualized interface or system rather than for a specific hardware implementation; multiple virtual instances of a system can often be run side by side on a single physical system (in machine virtualization for example); and physical resources can be protected.

While virtualization technologies certainly reduce the complexity of using a system, and especially when working across multiple heterogeneous computing environments, they are not widely deployed in high performance computing scenarios. As its name suggest, HPC seeks to obtain maximum performance from computing platforms. Extra software layers impact detrimentally on performance, meaning that in HPC scenarios users typically run the applications as close to the ‘bare metal’ as possible. In addition to the performance degradation introduced by virtualization technologies, choosing what details to abstract in a virtualized interface is itself very important. Grid and cloud computing support different interaction models. In grid computing, the user interacts with an individual resource (or sometimes a broker) in order to launch jobs into a queuing system. In cloud computing, users interacts with a virtual server, in effect putting them in control of their own complete operating system. Both of these

interaction models put the onus on the user to understand very specific details of the system that they are dealing with, making life difficult for the end user, typically a scientist who wants to progress his or her scientific investigations without any specific usability hurdles obstructing the pathway.

To address these problems, we have developed a software layer designed to implement the Software as a Service cloud paradigm for scientific applications that rely on high performance computing, mediated by the *Application Interaction Model* which we describe in Section 3.7, derived from the user requirements discussed in Section 3.4. This model is based on the insight that many e-infrastructures impose a steep learning curve on the majority of end users, who do not possess the technical expertise for the most part to compile, optimize, install, debug and finally launch their applications; they simply want to run their applications, obtain results and focus on their scientific endeavours. While an application may consist of a single execution of a computational code, it could also consist of a complex set of operations involving multiple codes, connected as a workflow; AHE enables all kinds of applications to be treated as simple “atomic” units, helping realize the original vision of a grid as “as distributed computing conducted transparently by disparate organizations across multiple administrative domains” [16].

3.6.1 Design Constraints

In addition to the functional requirements we discussed in Section 3.4, a number of constraints were placed on the AHE’s design. The problems associated with ‘heavyweight’ middleware solutions described in Section 2.4 have greatly influenced the design of the Application Hosting Environment. Specifically, they have led to the following constraints on the AHE design, in an attempt to simplify the end user’s experience of a distributed e-infrastructure:

- The user’s machine does not have to have specific client software installed for the middleware on the target grid resource. Instead the AHE client should provide a uniform interface to multiple grid middlewares.
- The client machine is behind a firewall that uses network address translation (NAT) [123]. The client cannot therefore accept arbitrary inbound network connections, and needs to poll the AHE server to find the status of an application

instance.

- The client machine needs to be able to upload input files to and download output files from a grid resource, but does not have GridFTP client software installed. An intermediate file staging area should therefore be used to stage files between the client and the target grid resource.
- The client has no knowledge of the location of the application it wants to run on the target grid resource, and it maintains no information on specific environment variables that must be set to run the application. Such information is knowledge possessed by the expert user, which should be maintained on a central AHE service.
- The client should not be affected by changes to a remote grid resource, such as if its underlying middleware changes from GT2 to GT4. Since AHE is intended to provide an interface to the target grid resource, a change to the underlying middleware used on the resource is not seen by the user, as long as it is supported by AHE.
- The client should not have to be installed on a single machine; the user should be able to move between clients on different machines and access the applications that she has launched. The user should even be able to use a combination of different clients, for example using a command line client to launch an application and a GUI client to monitor it. The client therefore must maintain no information about a running application's state. All state information should be maintained on a central service that is queried by the client.

These constraints have led to the design of a lightweight client for AHE, which is simple to install and does not require the user to install any extra libraries or software. It should be noted that this design does not remove the need for middleware solutions such as Globus on the target grid resource; indeed AHE provides an interface to run applications on several different underlying grid middlewares so it is essential that grid resource providers maintain a supported middleware installation on their machines. What the design does do is simplify the experience of the end user. We discuss AHE fully in Chapter 3.8.

3.7 The Application Interaction Model

The idea of an application is an instinctive abstraction to facilitate user interaction with HPC resources; it is the fundamental concept that users are interested in (since it is what they use to generate their scientific results) and can encapsulate all parameters and activities that a scientist wishing to use an HPC machine to perform an *in silico* experiment needs to deal with. Based on this insight, gained though the long experience of the author working in the field, and our analysis of the use cases presented in Section 3.2, we have derived the Application Interaction Model, designed to allow users to easily control virtualized applications running on remote e-infrastructures. Traditionally, HPC focuses on the concept of ‘jobs’ to describe distinct workloads submitted to a batch queue. We purposefully focus on the concept of *applications*. An application is a higher level concept than a job; although an application could be realized by a single HPC job, it could equally correspond to a coupled simulation, where two codes (launched as two HPC jobs) pass parameters between themselves, or a steered application which requires steering Web services to be initialized before the code is launched, or a workflow of arbitrary complexity. However the application is composed the user should still interact with a single entity to control the execution of all components of the application. As the application abstraction seemed to fit the case studies considered, an extensive and systematic requirements gathering exercise was not performed.

We define the Application Interaction Model as follows:

1. The virtualized application is the central entity in the Application Interaction Model.
2. An application does not necessarily correspond to a single computational code - it could be composed of multiple computational codes linked together in a workflow, or a computational code and associated steering web services. However it is presented to the user as a single application.
3. The application encapsulates all of the details of how to launch it, such as where the binaries that constitute the application are located, how to interact with individual resources and so on. These details are shielded from the user, who does not need to know anything about the underlying details.

4. Each instance of an application is controlled through a separate application instance, through which it is controlled. The application instance encapsulates all of the state associated with that run of the application, such as the input and output data, the application parameters and so on.
5. All user interaction occurs through the virtualized application instance, which causes the computational code(s) which constitute the application to be launched on back-end computational resources.
6. Operations on the application instance allow the user to stage data associated with the application to the resource where it is needed, launch, monitor and terminate the application. These operations have an effect on the codes running on remote grid resources.

A schematic representation of this interaction model is shown in figure 3.1. The principal motivation behind this approach is to simplify the use of e-infrastructures, by introducing an abstraction layer between the users and the high end computing resources available to them which hides the complexity of the latter, providing an abstract interface to scientific applications deployed on a grid. This abstraction layer takes care of the process of launching the application on one or more HPC resources, and reduces the interaction with an application to those operations most relevant to the user.

The Application Interaction Model implies that the task of deploying and configuring an application is taken care of by a system administrator, or a community's 'expert user'. This draws a parallel with many different communities that use applications on high performance computing resources, such as the UK Collaborative Computational Projects (CCPs) [120], where a group of programmers develop a code, which they then distribute to an end user community. Once the expert user has configured AHE to share an application, end users can use clients installed on their workstation, tablet or mobile phone to launch and monitor the application across a variety of geographically distributed computational resources.

3.8 The Application Hosting Environment

The Application Hosting Environment (AHE) is our implementation of the Application Interaction Model. AHE is based on two key concepts to promote usability: application

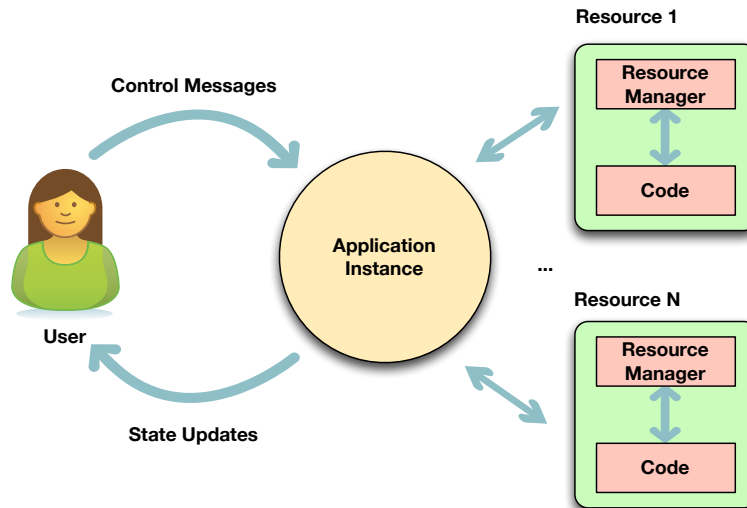


Figure 3.1: The Application Interaction Model underpinning AHE. The Application Instance is the central entity representing each instance of an application that a user launches. All user interaction is mediated via the Application Instance, which supports operations to launch, monitor and terminate the application, and to manage data sharing.

virtualization and community application sharing. Application virtualization allows developers to code against a single virtualized interface instead of the specific underlying software or hardware system, and it also allows multiple virtualized instances to be run side by side on a single physical system. The concept of an application in AHE does not necessarily imply a single computational code executed on a single HPC resource. A virtualized application in AHE can be comprised of more complex workflows, such as coupled simulations where multiple applications are required to pass data to each other, for example coupled quantum and molecular level simulations, made up of separate codes that exchange data via files [4].

AHE is a layer of middleware on top of existing grid technologies such as Globus [20] and Unicore [21], abstracting the details of the particular underlying grid middleware in use. AHE provides simple services that allow the scientist to easily manage simulations. Services are used to launch applications on grid resources, manage the location of input and output files associated with a simulation, stage files needed to run a simulation to their required destination and retrieve simulation output to locations easily accessible by the scientist.

AHE allows for scientific applications to be easily exposed as Web services, and run on a variety of different resources, from high performance grid machines belonging

to disparate members of a virtual organization and transcending security boundaries, to local clusters and even desktop workstations hosted within a single administrative domain. AHE does not mandate any required features for the applications it hosts, meaning that legacy community applications, often with many years of development investment, can be easily deployed on a grid without modification. It does this seamlessly, using secured, mutually authenticated transport protocols [124], and proxy credentials [125] where necessary. Using a single uniform interface, which can be accessed by a multitude of client tools including command line and graphical clients, the scientist can access a wide variety of different resources.

AHE is built around the idea of a community model. In this paradigm, an expert user is required to setup and configure AHE with details of a scientific application, the distributed (grid) infrastructure it is deployed on and then uses AHE's RESTful interface to share this scientific application transparently with a group of end users. The resources that a particular application is deployed onto are chosen based on the characteristics and requirements of the application. An end user can then launch and monitor applications through the AHE desktop GUI client, web client or command line client (described in §3.14), and any combination of these clients can be used simultaneously.

AHE manages the simulations started by the scientist, taking care of the many data files that need to be moved around the grid. By removing the need for the user to remember the many different incantations required to access numerous different high performance computers and manually managing simulations, AHE allows the scientist to concentrate on doing actual science. AHE uses very simple, lightweight, interoperable clients, meaning that the scientist can run their simulations in a number of different ways: for example the scientist can choose to launch simulations from a graphical client on their desktop machine, or create more complex scientific 'workflows' by scripting calls to command line clients. A mobile client has recently been released [126] that will run on a PDA or mobile phone, so that scientists can continue to do research while on the move.

3.9 Modelling Applications as Services

The requirements outlined in Section 3.4 and the design constraints discussed in Section 3.6.1 have led to the design of a software implementation of the model, the Application

Hosting Environment (AHE), the fundamental contribution of the first half of this thesis. The requirements and design constraints present the need for the following basic services:

- An application instance, a service that represents each individual application execution
- An application factory, which can be used to generate new application instances.
- An application instance registry, which records each instance of an application launched, so that it can be found again.
- An application registry, which allows users to discover each application hosted by AHE.
- A file staging service, to manage the transfer of files between resources.

AHE has evolved over several years of development, which has allowed us to evaluate different service development paradigms as a means to implement the tool.

Our initial AHE implementation of AHE used Web Services Resource Framework (WSRF) to model the above services. In Coveney *et al.* [12] we reported our initial implementation of this tool (AHE 1.0). Version 1 of AHE provides a first step towards the hosting of arbitrary grid applications; we built upon this success [42, 23, 127, 128, 19] to add new features and extend the grid application virtualization concept to more complex scenarios in AHE 2.0. The design of these two WSRF based implementations is described in Section 3.10. AHE 3.0 [1], described in Section 3.12, is a reimplementa-tion of the AHE concept, adding many significant new features to those found in AHE 2.0, with an unremitting focus on usability and reliability. We present a discussion of these two approach in Section 3.13.

3.10 Implementing the Application Interaction Model as WSRF Services

AHE 1.0 and 2.0 use the Web Services Resource Framework [53] to represent legacy scientific applications as stateful web services. A single WS-Resource represents each

instance of an application launched, and is used to maintain all of the properties associated with the application. The interface presented by the WS-Resource allows the user to control and monitor the application. The AHE 2.0 system consists of the following components:

3.10.0.1 App WS-Resource

The *App WS-Resource* represents an instance of the running application. It is a WSRF WS-Resource with the following ResourceProperties:

- The time the WS-Resource was created.
- The grid resource the application is/was run on.
- The back end middleware used to launch the application.
- The arguments to the application that were specified by the user.
- The location of input files.
- The location of out files.
- The status of the application (whether it is running, finished, failed etc.).
- The user that created the App WS-Resource.
- Details of the user's proxy credentials stored on a MyProxy server.

The App WS-Resource supports the WS-ResourceProperties operations specified in the WSRF specification, including *Create*, *Destroy* and *GetPropertiesDocument*. It also supports a range of operations to allow the user to control the application. The *Start* operation launches the application on the grid resource by submitting an appropriate job description to the resource manager running on the grid resource. Job descriptions can be submitted in either the JSDL [58] or JDD [129] and are created by applying an XSLT transform [130] to the WS-Resource's Properties document.

3.10.0.2 App Server Factory

The *App Server Factory* is a "factory" according to the Factory pattern [131]. It produces a new WS-Resource that acts as a representation of the instance of the executing application. The App Server Factory is itself a WSRF WS-Resource, and supports the

standard WS-ResourceProperties operations but not the WS-ResourceLifetime operations as it is a persistent service. Clients are not allowed to modify properties of the "App Server Factory", they are only allowed to retrieve them. The ResourceProperties for the App Server Factory are:

- The name of the each application the server is configured with.
- The set of grid resources the application is available on.
- The location of the application's job description template files.

The App Server Factory supports an extra option called *Prepare* which creates a new WS-Resource to represent an instance of the Application. The Prepare operation returns the end point reference (EPR) for the WS-Resource. Once a WS-Resource has been created, all further interaction between the user and the application is conducted via its operations.

3.10.0.3 App Server Registry

Each WS-Resource created by the App Server Factory is registered with the *App Server Registry*, which acts as a registry of all application instances launched by AHE 2.0 users. The service supports the standard WS-ResourceProperties operations, but the data returned by the GetResourceProperties operation is user specific, that is, the user is only returned the list of application instances that they himself have launched. This service allows the user to launch an application then close down their client then relocate their applications WS-Resource at a later date, meaning that no state information needs to be maintained on the client machine.

3.10.0.4 File Staging Service

In order to move data between the user's local machine and the grid resource they want to use, an intermediate file staging service is used. This means that a) the client does not need to know the specific details of how to transfer data to each target grid resource, and b) that the user's machine does not need to allow inbound network connections to allow files to be staged in. The file staging service can either use the WebDAV or GridFTP protocol to stage files. All input and output data associated with an application instance is maintained in a unique directory on the file staging service, and can be considered a part of the application's state.

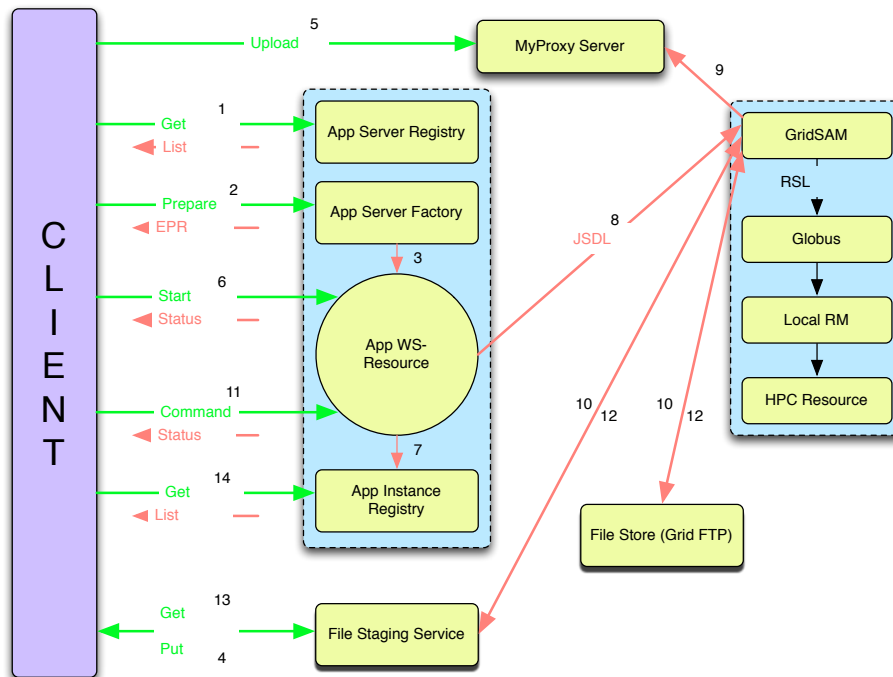


Figure 3.2: The architecture and message flow of the Application Hosting Environment. AHE 2.0 server builds a layer of middleware between the user and the grid to hide much of the complexity of launching grid applications.

3.10.1 The Workflow of Launching an Application

All user interaction is via a client that communicates with the AHE using SOAP messages. The workflow of launching an application on a grid resource running the Globus middleware, shown in figure 3.2, is as follows: the user retrieves a list of App Server Factory URIs from AHE 2.0 (1). There is an application server for each application configured in AHE 2.0. This step is optional as the user may have already cached the URI of the App Server Factories he wants to use. The user issues a *Prepare* message (2); this causes an App WS-Resource to be created (3) which represents this instance of the application’s execution. AHE 2.0 returns a list of possible resources on with the application can be run to the client. To start an application instance the user goes through the sequence: Prepare → Upload Input Files → Start, where Start actually causes the application to start executing. Next the user uploads the input files to the intermediate file staging service using the WebDAV protocol (4).

The user generates and uploads a proxy credential to the MyProxy server (5). The proxy credential is generated from the user’s X.509 certificate issued by their grid certificate authority. This step is optional, as the user may have previously uploaded

a credential that is still valid. Once the user has uploaded all of the input files he sends the *Start* message to the App WS-Resource to start the application running (6). The Start message contains the locations of the files to be staged in to and out from the target grid resource, along with details of the user's proxy credential and any arguments that the user wishes to pass to the application. The App WS-Resource maintains a registry of instantiated applications. Issuing a Prepare message causes a new entry to be added to the registry (7). A *Destroy* command sent to the App WS-Resource causes the corresponding entry to be removed from the registry.

The App WS-Resource creates a Job Submission Description Language (JSDL) [58] document for a specific application instance, using its configuration file to determine where the application is located on the resource and listing the application's input and output files, as well as any parameters that need to be supplied to the application. The JSDL is sent to the GridSAM [132] instance acting as interface to the grid resource (8), and GridSAM handles authentication using the user's proxy certificate. GridSAM retrieves the user's proxy credential from the MyProxy server (9) which it uses to transfer any input files required to run the application from the intermediate File Staging Service to the grid resource (10), and to actually submit the job to a Globus back-end.

The user can send command messages to the App WS-Resource to monitor the application instance's progress (11); for example the user can send a *Monitor* message to check on the application's status. The App WS-Resource queries the GridSAM instance on behalf of the user to update state information. The user can also send *Terminate* and *Destroy* messages to halt the application's execution and destroy the App WS-Resource respectively. GridSAM submits the job to the target grid resource and the job completes. GridSAM then moves the output files back to the file staging locations that were specified in the JSDL document (12). Once the job is complete the user can retrieve any output files from the application from the File Staging Service to their local machine. The user can also query the Application Registry to find the end point references of jobs that have been previously prepared (14).

3.10.2 Implementation

AHE 2.0 is based on a number of pre-existing grid technologies, principally GridSAM and WSRF::Lite [78]. WSRF::Lite is a Perl implementation of the OASIS Web Ser-

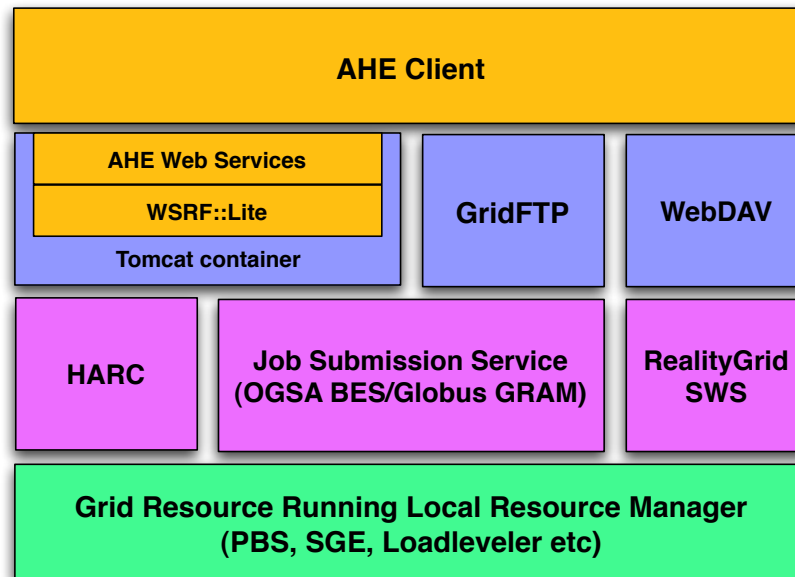


Figure 3.3: The layered architecture of AHE 2.0. The AHE 2.0 client, running on a user's desktop machine, interacts with the AHE 2.0 services running in a Tomcat Web services container, and also stages data via GridFTP or WebDAV. The AHE 2.0 services interact with job submission components such as OGSA-BES or Globus GRAM in order to submit applications to distributed grid resources. AHE 2.0 also has the ability to interact with HARC and RealityGrid steering services.

vices Resource Framework specification. It is built using the Perl SOAP::Lite [79] web services toolkit, from which it derives its name. WSRF::Lite provides support for WS-Addressing [133], WS-ResourceProperties [134], WS-ResourceLifetime [135], WS-ServiceGroup [136] and WS-BaseFaults [137]. It also provides support for digitally signing SOAP [46] messages using X.509 digital certificates in accordance with the OASIS WS-Security [138] standard as described in [80]. Reflecting the flexible philosophy and nature of Perl, WSRF::Lite allows the developer to host WS-Resources in a variety of ways, for instance using the Apache web server or using a standalone WSRF::Lite Container. AHE 2.0 has been designed to run in the Apache [139] container, and has also been successfully deployed in a modified Tomcat [140] container. Figure 3.3 shows the reuse of components in the AHE 2.0 stack, with the light coloured components being implemented as part of the AHE 2.0 development, and dark coloured components being reused from other projects. Not shown are the PostgreSQL database and the MyProxy server.

Due to the reliance on WSRF::Lite, AHE 2.0 server is developed in Perl, and is

hosted in a container such as Apache or Tomcat. The actual AHE 2.0 services are an ensemble of Perl scripts that are deployed as CGI scripts in the hosting container. This can either be a standard Apache web server, or a Tomcat container with a modified version of the standard Tomcat CGIServlet extended to provide a richer set of environment variables to the AHE 2.0 scripts. For performance reasons the AHE 2.0 server uses a PostgreSQL [141] database to store the state information of the App WS-Resources.

In the default implementation, GridSAM provides a web services interface for submitting and monitoring computational jobs managed by a variety of Distributed Resource Managers (DRMs), including Globus [100], Condor [67] and Sun Grid Engine [34], and runs in an OMII [142] web services container. Jobs submitted to GridSAM are described using Job Submission Description Language (JSDL) [58]. GridSAM uses this description to submit a job to a local resource, and has a plug-in architecture that allows adapters to be written for different types of resource manager.

3.11 AHE 2.0 Deployment

As described above, AHE 2.0 is implemented as a client/server model. The client is designed to be easily deployed by an end user, without having to install any supporting software. The server is designed to be deployed and configured by an expert user, who installs and configures applications on behalf of other users.

The initial release of AHE (version 1.0) required that the user install several additional components on their system, including a Postgres database, Apache web server and Perl interpreter. This proved to be too complicated for many of our target user communities, and was quickly remedied by the inclusion of AHE 1.1 in the OMII middleware toolkit [142]. The OMII installer takes care of deploying AHE 1.1 services, Postgres database management system, WSRF::Lite [78] plus supporting Perl modules in to a Tomcat container, making installation relatively easy. However, a number of users reported problems with deploying AHE 1.1 using the OMII installer, due to the relatively limited number of Linux distributions it supports.

Since the first AHE release, Virtual Box [143] became a popular open source system for running virtual machines (VMs) on a variety of operating systems. Virtual Box features a very simple mechanism for deploying new virtual machines. Therefore, in addition to the installation mechanisms we have supported previously, we made AHE

2.0 server available as a Virtual Box VM image. Based on the long term support version of the CentOS Linux distribution [144], AHE 2.0 VM includes all of the server side components required to run AHE 2.0 preinstalled and configured to run a sample application, with extensive documentation and examples available to allow users to customize AHE 2.0 server for their needs.

The AHE 2.0 client package comes with all of the Jar files required to run the client. To install the AHE 2.0 clients all an end user need do is download and extract the client, load her X.509 certificate into a Java keystore using a provided script and set an environment variable to point to the location of the clients. The user also has to configure their client with the AHE 2.0 server endpoints supplied by their AHE 2.0 server administrator. The end user can then launch and monitor applications with both the GUI and command line clients.

3.12 Implementing the Application Interaction Model as RESTful Service

AHE 3.0 is a complete re-implementation of AHE 2.0 in Java (AHE 2.0 and earlier versions were implemented in Perl). AHE 3.0 introduces a new workflow engine based on JBPM [145] allowing complex workflows to be created and integrated into AHE 3.0, and accessed by users as single applications. AHE 3.0 also incorporates an object relational mapping framework using Hibernate [146], which simplifies installation and development of AHE 3.0 by decoupling AHE 3.0 from the database used to maintain state. A RESTful web service interface based on the Restlet [147] library furnishes a simple and concise HTTP based interface for clients to access AHE 3.0 services, compared to the WSRF [53] based services used in AHE 2.0.

AHE 3.0 is a departure from AHE 2.0 and earlier releases, having undergone a complete redesign. AHE 3.0 comprises a number of modules which implement the core functionality. *AHE Runtime* controls the start-up and shut-down of the AHE 3.0 application life cycle; *AHE Engine* implements the core functionalities including the workflow engine as well as facilitating interactions between the different components; *AHE Connector* module implements the functionality required to connect to different types of middleware; *AHE Security* module handles the security component as well as

user management of AHE 3.0 application and grid middleware; *AHE Interface* module provides a RESTful Web service interface as well as command line access to AHE 3.0; the file module provides mechanisms to transfer files between different storage resources using GridFTP. A schematic showing the interaction of these different components is presented in figure 3.4.

RESTful web service provides a simple abstraction of AHE 3.0's functionalities to the user by exposing AHE 3.0 components as resources, each of which is identified by a global identifier (URI). This provides a clean and simple mechanism for end users to access AHE 3.0, making client tooling less complicated; it also means that AHE 3.0 can either be deployed via a servlet container such as Tomcat, or as a standalone server. A detailed discussion of the AHE 3.0 server components is presented below.

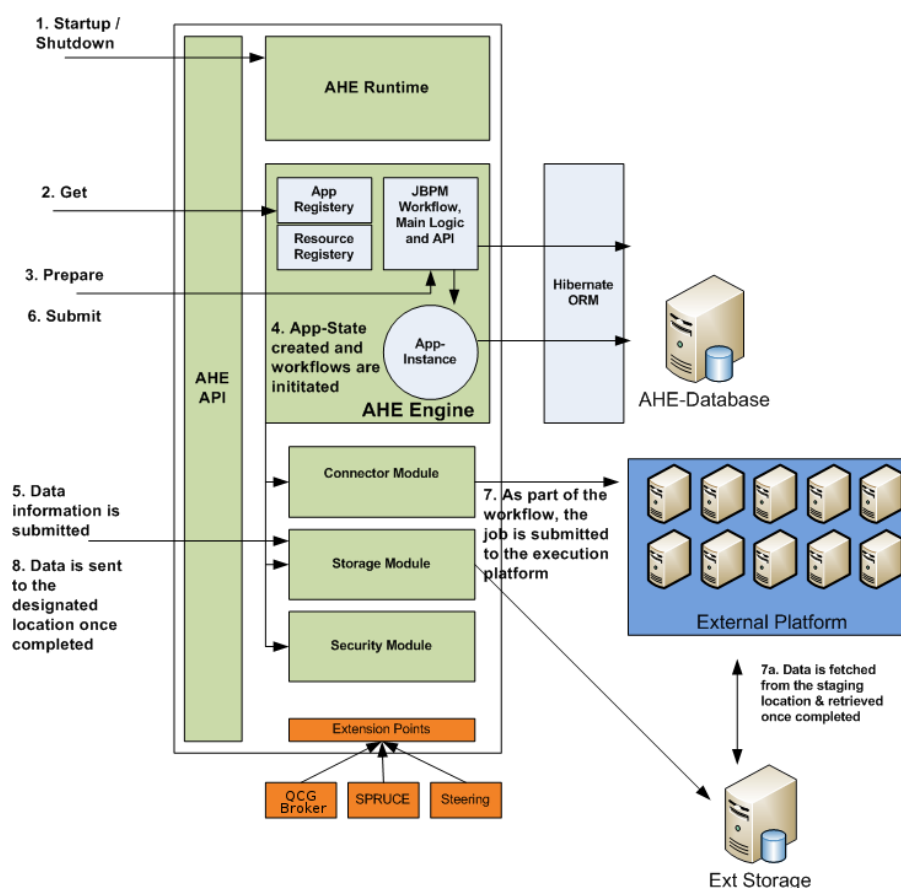


Figure 3.4: The architecture of the AHE 3.0 server, showing the relationship between the different software modules.

3.12.0.1 AHE Runtime Module

The AHE Runtime module is responsible for starting up and shutting down the server in the standalone mode. It also initializes all the components and user configurations as well as the basic registry information relating to users. In standalone mode, AHE 3.0 uses an embedded Jetty Server to provide web server functionality, including HTTPS with mutual user certificate authentication. In this mode, AHE 3.0 can be started from an executable file. AHE 3.0 can also be deployed as a Java servlet into a servlet compliant server such as Apache Tomcat [140]. AHE 3.0 can be deployed with an embedded H2 database [146] or use an external database through the Hibernate framework. More details about the deployment are discussed in Section 3.12.2.

3.12.0.2 AHE Engine Module

The AHE Engine module encapsulates the functionality through which AHE 3.0 virtualizes access to scientific applications. It provides methods to create an Application Instance object, used to represent an instance of a virtualized application. In addition, methods are provided to run and maintain the execution workflow for each virtualized application instance. The workflow describes how the data and computational code(s) associated with this application instance are processed, including details such as the back-end connector (*cf.* §3.12.0.3) and security mechanism to use.

The AHE Engine module also allows higher level workflows to be implemented. These workflows can control multiple application instances to create parameter sweep applications or complex chained application scenarios, in which data created by an application is used as the input for a second one and so on.

3.12.0.3 AHE Connector Module

The connector modules provide a set of classes that invoke external Java libraries which allow AHE 3.0 to act as a client to distributed resource managers (DRMs) such as Globus GRAM. The connector module provides a generic Java interface (using the adapter pattern) which adapters for different DRMs have to implement. This Java interface is used by AHE 3.0 to access external computational resources, providing a loosely-coupled relationship between AHE 3.0 and external client libraries. Connectors currently exist to allow AHE 3.0 to run applications via Globus 5.0[100], Unicore 6 [21] and QCG OGSA-BES [64]. Each connector implementation translates the AHE

3.0's internal application state model into specific directives to the relevant DRM, such as the number of cores to use. The extensible interface framework means that interfaces to other DRM systems can easily be added as necessary. Each connector module is responsible for trapping errors from the underlying DRM and mapping it to an AHE 3.0 error state, which is presented to the user.

3.12.0.4 AHE Interface Module

This module contains library code used to interface with AHE 3.0. This includes a bridge between the RESTful Web service interface (*cf.* §3.12.0.7) and the AHE Runtime Module (*cf.* §3.12.0.1). The AHE 3.0 REST Web services exposes the main AHE 3.0 functionalities and components as resources which can be controlled by performing operations on those resources.

3.12.0.5 AHE Security Module

The AHE Security Module provides a number of important functions, including user management, authorization and authentication control, platform credential management and integration with Audited Credential Delegation [7] (see Section 3.15.1.1). AHE 3.0 provides a mechanism to delegate security control to ACD; these security functions include user authentication and management as well as virtual organization support and proxy generation for any specified virtual organization. In ACD mode, AHE 3.0 contacts ACD using RESTful web service calls to authenticate users as well as to request the generation of proxy credentials on a per user basis. AHE 3.0 itself is also able to authenticate users via SSL certificates or the more standard username/password credential combination. The security module is able to perform command level authorization, as well as platform credential management such as maintaining private key and certificate information for a user which may be required for him/her to be granted access to particular computational resources and data. Additionally, the security module is able to request updated proxies from a MyProxy server when a certificate is about to expire.

3.12.0.6 AHE Transfer Module

In AHE 3.0, input data is transferred directly from a location specified by the user to the computational resources where an application is to be run. Once a job is completed, the AHE 3.0 server takes care of staging any output data back to the user specified location.

The AHE Transfer module provides a mechanism to set up the security credential used to authenticate transfers and then initiate transfers between two different storage components through the AHE Java file transfer interface. Different transfer mechanisms have been implemented using this Java interface. Currently, file transfer functionalities make use of the jGlobus and UCC libraries to stage data using the GridFTP [69] and Unicore transfer protocols. The Java interface makes it easy to add new transfer protocols in future should they become necessary.

3.12.0.7 RESTful Web service

One of the main features of AHE 3.0 is the implementation of the RESTful web services interface, which replaces the WSRF interface used in AHE 2.0 and earlier versions. RESTful web services expose resources addressable via HTTP and operated on using HTTP operations such as POST and GET. This provides a secure and straightforward universal end point for AHE 3.0 to provide services to users. AHE 3.0 uses the Java Restlet library for its RESTful implementation [147]. The Restlet library was chosen to underpin the AHE 3.0 server due to the many features it provides, including the ability to develop services that run as standalone services or which can be deployed in a servlet container such as Tomcat (using either the J2SE or the J2EE version of the library), multiple native data representation formats such as XML and JSON, and scalability as well as security support.

The AHE 3.0 REST command structure is grouped into a number of resources including: User, AppInstance, AppReg (application registry), Resource, PlatformCredential and Cmd (general commands). Each of these resources can be viewed or modified using the GET, POST or DELETE HTTP operations when applied to a suitable AHE 3.0 resource URI. A typical AHE 3.0 URI consists of several components; the domain URI followed by the user identifier and the AHE 3.0 resource that will be operated on. The URI is followed by the command and argument if required.

3.12.0.8 AHE Workflow Engine

A key component of AHE 3.0 is the workflow management system built on the JBPM framework. JBPM is a lightweight Java workflow engine, with workflows described using the Business Process Modelling Notation (BPMN) 2.0 specification which calls specific Java classes, scripts or Drool rules to perform arbitrary functions. This allows

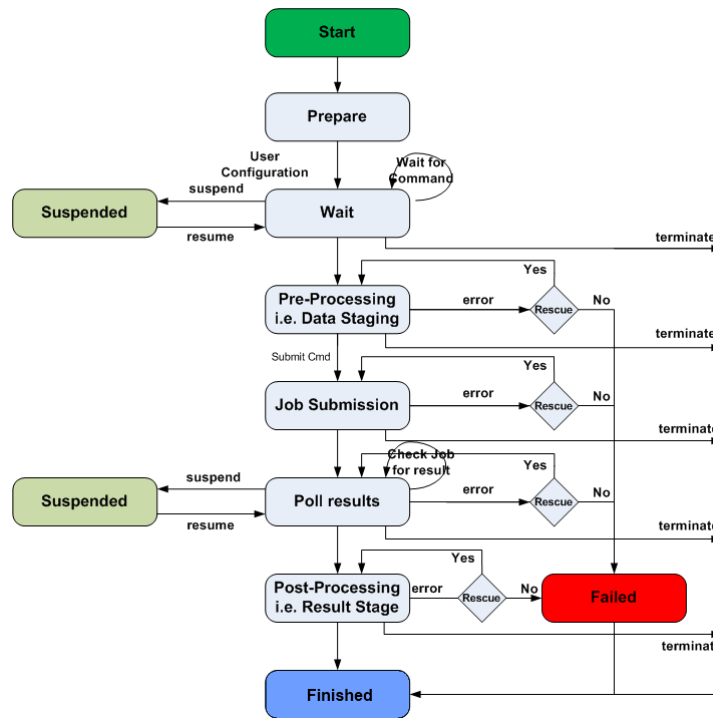


Figure 3.5: The application life cycle. AHE 3.0 server manages the transition of an application instance through a number of states, in order to stage data, execute an application, and handle failures.

new complex workflows and scripting functionalities to be introduced quickly to extend AHE 3.0. JBPM supports workflow persistence using the Hibernate framework meaning that, in the event of a server crash, the workflow can be recovered quickly and seamlessly. There is also a wide range of tools available to plug in to the JBPM framework, including workflow editors, which eases the integration of JBPM with AHE 3.0.

By using a workflow engine, further functionalities and workflows can be introduced into AHE 3.0 applications. This allows the expert user to tailor customized workflows to complex tasks, such as coupled model applications. It also allows additional features or functionalities to be added, such as fault tolerance, and to integrate AHE 3.0 with external services such as SPRUCE [107] in order to submit urgent computing jobs, and RealityGrid Steering [148] which allows scientist to interact with running applications.

3.12.1 AHE 3.0 Application Life Cycle

AHE 3.0 manages the whole life cycle of an application when invoking AHE 3.0, from input data staging, through job execution, to output data staging; during this process

the application transitions through a number of different states. This life cycle is shown in figure 3.5.

The process starts when a prepare command is received by AHE 3.0. This puts AHE 3.0 in a waiting mode, allowing the user to set up additional configuration details required for the application or workflow submission. Once a start command has been submitted, AHE 3.0 server proceeds to first stage any input data that the user has attached to the application instance; once that is completed, it is then submitted to the execution platform. Once the application has been submitted to an external execution platform, AHE 3.0 goes into a polling state, checking regularly for the completion of the application. When the job has completed, any output data is staged to the location specified by the user and the job submission process comes to an end.

If errors occur during certain stages of the AHE 3.0 workflow process, AHE 3.0 captures the error and allows the user to fix this error and attempt to execute the same step again. This workflow is modelled and executed using the JBoss jBPM workflow library and additional components can be added to the workflow if necessary.

In practice, a user has to go through the following steps in order to run an application:

1. AHE Runtime initializes all components, populates the internal data structures and ensures that the data held in the state database is synchronized with the AHE 3.0 data structures.
2. The user queries the application registry to see what applications are available.
3. The Prepare command is issued which tells AHE Engine to create a persistent App-Instance Object that keep tracks of the status and state of an executing application, which in turn initiates the AHE 3.0 workflow process. An App-Instance object is a representation of a virtualized application initiated by the user. This allows AHE 3.0 to keep track of the state and progress of the virtualized application.
4. This App-Instance object is persistent and stored in a local database using the Hibernate Framework, which allows AHE 3.0 server to be database agnostic. In particular:

- (a) the App-Instance object is associated with a user/group and has a unique identifier;
 - (b) active App-Instance data/objects are held in a registry and checked by AHE Engine to see which processes can be operated on each App-Instance, such as when and how they can be run, when data can be checked or retrieved, and so on.
5. Input data files required by the application are staged to the target resource. AHE Server records the location and transfer protocol specified for each individual data file and passes that information to the relevant connector module so the job manager knows how to stage the data and retrieve the results if necessary.
 6. The user next issues the submit command.
 7. AHE Workflow module then schedules the execution of the application using JBPM and quartz scheduler [149]. This allows complex workflows to include asynchronous tasks, as well as multi-thread/concurrency support.
 8. AHE Engine deals with the security interface requirements and submits tasks to external execution platforms. AHE 3.0 polls the external execution platform (if it is configured to do so) and retrieves any output data once the application is completed. JBPM allows additional features to be added in order to create more complex workflows incorporating AHE 3.0 plug-in components. JBPM is persistent so that all events are logged. If the server crashes, the workflow state stored in a database can be retrieved and reinitialized.
 9. Once the application has completed, the data is retrieved and sent to the scratch disk (temporary file storage) or copied to an external storage resource specified by the user, allowing him/her to access it.

3.12.2 Deployment of AHE 3.0

AHE 3.0 can be deployed as a standalone application via the Jetty Server using an embedded database or, in a more complex environment, AHE 3.0 can be deployed as a Servlet hosted within a Servlet compliant server such as Apache Tomcat and configured to use databases supported by the Hibernate framework.

In the simplest configuration, the standalone mode allows AHE 3.0 to be executed as an application which launches the Jetty Server with the option of invoking an embedded database or any external database supported by the Hibernate Framework. In this configuration, the user simply downloads the AHE 3.0 executable, configures the Hibernate configuration file to set up the database connectivity and runs the program.

With server or network constraints, AHE 3.0 can be hosted inside a Servlet compliant server such as Apache Tomcat and be configured to use any databases supported by the Hibernate framework. A user should then download the AHE 3.0 servlet version, deploy it on the Servlet server and configure the database configuration file to ensure AHE 3.0 runs correctly. Once AHE 3.0 is running, the system administrator configures user management, hosted applications as well as resources and credentials.

Whichever way the server is deployed, end users can access it either using a web browser, via the web client interface, or using the GUI or command line client tools. The client tools simply require Java to be available on the client machine; after setting an environment variable and running a configuration script these can be easily run.

3.13 AHE 3.0: Comparison with AHE 2.0

Our efforts to refactor AHE 3 to expose a RESTful interface, as well as redesign the AHE server in version 3.0 have not only been done to enhance user experience, but also to improve performance. In order to evaluate the benefits of this work, we ran performance tests comparing the performance of AHE 2.0 against AHE 3.0.

Our experimental set up consisted of a server running both AHE 2.0 and AHE 3.0, with both systems configured to launch applications via the QCG-Computing middleware onto a 96 node cluster within the Centre for Computational Science at UCL. The tests we performed used a work station to submit batches of applications to AHE 2.0 and AHE 3.0 in turn, measuring the time taken to submit these batches. The application launched was a simple code designed to sort a list of words into alphabetical order, but since we are only interested in the time performance of the AHE server itself, we only measured time taken to submit the application rather than measuring the time the application takes to execute (which would be affected by the cluster load), and the cluster was dedicate to the experiment while the tests were performed. The The tests themselves were implemented as JUnit tests calling the AHE client API, while JUnit,

Number of Jobs	Average Time to Submission (Sec)	
	AHE 2.0	AHE 3.0
10	150.658	11.267
20	297.299	20.293
30	445.128	26.055
40	599.798	32.326
50	778.178	39.888
60	890.562	42.999
70	1044.253	48.899
80	1189.409	62.225
90	1491.252	65.443
100	1640.279	69.903
110	1989.628	80.953
120	2139.565	118.927
130	2313.931	125.504
140	2491.821	141.491
150	2661.946	169.865
160	2933.978	212.076
170	3016.636	217.476
180	3190.618	267.429
190	3397.940	287.478
200	3532.591	295.899

Table 3.1: Average time taken to submit jobs using AHE 2.0 and AHE 3.0

executed via the Eclipse development platform, was used to measure the time taken to perform the tests. Each test was repeated three times, and the mean time taken for each test calculated. The results are plotted in figure 3.6.

As the results presented in table 3.1 and figure 3.6 show, AHE 3.0 performs far faster than AHE 2.0, and the time taken to submit jobs using AHE 2.0 is much more variable, . This is due to the fact that AHE 3.0 is developed in Java whereas AHE 2.0 was developed with Perl/WSRF::Lite; AHE 3.0 exploits a simple RESTful interface, whereas AHE 2.0 uses the far more complicated WSRF extension to Web services, which increases the complexity of both client and server. Application submission is also faster in AHE 3.0 because the system implements a buffered queuing system between the AHE server and the connector modules, which has the effect of allowing the submission interface to process more simultaneous requests, compared to AHE 2.0.

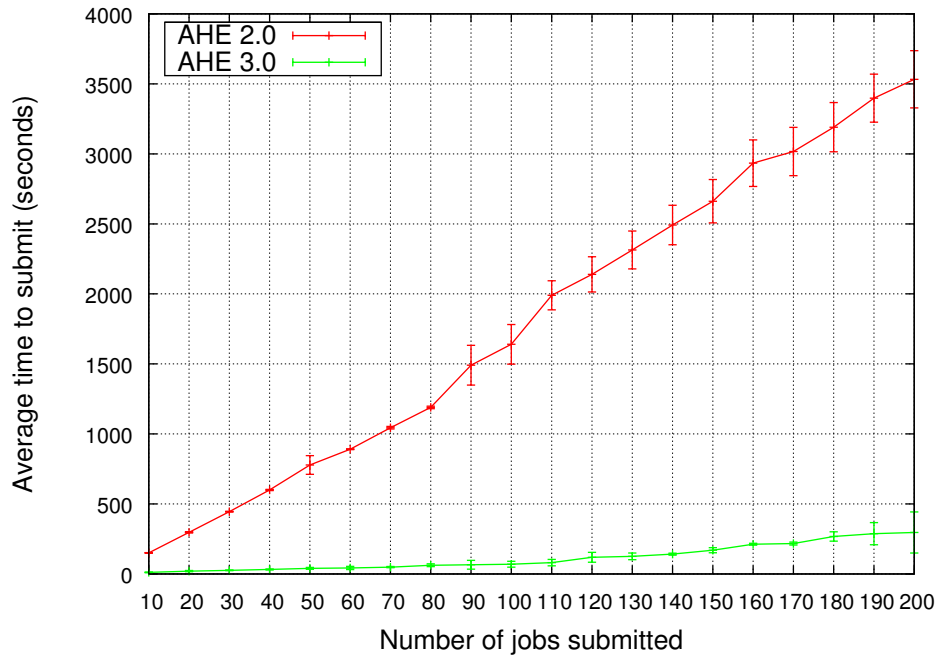


Figure 3.6: Comparison of the mean time required to submit using AHE 2.0 and AHE 3.0 for batches of 10 to 200 jobs. Bars show the standard deviation of the result mean.

3.14 AHE Client Tools

The AHE server maintains all state information about a particular application instance. This means that client tools, which work with all versions of AHE, need to store no information about individual application runs, and consequently very simple clients containing little configuration data can be created. It also means that clients can inter-operate, with one client used to launch an application and another used to monitor for example.

The GUI client uses a wizard to guide a user through the steps of starting their application instance. The wizard allows users to specify constraints for the application, such as the number of processors to use, choose a target grid resource to run their application on, stage all required input files to the grid resource, specify any extra arguments for the simulation, and set it running. The same functionality is provided by a set of command line clients, which have the added benefit that they can be called from a script to create complex, multi-application workflows.

Once an application instance has been prepared and submitted, the AHE GUI client allows the user to monitor the state of the application by polling its associated App WS-Resource. After the application has finished, the user can stage the applica-

tion's output files back to their local machine using the GUI client. The client also gives the user the ability to terminate an application while it is running on a grid resource, and destroy an application instance, removing it from the AHE's application registry. The AHE GUI client can be seen in figures 3.7 and 3.8.

The AHE client attempts to discover which files need to be staged to and from the resource by parsing the application's configuration file. It features a plug-in architecture which allows new configuration file parsers to be developed for any application that is to be hosted in AHE. The parser will also rewrite the user's application configuration file, removing any relative paths, so that the application can be run on the target grid resource. If no plug-in is available for a given application, then the user can specify input and output files manually.

The simple REST endpoints exposed by AHE 3.0 server mean that in practice any tool which can perform HTTP POST and GET operations (such as the UNIX `curl` command) can be used as clients. However, a Java client API has been developed which not only provides methods to call AHE 3.0 server commands, but also provides auxiliary functions such as data staging. This API has been used to produce both graphical and command line clients. It also allows applications hosted in AHE to be accessed from high level tools, and integrated with workflow engines such as GridSpace and Taverna (see Section 4.8).

In addition, an AHE web client has been developed for AHE 3.0 to provide a simple interface for the end user when interacting with the AHE server via a web browser. The web client interface has been developed using the Google Web Toolkit (GWT) and communicates with the AHE 3.0 server through its RESTful interface. The web client can be deployed on Java servlet compliant servers such as Tomcat or JBoss AS. The web client also allows the user to administer and configure an AHE 3.0 server, providing capabilities to manage users, certificates, applications, target computational resources and the server itself. The web client also allows end users to transfer files, launch and monitor AHE jobs all through a web browser.

3.15 Common Core Foundations

Versions 1.0, 2.0 and 3.0 are all intended to provide community specific portals to a distributed e-infrastructure, and can be run without the intervention of resource ad-

ministrators. As such, AHE builds on a number of commonly deployed distributed e-infrastructure technologies, described in this section.

3.15.1 Security

As discussed previously, AHE seeks to build on top of existing grid middleware infrastructures, and hence implements a security infrastructure designed to interoperate with the security infrastructures used by these grids. In the case of all of the grids that we interact with, these security mechanisms are based on X.509 certificates [61] and proxy credentials [125] derived from them. Proxy credentials allow services such as AHE to perform actions on behalf of a user without having access to a full certificate. In common with many other Web services based applications, all communication between AHE client and server is secured with Transport Layer Security (TLS) [124] by means of the user's X.509 grid certificate.

Briefly, the security mechanism works as follows: AHE server is configured to use a server certificate issued by a public certification authority that all clients trust. Each user obtains a user certificate (usually from the same authority) and pre-loads it in to his AHE client; the user certificate should also be trusted by all of the grid resources that the user accesses, where it is used to identify the user. When starting the AHE client, the user is required to enter a password to unlock his certificate before the client can be used.

Prior to launching the application, the AHE client allows the user to generate a short lived proxy certificate from his grid certificate and upload it to a MyProxy server. The user then queries the AHE server and launches his application over mutually authenticated HTTPS links. The AHE server associates all applications launched by a single user with the distinguished name (DN) of their certificate. When the user launches an application, a Start message is sent from the client to the server containing details of the application run and also the details of the user's proxy certificate. The AHE server uses this proxy certificate information in one of two ways. In the case of submission to GridSAM, the AHE server will include the proxy details in the JSDL document that it generates, contained in extensions to the JSDL standard specific to GridSAM. In the case of direct Unicore BES or Globus GRAM 4 submission, the AHE server will use the proxy details to retrieve the user's proxy certificate from the MyProxy server and

use it to submit the application on the users behalf.

By default, AHE will allow querying and application launching by anyone with a certificate generated by a trusted certificate authority. When a user launches an application, AHE server will attempt to submit it to the target resource on their behalf. The question of user authorization is left to the particular resources; if a user is not authorized to access a particular grid resource then the application will fail. AHE services delegate responsibility for establishing HTTPS connections to the Tomcat container it is running in. It is therefore possible to limit the set of users authorized to use AHE services by configuring a user access list in Tomcat.

3.15.1.1 Usable Security in AHE 2.0/3.0

Efforts to address the usability of e-infrastructures are hampered by existing security mechanisms imposed on users. Typically, these require a user to obtain one or more digital certificates from a certificate authority, as well as to maintain and renew these certificates as necessary. The difficulty in doing this leads to widespread certificate sharing and misuse and a substantial reduction in the number of potential users [76]. In order to remove this barrier, we have coupled AHE 3.0 to Audited Credential Delegation (ACD) [7]. ACD is a usable security system that accommodates the security requirements of both end-users and resource providers, offering facilities to authenticate, authorize and audit all transactions.

The main advantage of ACD is that it entirely removes the use of digital certificates from end-users' experience, minimising the usability problems caused by such credentials while addressing resource providers' concerns regarding securing access to their shared resources, tracing the users responsible for performing specific tasks on their resources. ACD enables users to invoke security credentials they are familiar with such as their institutional username/password combination (using Shibboleth [150] for example); assuming that they are authenticated it issues a digital certificate to them when necessary in the background.

When run without ACD, the AHE security model requires each individual user to have a digital certificate, which carries with it the need to go through a lengthy credential acquisition process. To remove the need for such a certificate, we have integrated ACD with AHE. The first step of the integration requires understanding the interac-

tion between AHE and ACD, in other words, the functional and administrative tasks that can be performed within the integrated system. The administrative tasks offered by ACD include VO creation, certificate assignment, adding users, resetting user passwords, creating user roles, assigning tasks to roles, and assigning users to roles. The functional tasks offered by AHE include: Prepare Job, Submit Job, Monitor Job, download and Terminate Job. AHE's functional tasks are the same as the tasks permitted for any authorized user on a computational resource that uses the Globus or UNICORE middleware, for example. Therefore, the permissions assignment to the VO is done by the resource owner first, then the VO administrator re-assigns these permissions to the roles in the VO according to the VO authorization requirements.

In the integrated ACD+AHE environment, the authorization requirements determined by the VO administrator are expressed through the introduction of two roles: VO Administrator and Scientist. The former is permitted to perform all the administrative operations above in addition to terminate, monitor and download any job submitted to a resource. The latter is permitted to perform all AHE operations in such a way that a person who submitted a specified job can only perform AHE functional operations on this application. As a result, two users running applications invoking different data will not be able to view the results of each other's activities.

3.16 Building on Other Middleware Tools

Since the release of AHE 1.0, we have continued to enhance the features and usability of the tool, and worked with several projects seeking to exploit large scale grid computing to deploy AHE as a key piece of infrastructure. These include the EU funded ImmunoGrid project [19], where AHE has been used as a backend engine of a web portal to allow researchers to run simulations on federated, international resources, and the ViroLab [151] project, where a grid-based workflow built on top of AHE automates the whole process of building, running and analysing ensembles of molecular dynamics simulations to calculate the binding affinities of drugs to HIV target enzymes [111].

We have integrated AHE with a number of other middleware tools and services that have matured since the original release to provide a rich problem solving environment. Features include the ability to co-reserve time on resources in advance, launch cross-site applications, and launch steered applications. Figure 3.3 shows the layered

architecture of AHE and how it interfaces with these services. The services are described in more detail later in this section.

3.16.1 Computational Steering

The RealityGrid steering system [148] is a WSRF-compliant middleware tool and associated libraries that allow parameters in a simulation code to be marked as steerable, which means that they can be monitored and modified in real time as an application is running, and hence change the course of the simulation. This is useful for a number of reasons: a scientist can monitor the progress of a simulation as it is running, and either stop it or alter the simulation parameters if a problem occurs, preventing valuable compute time from being wasted [152]. Steering also allows the scientist to checkpoint a simulation as it is running should an interesting state occur, and then use this checkpoint to spawn new jobs and investigate the parameter space further from the saved starting point. In order to do this effectively, the user needs to be able to schedule access to a machine at a time convenient to herself (textitcf. §3.16.3), rather than waiting for the application to progress through the machine's queuing system.

The RealityGrid system provides Web services that establish and mediate communication between users and their simulation codes. When a simulation code begins to run on an HPC resource, it publishes its steerable parameters to these services. The user periodically queries the steering services using lightweight client tools to check if the application has started to run, and to monitor and modify the parameters when it has. If parameters are modified by the user, the changes are passed back to the simulation code by the steering service. Since the RealityGrid Steering services provide a consistent interface for the exchange of simulation parameters, the RealityGrid toolkit can also be used to couple models of different scale, using diverse simulation codes, potentially running on various computational resources [152].

In normal usage, a user wishing to launch a steered application will complete the following procedure: firstly, they will launch a setup command on their local machine which will initialize a Steering Web Service (SWS) on a steering server, and register it with a steering registry service. Each instance of an application corresponds to one SWS. Next, the user submits the application in the preferred way (e.g. by submitting a PBS script to the local queuing system on their target grid resource). The

application's environment must be set up with an environment variable containing the Uniform Resource Identifier (URI) of the SWS. Once the steered code starts to run it uses the contents of this environment variable to establish contact with the SWS. After submitting the application, the user launches the RealityGrid steering client on a local machine which connects to the SWS to monitor and steer the simulation's parameters.

In order to simplify these procedures, we have automated the manual process described above within AHE 2.0. An extension to the application description file allows an application to be marked as steerable; when a user launches such an application, AHE starts up the appropriate SWS service, sets the environment variable and launches the application. The AHE client enables a steering button on the application monitoring panel which allows the user to launch the RealityGrid steering client and connect it to the appropriate SWS.

3.16.2 Geographically Distributed Parallel Computing

While the MPI specification primarily concerns itself with intra-machine process communication, the prevalence of grid middleware tools such as Globus on HPC resources has facilitated the development of inter-machine versions of MPI which use grid middleware to create secure communication channels between resources that are part of the grid [153]. One of the most widely used implementations of MPI over grid middleware is MPIg [101], formerly called MPICH-G2, developed at Argonne National Laboratory.

The benefits of running simulations across multiple machines on a grid are twofold. Firstly, it allows problems to be solved that cannot be practically addressed by a single HPC resource (for example, because it requires more memory or CPU power than one resource is able to provide) to be run on a 'meta-computer' made up of multiple individual compute resources. Secondly, it permits one to tackle large problems which will potentially take a long time to run on a single machine (due to the load on the machine, meaning that the job will have to wait in the queue for a long time) by harvesting smaller numbers of processor cores from multiple machines, meaning that the application could be turned around more quickly, by avoiding long queue wait times.

Since they run concurrently at a number of sites, MPIg/MPICH-G2 applications [101] require guaranteed access to multiple resources at a given time. One way to

establish this is through the implementation of co-reservation systems (discussed more fully in the next section). Examples of MPIg/MPICH-G2 applications include Nektar [24], a fluid dynamics application to simulate human arterial blood flow, and Vortonics [24], an application used to investigate vortex knot dynamics in fluid turbulence.

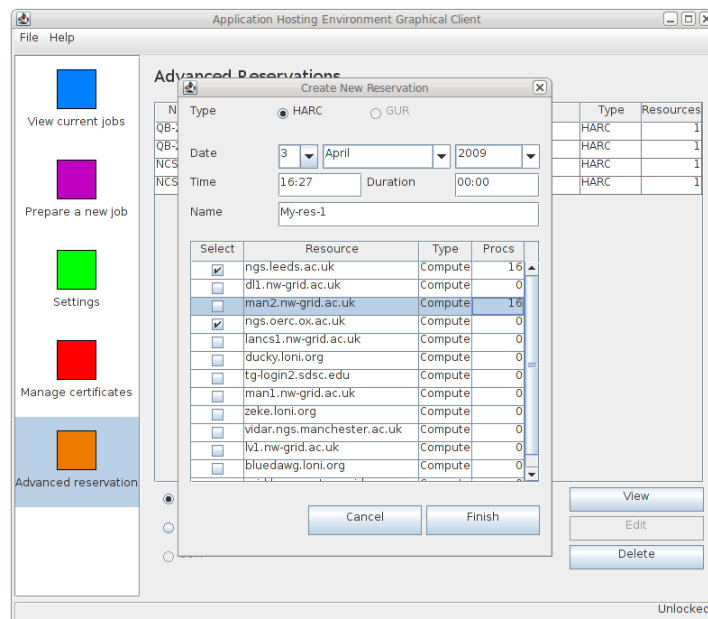
In order to launch a cross-site MPIg application, a user has to first compile a code against the MPIg libraries on each HPC resource being used. They then have to stage their input data to the resources that they want to use, having first established with the machine operators that the resources will all be available at the same time. Since MPIg is heavily dependent on Globus, making use of Globus managed communication channels between machines amongst other things, applications must be described by Globus JDD or RSL documents, and submitted using Globus client tools to the Globus MultiJob service. This service takes care of launching separate components at each target site, and setting up the communication between them.

To simplify this process, we have extended AHE's internal state model to allow it to manage multi-site jobs. An XSLT transform is performed to convert this internal representation into a JDD document which is submitted directly to a Globus MultiJob service, something that was not possible with AHE 1.0 (which used the GridSAM job submission tool). Used in combination with the HARC software described in the next section, cross-site MPIg jobs can be launched into pre-reserved time slots across multiple machines.

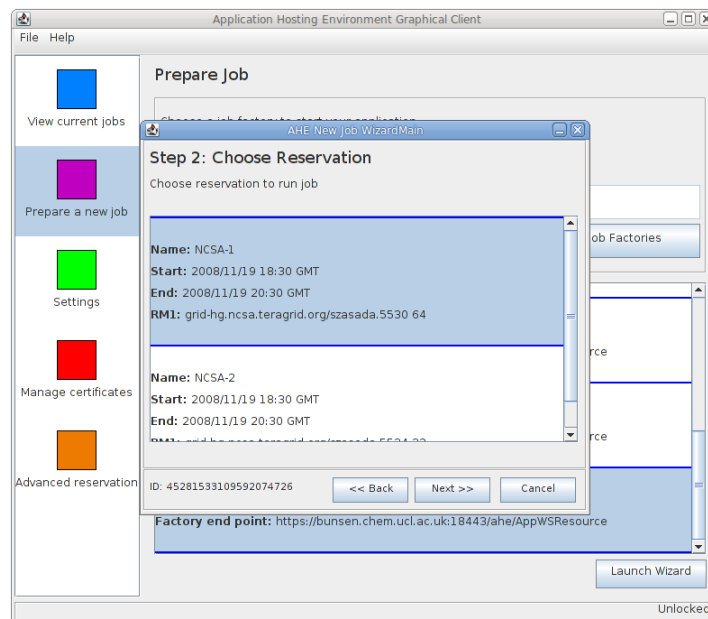
3.16.3 Advanced Co-reservation of Resources

The on demand access mechanisms described in Section 2.8 provides ways for a distributed e-infrastructure user to access resources at a time that suits them. The Highly Available Robust Co-scheduler (HARC) [104] is one system designed not only to allow advanced reservations of time to be made on a single machine, but to enable cross-site reservations of time to be made in a consistent manner on a number of compute resources, together with the switched light path networks that connect them. Similarly, QCG-Computing allows for advanced reservations to be made on distributed e-infrastructures.

Our work to integrate advanced reservation via HARC and QCG-Computing in to AHE has been partly motivated by the need to integrate the tools described in Sections

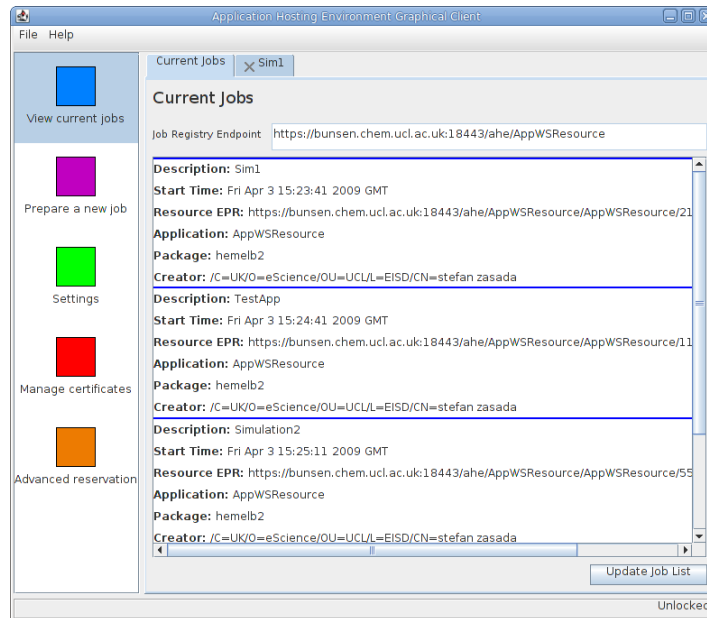


(a)

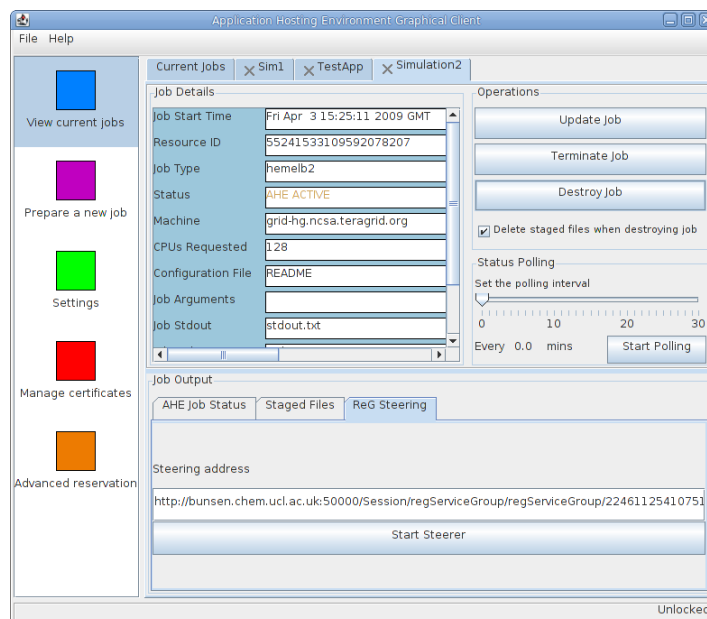


(b)

Figure 3.7: The AHE graphical client has been extended to support advanced reservation, for example allowing users to (a) create new reservations, (b) launch an MPI application into an existing reservation.



(a)



(b)

Figure 3.8: The AHE 2.0 graphical client gives the user the ability to (a) query the registry of launched applications and (b) monitor and steer a running application.

3.16.1 and 3.16.2, both of which require interactive access to resources, but is also necessary for several of the biomedical projects which use AHE, including the case studies discussed in Section 3.1. Many of these use cases also require access to high QoS, high bandwidth, low latency network connections [42]. Since HARC can also co-reserve network lightpaths along with compute time, the network between resources is elevated to a schedulable, ‘first class’ resource [42].

We have modified the AHE graphical client to include a new panel which allows the user to create and manage co-reservations. A modified job submission wizard allows the user to either select an advanced reservation that have been previously made, or specify the resource parameters in the same way as AHE 1.0. In the former case, the AHE client passes the reservation ID for each reserved resource through to the AHE server, which includes them in the Globus JDD file that it submits, leading in turn to the job being run in the reserved time slot(s). Figures 3.7 and 3.8 show several components of AHE being used to submit cross-site and steerable jobs that use reservations.

3.16.4 Standards Compliant Submission

As noted in Section 2.3.1, the effort to build distributed e-infrastructures using a service oriented paradigm has led the Open Grid Forum [154] (formerly the Global Grid Forum) to propose the Open Grid Services Architecture (OGSA) [56]. OGSA aims to address some of the deficiencies perceived in earlier grid middleware by building upon the practical experience gained from using those systems. OGSA has been put forward as an architecture for producing interoperable grid middleware using open standards and industry standard technologies such as Web services. A key component of the OGSA architecture defined by the OGF is the Basic Execution Service (BES) [57].

This Basic Execution Service specification defines Web services interfaces for creating, monitoring, and controlling computational jobs. Clients submit jobs to the BES described in the Job Submission Description Language (JSDL) [58]. BES provides a standard job submission interface to grid resources, and is being implemented in several common middleware stacks, including version 6 of the Unicore [21] grid middleware.

As mentioned previously, AHE 1.0 launches applications via the GridSAM middleware. The original GridSAM [132] interface was a forerunner of BES; current versions of GridSAM now support the BES standard as well as their legacy submission

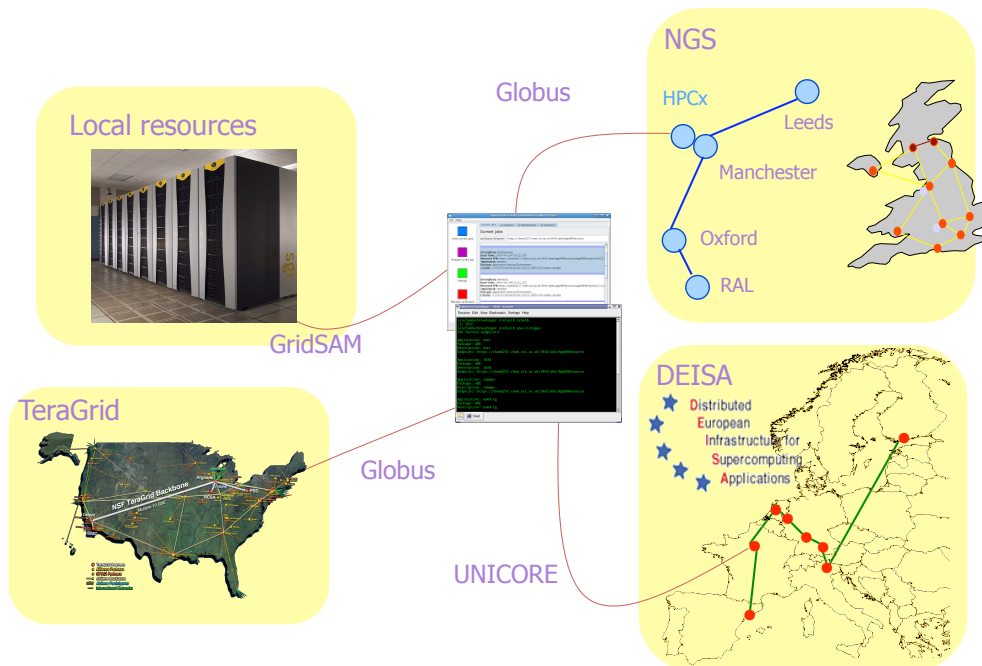


Figure 3.9: The use of standards compliant job submission protocols means AHE can act as a client to many different e-infrastructures and local computational resources, giving the user a single, consistent interface to a wide range of different resources.

interface. Our modifications to both versions 2.0 and 3.0 of the AHE server now mean that it can act as a generic client to services that implement the BES interface, and has been tested with the latest GridSAM release, as well as QCG-Computing and Unicore version 6.

Although Globus 4 GRAM does not implement the BES standard, we have also modified AHE so that it can act as a client directly to GRAM 4 (previously AHE used GridSAM as an intermediate interface to submit to Globus 2 GRAM [20]). Both of these developments mean that AHE can launch application on a much wider range of grids than was previously possible. We have previously described how AHE facilitates user level federation of resources from multiple grids [155]. With these further extensions, AHE can now be used as a single interface to a vast range of compute grids running different middleware stacks, as well as resources local to the user such as clusters and workstations, completely transparently to the user. Figure 3.9 shows how AHE provides a centre point for user-level federated e-infrastructure access.

3.17 Summary

By narrowing the focus of the AHE middleware to a small set of applications that a group of scientists will typically want to use, the task of launching and managing applications on a grid is greatly simplified. This translates to a smoother end user experience, removing many of the barriers that have previously deterred scientists from getting involved in grid computing. We envisage that AHE instances will be deployed following a community model, where the expert users in a particular community will be responsible for deploying and configuring AHE, and will use it to share their application with a group of end users.

AHE takes a slightly different approach from projects developing web portals to facilitate user access to the grid. AHE provides a greater amount of flexibility than a portal - for example the AHE command line tools can be called from scripts to automate common tasks. A certain class of scientific user have previously avoided using portals precisely because of the constraints they put on the user. However, should a portal project want to make use of AHE, the web service interface provided by the AHE server could easily be wrapped in a portal.

Since AHE uses interfaces such as GridSAM for job submissions, it sits in front of existing grid middlewares such as Globus, and allows a group of users to launch applications on the grid without needing any modification to be made to existing grid infrastructure or policies by the grid administrators.

In a production environment we have found AHE to be a useful way of providing a single interface to disparate grid resources. As well as a number of scientific end users, both inside and outside UCL, using AHE servers that we have deployed to run simulations, a number of groups worldwide are either deploying or evaluating AHE. These groups include both scientific projects looking to deploy one or two applications for use by project members and grid resource providers aiming to host a number of different applications for use by a wide community. By representing the execution of an application as a stateful web service, it is easy to build systems of arbitrary complexity on top of the original AHE services.

Chapter 4

Usability Evaluation of the Application Hosting Environment

In this chapter we present our comparative usability study, comparing the usability of the Application Interaction Model as realized in the Application Hosting Environment, with other widely adopted grid middleware systems. We go on to look at the uptake of AHE since its initial release across a range of different projects.

4.1 Evaluating Usability

One of the key purposes of developing distributed e-infrastructure technologies has been to provide a platform for the facilitation of large-scale scientific projects [72, 44]. Conversely, one of the central problems associated with the uptake of grid technologies by scientists has been the lack of ease of use of existing tool kits [18, 73, 74, 75], because current tool kits are too cumbersome for a user to rapidly build prototype distributed applications. Existing tool kits are often themselves resource intensive and require the user to install additional software packages that are not relevant to the tasks they want to perform, but are nevertheless required for the toolkit to operate.

In addition to this, some toolkits have previously required users to install custom patched libraries for the middleware to work. This leads to system administrators having to maintain multiple copies of libraries on machines using the middleware. These heavyweight middleware requirements present a barrier to use that must be overcome before anyone can be productive in their use of e-infrastructure resources. In many cases, a research group might have one or two users accessing e-infrastructure resources; these users often have to maintain their own client middleware tools, fix

problems with digital certificates and apply firewall policies that allow the grid middleware tools to operate. Such heavyweight requirements lead to users abandoning the grid all together, or at least not taking advantage of the power to be gained by accessing an infrastructure of heterogeneous resources, perhaps resorting to only one or two of the available resources to run their simulations in conventional batch mode. At a stroke, users are prevented from being ambitious and thinking about what could be achieved through coordinated access to a set of powerful resources.

Another criticism levelled at many grid middleware systems is the difficulty in using the associated security infrastructure [76], particularly with reference to user management of digital certificates. Many of the existing computational grid environments use Public Key Infrastructure (PKI) and X.509 digital certificates as the cornerstone for their security architectures to provide secure authentication and authorization. However, it is well documented that security solutions based on PKI lack user friendliness for both administrators and end-users [156], which is essential for the uptake of any grid security solution. The problems stem from the lengthy, multi-step procedure for acquiring X.509 digital certificates and requesting authorization to access remote resources, which involves the creation of proxy certificates as part of the authentication process. For many users, all this rigmarole is simply too much to take. They either give up or engage in practices which weaken the security of the environment, such as the sharing of the private key of a single personal certificate to get on with their tasks.

In order to design usable e-infrastructure solutions, it is fundamental to understand end-users and resource providers' requirements. End-users, such as scientists who are not grid technology experts, are concerned with the results of the analyses they perform on grids rather than how to acquire and use digital certificates and to install, maintain and use middleware tools, in order to launch applications in unconventional ways.

4.2 Usability Study Objectives

Our observations relating to the usability of widely deployed middleware solutions outlined above have led us to ask how the user experience can be improved. To answer these questions, we have developed application hosting environment (AHE), described in the previous chapter, to make managing applications easier and Audited Credential Delegation (ACD, see §3.15.1.1) to make dealing with security more simple for end

users. To evaluate the usability of AHE and ACD, we conducted a comparative study [6], to firstly compare the usability of AHE to two of the most widely deployed grid middlewares, Globus and UNICORE and, secondly, to compare the process of using AHE with a digital certificate to using AHE combined with ACD so as to invoke only local username/password credentials to access the grid.

Specifically, our study set out to validate the following two hypotheses:

- (i) The application-interaction model implemented in AHE is more usable than the resource interaction model implemented by many other grid middleware toolkits.
- (ii) The username/password authentication implemented by ACD is more usable than the certificate-based authentication implemented by other common grid toolkits.

To test these hypotheses we compared five different aspects of usability: (a) whether a user was able to complete a fixed task with each tool; (b) how quickly a task could be carried out; (c) the user's satisfaction with the way that they performed the task that they carried out; (d) the user's perceived difficulty in using the tool; and (e) the user's overall impression of the tool that he/she used. The methodology used to assess these usability characteristics is described in the next section, and the results are presented in Section 4.4.

4.3 Study Methodology

Our usability study comprised two sections. Globus and UNICORE are the *de facto* standard middleware tools used to access contemporary production grids to which we have access. By default, Globus is accessed via command line tools to transfer files and submit and monitor jobs. UNICORE has both command line and graphical clients to launch and monitor applications, as does AHE. The first part of our study compared the usability of the Globus command line clients with the usability of the AHE command line client, and the usability of the UNICORE Grid Programming Environment (GPE) graphical client (which we ourselves found easier to use than the full UNICORE Rich Client) with the usability of the AHE graphical client. The version of Globus used was 4.0.5, submitting to pre-WS GRAM; version 6.3.1 of UNICORE was used, with version 6 of the GPE client. AHE version 2.0 was used for the AHE based tests, with a pre-release version of AHE+ACD used for the security tests. At the time these tests

were performed, AHE 3.0 was not available which is why AHE 2.0 was used. The client tools for both AHE versions are identical, so we do not expect the results to differ if the tests were performed again using AHE 3.0.

The remaining part of our usability study set out to evaluate our second hypothesis. We compared a scenario where a user was given an X.509 certificate and had to configure it for use with AHE to a scenario where a user invoked ACD to authenticate to AHE. Both sections of the study can be considered as representing ‘best case scenarios’. Firstly, all tools were installed and preconfigured for the user. An actual user of XSEDE [97] or PRACE [62] would most likely have to install and configure the tools herself. In the security section of the study, the user was given an X.509 certificate to employ with AHE. In reality, a user would have to go through the process of obtaining a certificate from her local Certificate Authority, a time consuming task that can take between two days and two weeks [157].

In passing we note that while other middleware tools, and other interfaces to Globus and UNICORE, certainly do exist, these interfaces are often community specific and not available to all users. Our tests evaluate the default minimum middleware solutions available to PRACE and XSEDE users.

4.3.1 Participants

Some usability experts (notably Nielsen [158]) maintain that five is a suitable number of subjects with which to conduct a usability study, since this number of people will typically find 80% of the problems in any given interface. However, our study does not seek bugs in a single interface: it asks participants to compare the features of several middleware tools to find which is most usable. To do this we need a sufficient number of participants to be sure that our results are statistically significant. To determine the minimum number of participants required, we conducted a power analysis [159], calculating the probability that a statistical test will reject the null hypothesis or alternatively detect an effect. In order to determine the statistical significance of our results, we used a one-tailed paired *t*-test. For a reasonable statistical power of 0.8 (i.e. the probability that the test will find a statistically significant difference between the tools), we therefore determined we would need a minimum of 27 participants, plus a few more to allow for those who might drop out for various reasons.

We recruited a cohort of 39 participants consisting of UCL undergraduate and postgraduate students, each of whom received a £10 Amazon Voucher for taking part in the study. These participants came from a wide range of backgrounds in the humanities, sciences and engineering, but none had any previous experience in the use of computational grids. This cohort is therefore analogous to a group of new users of computational e-infrastructures (e.g. first year PhD students) in terms of educational background and experience.

4.3.2 Tasks

As discussed, our usability study was split into two sections. In the first section participants were asked to compare Globus, UNICORE and AHE by performing three separate tasks:

(i) *Launch an application* on a grid resource using the middleware tool being tested.

The application in question (pre-installed on the grid resource) sorted a list of words into alphabetical order. The user had to upload the input data from their local machine and then submit the application to the machine.

(ii) *Monitor the application* launched in step 1 until complete.

(iii) *Download the output* of the application back to the local machine once it has completed.

The second section compared the use of X.509 certificates to ACD authentication. In this section, users were asked to perform the following two tasks:

(i) *Configure the AHE client with to use an X.509 certificate*, and then submit a job using the graphical client.

(ii) *Authenticate to AHE using an ACD username and password*, and then submit a job using the graphical client.

In order to avoid the typical queue waiting problem when using HPC resources, all of the tests ran the application on the same server, based locally in the Centre for Computational Science at University College London, which was used solely for the purpose

of running the usability test application. Participants were provided with documentation on the tools they were asked to use, copies of which can be found in Appendix D.

4.3.3 Data Collection

Prior to beginning the tasks outlined above, each participant was asked a number of questions related to their academic background, general IT experience and previous experience of using grid middleware tools. After each task, we asked the participants to rate the difficulty of the task and their satisfaction with their performance of the task, using a Likert scale [160] (i.e. five options from *strongly agree* to *strongly disagree*). In addition, we timed how long it took the user to complete each task. After using each tool, we asked the participant to evaluate it using the System Usability Scale (SUS) [161], via ten questions about his impression of the tool giving a standard measure of usability scored out of 100, which is suitable for making comparisons between tools. After completing the two sections of the study, each participant was able to give freeform comments on impressions of the tools used, if desired. While performing each task, an observer watched each participant and recorded whether or not the task was completed successfully. The specific wording of the tasks the users were asked to perform, and the questions they were asked to respond to, can be found in Appendix C.

4.3.4 Delivery

To ease the process of data collection and tabulation (and the timings of tasks), we developed a simple web platform from which to deliver the usability study. The study was conducted in the Centre for Computational Science at University College London. Each participant in the study was assigned an ID number, which they used to log on to the delivery platform. All of the various usability metrics were then recorded against this ID in a database. Before starting the study, the delivery platform displayed a page explaining to the user the purpose of the study. The observer also explained to the participant that he was not able to provide any assistance or answer questions relating to the tasks being performed.

The delivery platform provided web forms on which participants could record the answers to the questions outlined in the previous section. The delivery platform also described the operations that the user had to carry out. Prior to performing the task, the

	Middleware Tests				Security Tests	
	Globus Toolkit	AHE CLI	UNICORE GUI	AHE GUI	AHE with Cert	AHE with ACD
Percentage of successful users	45.45	75.76	30.30	96.97	66.67	96.97
Percentage of users satisfied with tool	27.27	53.54	47.47	79.80	51.52	87.88
Percentage of users who found tool difficult to use	45.45	25.25	26.26	5.05	27.27	0.00

Table 4.1: Summary of statistics collected during usability trials for each tool under comparison.

user had to click a *Start* button, which set a timer running for the task, and a *Stop* button when it was completed. When performing a task, the user was given a documentation snapshot, taken from the tool’s documentation, that instructed them how to perform the task (included in Appendix D). As noted, all of the tools were preconfigured on the machine used by the participant to perform the tasks. Each of the tasks in the two sections was assigned in a random order, to minimize the risk of bias entering the study.

4.4 Results

Our usability tests show very clear differences between the different tools tested, based on the usability metrics defined in Section 4.2. Table 4.1 presents key measurements from our findings. Due to problems with the delivery platform (such as web browser crashes half way through a set of tests), the results from six participants have been excluded from our results, meaning that the results presented have been gathered from a cohort of 33 participants.

We applied a 1-tailed, paired t -test to our results to determine the statistical significance of any differences between the tools being compared. We compared the Globus command line client with the AHE command line tool, and the UNICORE graphical client with the AHE graphical client. We also compared the AHE using a digital certificate to the AHE using ACD authentication. The P -values of these t -tests are shown in table 4.2, along with mean scores for the five different metrics. A $p < 0.05$ shows that the difference between the tools is statistically significant.

Our first usability metric looked at whether or not a user could successfully com-

plete the set of tasks with a given tool. Table 4.1 summarizes the percentage of participants who were able to complete all tasks for each tool. Although the failure data is measured on an ordinal scale (Success, Failed etc.), we have converted it to numerical data in order to more easily compare results. The mean failure rate is shown in table 4.2, with a lower score meaning there were less failures when using the tool. Also shown in table 4.2 are the P -value scores; the AHE command line was found to be less failure prone than the Globus command line ($t(33) = 1.41, p < 0.05$), the AHE GUI was found to be less failure prone than the UNICORE GUI ($t(33) = 1.07, p < 0.05$) and AHE with ACD was found to be less failure prone than AHE with X.509 certificates ($t(33) = 1.03, p < 0.05$).

Our second usability metric was a measure of how long it took a user to complete an application run. Figure 4.1(a) plots the mean times taken to complete the range of tasks with each tool. Again, the differences are statistically significant as shown in table 4.2, with participants able to use AHE to run their applications faster than via Globus or UNICORE, and AHE with ACD faster than AHE with X.509 certificates.

Our third usability metric measured user satisfaction with the tools used. In table 4.1 we have summarized the percentage of participants who reported being either Satisfied or Very Satisfied with a tool. The Likert scale data is again ordinal, but we have converted it to numerical data in order to compare it, according to commonly practice [162]. The mean satisfaction level is reported in table 4.2, a higher score meaning that a user was more satisfied with the tool. Again, users reported being more satisfied with the AHE than with other tools, and with ACD than with X.509 certificates, as show by the P -value scores table.

Our fourth usability metric looked at how difficult a user perceived a tool to be. Again the percentage of users who found a tool difficult or very difficult is summarized in table 4.1. The mean difficulty scores are shown in table 4.2, with a higher score meaning that the tool was perceived as being more difficult to use. The AHE GUI client was perceived as being less difficult to use than the UNICORE GUI client ($t(33) = 1.73, p < 0.05$), the AHE command line interface was perceived as being less difficult to use than the Globus command line tools ($t(33) = 2.55, p < 0.05$), and AHE with ACD was perceived as being less difficult than AHE with digital certificates ($t(33) = 1.52, p < 0.05$).

	Middleware Tests				Security Tests	
	Globus Toolkit	AHE CLI	UNI-CORE GUI	AHE GUI	AHE with Cert	AHE with ACD
Mean SUS score. 100=most usable	37.50	51.89	52.95	69.47	51.21	72.12
SUS score <i>P</i> -value	6.14821×10^{-4}		2.55007×10^{-4}		1.10235×10^{-7}	
Mean task time (seconds)	667.73	606.18	587.33	388.70	309.00	131.39
Mean task time <i>P</i> -value	3.55845×10^{-2}		3.49719×10^{-4}		8.97553×10^{-5}	
Mean failure rate. 5=most failure prone.	2.11	1.41	2.16	1.07	1.45	1.03
Failure rate <i>P</i> -value	6.23844×10^{-8}		6.60682×10^{-14}		8.18544×10^{-4}	
Mean perceived difficulty. 5=most difficult	3.15	2.55	2.44	1.73	2.67	1.52
Perceived difficulty <i>P</i> -value	7.27699×10^{-7}		3.23194×10^{-7}		2.27949×10^{-5}	
Mean satisfaction. 5=most satisfied	2.68	3.29	3.34	4.05	3.12	4.27
Satisfaction <i>P</i> -value	7.27699×10^{-7}		1.28956×10^{-6}		7.50398×10^{-6}	

Table 4.2: The mean scores and *t*-test *P*-values for our five usability metrics, comparing the AHE and Globus command line clients, the AHE and UNICORE graphical clients, and the AHE with and without ACD.

Our final usability metric measured a participant's overall impression of a tool using the SUS usability scale. The mean SUS score is shown in table 4.2, with a higher score meaning the tool is more usable. Again, we found statistical significance, with AHE GUI being rated as more usable than the UNICORE GUI, AHE command line being rated higher than Globus, and AHE with ACD being rated higher than AHE with digital certificates, as summarized in table 4.2.

4.5 Discussion of Results

The results presented in the previous section clearly confirm our hypotheses, that the application interaction model used by the AHE is more usable than the resource interaction model implemented in the UNICORE and Globus toolkits, with AHE found to be more usable for each of our defined usability metrics. We believe the reason for

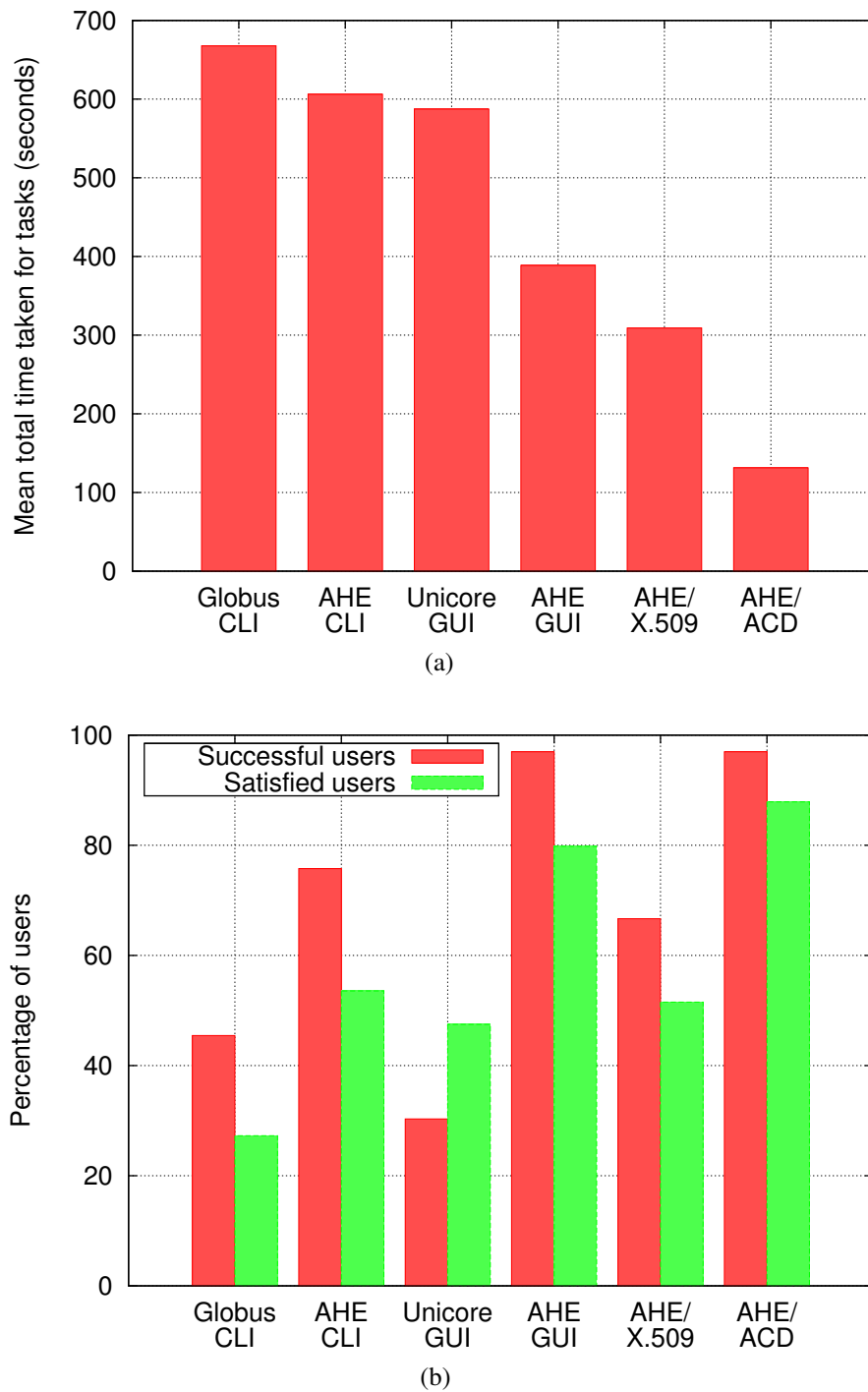


Figure 4.1: (a) Mean time taken to complete a range of tasks with each tool; (b) a comparison of the percentage of users who were satisfied with a tool and the percentage who could successfully use that tool.

this is due to the fact that AHE hides much of the complexity of launching applications from end users, meaning that (a) there are less things that can go wrong (hence the lower failure rate) and (b) there are less things for a user to remember when launching an application (hence the higher satisfaction with and lower perceived difficulty of AHE tools). The fact that the AHE model encapsulates input and output data as part of an application's instance (and stages data back and forth on the user's behalf) means that application launching is faster via AHE.

In the case of ACD security, the familiar username and password were clearly found to be more usable than X.509 certificates, but it should also be stressed that the scenario modelled here represented the 'best case' scenario, where a user was already given an X.509 certificate with which to configure their client. As previously noted, in the real world a user would have to go through the laborious process of obtaining an X.509 certificate from their certificate authority, which renders the ACD solution far more usable still.

The failure rate when using a tool is dependent on all of the subtasks being completed successfully; if one task failed, it meant that the following tasks could not be successfully completed (marked 'Failed due to previous' by the observer). This is, however, analogous to real world scenarios where, for example, a user will not be able to download data from a grid resource if his job is not submitted correctly.

We noted particular problems experienced by participants using the UNICORE middleware, related to staging files and configuring job input files. However, these problems were not noted by the participants themselves, due to the jobs appearing to submit properly. Figure 4.1(b) plots the percentage of users reporting satisfaction with a tool alongside the percentage of users who successfully used that tool. Curiously, more users reported satisfaction with the UNICORE client than were able to use it successfully, suggesting that many participants did not realize their jobs had not completed successfully.

The freeform comments made by users of the individual systems also provide some illuminating insights as to their usability. Regarding the use of ACD security with AHE, one participant reported "To deal with security issues a user is much more at ease with a simple username/password system. The use of certificates just complicates the process unnecessarily". Another participant highlighted the problems involved in

AHE Version	Client Downloads	Server Downloads
1.0.0	22	16
1.0.1	138	91
2.0.0	59	78
3.0.0	N/A	47
<i>Total downloads = 404</i>		

Table 4.3: AHE downloads from RealityGrid.org and Sourceforge.org to January 2014. Note a standalone client release for AHE 3.0 has not been provided.

learning the command line parameters required to use the Globus Toolkit, reporting “there were difficulties in accessing the outside servers i.e. adding gsiftp:// or when to input the whole path into the command line”.

4.6 Community Uptake of the AHE

The AHE has been distributed through a number of different channels, in order to reach as wide an audience as possible. Initially, the AHE client and server were distributed as installation tarballs through the RealityGrid website. This method of deployment required the end user of the AHE server to deploy their own hosting and database infrastructure (e.g. Tomcat and PostgreSQL). To simplify server installation, AHE version 1 was subsequently integrated with OMII Toolkit [142]. This meant that the AHE server components and databases were automatically installed and configured by the OMII stack installer.

The trend towards server virtualization means that many ‘virtual’ server instances can be run on a single physical machine, aiding hardware utilization and facilitating software deployment. The release of AHE 2.0 took advantage of this growing trend, with the AHE server provided as a preconfigured virtual machine image, which could be easily downloaded and deployed by a reasonably experienced user. AHE 3.0 was released as a set of standalone services, through the SourceForge website. In addition, AHE has been made available from other sources, such as the NeSCForge portal, and the VPH ToolKit. Table 4.3 summarizes the client and server downloads made through the RealityGrid portal and SourceForge website.

4.6.1 Hosted Applications

The AHE was designed as a hosting environment for generic legacy scientific codes, and as such it has been used by different user communities to host a wide range of

different application. The application hosting process involves the expert user in a community compiling the application to be hosted on the set of target resources that it is to be used on, generating JSDL template documents describing the application, and configuring the AHE server with details of where the application is installed. Optionally, the expert user can also create a Java parser plugin to process the input file of the application they are hosting, to assist with data transfer.

An overview of some of the most widely used applications run via AHE is listed below:

- **NAMD** - Several members of the Centre for Computational Science (CCS) at UCL are using the AHE to run NAMD simulations, modelling the behaviour of HIV-1 protease, using both the GUI launching client and creating shell scripts that implement application workflows by calling the AHE command line clients. NAMD is also used via AHE in the School of Crystallography at Birkbeck College, to run jobs on UK National Grid Service (NGS) nodes.
- **LAMMPS** - CCS: Within the Centre for Computational Science researchers are using the AHE to launch LAMMPS simulations on the ARCHER, PRACE and XSEDE. Researchers from the Centre for Applied Marine Sciences at the University of Wales, Bangor, have also used the AHE to launch LAMMPS simulations on the NGS.
- **DL_POLY** - Members of the Condensed Matter and Materials Physics group at UCL have used the AHE to launch DL_POLY jobs on the UK NGS.
- **LB3D** - Currently LB3D is being hosted and used on XSEDE and PRACE by members of the CCS at UCL.
- **Gromacs** - The Structural Bioinformatics & Computational Biochemistry group at Oxford use Gromacs in the AHE to run on local resources and previously on the UK NGS.
- **CHARMM** - The Computer Simulations of Biomolecular Systems at the group at the University of Southampton has instrumented the CHARMM code with the RealityGrid steering API, and hosted the steered version of the application in the AHE.

- CHASTE - Researchers in the Department of Computer Science at the University of Oxford use AHE to run CHASTE simulations on ARCHER.

4.7 Case Studies

To illustrate the diverse ways in which AHE has been used, in this section we present several case studies describing the projects that use AHE, and how they use it. Whilst this is not a formal usability study, it captures the subjective experience of many projects that have used the AHE software.

4.7.1 RealityGrid and Materials Science

The RealityGrid [163] was one of the original EPSRC funded e-Science projects, and was further supported under an EPSRC Platform Grant. The project provided an incubator for the AHE, and the needs of RealityGrid scientists drove the initial design and development of the AHE. RealityGrid scientists, based in the Centre for Computational Science at UCL and elsewhere, have integrated the AHE into their daily scientific activities [127, 155]. Researchers who have moved on from the RealityGrid project have taken AHE with them, and have been responsible for introducing it to new user communities.

4.7.1.1 Clay-polymer Nanocomposite Simulations

Clay-polymer nanocomposite materials have recently attracted a great deal of attention as they offer enhanced mechanical and thermal properties compared to conventional materials. Complete understanding of the materials properties of these composites requires accurate knowledge of the elastic properties of their components. RealityGrid researchers have exploited the capability of large scale resources such as the UK's HECToR machine to model, in full atomistic detail, clay systems of up to approximately 10 million atoms whose dimensions approach those of a realistic clay platelet [128]. These exceptionally large scale simulations show emergent behaviour: at length scales greater than 15 nm collective thermal motion of clay sheet atoms produces low amplitude, long wavelength collective undulations of the clay sheets themselves, implicitly inhibited by the small system sizes commonly encountered in atomistic simulation. These "mesoscopic" bending fluctuations allow the calculation of clay materials properties, which are hard to obtain experimentally due to the small size of a clay platelet.

The ten million atom model requires more than 1024 processors on a resource such as HECToR, together with the near linear scaling performance of LAMMPS [164], a high performance molecular dynamics code. Job submission is handled by the Application Hosting Environment to facilitate access to grid-enabled computing resources. AHE is used to manage the full simulation execution chain, dealing with staging input and output data from the machine, and giving the user a remote monitoring tool to assess the progress of their simulations. The high performance computing power available from HECToR, coupled with easy-to-use middleware, has enabled RealityGrid researchers to achieve high levels of scientific productivity, even with such extremely demanding simulations.

4.7.2 The Virtual Physiological Human Initiative

Patient-specific medicine refers to the tailoring of medical treatments based on the characteristics of an individual patient [10]. Decision support systems based on patient-specific simulation hold the potential to revolutionize the way clinicians plan courses of treatment for various conditions, such as viral infections and lung cancer, and the planning of surgical procedures, for example in the treatment of arterial abnormalities. Since patient-specific data can be used as the basis of simulation, treatments can be assessed for their effectiveness with respect to the patient in question before being administered, saving the potential expense of ineffective treatments and reducing, if not eliminating, lengthy lab procedures that typically involve animal testing.

The Virtual Physiological Human (VPH) is a methodological and technological framework that enables the collaborative investigation of the human body as a single complex system [165]. The collective framework makes it possible to share resources and observations formed by institutions and organizations creating disparate but integrated computer models of the mechanical, physical and biochemical functions of a living human body. The VPH Initiative currently comprises over 20 large-scale EU funded research projects working in such diverse areas as heart, musculoskeletal and cancer modelling. In addition the VPH Network of Excellence project manages and coordinates the activities of the various VPH-I endeavours, and itself funds several small scale exemplar projects.

However, such simulations typically require access to supercomputing class re-

sources. If large scale simulation is going to be used in anything approaching a routine way, a single computational resource is insufficient, and indeed all the resources provided by a single computational infrastructure often cannot provide sufficient on-demand power to support clinical use; only by federating the resources of multiple resource providers can such power be achieved. To support VPH, DEISA set up one of the first Virtual Communities, managed by the VPH NoE, and intended to provide VPH researchers with access to large scale computing facilities with which to conduct their research. The VPH Virtual Community was awarded 2 million standard DEISA core hours for 2009, renewed again in 2010 and 2011 to provide access to the HECToR machine (Cray, UK) and the SARA machine (IBM Power 6, Netherlands). This allocation was also used to support the smaller Virolab project, working in the VPH related area of patient specific HIV modelling.

The Application Hosting Environment has been used to support VPH research in a number of different ways. AHE is a key component of the ‘VPH ToolKit’ being developed by the VPH Network of Excellent project [166], meaning that it is being used by a wide community of researchers across Europe and beyond. As well as allowing users to download AHE as part of the ToolKit, a centralized AHE installation managed by the NoE project provides a single interface from which VPH researchers can access the DEISA resources available as part of the VPH Virtual Community.

4.7.2.1 ContraCancrum

The ContraCancrum VPH-I project [167] sought to develop a multilevel platform to simulate malignant tumour development, and the response of tumour and normal tissue to different treatment regimes. The project aimed to aid understanding of the phenomena of cancer, and optimize the treatment of the disease. ContraCancrum deploys two important clinical studies for validating the models, one on lung cancer and one on gliomas. The ContraCancrum ‘virtual laboratory’ consists of a number of different simulation techniques that can be usefully employed by the clinical oncologist. All of these simulation paradigms have one thing in common: they are all driven by the use of clinical data sets supplied by the clinical partners in the project.

One such simulation paradigm aims to create molecular level simulators which have an impact in personalized drug treatment of targeted therapy. The simulators

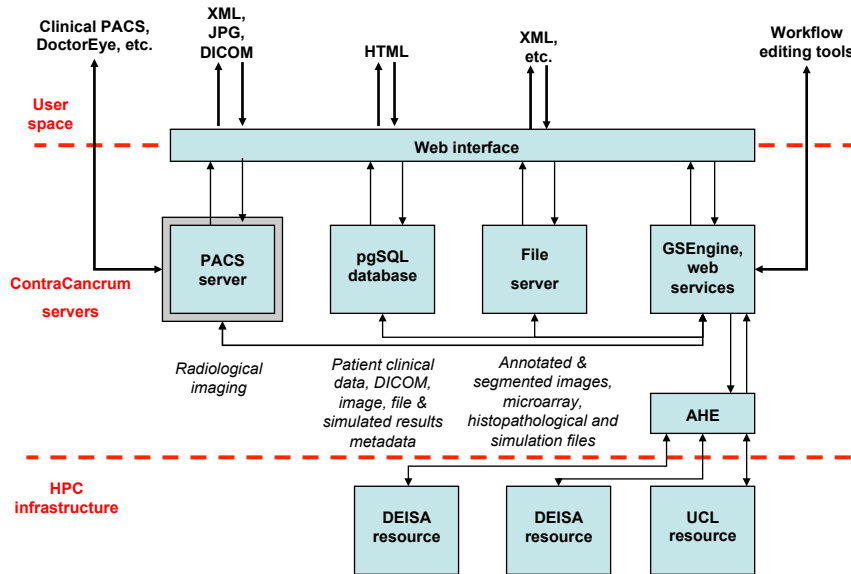


Figure 4.2: The architecture of the Individualized MEDicine Simulation Environment (IMENSE) system. Access to the different components is mediated through a web portal interface, and includes access to high performance computing (HPC) infrastructures such as DEISA to execute simulation codes, mediated by the AHE.

developed in this study have the potential to select targeted drugs in the patient's individualized context by ranking available drugs to patient's specific genotypes, to explain effects of mutations on activation of key proteins and drug resistance. The epidermal growth factor receptor (EGFR) is a major target for drugs in treating lung carcinoma since it promotes cell growth and tumour progression. Molecular dynamics simulations are used to model drug-EGFR binding, in an attempt to rank drug suitability on a patient specific basis.

The multiple simulation paradigms used by the ContraCancrum project all make use of patient specific datasets as the starting point for their simulations. ContraCancrum developers have built a centralized platform to tie together anonymized patient data set storage and viewing with the simulation techniques developed in the project, accessed via a web portal. The Individualized MEDicine Simulation Environment [5] allows ContraCancrum clinicians and researchers to view clinical datasets, and to launch simulation workflows on those datasets. These workflows make use of a range of different resources, from workstations to the largest scale HPC machines available on DEISA in the case of the molecular dynamics simulations. Workflows are orchestrated using the GridSpace workflow engine [168], which in turn launches applications

using the AHE. AHE acts as an interoperability layer meaning that the workflow engine is able to submit application to workstations, departmental clusters and DEISA resources as necessitated by the workflow. The architecture of the IMENSE system is depicted in figure 4.2.

4.7.3 ImmunoGrid

Researchers in the School of Crystallography at Birkbeck College, University of London, have used AHE as the basis of a web portal developed to allow researchers in the EU FP6 funded ImmunoGrid project to access both NGS resources, and local clusters belonging to members of the ImmunoGrid project and running the GridSAM middleware. The purpose of ImmunoGrid [19] is to model the mammalian, and specifically human, immune system using grid technologies. It aims to simulate the immune system at different physiological levels, and as such prefigured the EU FP7 funded Virtual Physiological Human Initiative described in the previous section.

ImmunoGrid integrates processes at molecular, cellular and organ levels, with the web portal allowing project researchers to launch a bespoke project simulation code, called CIMMSIM, on compute clusters provided by partners in ImmunoGrid. The portal acts as a front end to the AHE, meaning the project members need only a web browser installed on their desktop machine to be able to launch simulations on a wide variety of distributed resources made available to project researchers, and use GridSAM to provide an access mechanism to the various resources contributed by project partners. The portal, developed in PHP, wraps around AHE commands, with AHE used to manage simulation execution and data staging.

4.7.4 ViroLab

A major problem in the treatment of AIDS is the development of drug resistance by the human immuno-deficiency virus (HIV). HIV-1 protease is the enzyme which is crucial to the role of the maturation of the virus, and is therefore an attractive target for HIV/AIDS therapy. Although several effective treatment regimes have been devised which involve inhibitors that target several viral proteins [108], the emergence of drug resistant mutations in these proteins is a contributing factor to the eventual failure of treatment. Doctors have limited ways of matching a drug to the unique profile of the HIV virus as it mutates in each patient. A drug treatment regimen is prescribed us-

ing knowledge-based clinical decision support software, which attempts to determine optimal inhibitors using existing clinical records of treatment response to various mutational strains. The patient's immune response is used as a gauge of the drug's effectiveness and is periodically monitored so that ineffective treatment can be minimized through an appropriate change in the regimen.

Computational methods exist for determining biomolecular binding affinities. In a recent study [127], the effectiveness of the drug saquinavir was tested against the wild-type HIV-1 protease, along with three drug-resistant strains using free energy methods in molecular dynamics (MD) simulations. The protocol implemented by the study gave accurate correlations to similar experimentally determined binding affinities. Furthermore, the study made use of a tool, the Binding Affinity Calculator (BAC) [111], for the rapid and automated construction, deployment, implementation and post processing stages of the molecular simulations across multiple supercomputing grid-based resources. BAC automates binding affinity calculations for all nine drugs currently available to inhibit HIV-1 protease and for an arbitrary number of mutations away from a given wildtype sequence.

The main objective of the EU FP6 ViroLab [151] project was to develop a Virtual Laboratory for HIV drug resistance that facilitates medical knowledge discovery and decision support in the prescription of treatments. At the core of the ViroLab Virtual Laboratory is a rule-based ranking system, used to rank the effectiveness of drug combinations using patient derived data. Central to the Virtual Laboratory is a JRuby workflow engine, called GridSpace [168], used to edit workflows developed in the Ruby scripting language. Within the project, developers worked to be able to launch applications on remote grid resources using the AHE from the GridSpace environment, ultimately to implement the BAC workflow. This simulation based approach was then be used to augment the rule based drug ranking system developed by the project, when insufficient data exists to confidently make rule based predictions.

The scope of BAC is enormous as it offers an automated in-silico method for assessing the drug resistance for any given viral strain, and indeed the BAC is also being employed in the ContraCancrum project (*cf.* §4.7.2.1). The AHE has been instrumental in managing the large scale simulation workflow required by BAC. The number of clinically interesting drug resistant mutational patterns is far larger than the avail-

able crystal structures of HIV-1 protease. Mutational protocols convert one protease sequence with available crystal structure into another that diverges by a small number of mutations. It is important that such mutational algorithms are followed by suitable multi-step equilibration protocols, using chained molecular dynamics simulations, to ensure that the desired mutant structure is an accurate representation. Previously these have been difficult to perform on computational grids due to the need to keep track of large numbers of simulations. AHE is used to manage all of the simulations required to construct a chained simulation, with scripting of the AHE command line clients to create complex application workflows, linking the output of one simulation into the input of the next. This approach has greatly simplified the scientist life, reducing the time taken tracking and marshalling input and output files between remote grid resources.

The work of the ViroLab project is being extended through the recently funded VPH-Share project.

4.7.5 MAPPER

Today scientists and engineers are commonly faced with the challenge of modelling, predicting and controlling multi-scale systems which cross scientific disciplines and where several processes acting at different scales coexist and interact [169, 170]. Such multidisciplinary multi-scale models, when simulated in three dimensions, require large scale or even extreme scale computing capabilities. The MAPPER project [171] developed computational strategies, software and services for distributed multi-scale simulations across disciplines, exploiting existing and evolving European e-infrastructure.

To facilitate such an infrastructure, the MAPPER project developed and deployed a multi-tiered software stack [4]. The MAPPER software stack sought to deploy a set of services to facilitate the execution of multi-scale scientific applications, that is single applications composed of multiple different single scale models, with each model usually executed by a different application code. This set of services, building on computational resources from the EGI [172] and PRACE [62] initiatives, as well as testbed resources and services run by the MAPPER project, constituted a European wide infrastructure for multi-scale modelling and science. By necessity, some of these components were run on on target compute resources (such as those operated by PRACE

and EGI) and some components were run at a higher level, on resources operated by the MAPPER project.

MAPPER made a distinction between loosely coupled and tightly coupled application scenarios, the difference been that tightly coupled applications require constant communication between components, whereas loosely coupled applications were executed in a chain of dependant steps. To drive the development of the multi-scale modelling infrastructure, the MAPPER project worked with exemplar applications from five representative scientific domains (fusion, clinical decision making, systems biology, nano science, engineering).

The nano science application domain involved a loosely coupled application consisting of three levels of simulation. The lowest level simulates the electronic degrees of freedom, using the Car-Parrinello Molecular Dynamics (CPMD) code. This code is a parallelized plane wave/pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics. The high level of accuracy of this method provides a mechanism for deriving accurate atomic charges which can be used in classical molecular dynamics, where the electronic degrees of freedom are removed. The atomic charges are passed to the initial models simulated using LAMMPS classical molecular dynamics code. To increase the size and length of simulation we used the classical molecular dynamics simulation to create input model parameters for Coarse-Grained Molecular Dynamics (CGMD) simulations, again using the LAMMPS code. The CGMD simulations have reduced degrees of freedom, by combining atoms into single larger particles. The parameters transferred between these levels are the interparticle positions and interactions, calculated to reduce the structural details of the simulation.

The three components of the loosely coupled application scenario, were deployed on resources appropriate to their processor requirements. The architecture of the system is shown in figure 4.3. First, the CPMD application was executed, with it's output processed into a form that can be used as input to the LAMMPS MD code, and the data transferred to the resource running LAMMPS. Once the first LAMMPS model had completed, its output is similarly processed into a form that can be used by the second LAMMPS model, and the data was transferred to a PRACE HPC resource. The sequential execution of the different scale models was managed by the GridSpace

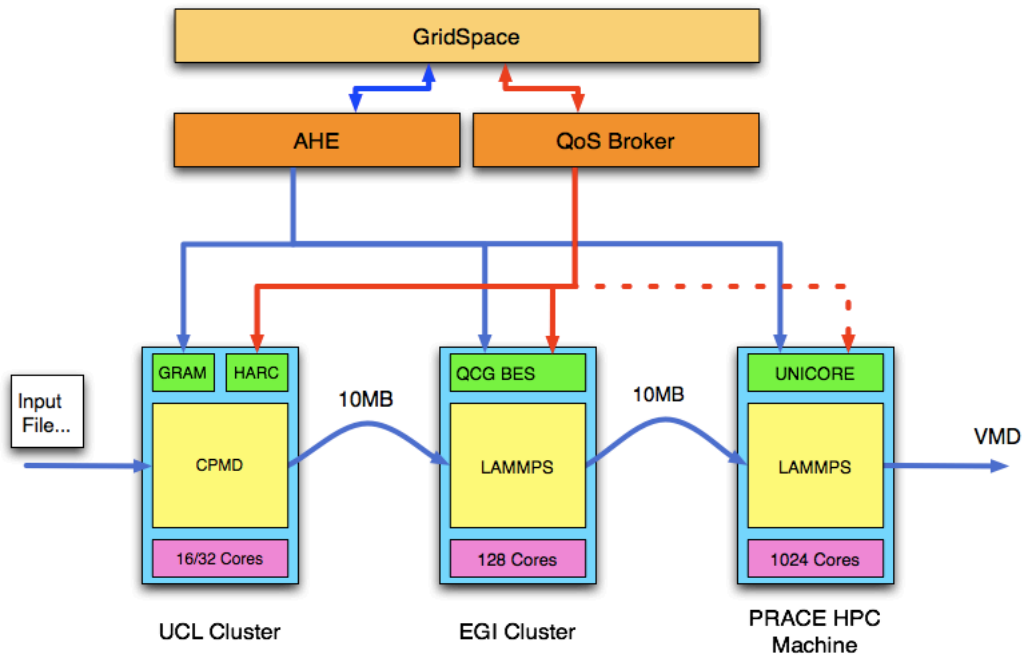


Figure 4.3: The role of AHE in the MAPPER architecture was to act as an interoperability layer, allowing simulation components to be submitted to resources running a range of back end middleware interfaces, in a way that was transparent to the user.

workflow engine, using the Application Hosting Environment as an interoperability layer. The QoS Broker was used to co-reserve resources, to ensure that the different sub-models were able to execute before their subsequent sub-models were executed.

4.8 VPH Share

Like its predecessors, AHE 3.0 is being actively used by several large research projects. AHE provides the principal HPC access tool in the VPH-Share project [173], a currently funded endeavour within the Virtual Physiological Human (VPH) initiative [174], concerned with patient-specific biomedical modelling and simulation [3]. The aim of this project is to develop a set of intelligent services and supporting network infrastructure that will facilitate the exposure and sharing of data and knowledge. In particular, it is developing a multi-scale framework for the composition of new biomedical workflows to promote collaboration within the VPH community.

As part of this infrastructure, VPH-Share is developing a cloud platform that will allow users to easily access computational as well as data resources. AHE and ACD together constitute the HPC gateway service for VPH-Share, allowing simulations that

require more computational power than the VPH-Share cloud infrastructure is able to provide to be seamlessly run on HPC resources. AHE and ACD are deployed based on the Software as a Service (SaaS) model. AHE is responsible for handling the execution life cycle of virtualized applications on computational resources, while ACD bridges the gap between different security infrastructures used by the execution platform and those remote resources. This allows simulation workflows to be deployed which combine resources from a cloud provider such as Amazon in order to execute single core and small scale parallel simulations, but that can switch to high performance computing, accessed via AHE, to run parts of the workflow that require more computational power. The ability of the Taverna workflow system used by VPH-Share to call AHE's RESTful interface allows applications hosted in AHE to be included as components in Taverna workflows.

The system is being used in production runs by VPH-Share scientists to run the Chaste code [175] to model personalized treatments of cardiac arrhythmias in patients. AHE allows the researchers to launch simulations using the Chaste on the ARCHER HPC machines in the UK (part of PRACE), marshal input and output data and manage parameter sweeps. It also allows data to be staged in and out of the EUDAT [70] data storage infrastructure as necessary.

4.9 Conclusions

Our study confirms that AHE is more usable than both UNICORE and Globus, and that familiar username/password authentication provided by ACD is more usable than X.509 certificates. While it may be argued that there are other grid interfaces which can be deployed for users (web portals for example), such interfaces are usually non-standard, and are in practice not available for users on the TeraGrid or DEISA. The UNICORE and Globus tools we tested embody the user-resource interaction model, whereby users have to interact with a machine to stage files and submit jobs. AHE implements the user-application interaction model, whereby users interact with their applications, leaving the AHE to interact with resources on their behalf. AHE can be seen as a hybrid Software as a Service (SaaS)/Platform as a Service (PaaS) architecture, common to many cloud computing systems; we believe that the usability findings reported in this thesis will therefore be useful to both the grid and cloud communities.

By removing the digital certificate from the user's experience, we have shown that users can begin launching applications faster, and with less chance of failure. We have shown that both AHE and ACD are important tools for the computational scientist; while it is of course possible that a junior computational scientist could spend time learning the intricacies of a particular middleware tool, the time they spend doing so is time not engaged in scientific research. The real power of our approach is that computational scientists can use a simple tool to launch applications, without worrying about the particular incantations required by a machine to run a job. The interoperable nature of AHE means that it provides a single interface to a wide range of resources, from HPC grids to departmental clusters and workstations [8]. Indeed, AHE with ACD is currently being used as the basis of a practical introduction to computational chemistry course run for undergraduate students at University College London. The study reported in this thesis focused solely on the 'best case' scenario, where all middleware tools and applications were pre-deployed, and the study participants were used to evaluate and compare the tools' interfaces. In future we plan to extend the study by examining aspects relating to the usability of middleware deployment.

Since its initial release, the Application Hosting Environment has been taken up by a wide and diverse community of users, and employed in a number of different ways. Not only has AHE been used by individual researchers to host and launch their scientific simulations codes, it has also been used as the basis of web portals and as an interoperability layer in several distributed computing systems. AHE's exposure through the Virtual Physiological Human toolkit means that it has been taken up by many researchers across Europe outside of the traditional e-Science community, and by many users whose first exposure to grid computing has been through AHE.

4.10 Summary

In this chapter we have seen how a comparative usability analysis has rated our Application Interaction Model, as realized in the Application Hosting Environment, as being perceived to be more usable than widely deployed similar solutions. We have also seen how this level of usability has meant that AHE has found widespread uptake through a number of large-scale international research projects, as indicated by the number of times the software has been downloaded (shown in table 4.3). The development of new

AHE features has been, and will continue to be, driven by requirements gathered from the AHE community, and as the nature of that community has expanded, so have the features of AHE.

Chapter 5

Agents, Metascheduling and Mechanism Design

In this chapter we look at the design of auction mechanisms as a means of efficiently allocating resources. Many common types of auction mechanism are reviewed, as well as the theoretical underpinnings of mechanism design. We go on to look at the concept of software agents, and the relationships between software agents and mechanism design.

5.1 Resource Allocation

Essential to realising the vision of a computational e-infrastructure as ubiquitous, seamless to use and transparent as the electrical power grid, as proposed by Foster *et al.* [17], is the metascheduler. The metascheduler is a component of the e-infrastructure system responsible for efficiently distributing jobs between e-infrastructure resources, taking into account factors such as machine load and cost models. A metascheduler provides a point of contact between the user and the e-infrastructure, placing jobs submitted by the user onto appropriate computational resources. The metascheduler means that the user does not have to deal directly with each machine on the e-infrastructure, for example logging on to several resources when deciding where to run a job to find the one with the least load. The metascheduler also means that expensive HPC resources are used as efficiently as possible, ensuring that one machine is not idle while another has a large queue of jobs.

Czajkowski *et al.* [176, page 277] give several reasons for the need to use metaschedulers in complex e-infrastructure systems. These are the ability to virtual-

ize resources, thus prevent users from having to learn every detail of the machines they wish to access; Policy enforcement, that is, deciding where about jobs should be run based on a scheduling policy; Protocol conversion, the ability to translate between different back end middlewares, without the end user needing to maintain different sets of client tools.

Much effort has gone in to developing metaschedulers for e-infrastructure systems, for example [177, 178, 179, 180, 181]. As noted in [179] metaschedulers have a number of challenges to overcome: the heterogeneity of systems they are dealing with - systems will have different operating system, access policies, cost models and state reporting mechanisms; the variety of applications that they are used for - for example, single processor task farming jobs or complicated workflows using parallel applications; the interface provided by the broker to the user, be it a web portal or command line, which interacts with the job submission mechanism. Many metaschedulers address this last point by providing a metascheduler and job submission service combined [182].

The job of the metascheduler is to match jobs to available resources. As noted in Section 2.1, the majority of HPC resources on an e-infrastructure will have their own local scheduling mechanism, responsible for allocating jobs submitted to the resource between the available processors. As noted by Thain *et al.* in [67]:

“Grid computing cannot be served by a centralized scheduling algorithm. By definition, a Grid has multiple owners. Two supercomputers purchased by separate organizations with distinct funds will never share a single scheduling algorithm”.

The above quote illustrates the fact that the e-infrastructure wide metascheduler has to work with the different scheduling policies of all of the different resources shared on the e-infrastructure. Metaschedulers need to work with the different local scheduling mechanisms present on the resources in the e-infrastructure, and also take in to account the fact that very often users will be able to submit jobs to the various resources in a number of ways, for example via the metascheduler, or by logging directly in to a submission node on the resource. Because of this, a metascheduler cannot maintain its own local view of resource utilization, based on the jobs that it has been used to schedule, as a basis for making scheduling decisions. The metascheduler needs to

periodically update its view of the utilization of the whole e-infrastructure, for example by querying information services on each resource to discover the state of the queue.

A very naïve metascheduling approach, designed to ensure jobs complete in the fastest possible time, [183], is to start the same jobs on all resources available, then, once the job has completed on one resource, kill it on all of the other resources. Obviously this method is extremely wasteful of resources, as some resources will spend time working on the job before it is completed. Some metascheduler approaches just send jobs between resources in a round robin fashion, with no consideration of the utilization of the resource. Some approaches use Service Level Agreements (SLAs), where a provider agrees to make their resources available to some user or community at a particular time. Again this results in poor utilization of resources, for similar reasons that advanced reservation can lead to under utilization of resources discussed in Section 2.9.

Several projects have examined the ideas of computational economies or markets [184, 185, 186], that is the idea of developing cost models to be associated with e-infrastructure resource usage, and its application in metascheduling [187]. The ability to associate cost with units of processing work essential if the computational grid is to be truly analogous to the electrical power grid, and encourage commercial resource providers to form e-infrastructures which can be used to supply computational power to customers in the way that the electrical power grid supplies electricity. The ability to do this also depends on the provision of robust, distributed accounting services which can be used to generated billing information.

There are many more sophisticated metascheduling approaches that have been applied to computational e-infrastructures. Sections 5.1.1 to 5.1.8 review of some common metascheduling systems employed in various e-infrastructure systems, and the scheduling policies and cost models that they support.

5.1.1 Condor

Condor [67, 188] is a software system for the management of high throughput computational environments, that is computing environments that can deliver large amounts of processing capacity over long periods of time, made up of collections of distributively owned computing resources. It can be used to manage workload on a dedicated

cluster of computers, or to distribute work to pools of idle desktop computers on a cycle scavenging basis. The Condor system is motivated by the question:

“Can we satisfy the needs of users who need extra capacity without lowering the quality of service experienced by the owners of under utilized workstations.” [67]

For Condor systems made up of pools of user desktop machines, the user is given full control over the use of their machine, for example by allowing any Condor run processes to be killed when a user resumes work on the machine. This ensures that Condor’s use of the machine does not interfere with its owner/user’s use of it, and cause them to withdraw if from use in the pool. Originally, Condor pools consisted of computational resources contained within a single administrative domain. Condor-G [188] is an extension of Condor which builds on the Globus middleware to provide secure inter-domain communication and standardized access to remote batch systems. Condor can be used at both the front end and back end of the middleware environment, either as a reliable submission and job management service, or as a fabric management system (that is a grid enabled pool of resources), with the Globus middleware providing services in between.

The Condor system consists of Agents (which are used to manage job execution), Resources (the computational resources on which jobs run), a Matchmaker (to which Agents and Resources advertise themselves, which is responsible for matching job requirements with resources). Once introduced, the Agent must contact the resource to check that the offer to run the job is still valid. To execute a job both sides (Agent and Resource) start new processes: the Agent starts a Shadow process, which is responsible for providing all the information needed to execute a job. On the resource side a Sandbox is started which provides a safe execution environment in which the job can run. In the case of Condor-G the resource can be a HPC cluster running the Globus middleware.

Condor draws a distinction between scheduling and planning, which they define as:

- Planning is the acquisition of resources by users, with users typically being interested in increasing the factors which affect them, such as turn around time and

job throughput.

- Scheduling is the management of a resource by its owner. Typically a resource owner will want to increase utilization of their expensive machine, without losing customers (by, for example, scheduling so many jobs that the customer has to wait an unacceptable amount of time).

In the Condor view, both users and resource providers retain their independence, but there is feedback from one to the other. The fact that resources in a Condor pool are often upgraded in a haphazard way adds to the heterogeneity of the system, and resource owners powering their machines on and off at different times creates a dynamic resource pool. Matchmaking is used by the Condor system to bridge the gap between planning and scheduling. Matchmaking occurs in four steps:

- (i) The Agents and Resources advertise their characteristics and requirements in Classified Adverts (ClassAds).
- (ii) The Matchmaker scans the available ClassAds and creates pairs that satisfy each others constraints and preferences.
- (iii) The Matchmaker next informs both parties of the match.
- (iv) Finally the matched Agent and Resource make contact, possibly carry out further negotiation, and co-operate to execute the job.

ClassAds are sets of uniquely named expressions, with no specific schema, so both users and resources can advertise any attributes they wish. Two special attributes, Rank and Requirement, are used to specify desirability and constraints respectively, with Rank used to choose between a set of compatible matches (the one with the highest rank is chosen). Condor-G can also plan around a schedule, if a remote resource scheduler publishes information about its workload, or can be used to perform more naïve planning, for example by submitting the job to several sites, and then upon completion of the job cancelling it at the remaining sites. Condor also has the ability to schedule within a plan; once an Agent and Resource have negotiated on access to the resource, the Agent considers itself to be the owner of the resource until told otherwise. It can now plan as many jobs as it want to execute on the resource. The flexible design of the

Condor matchmaking system has been used as the basis of the resource broker in the gLite [22] middleware.

5.1.2 Nimrod/G

Nimrod [189] is a tool for the parametrization of serial programs to create and manage embarrassingly parallel, task farming style applications. Nimrod provides a simple declarative parametric modelling language to describe a parametric experiment. The original Nimrod system was designed to manage task farming style workloads within a single administrative domain. The unsuitability of Nimrod for running applications on resources provided by a distributed grid infrastructure, where the resources are spread across institutional domains and each have their own usage policies, cost models and queuing systems, led to the development of Nimrod/G, which makes use of the Globus middleware for job launching and resource discovery. Users interact with Nimrod/G via a client tool, which allows the user to modify parameters related to the time and cost of their experiment, and also allows users to monitor jobs that have been dispatched.

The Nimrod/G system consists of a parametric engine responsible for persistent experiment control management. It parameterizes the simulation (that is, splitting up a workload to be distributed among a task farm), creates and dispatches jobs to the resources on the e-infrastructure. It takes as its input the experiment plan described using the Nimrod parametrization language. It maintains the state of the whole experiment in persistent storage, to provide a level of crash recovery.

The parametric engine interacts with a scheduler, which is responsible for resource discovery, resource allocation and job assignment. The resource discovery algorithm queries e-infrastructure information services (such as Globus MDS), identifies authorized machines and keeps track of resource status information. Scheduling in Nimrod/G follows an economic model, whereby it attempts to execute work on a user's behalf at a given cost and to a given deadline, and can use a variety of parameters in order to arrive at the scheduling policy needed to optimally complete the application execution (for example: free nodes, queue Lent, resource architecture). It implements four scheduling algorithms to allow users to optimize different parameters of the scheduler: *cost optimization*, *cost-time optimization*, *time optimization* and *conservative time optimization*.

5.1.3 Gridbus

Gridbus [179] is a scheduler for distributed data intensive applications on heterogeneous global e-infrastructures. Its architecture is comprised of three layers: the Interface layer, the Core layer and the Execution layer. This three layer architecture is designed to isolate the logic of the broker (the Core) from interactions with specific client tools and e-infrastructure services. The Interface layer communicates with entities external to the broker, such as job submission portals and other client tools. The Interface layer translates the inputs from the external entities into inputs for the Core layer, namely a description of the application requirements, a set of services which can potentially run the application, and the set of credentials required to access the service.

The Core layer contains functional components that can be classified into two categories: entities and workers. Entities exist as information containers, representing the properties, functions and states of elements involved in the execution of jobs. Workers represent the functionality in the broker, implementing the logic of the scheduling framework. A number of workers are utilized to schedule job requests; individual workers are used to monitor the state of services (the availability of resources), match jobs individually to resources (based on minimum requirements specified by the job), and to dispatch jobs to resources (via the Execution layer). In addition to compute cycles, job requirements can also be specified in terms of the data needed to execute the job. The broker makes use of many different services within the e-infrastructure architecture, such as the Grid Index Information Service (GIIS), and file transfer services.

The Execution layer takes care of dispatching jobs to the various different middlewares with which the broker interacts, and monitors the execution of the job on the target resource.

The plug-in architecture of Gridbus means that new schedulers can be plugged in to the system that satisfy user constraints in different ways. The default scheduler has been designed to support the computational economy paradigm, and assigns costs to data and compute services, which is used in combination with the scheduling policy to decide on the optimal scheduling of jobs. Market pricing information is periodically updated by polling a market information service.

5.1.4 EMPEROR

EMPEROR [181] is a metascheduling framework that can be used to implement dynamic job scheduling solutions based on performance criteria. The EMPEROR framework scheduling schemes take in to account dynamic resource predictions, which are used as the basis for scheduling workloads across the e-infrastructure. The resource prediction algorithms used are based on time series analysis to exploit information about past usage. The resource usage measures are based on host load and memory usage. Predictor models are devised based on the statistical characteristics of the resource utilization; Auto Regressive (AR) predictors are used in cases where the load/memory sequences are nearly stationary, while Auto Regressive Integrated Moving Average (ARIMA) models are used where there are rapid fluctuations in usage.

The EMPEROR scheduling process is comprised of the following steps:

- (i) Firstly the user submits their job to the EMPEROR system
- (ii) EMPEROR queries the Globus MDS server to obtain information about available hosts, and the predicted load and memory of each of the underlying resources, and applies the appropriate running time estimation algorithm.
- (iii) The EMPEROR Resource Prediction Systems (RPS) running on each e-infrastructure resource periodically update the MDS database with their prediction information.
- (iv) EMPEROR selects the system that is likely to minimize the run time of the job, and submits the job to this resource.

EMPEROR is designed to operate within the OGSA framework, the only non-standard part of the e-infrastructure architecture being the EMPEROR Resource Prediction System running on each resource. The Resource Prediction System is the key to EMPEROR's job placement strategy. It implements the forecasting models used by the system, and can switch between different prediction models on the same host, necessary due to the changing characteristics of resource usage. It can operate on any type of resource trace (they use resource load and memory consumption), meaning that it can be easily modified to support different types of resource as they are brought to the e-infrastructure.

5.1.5 XML-Based Policy Framework in EZGrid

The EZGrid project [190] aims to promote controlled resource sharing and efficient job execution on a distributed e-infrastructure. It makes use of Globus services such as GRAM for job submission and GSI for security. It uses Policy Engine, a policy based framework to facilitate authorization and accounting. Policy specification and evaluation are expressed in XML, with policies transferred around the e-infrastructure using GridFTP. Policies can be set specifying when and how many jobs a user can run on a system, and the number of 'credits' in the system assigned to a user. The resource usage rule file is evaluated when the resource requests are obtained from the user.

5.1.6 Community Scheduler Framework

The Community Scheduler Framework (CSF) [191], developed by Platform Computing, is an open source implementation of a service oriented scheduler framework, designed to be integrated with the Open Grid Services Architecture. It is based around the concept of "community schedulers", that is a scheduler that is designed to assign jobs between the resources of a single virtual organization (VO). In this context a VO can be thought of as a community of users and the resources that are shared for the benefit of the community. In the community scheduler model, multiple VOs can have schedulers that schedule to the same e-infrastructure resources, if those resources are part of multiple VOs.

The CSF provides basic capabilities for scheduling, and provides a framework for implementing community schedulers, which individual VOs can use to produce their own bespoke metaschedulers, tailored to their individual needs. The CSF metascheduler comprises several services, namely: the Job service, the Reservation service, the Global Information service, the Queuing service and the Resource Manager Adapter Service. The Job service provides an interface for placing jobs on a resource manager (for example a Globus GRAM service, or local job manager), and managing the job once it has been dispatched, and provides basic matchmaking facilities between job requirements and available resources. It uses the Global Information service to store details on the state of submitted jobs. The Reservation service allows both end users and the Job service to reserve resources, to guarantee that the resource is available to run the job. Once a reservation is made for a particular job, the Job service sends the

job to the resource manager responsible for making the reservation. Policies can be associated with the Resource service to limit which users are able to make reservations, and how many nodes on a cluster a user is allowed to reserve. The Global Information service provides a repository of information required by the metascheduler, and is built on the Globus Toolkit's Index Service. It provides a consistent interface for persistent state information required by other CSF services. The RM Adapter service provides a consistent e-infrastructure interface for communicating with underlying resource managers. In order to communicate with resource managers that do not have an RM Adapter service bridge, the CSF also supports job submission via a Globus GRAM service.

The Queuing service is responsible for scheduling jobs based on policies defined at the VO level. It uses the policies to map jobs to resource managers, for example, the Fairshare policy that it implements ensures that all users in a VO have fair access to resources, rather than allowing a single user to monopolize a queue. CSF implements a plug-in framework, whereby new scheduling algorithms can be written as plug-ins for the system. Scheduling happens in two phases: firstly the scheduler is passed a queue of jobs which it orders based on its scheduling algorithm (for example first come first served), secondly the jobs in the queue are matched to available resources. If there are multiple scheduling policies in the Queuing service the appropriate schedulers will be called in order. In this way multiple scheduling algorithms can be combined to affect the order of the job queue.

The CSF comes with two default scheduling algorithms. The first of these is based on first come first served; jobs are matched with resources in the order that they are submitted to the queuing service. The second is a throttling scheduler, which ensures that one RM does not receive too many jobs at the same time.

5.1.7 Kalman Filter Based Resource Scheduling

The predictive resource scheduling approach developed by Chapman *et al.* [192] uses a linear Kalman filtering technique to predict future resource utilization, and hence where a particular job is likely to run in the shortest time. The Kalman filter technique is based on predicting future CPU utilization, and exploits the fact that users submit jobs in repetitive patterns. The advantages of using Kalman filter prediction are that

the predictor does not require a training phase, and that the entire history of the system does not need to be maintained to make predictions.

The Kalman filter based predictive scheduler has been used in conjunction with Condor to schedule jobs on a Condor managed cluster, and demonstrated that the predictive approach taken leads to reduced user waiting times, compared to scheduling based on shortest queue length. The utilization factor chosen, on which predictions are made, is the number of unclaimed resources in the system, that is, the number of resources in the Condor environment that are not being used. At each site the predictor reads the number of resources available at a regular interval to update its state space model. This is then used to predict the number of resources that will be available at the next time step, or by feeding the predictions back in to the predictor, at a number of time steps in the future. Jobs are submitted to the site with the predicted highest number of unclaimed resources.

5.1.8 Conservative Scheduling using Predicted Variance

The conservative scheduling approach of Yang *et al.* [193] uses a time series prediction technique to predict the average CPU load on a resource at a future point in time and over a future time interval, and to predict the variance in CPU load over a future time interval. These predictions are then used to balance data loads between processors within a distributed processing task so that each processor finished executing its task at roughly the same time, a form of load balancing known as *time balancing*.

In order to efficiently distribute data between heterogeneous processing units, the CPU load over the time interval that the application will run is determined, which is then used to predict the variation in CPU load. This is then used to distribute data between processors, to minimize the runtime of the loosely synchronous distributed application. Their conservative scheduling approach always allocates less work to the CPUs displaying the highest variance. The experimental validation of the technique focused on running a data intensive astrophysics application distributed across workstation clusters based at several US universities. It showed that their technique performed better, using several different scheduling algorithms, than techniques based on mean CPU load alone.

Scheduler name	Wait time prediction	Application model	Cost model	Job management
Condor	No	Single job	No	Yes
Nimrod/G	No	Parameter sweep	Yes	Yes
Gridbus	No	Parallel	Yes	Yes
EMPEROR	Yes	Single job	No	Yes
XML-Based Policy Framework	No	Parallel	No	N/A
Community Scheduler Framework	No	Parallel	No	Yes
Kalman Filters	Yes	Single job	No	Through GridSAM
Predicted Variance Technique	Yes	Data parallization	No	N/A

Table 5.1: Comparison of features of different e-infrastructure metaschedulers

5.1.9 Comparisons of Metascheduling Approaches

Computational e-infrastructure are used as a mechanism to share a wide variety of resources, and support the execution of a wide variety of scientific and engineering applications. As such, there are many different approaches to the problem of scheduling work across the resources on a particular e-infrastructure, many of which are designed around the type of applications that users want to run on the particular e-infrastructure that they are targeted at. The metascheduling approaches discussed in Sections 5.1.1 to 5.1.8 take a number of different approaches to solve the problem of efficiently distributing jobs amongst the constituent resources of a e-infrastructure. Table 5.1 compares the salient features of the different approaches, namely, the types of jobs the scheduler supports, whether the scheduler uses any method of resource utilization prediction to determine when a job will run, whether the scheduler supports a resource usage cost model and whether the scheduler incorporates a job submission and management engine. While some of the metascheduling techniques discussed might be applicable to many different scenarios, the table considers those scenarios to which the techniques were applied in the literature discussed above.

The most striking conclusion we can draw from the table is that since 5 out of

the 8 systems considered do not support parallel applications, they are not suitable to HPC e-infrastructure usage. However, all of the schedulers employ some techniques which could be usefully applied to HPC e-infrastructure scheduling. As can be seen from the table, there are several approaches which use predictive models to estimate when a job will run, and several models which include a cost model to allow users to price the jobs that they want to run. The matchmaking approach taken by Condor connects jobs with the resources that match their requirements. The schema free nature of the Condor ClassAds mean that providers and users can specify any attributes they like, with the caveat that the attributes specified may not be able to be matched to any resources. The ClassAds system has a drawback in that stale ClassAds registered with the system (ones which no longer reflect the availability of a particular resource) can lead to bad matches and thus poor utilization. The Kalman filter predictor shows how a future resource utilization prediction model can be used within a Condor system to reduce the wait time for running a job.

The approach taken by Nimrod/G allows for users to specify the maximum cost of their job, and to specify the speed with which they want their job to be turned around. Similar to Condor, the Nimrod/G approach is designed to schedule work in a task farm and does so with embarrassingly parallel problems that can be decomposed in to parameter sweeps.

The Condor and Nimrod/G systems are aimed at their original use cases, that is running multiple single CPU jobs on loosely coupled clusters of workstations, in a task farming or cycle scavenging fashion.

The Community Scheduler Framework provides a simple and extensible scheduling system which virtual organizations can use as basis on which to develop metaschedulers tailored specifically to their needs. The mappings between resources and jobs are based on scheduler policy files; by default the scheduler does not take in to account any prediction of future resource usage, or the cost involved in running jobs. By default a first come first serve queue of jobs is maintained which are mapped to resources as they become available. Although jobs can be submitted through the Globus GRAM interface, jobs that need to make use of advanced reservation can only be submitted to resources running the Platform Computing LSF interface, since the Globus GRAM does not provide mechanisms to make reservations.

The Gridbus metascheduler is built on many underlying grid services, and provides a dynamic cost model which allows for resources to advertise and modify their prices, introducing a computational market. However the scheduler does not take in to account predicted resource utilization, so does not give the user to specify their scheduling requirements based on both the cost and runtime of their job.

XML based policy framework approach provide a static mechanism for resource administrators to limit use of their resources to certain users or subsets of users, but does not attempt to minimize job run times by modelling predicted future resource utilization, or provide a dynamic costing model to allow jobs to be priced.

The EMPEROR metascheduler is designed to run in the OGSA grid framework, and therefore should be able to schedule in any e-infrastructure that implements OGSA. The predictive model used to estimate future resource utilization uses a measure based on CPU load memory usage, which is appropriate for the single CPU resources used to test the system, but not an appropriate measure of utilization on HPC resources using complex queue management systems. The scheduler also features no cost model to allow for the price of jobs to be set.

Conservation scheduling technique using predicted load variance is slightly different to the other schedulers discussed here, in that it is concerned with time balancing data intensive jobs distributed around a number of loosely coupled workstation CPUs. It demonstrates how a model of predicted utilization can be combined with scheduling algorithms to reduce job run time compared to approaches that use no predictive modelling. It does not however feature a cost model, assuming that the cost of the workstations it is scheduling between is fixed, or irrelevant.

As discussed, several of the approaches under consideration have shown than modelling the future resource utilization in a e-infrastructure allow better scheduling decision to be made [192, 193]. All of the metaschedulers described above include some job management component, to take care of job submission and execution management. The metaschedulers that take in to account some notion of resource cost do so in a relatively unsophisticated way; resources do not vary their cost based on utilization to create a true market.

The majority of metascheduling approached described above that have some predictive model of future resource utilization use CPU and memory utilization as mea-

tures on which to predict when a resource will become free. This approach is applicable on single CPU resources, but is not expressive enough to cover the HPC clusters managed by a local scheduler which implements its own scheduling algorithms, such as Back Fill. The model proposed by Smith *et al.* [194] (discussed more fully in Section 6.2) takes in to account the need to make predictions based on the utilization of local queuing systems.

5.2 Auctions

A seller wishing to sell an item will usually desire to sell it for the highest price possible. Where an established market does not exist in order to set the price for the item to be sold, the seller is able to devise their own set of rules governing the sale, called the mechanism. Mechanism design is the process of constructing a mechanism which the seller hopes will maximize the expected selling price (called the *optimal mechanism*). A full discussion of mechanism design and auction theory is beyond the scope of this thesis, but we refer the interested reader to Bigmore [195, pages 523-536]. The challenge when designing a mechanism is to allow the principal (the person selling an item) to discover the private information held by the actors in the game (called the agents), that is the maximum they are prepared to pay for the item being sold. The set of mechanisms that are feasible, i.e. which could be put into practice, are called *auctions*.

The aim of an auction is to reveal to a seller of an object the maximum value that a set of buyers assign to the object for sale. There are a number of commonly used auction mechanisms devised to do this, which are applicable to different situations. In most cases an optimal mechanism will achieve the highest price possible for the object that the seller is selling.

In economic terms, auctions are defined as principal-agent problems, with the seller termed the principal and the buyer the agent. Such problems are found in many areas of economics from government redistribution of tax revenue to the insurance industries design of insurance contracts [195, page 527]. We consider some of the more popular ones below.

5.2.1 Take it or Leave it Auction

The simplest type of auction, fulfilling the definition given above, is not even usually considered an auction. The *Take it or Leave it Auction* is usually employed in traditional retail situations. An item to be sold is given a fixed price by the vendor, and the buyer either pays the price and takes the item, or leaves it.

5.2.2 Sealed Bid Auction

In a *Sealed Bid Auction* (also called a *First Price Sealed Bid Auction*), each prospective buyer writes their maximum valuation for the item to be sold on to a piece of paper and then seals it in an envelope, so that no bidder is aware of the valuation placed on the object by the other bidders. The seller then sells the object to the highest bidder, using a predetermined protocol to deal with ties, such as selecting a winner at random from the set of highest bids.

5.2.3 English Auction

The *English Auction* is the type of auction most usually associated with the term auction. Traditionally an English Auction involves an auctioneer inviting ascending verbal bids from a group of buyers, until no one is willing to go above the last bid made. Whoever made the final bid buys the object at the price they bid.

5.2.4 Dutch Auction

In a *Dutch Auction*, the auctioneer starts by announcing a high price, then subsequently lowers the price by a set amount at each call, until one of the set of buyers calls a halt. The buyer who stops the auction wins the item for sale at the last price called when he intervened.

5.2.5 Vickrey Auction

The *Vickrey Auction* [196] is similar in protocol to the First Price Sealed Bid auction, with the winner being the bidder that makes the highest bid, but differs in that the winning bidder pays the price of the highest losing bid. While it may at first seem non-optimal for the seller to sell their item at a lower price than they necessarily had to, the Vickrey Auction actually produces higher bids than the First Price Sealed Bid Auction, as buyers have nothing to lose by bidding below their true valuation of the item being sold. Bidding below his true valuation in fact lessens the chance a buyer has of winning

the item without reducing the price he would pay for the item. Bidding above his true valuation would not benefit the buyer either, as in order to win, some other potential buyer must have submitted a bid at least equal to the first buyer's valuation. At best the first buyer would end up paying their true valuation, but could well pay more.

5.3 Optimum Auction Design

In order to discuss the design of optimum auction mechanisms, it is necessary first to define some terms widely used in the mechanism design literature. As stated above, auctions are principal-agent problems, where the principal is the seller of an item and the agents are the buyers. Each agent is said to have a *type*, which is the maximum value the agent places on the item being sold. The type of each agent is hidden from the principal, and the purpose of a mechanism is to force the agents to reveal their type.

Auctions are characterized by the presence of asymmetric information (the type of each agent). In a *private value* auction model, the value that each agent places on the item being sold is known only to himself. In a *pure common value* model, the value of the sale item is the same for everyone, but agents hold different private information about what the value actually is, and may have access to different signals in the environment that influence their decisions (for example, the amount of natural gas available to the holder of a gas drilling licence that is being auctioned). A general model that encompasses both of these auction types combines both the agent's own private information, and common signals from the environment [197, pages 248-251].

5.3.1 Revenue Equivalence Theorem

The *Revenue Equivalence Theorem* underpins much of the research in to auction theory. Following on from the work of Vickrey [196], it has been found that the equivalence of expected revenue in a wide range of different auction mechanisms applies very generally [197]:

“Assume each of a given number of risk-neutral potential buyers of an object has a privately-known signal independently drawn from a common, strictly-increasing, atomless distribution. Then any auction mechanism in which (i) the object always goes to the buyer with the highest signal, and (ii) any bidder with the lowest-feasible signal expects zero surplus, yields

the same expected revenue (and results in each bidder making the same expected payments as a function of signal).”

Revenue Equivalence applies to both private-value auctions and common-value auctions with independent bidders signals. It means that all of the standard auction types discussed above, plus many non-standard mechanisms, yield the same expected revenue. For a full proof of the theorem, readers are referred to Klemperer [197].

5.4 Double Auction

The standard auction mechanisms considered above involve a single seller designing and controlling the mechanism. By contrast *Double Auction* consider situations where buyers and sellers are treated symmetrically, with buyers submitting bids and sellers submitting asks. One of the most famous examples of a Double Auction is the *k-double auction* proposed by Chatterjee and Samuelson [198], in which a single buyer and a single seller submit respectively a bid (b) and an ask (s), and if the bid exceeds the ask the trade proceeds at the price $kb + (1 - k)s$, where $0 \leq k \leq 1$. Since both buyer and seller have incentives to misrepresent their true values, this is not necessarily an optimum mechanism.

5.5 Reverse Auction

Reverse Auctions [199, 200] (also called *Procurement Auctions*) are a relatively recent innovation, made possible by the communications infrastructure provided by the Internet. Unlike the traditional or forward auctions described above, a reverse auction consists of a single buyer and multiple sellers. The process is initiated by the buyer contacting multiple sellers and issuing a request for quotation (RFQ), specifying what they would like to purchase. Within a set time frame the group of sellers then enter (potentially multiple rounds of) quotes at which they will supply the requested goods or service. Once the bidding has closed the buyer contracts the seller that most closely matches their needs, be this lowest cost or some other factor, to supply the service. The key difference between the auctions described above (single seller, multiple buyers) and the reverse auction (multiple sellers, single buyer) is that the reverse auction is initiated by the buyer rather than the seller.

Reverse Auctions are frequently used in business to business industrial procure-

ment to solicit quotes for required goods or services. Multiagent systems have been used to manage trading in several reverse auction situations, including e-Commerce applications [201] and in supply chain management [202].

5.6 Multi-attribute Auctions

In the majority of the auction mechanisms considered above, the buyer has a single objective in mind: to obtain the item on sale at or below their maximum valuation of the item. However in some auction scenarios, especially reverse auctions, price is not always the most important factor [203], and buyers can make decisions on other factors that they consider important (for example, time to delivery of a contract). Adding extra attributes other than price compounds the problem of choosing an optional auction mechanism, although some work has been done to show how, where actors' preferences are known, multi-attribute auctions can be reduced to single-attribute auctions [204].

5.7 Combinatorial Auctions

All of the different auction mechanisms considered above have considered situations where a single unit is being auctioned. *Combinatorial Auctions* (also called *Multi-Unit Auctions*) [205] are the class of auction where multiple items are bought or sold in the same auction. Combinatorial Auctions can be seen as extensions to the types of auction described above, for example, an English Auction in which a single seller is selling multiple items or a fraction of a single item to a group of buyers, or a Reverse Auction where a single buyer is looking to buy a set of services from one or more seller.

Combinatorial Auctions can take one of two forms: *Simultaneous Auctions* or *Sequential Auctions*. Simultaneous Auctions, first considered by Wilson [206] involve a group of sellers bidding for a fraction of an item. Sequential Auctions are the class of Combinatorial Auctions where multiple units are sold sequentially (for example through multiple rounds of single unit auctions).

Much current research on combinatorial auctions has found that it is very difficult to achieve efficient outcomes [197], due to the multi-dimensional private signals held by bidders, for example due to the tendency of buyers to reduce demand when bidding on large numbers of units in order to reduce the sale price.

5.8 Software Agents

The term ‘software agent’ is used to describe systems that act on behalf of a user to find and filter information, negotiate services, automate tasks or collaborate with other agents. Agent technology has been applied to many diverse areas of computer science. One such area is the field of optimal staff time-tabling [207], using agents to produce staff timetables within business settings. Another example is in the area of process planning, using software agents to organize tools, people and systems in distributed industrial organizations [208].

Although there is some disagreement about exactly what a software agent is and what constitutes a software agent, general consensus in the literature [209, 210, 211] coalesces around the definition proposed by Wooldridge and Jennings [212], which is a software system that satisfies the following properties:

- **Autonomy:** agents operate without the direct intervention of humans or others.
- **Social Ability:** agents are able to interact with other agents via an agent communication language.
- **Reactivity:** Agents perceive their environment and are able to respond to changes in it.
- **Pro-activeness:** Agents do not simply respond to changes in their environment, they exhibit goal directed behaviour.

Wooldridge and Rao [213] go on to define the concept of rational agents as software entities that perceive their physical environment through appropriate sensors, have a model of and can reason about the environment of which they are part, and based on their own internal state can take actions that change the environment.

Frequently an intelligent agent will be characterized by its beliefs, desires and intentions (BDI)[214, 215]. The BDI agent architecture defines explicit data structures that correspond to these characteristics. Beliefs are the knowledge that the agent has about the world when it is started up. The goals that the agent wants to achieve are represented as desires. During the course of its execution, the goals that the agent wants to achieve are pushed on to the intention stack. Another common approach to building intelligent agents is the Markov Decision Process architecture [216]. MDP

assigns utilities to states and probabilities to transitions between states, and applies an algorithm to generate a mapping between states called a policy. The policy is a complete specification of what an agent should do in each state, based on the probable outcomes of every action. The need to establish the outcomes of every possible action in every possible state makes finding such policies intractable in many real world cases, and a major drawback to the MDP approach.

In addition to the different approaches taken to building agents, agent goal types themselves can vary [217], for example an agent may be blindly committed to a goal (i.e. it holds on to the goal until it achieves it), or single minded (it will drop the goal if it believes it can never be achieved).

Bradshaw [218] defines the concept of an agent as a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents. The concepts of continuity and autonomy are important, because they mean that an agent will respond to changes in its environment in a flexible and intelligent manner, without requiring constant human intervention.

As can be seen from the definitions of agenthood presented above, software agents are characterized by their ability to operate without user intervention to achieve the goals that they have been set, to respond to environmental changes, and potentially to interact with other agents to achieve their goals.

Very often software agent components are developed using an agent framework, such as the Java Agent DEvelopment framework (JADE) [219] or ARCHON [220], to take care of communication protocols between agents and speed up software development, or to facilitate the testing and simulation of agent interaction environments [221]. It has been recognized [210] that agents provide a powerful level of abstraction which can be used to define a radically new way of engineering software. Agent oriented software development methodologies such as GAIA [222] attempt to formulate a new software engineering paradigm based on software agents.

5.8.1 Multiagent Systems

A multiagent system is a collection of software agents that co-operate with each other to solve problems that are beyond the ability of a single agent. According to Wooldridge [223][page 105] “There’s no such thing as a single agent system”: all but the most

trivial systems contain numerous components which need to interact with each other. In a multiagent system, different agents within the system have control over, or are able to influence, different parts of the environment. The key abstractions in a multiagent system are the agents themselves, the interaction between agents, and the organization of the interacting agents. Weyns *et al.* [210] describe multiagent systems thus:

“Multiagent systems provide an approach to solve a software problem by decomposing the system in to a number of autonomous entities embedded in an environment in order to achieve the functional and quality requirements of the system.”

Multiagent systems have been applied to areas as diverse as business process management [224] and resource selection games [225] to controlling the distribution of power in the electrical grid [226] and the modelling of a single human decision maker [227].

5.9 Agents and Distributed e-Infrastructures

Agent technology has the potential to automate much of the mundane footwork carried out by the scientist when conducting scientific studies using computational e-infrastructure resources. Indeed, some work has already been carried out investigating the use of agent technologies to manage scientific workflows, for example in the field of medical decision support [228] and in generic virtual laboratories [229].

Foster *et al.* [230] have recognized that many of the objectives of the agent and grid research communities intersect, but are being pursued from different perspectives. They characterize this intersection as brain meets brawn; the brawn is the infrastructure, tools and middleware developed by the grid community. The brain is the autonomous problem solving capabilities of software agents. They argue that the dynamic nature of e-infrastructures create environments where agent technology, which can cope with the change and evolution of the environment, can fulfil a number of different roles, and highlight five different areas where e-infrastructures and agent based systems intersect:

- **Autonomous Service.** Grids are built on the concept of services - entities which provide capabilities to a client via a well defined message exchange interface.

Based on this definition, agents can be seen as services that display autonomous action, for example in a e-infrastructure load balancing application.

- **Rich Service Models.** Both agents and e-infrastructure systems consist of dynamic and stateful service. e-Infrastructure research has concentrated on providing robust service architectures and well defined interfaces, but the agent community have not considered this a priority.
- **Negotiation and Service Contracts.** In an e-infrastructure system it can often not be assumed that a particular service will be able to provide a particular capability to a user. If the system is to have any kind of predictable behaviour it becomes necessary to obtain commitments about the level of service provided. Agent research has invested significant effort in developing negotiation algorithms which can be used to obtain such commitments dynamically.
- **Virtual Organization Management.** Grid resources coming together to form a VO can be viewed as a dynamic service composition problem. The VO require the roles and responsibilities of each of the participant services to be defined, another problem to which agent based negotiation strategies can be applied.
- **Authentication, Trust and Policy.** Identity management and mappings require policy specification and enforcement. Agent based trust and repudiation techniques can be applied here to provide a more dynamic security model to the e-infrastructure environment.

As can be seen from the points discussed above, there are a number of areas where e-infrastructure and agent technology intersect, and where techniques gained from one field can be applied to the other for mutual gain.

5.10 Computational Mechanism Design

Multiagent systems have been applied to many different auction situations, from deriving bidding strategies in combinatorial auctions [231] (auctions where the buyer is bidding on multiple sales) to decentralized multiproject scheduling [232]. The wide variety of situations where multiagent systems can be applied to auctions has led to the development of agent development frameworks specifically for producing agent based

auction systems [233]. Dash and Jennings [234] have shown how *computational mechanism design* (CMD) can be used to address two fundamental issues when designing multi-agent systems: the design of the protocols that govern interaction between agents, and the definition of each agent's strategy. The key strength of applying CMD is that it provides an elegant mathematical framework on which to base MAS systems.

CMD agents are assumed to act in a game theoretic way, which helps facilitate the design of systems that allow system-wide properties such as efficiency and stability to emerge in equilibrium, and helps simplify the design of self interested agents.

5.11 Summary

In this chapter we have looked at the field of mechanism design, and seen how mechanism can be used to formulate auction rules that allow resources to be optimally allocated. A number of common mechanisms are shown to do this with the same level of efficiency due to the Revenue Equivalence Theorem. The growth of the Internet has led to the development of new types of mechanism, such as the Reverse Auction, to assist with and drive down the cost of business to business procurement.

We have also looked at the ideas behind software agency, and seen how multi-agent systems have been applied to a wide range of different problems, and looked at some of the possible problems in grid computing to which MAS could be applied. Finally we have looked at the intersection of mechanism design and MAS, and how computational mechanism design can be used to assist with the design of multi-agent systems.

Chapter 6

Design and Implementation of a Resource Allocation Market Place

In this chapter we review the requirements for a distributed resource allocation system presented in Chapter 3 and look at how the technologies discussed in Chapter 5 can be used to meet those requirements. We go on to present a design for a system that meets these requirements, and discuss our implementation of this design.

6.1 Designing a Resource Allocation System

The requirements described in Section 3.4 that relate to the resource allocation mechanism are that it must be user initiated, capable of allowing users to specify their requirements for an application run and allow users to request access to multiple resources (requirements 5, 6 and 12), which to reiterate are:

- (v) The process of submitting an application should be initiated by the user and done at the user's convenience, rather than at a time specified by the computational resource provider.
- (vi) With current systems the onus is on the user to choose the resource on which they want to run, meaning that they often choose the one they think will be able to run their job fastest, or the one they are most comfortable using. Instead of requiring users to choose resources, our system should allow the user to specify requirements for their application run, such as the time they need the results to be produced by, or the maximum cost they are willing to pay in order to run the application.

- (xii) Users may require access to multiple resources in order to run their application. For example, for an application that consists of a simulation code and a coupled visualization engine, the user would need access to a compute resource and a visualization resource.

In Chapter 5 we reviewed a series of existing solutions and technologies which could be applied to satisfy these requirements. None of the existing resource allocation technologies we reviewed meet fully the requirements, and so we must address the requirements by designing our own system. Firstly, we need to fully capture the terms a user needs to be able to specify their requirements when requesting that their application be run on a resource. Secondly, we need to design a mechanism that meets the requirements outlined above, and specify how this can be implemented as a software system.

As we discussed in in the previous chapter, open markets provide a means to efficiently allocate resources in a decentralized manner, and auctions provide an efficient means of valuing an item to be sold, or in the case of a reverse auction allowing a buyer to obtain good value for the item that they are buying by forcing sellers to compete for custom.

Section 5.2 lists several type of common auction mechanism. From the list of mechanisms, the one that most closely fits requirement 5 is the reverse auction mechanism, in that it allows the buyer to initiate the bidding process. It also fits the model of a user with access to multiple e-infrastructure resources: the user is the buyer of services (compute time) from the resources. The sellers, in this case the compute resources, possess private information based on the load on their machines, and the range of costs they are willing to accept per core hour in order to maximize utilization of their resources. This allows them to alter their bidding strategy based on how under- or over-utilized their machine is at any given time.

Requirements 6 and 12 require us to extend the basic auction mechanism to make it multi-attribute (since buyers need to make their decision on which resource to select based on the attributes important to them, such as cost or time to solution) and combinatorial (so that buyers can bid for a range of resources from a group of sellers).

Therefore, we will base our resource allocation system on a combinatorial, multi-attribute reverse auction, in which resource providers compete for workloads offered

by users. We name this system the Resource Allocation Market Place (RAMP).

6.1.1 Design Constraints

In order to satisfy the requirements discussed above while maintaining a focus on usability, we have placed the following constraints on our resource allocation system:

- The user should not have to configure the details of every resource that he potentially want to user. The system should automatically discover potential resources as they become available.
- The user should be able to specify what he requires from a resource in order to run their application. Any requirements explicitly specified must be met by the responding resource. However, if no resource can satisfy the requirements after N rounds of bidding, a resource can make its best offer to the user.
- The RAMP system should be accessible through the same client used to access the AHE. The user should not be concerned with any details of where or how an application is run.

6.1.2 Functional Specification

Following on from the requirements and design constraints discussed above, we can derive the following functions, which our RAMP system must possess to address the problem of flexible, distributed resource allocation:

- When the user submits a request for quotation document, a combinatorial reverse auction is initiated between the user and a set of resources to satisfy the request.
- When a resource receives a request for quotation, it examines whether it can meet the request, and either makes an offer or refuse to participate.
- When a user revives an offer, it is held in a list sorted by relevant factors (cost, end time etc.). If the offer is lower than the current bid price, the request for quotation is modified with the new offer in subsequent bidding rounds.
- When a user decides to accept an offer, it initiates a two phase commit process, that ensures that all required resources are available, or none are bought. The user is supplied with a reserved slot on the machine

- When a user and resource have agreed on a deal, it is logged with the market maker who debits the user's credit account by the amount of the sale and credits the resource's account.
- When a user sends a cancellation request, the resource evaluates the request and informs the user whether or not the reservation can be cancelled. If it can, the market maker is informed to credit the user's account from the resource's account.

6.2 The Architecture of the RAMP System

The usefulness of multi-agent systems (MAS) in distributed e-infrastructure environment is discussed in Section 5.9. In addition to the agent programming paradigm, agent development environments such as JADE [219] provide a framework in which much of the software tooling required to develop agents, establish inter-agent communication as so on is already provided, much simplifying the process of developing multi agent systems.

In the context of the reverse auction based metascheduler, multi-agent systems also bring a number of other benefits. One of the drawbacks of some of the meta-schedulers discussed in Section 5.1 is that they rely on central information services to aggregate data from resources and maintain their world view. The obvious limitation of this approach is that scheduling decisions may be made on out of date data. In the MAS approach proposed, each agent is responsible for maintaining the view over its own sphere of the world, meaning that the data used to make scheduling decisions is more current. This accords with the devolved nature of a distributed e-infrastructure, and especially federated e-infrastructures.

The application of multi-agent systems to auctions discussed in Section 5.10 shows that trading systems and economies can be successfully built from interacting software agents. This model is thus also applicable to the distributed e-infrastructure economy that is developing as commercial providers trade compute power on an open market (badged as 'cloud' HPC). Finally, MAS provides a software development framework featuring a high level of abstraction for building autonomous, rationally functioning software systems. The distributed nature of e-infrastructure systems coincides with the distributed, multi-agent system model of programming, and leads to the

development of fault tolerant peer-to-peer systems, in which the failure of one components does not have too adverse an impact on the rest of the system.

This makes the MAS paradigm ideally suited to develop our distributed resource allocation system. Within the system, software agents can act on behalf of the different entities involved, principally users and resources. Two different BDI (*cf.* §5.8, i.e. one characterized by its beliefs, desires and intentions) types of agents will feature in the system:

- Resource management agents, responsible for maximising the utilization of a resource. A resource agent is run on each constituent resource of the distributed e-infrastructure. It maintains a predictive model of resource availability, which it uses to decide when it is able to run a job. It can vary the cost of the offers to run jobs that it makes to encourage jobs to run when the machine is free by lowering its prices, or increasing the cost when the machine is overloaded with jobs to maximize revenue, based on a set price range configured by the resource administrators.
- User agents, responsible for gaining access to resources at a cost and availability specified by the user. The user agent runs on the client machine, and negotiates with the resource management agents for the most appropriate resource to run a particular application. Users provide the agent with a description of their cost and time requirements for the job in RFQL (*cf.* §6.2.2); they can either ask for the job to be run in the fastest time possible, at the least cost, or at a specified maximum cost and/or wait time. The agent's goal states consist of minimising the cost of the job, minimising the wait time of the job, or at least matching the specified requirements. The agent then initiates multiple rounds of bidding with the resource management agents until the requirements are achieved and the application is launched, or if the requirements cannot be met the user is presented with the best offer received. If no offer is received, the application fails. The user will be able to specify both static and dynamic constraints on their job, as defined in the RFQL schema (*cf.* §A).

In addition, a banking agent acts as a collation point for all successfully actioned requests. The implementation of these agents is described fully in Section 6.3.

6.2.1 Developing the Negotiation Protocols

The process of participating in a reverse auction requires the agents involved to communicate in a structured way. Fortunately, a standardized way exists to achieve this. The Foundation for Intelligent, Physical Agents (FIPA) exists to develop standards relating to software agent technologies. The standards that FIPA develop provide a mechanism for software agents to be mutually understood, regardless of underlying implementation technologies. The FIPA Agent Communication Language (ACL) [235, pages-10-17] specifies the structure of inter-agent messages, and defines a set of *communicative acts* (CAs), performed by the act of communicating. These CAs, along with a bespoke content language, allow agents to participate in a reverse auction.

We define three different procedures for agents to communicate in different circumstances:

- The reverse auction negotiation - the actual negotiation process required to conduct a reverse auction.
- The banking update negotiation - the process of notifying the banking agent to record a successful auction result.
- The cancellation negotiation - the process of a user cancelling a request.

These protocols are described in detail below.

6.2.1.1 The Reverse Auction Protocol

The reverse auction algorithm developed is adapted from that described by Matsuo *et al.* [201]. The auction consists on N rounds of open bidding, where all sellers can see the bids made by other sellers. As the auction must be based on multiple attributes, the user is able to specify their requirements through a request for quotation (described in §6.2.2).

The auction is combinatorial, meaning that multiple units can be requested. Each sub-request should be treated as a separate auction in the system. This means that an inconsistent state could result, where some parts of an overall request are successful and others are not. Therefore, the auction protocol incorporates a two-phase commit process to ensure the availability of all requested resources.

Briefly the algorithm flow is as follows, and is shown in figure 6.8:

- (i) A User agent initiates the auction by advertising their requirements with Resource agents via an RFQ (FIPA: Call for Proposals).
- (ii) Resource agents evaluate the RFQ and decide whether they can satisfy the request (or section of a request in the case of a combinatorial request), based on their utilization and CPU hour cost. The Resource agents which can satisfy the request make bids, which are propagated to the User agent (FIPA: Propose). If the Resource agent cannot accept the request, it notifies the User agent (FIPA: Refuse)
- (iii) The User agent evaluates the requests it has received, and stores them in a ranked list if they meet its requirements, or else rejects them to the submitting Resource agent (FIPA: Reject Proposal). When the next round of bidding commences, the User agent modifies its RFQ with to correspond to the best offer it has so far received, which is sent to resource agents as its revised request. Steps 2 and 3 are repeated N times.
- (iv) After N rounds of bidding, the User agent evaluates the final set of bids received.
- (v) The User agent selects the most optimal bid or bids that matches requirements and notifies winning Resource agents(s) (FIPA: Accept Proposal), or if no bid or bids match the requirements, the closest matching set are presented to the user for approval. The user can also configure the system to allow them to manually approve all bids.
- (vi) The resource agent(s) holds a slot for the winning bid on the machine queue by creating a reservation in the queuing system, and sends an acknowledgement back to the User agent that they are willing to proceed (FIPA: Agree) along with a reservation ID for the requested slot on the relevant resource. If the Resource agent cannot now satisfy the request (because more jobs have now been queued on the system for example), the agent withdraws from the auction (FIPA: Refuse). This is the *voting phase* of the two-phase commit.
- (vii) The User agent works through all offers received until all parts of the request have been agreed to. Where a winning bid is subsequently refused, the User agent contacts the next best bid and so on until all available bids are exhausted.

- (viii) If all sections of a request are agreed to, the User agent notifies all successful resource agents (`FIPA: Confirm`) and sends a digitally signed copy of the RFQ and the reservation ID back to the Resource agent, signed using the user's personal X.509 credential. This establishes that the user has agreed to the reservation. The Resource agent acknowledges this message (`FIPA: Confirm`). This is the `commit` phase of the two-phase commit.
- (ix) If all sections of a request cannot be agreed to, the User agent cancels all requests received (`FIPA: Cancel`).
- (x) When a Resource agent receives the (`FIPA: Agree`) message (step 6 and creates a reservation slot in its queuing system, it begins a timer process. If the Resource agent does not go on to receive a (`FIPA: Confirm`) message within a given time period, it cancels the reservation slot in the queue.

This is the protocol employed by the user and resource agents to negotiate access to computational resources at user specified time periods. The sequence of FIPA operations are shown in figure 6.1.

6.2.1.2 Reservation Notification Protocol

This protocol is used to inform the banking agent that a request has been successfully fulfilled. After a successful negotiation, the Resource agent proceeds as follows:

- (i) The Resource agent takes the signed message from the user agent, containing the RFQ and reservation ID, and digitally signs it itself, using its own certificate.
- (ii) The Resource agent sends this signed document to the Banking agent (`FIPA: Request`).
- (iii) The Banking agent confirms the digital signatures applied to the message to establish that the veracity of the message, and then debits the user's account in accordance with the request and credits the resource's account commensurately. It then notifies the requesting Resource agent (`FIPA: Agree`).
- (iv) If the Banking agent cannot validate either signature, it responds to the requesting Resource agent with (`FIPA: Refuse`).

The sequence of operations is shown in figure 6.2.

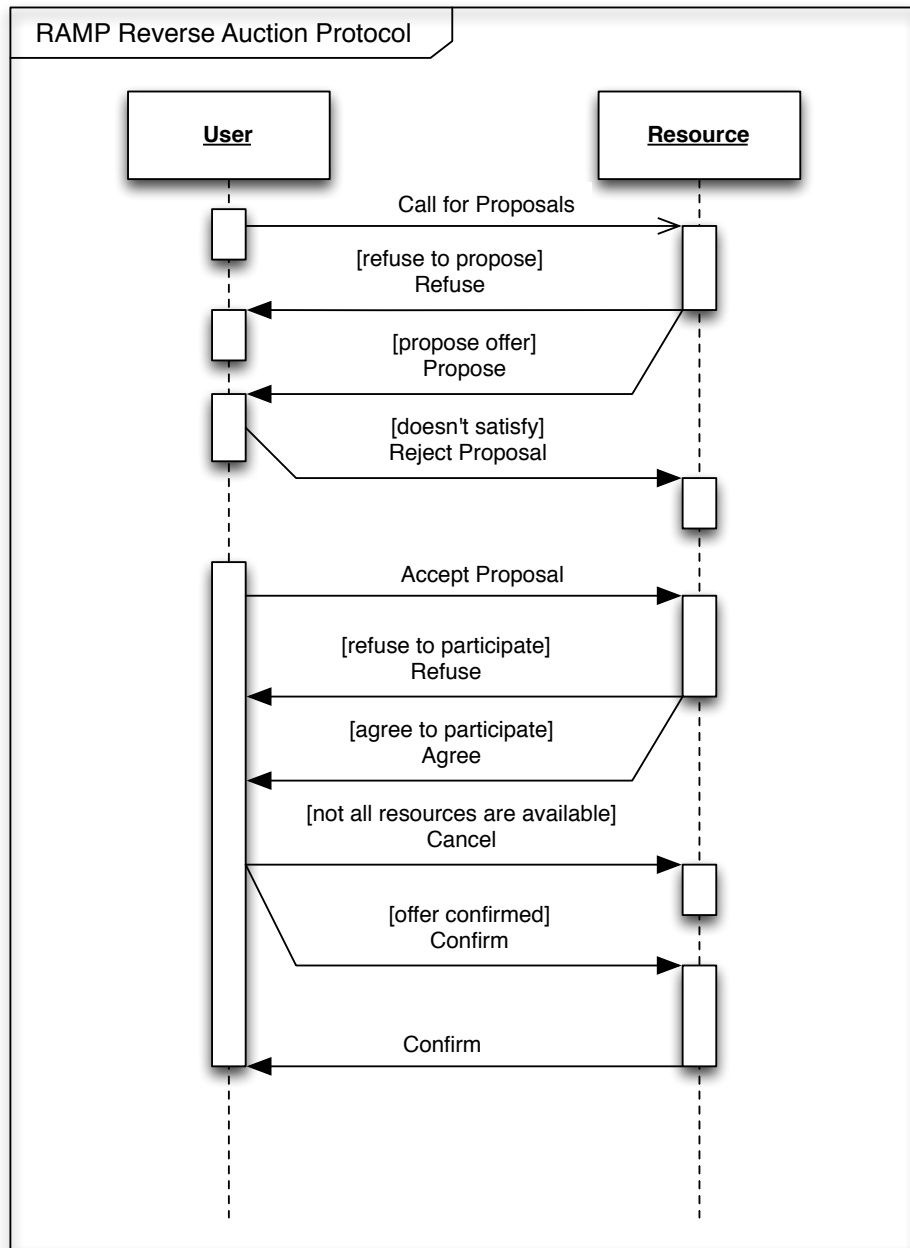


Figure 6.1: The sequence of FIPA messages between a user and a resource in an auction negotiation with $n=1$ rounds of bidding.

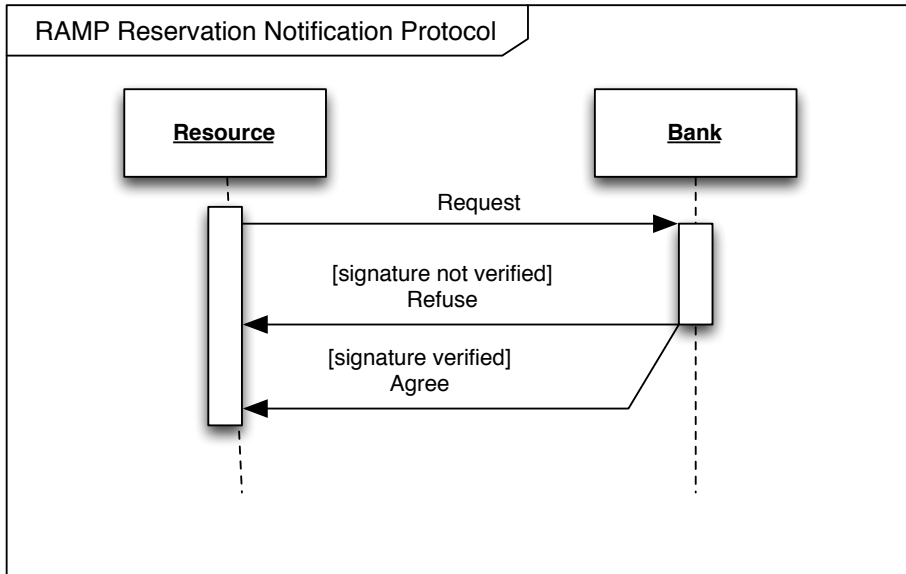


Figure 6.2: The sequence of FIPA messages between a resource and a banking agent required to notify the bank of a successful auction.

6.2.1.3 Cancellation Protocol

If a user needs to cancel a reservation once made, they can do so through the RAMP system. If the Resource agrees to the cancellation, the user’s account will be re-credited with the cost of the resource slot. The following communication protocol is observed:

- (i) The User agent contacts the relevant Resource agent with the request to cancel (FIPA: Request).
- (ii) The Resource agent decides whether or not to accept the cancellation request and either agrees (FIPA: Agree) or disagrees (FIPA: Refuse). If the Resource agent agrees, it sends the User agent a signed copy of the request along with the agreement message.
- (iii) If the User agent receives an agreement, it too signs the request message, and sends the message with both signatures to the Banking agent (FIPA: Request).
- (iv) The Banking agent confirms the digital signatures applied to the message to establish that the veracity of the message, and then credits the user’s account in accordance with the request and debits the resource’s account commensurately. It then notifies the requesting User agent (FIPA: Agree).

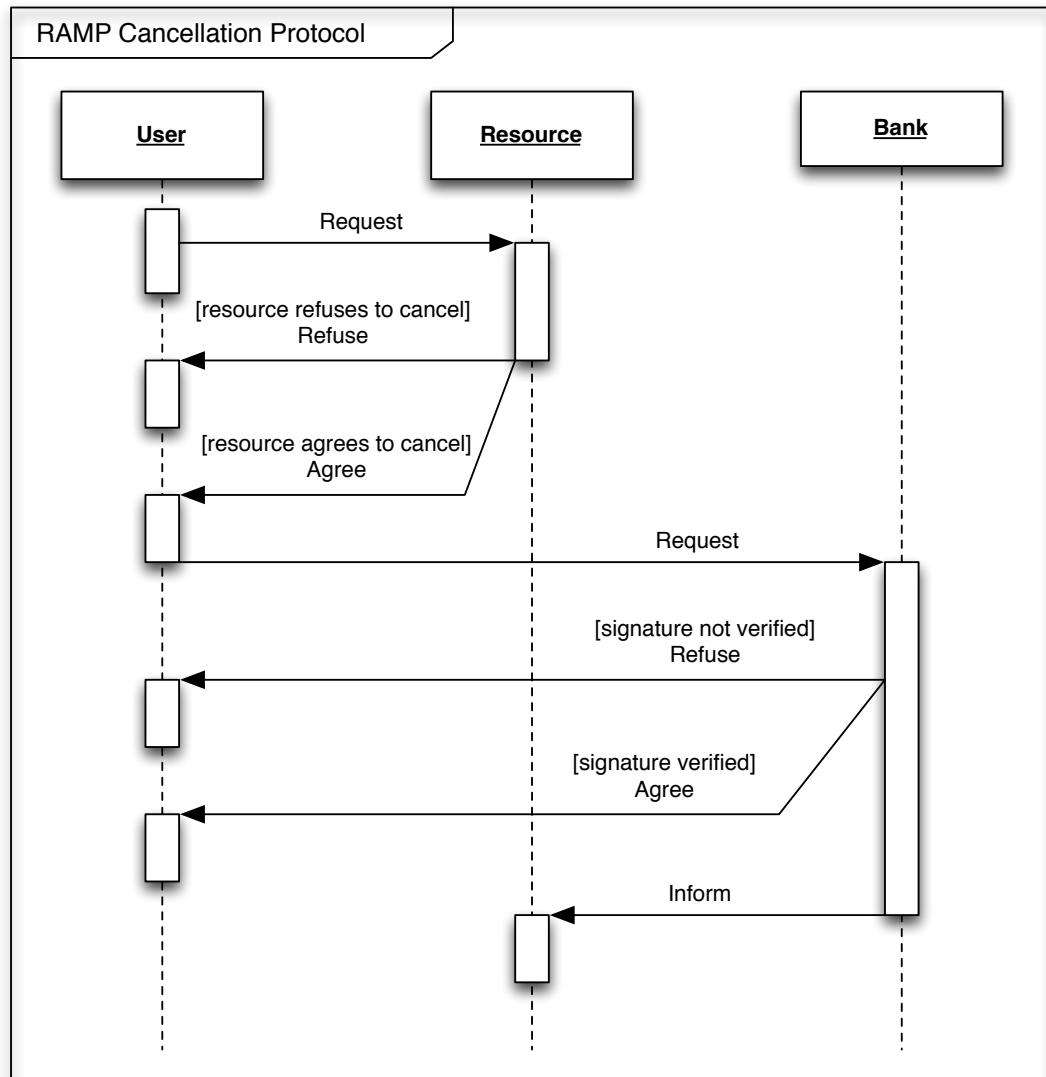


Figure 6.3: The sequence of FIPA messages between a user, a resource and a banking agent when a job is cancelled.

- (v) If the Banking agent cannot validate either signature, it responds to the requesting User agent with (FIPA: Refuse).
- (vi) Finally, the Banking agent informs the resource that the cancellation has taken place (FIPA: Inform), which leads to the Resource cancelling the respective reservation in the queuing system.

This sequence of operations is depicted in figure 6.3.

Note: The system does not provide a capability to cancel multi-unit requests. If a whole multi-unit request must be cancelled, each sub-unit must be cancelled individually.

6.2.2 Specifying a Quotation Request

As we discussed earlier, the overriding concern of the HPC grid user is the time to solution for the problem that she is working on, with a secondary concern of how much the application will cost to run. Our task is to identify the terms which a user needs to specify her requirements from a machine in order to run her application, and to capture these terms in a request for quotation language (RFQL) which provides a standard way of requesting quotations to run applications from resources. Since the AHE takes care of maintaining information such as which resources have which applications installed, the RFQL need not contain terms too specific to the instance of an application the user wants to run (such as the location of a binary); instead it needs only to contain the terms the user needs to specify their requirements from a machine.

To help derive these terms, it is useful to consider the relevant non-application specific terms used by common grid job submission languages. In table 6.1 we list the relevant terms from the OGF standard Job Submission Description Language (JSDL), the Globus 4 Job Description Definition (JDD) language, and the Unicore 5 job description language.

These terms provide a useful starting point from which to build a request for quotation language. Our language also needs to contain terms to allow the user to specify cost and deadline requirements, and aspects that might affect the performance of the application, such as operating system running on the resource, the maximum RAM available, or the CPU architecture. The terms used in our RFQ language are described below:

- **CPUHourCost** - the maximum cost per core hour that the user is prepared to pay in order to run her application.
- **EndDate** - the date by which the user requires her application run to be complete.
- **EndTime** - the time by which the user requires her application run to be complete.
- **StartDate** - the time after which the user needs her application run to start. This is useful if the application run is part of a workflow and depends on a previous application run completing before it is able to start.

Parameter description	JSDL	JDD	Unicore
The operating system requested	OperatingSystemType	N/A	Operating system
The version of the operating system required	OperatingSystemVersion	N/A	N/A
Architecture of the processors required	CPUArchitecture	N/A	N/A
The minimum speed of the processors required	IndividualCPUSpeed	N/A	N/A
The wall clock time per individual processor	IndividualCPUTime	maxTime	Runtime
The number of processors per node	IndividualCPUCount	N/A	CPUs
The inter-node network bandwidth	IndividualNetworkBandwidth	N/A	N/A
The RAM per node	IndividualPhysicalMemory	minMemory	Memory
The total CPU time requested	TotalCPUTime	maxCPUTime	N/A
The total number of processors required	TotalCPUCount	cpuCount	N/A
The minimum disk space required	TotalDiskSpace	N/A	N/A
Number of nodes required	TotalResourceCount	hostCount	Nodes

Table 6.1: Resource requirement attributes from several job description languages

- **StartTime** - the time after which the user needs her application to start.
- **OperatingSystem** - the operating system that the user requires the grid resource to be running.
- **OSVersion** - the version of the operating system that the user requires the grid resource to be running.
- **Architecture** - the CPU architecture that the user requires the grid resource to consist of.
- **CPUSpeed** - the minimum CPU speed that the user requires of her target resource.
- **WallTime** - the maximum time that the application will run for.

- **TotalDiskSpace** - the total disk space that the user needs to be available in order to run her application (this term and the NodeDiskSpace term are mutually exclusive).
- **NodeDiskSpace** - the total disk space the available on each compute node (this term and the TotalDiskSpace term are mutually exclusive).
- **InterNodeBandwidth** - the minimum network bandwidth that user requires between the nodes on the target resource.
- **RAMPerCore** - the minimum amount of RAM that the user requires to be available per compute core.
- **TotalCores** - the total number of compute cores that the user needs to have access to (this term and the NodeCount/NodeCores terms are mutually exclusive).
- **NodeCount** - the number of compute nodes that the user requires access to (this term and the TotalCores term are mutually exclusive).
- **NodeCores** - the number of cores per node that the user requires access to (this term and the TotalCores term are mutually exclusive).

These terms are expressed using XML syntax, formally defined by an XML schema. This is listed in Appendix A. Several of the attributes are required in each instance of RFQL: CPUHourCost, EndDate, EndTime and either TotalCores or both of NodeCount and NodeCores. The other attributes are optional, and it is assumed that the user is not interested in making a decision based on any attribute which is not specified. If an attribute is present, a resource must be able to satisfy it before responding to the RFQ. An example RFQL document is show in listing 6.1.

The schema allows for multiple requests to be made within a single RFQL document, meaning that a user can request a combination of resources in order to perform a workflow, or run a highly distributed application. No formal mechanism is provided to specify dependencies between individual requests, but the StartTime and StartDate terms allow the user to request resources sequentially in time.

The approach we have taken with RFQL is to define a small vocabulary that captures the computational requirements that the user is interested in. This compares with

the Condor ClassAds approach, which allows users and resource providers to define arbitrary terms in the job descriptions. We believe that our small, well defined vocabulary is the correct approach to take here, as it aids system development and improves the likelihood of request/resource matching. We do not expect users to code RFQL by hand, but instead generate documents automatically through our interface tooling.

Listing 6.1: Sample Request For Quotation Document

```
<?xml version="1.0"?>
<RequestForQuotation xmlns="http://www.realitygrid.org/AHE/RFQ" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.realitygrid.org/AHE/RFQ/rfg.xsd">
  <Request>
    <CPULHourCost>20</CPULHourCost>
    <EndDate>2009-07-10</EndDate>
    <EndTime>13:20:00</EndTime>
    <OperatingSystem>LINUX</OperatingSystem>
    <Architecture>x86_64</Architecture>
    <RAMPerCore>1024</RAMPerCore>
    <Cores>
      <Nodes>
        <NodeCount>4</NodeCount>
        <NodeCores>4</NodeCores>
      </Nodes>
    </Cores>
  </Request>
</RequestForQuotation>
```

6.2.2.1 Dynamic Service Level Agreements

An RFQ, plus a response from a resource that satisfies the request, constitutes a contract between the user and the resource to allow the user access to the specified number of processor cores on the resource, for the specified period of time. It can be considered a dynamic service level agreement to provide a specific, one time service to a user at a defined cost. Enforcement of the SLA is beyond the scope of this thesis however.

6.3 Implementation

We used the agent specifications and communication protocols, along with the Request for Quotation notation, to implement a multi-agent system in Java using the JADE

framework. JADE was chosen because it provides a distributed agents platform, supports coordination between different FIPA compliant agents and provides a standard implementation of the FIPA-ACL communication languages. Agents can be quickly constructed by extending the `jade.core.Agent` class. The capabilities of the agent are then defined by implementing behaviour classes which extend subclasses of the `jade.core.behaviours.Behaviour`.

Below we review the three agent types we have defined, and discuss the implemented behaviours that provide their capabilities.

6.3.1 User Agent

As mentioned, the User agent is responsible for purchasing resources on the instruction of its owner. Each user has a single User agent to manage their requests. Since it initiates and manages the auction process, it is the most complicated agent, comprising the greatest number of behaviours. The hierarchy of behaviours is shown by the class diagram presented in figure 6.4.

- `RequestAQuote`: This behaviour is responsible for taking user requests in the form of RFQ documents, translating them to the inter-agent communication ontology used by the RAMP system and imitating and managing the rounds of bidding in the auction. The number of rounds and interval between rounds are user configurable parameters.
- `RequestManager`: The behaviour is used by the `RequestAQuote` behaviour to manage the individual rounds of bidding. It extends `jade.core.behaviours.TickerBehaviour` to fire an event (another bidding round) as set time intervals.
- `ProcessOffers`: This behaviour processes offers received from resource agents during the bidding process, and is responsible for sorting the offers received. A simple sorting algorithm is used, sorting first on cost, then deadline, then the order offers are received in.
- `ResourceNotifier`: This behaviour executes once the `RequestAQuote` behaviour completes, and is responsible for finalising the purchase of the resources requested. It implements the two phase commit required to ensure

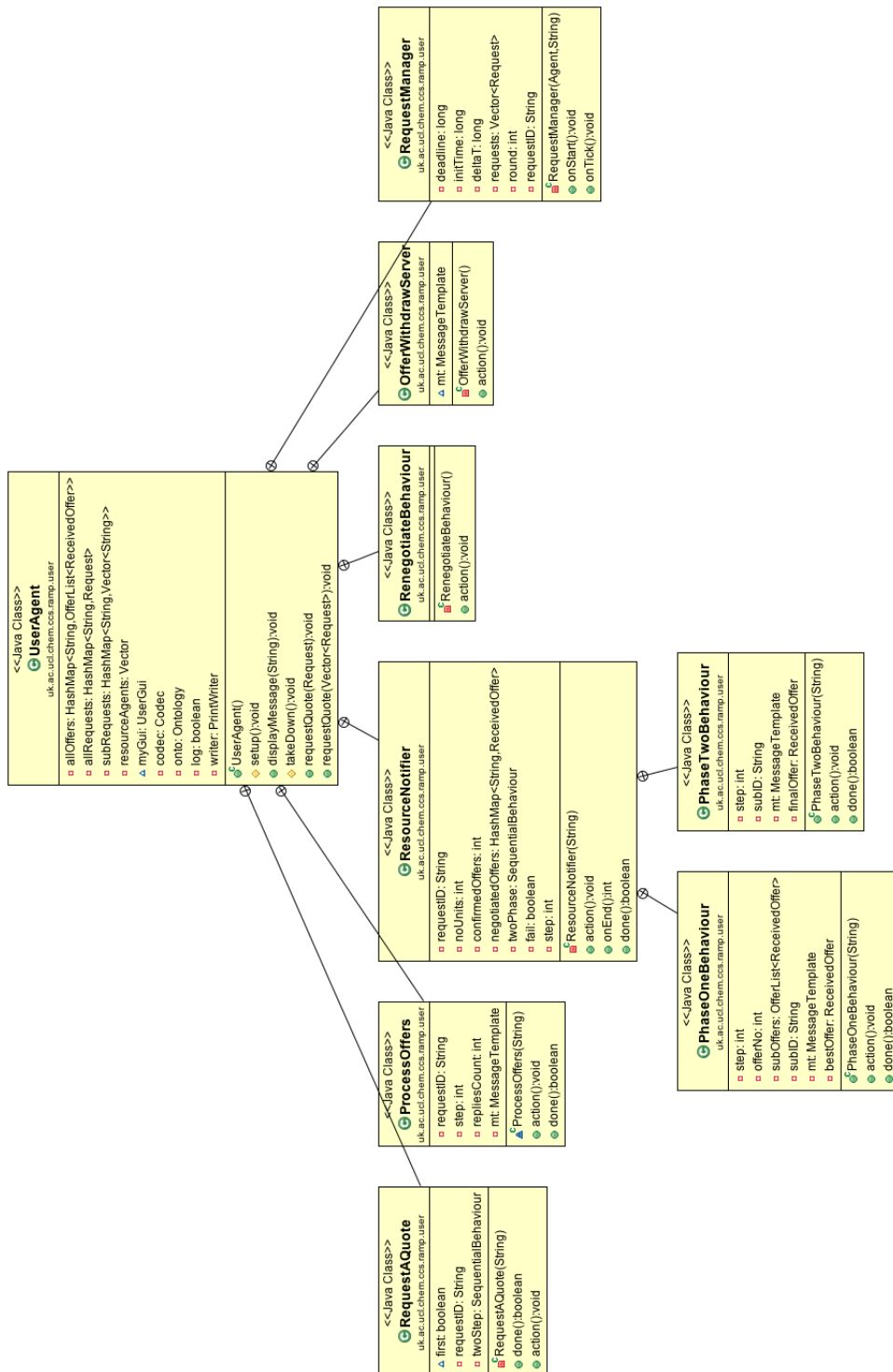


Figure 6.4: The hierarchy of behaviours implemented by the User agent.

consistency when purchasing multiple units in an auction through two sub behaviours:

- `PhaseOneBehaviour`: This behaviour implements the *voting phase* of the two phase commit, instructing successful resources that the user would like to accept their offer, and processing confirmations or rejections from those resources.
- `PhaseTwoBehaviour`: This behaviour implements the *commit phase* of the two phase commit, confirming the offer acceptance if all required resources are available, or cancelling the transaction if not.
- `RenegotiateBehaviour`: This behaviour is used by the User agent to initiate the cancellation procedure (see §6.2.1.3).

The user initiates resource auctions by passing the User agent one or more RFQ documents. To simplify this process, the User agent provides a graphical user interface, shown in figure 6.5. This GUI allows the user to load RFQ documents, initiate and monitor auctions, and also view and manage purchased resources.

6.3.2 Resource Agent

Each resource that participates in the resource allocation market place runs a resource management agent, which is responsible for responding to requests for quotation and negotiating the sale of CPU time. In order to do this, the Resource agent implements four distinct behaviours, shown in figure 6.6 and described below:

- `RFQResponseServer`: This behaviour listens for requests for quotations made by user agents, evaluates those requests and then either submits an offer in response to the request, or else declines to participate.
- `PurchaseOrdersServer`: This behaviour listens for the acceptance of offers from user agents. When an offer is accepted, this behaviour creates a tentative reservation within the machine's queuing system to correspond to the offer. If the requested resource is no longer available, this behaviour declines the acceptance of the offer. It implements the voting phase of the two phase commit protocol on the Resource agent side.

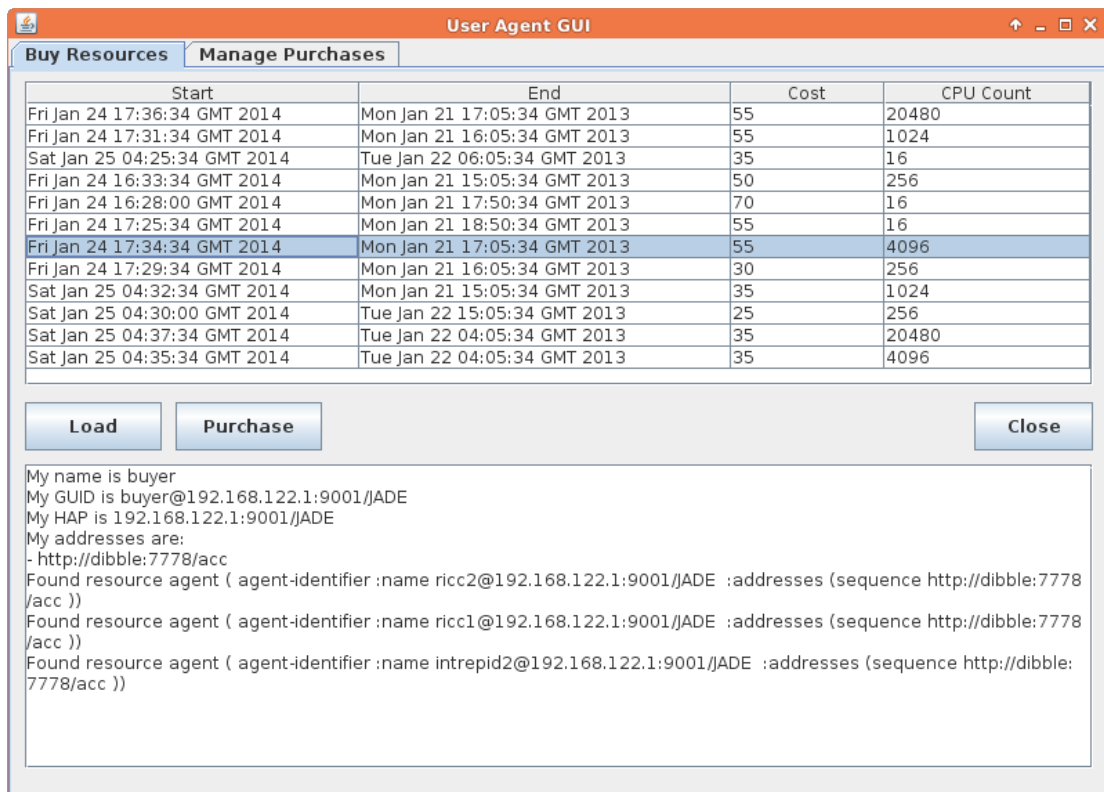


Figure 6.5: The User agent graphical user interface.

- **FinaliseServer:** This behaviour listens for messages from the user agent confirming the finalization of an offer. When an offer is finalized, it stops the time out set on the queue reservation, locking it in. It implements the commit phase of the two phase commit protocol on the Resource agent side. Once an offer is fully confirmed, this behaviour is responsible for initiating the Reservation Notification Protocol described in Section 6.2.1.2.
- **CancelServer:** This behaviour provides the cancellation capabilities capabilities, as described in Section 6.2.1.3.

6.3.2.1 Modelling system availability

The Resource agent maintains an internal representation of the resource that it manages in order to be able to respond to request for quotation. Obviously, the resource must know details of the resource in terms of the CPU types available, memory per node and so on. The administrator of the resource therefore configures these static properties via a configuration file, prior to running the Resource agent. These static properties correspond to the terms of the RFQ specification (*cf.* §6.2.2), but exclude the two

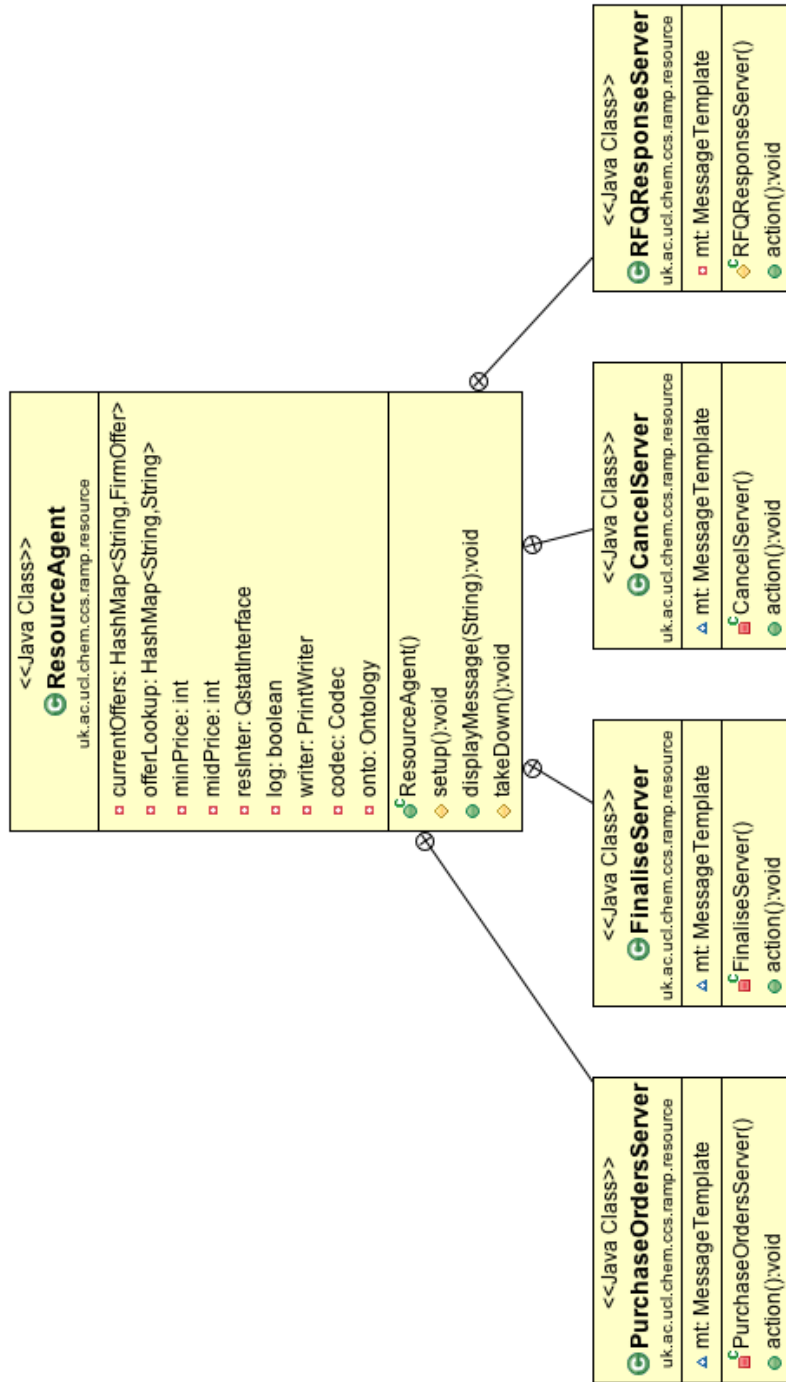


Figure 6.6: The hierarchy of behaviours implemented by the Resource agent.

dynamic properties relating to price and time, which the Resource agent derives from the queuing system.

Note, we consider a resource to be made up of homogeneous compute nodes, although this does not always conform to reality, whereby a large HPC system could be made up of CPUs with different speeds, nodes with different memory sizes and so on. Systems that comprise heterogeneous architectures can be supported by running multiple instances of the Resource agent, one for each distinct part of the machine.

The resource agent will examine each RFQ that it receives, and then decide whether to make an offer or not. Evaluating a request involves two steps:

- (i) First, the Resource agent checks the static terms of the request (such as the requested CPU type) against its internal resource model. If the resource cannot satisfy the this part of the request, then the Resource declines the offer to participate.
- (ii) Next, the Resource agent examines the current load on the resource. If sufficient free CPUs exist at the point in time that they are required, the resource agent calculates an offer price (see below), checks that this offer price meets the request, and then makes an offer.

6.3.2.2 Interfacing with the Queuing Systems

The Resource agent must interact with the resource it manages to obtain a view of system utilization which can be used to respond to requests for quotation, and generate offer prices for those responses. The default implementation interfaces directly with the queuing system to obtain a measure of the load on the resource (in terms of available CPUs) at the point in time when the requested job must be satisfied. While providing an adequate model of system usage, the default queuing system is not without its drawbacks, in that when examining the queuing system to evaluation future availability, it makes calculations based on the wall time specified by the user for each running job. Often, a user will use the system default wall-time, meaning that a job could finish long before the queuing system expects it to. Obviously, the queuing system copes with this by running the next job in the queue, but this can lead to situations where the Resource agent expects the system to be unavailable when it is not.

However, the Resource agent features a plug-in architecture that allows the resource administrators to substitute an alternative resource interface where available. For example, a resource administrator could implement an interface that queries a resource availability modelling service, such as the QBETS batch queue prediction service [236], deployed on XSEDE. The QBETS service can be used to predict a statistical upper bound on how long the job is likely to spend waiting in the queue prior to execution and given the job characteristics and a start deadline, can calculate the probability that the job begins execution by the deadline. The Resource agent can then use this information to make offers, rather than relying on the basic queuing system interface.

6.3.2.3 Pricing model

The Resource Agent is responsible for setting the offer price made to resource requests coming from User Agents. The price offered is varied based on the load on the machine at the time the job must be run. If the machine is lightly loaded, the resource makes a low offer, in order to attract more work to the machine. If the resource is heavily loaded, the machine offers a higher price, or if it is saturated, refuses to participate in the auction at all.

These prices depend on two resource administrator defined parameters, the `start price` and the `minimum price`. As its name suggests, the minimum price is the lowest price a Resource Agent will ever offer in an auction. These values are used to calculate the decrement by which the request is reduced by the resource when making an offer, according to the following formula

$$dec = \frac{(sp - mp)}{s \times (1 - l)}$$

where sp is the starting price, mp is the minimum price, and l is the percentage of the machine that will be allocated at the time the job is to be run, expressed as a decimal. Although the number of bidding rounds is controlled by the User Agent, the Resource Agent anticipates that there will be multiple bidding rounds using the s parameter, so that total decrement is not applied in one go, but gradually over several bidding rounds. This gives a value to decrease the request price by, dec , which results in a bespoke spot price for the resource at a given point in time and in response to a user's request.

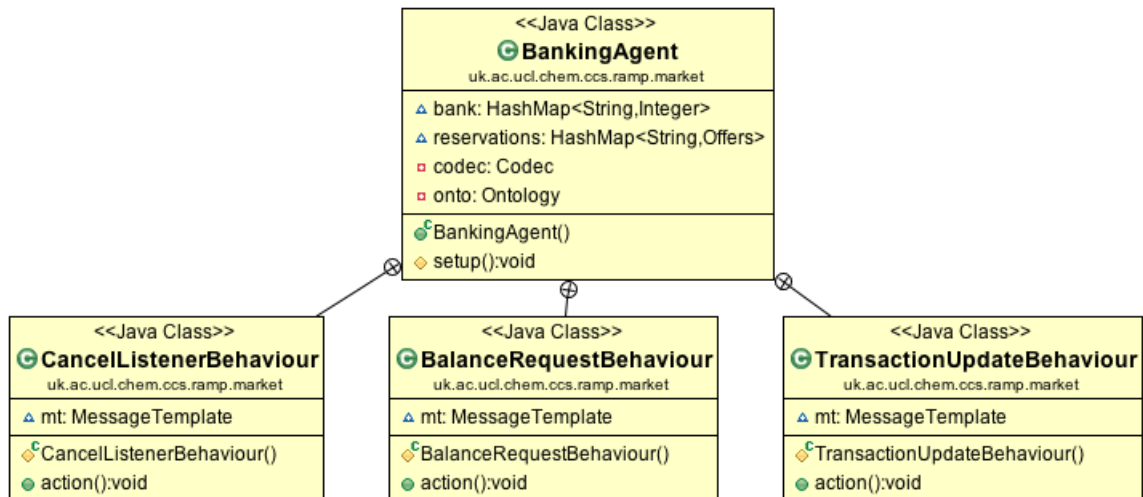


Figure 6.7: The hierarchy of behaviours implemented by the Bank agent.

6.3.3 Banking Agent

The banking agent is tasked with recording transactions between users and resources, and can provide an overview of the overall system of resource trading. To do this it implements three behaviours, shown in figure 6.7 and described below:

- `TransactionUpdateBehaviour`: This behaviour listens for and processes update messages from the Resource agents on the completion of successful transactions, to update the internal balances of Resource and User agent.
- `CancelListenerBehaviour`: This behaviour listens for and processes cancellation messages, and notifies both User and Resource agent when the cancellation is complete.
- `BalanceRequestBehaviour`: This simple behaviour can provide a user or resource with a statement of their balance on receipt of a digitally signed request.

In order to verify the digital signatures appended to transaction update messages, the Banking agent maintains a record of the public key of participants in the market place. This is done when accounts are credited by the Banking agent administrator. Within the RAMP system, a standard virtual currency is used.

6.3.3.1 A Note on Banking

The Banking agent is designed and implemented as a technical way to keep track of deals made between users and resources, but it is not intended to answer the policy

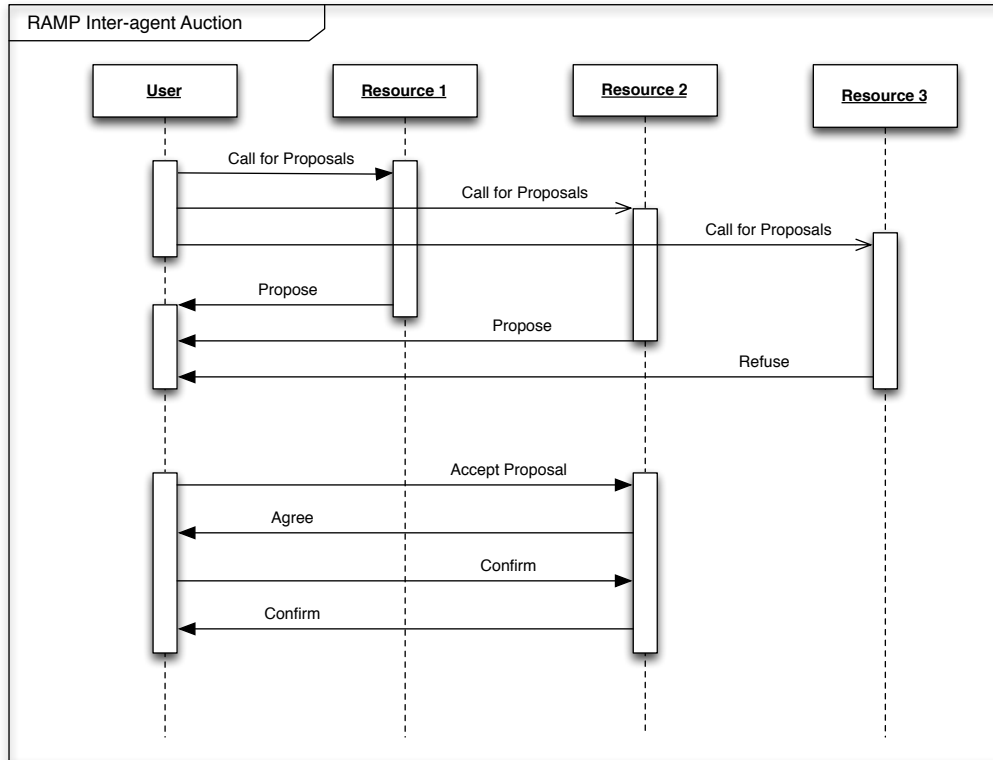


Figure 6.8: The sequence of messages between a user agent and three resources required to implement a single round of bidding for a single unit auction. Inter-agent communication is performed using messages specified in the FIPA protocol. In this example, resource agents 1 and 2 respond to the RFQ with offers to accept the workload, while agent 3 refuses. The user agent accepts the offer from agent 2 using a two phase commit.

questions that address how usage of resources is reconciled with real-world cash payments, which is outside the scope of this thesis. Some of those policy questions relate to how the currency used in the system converts to real world currencies, how deals are enforced, and how overdrafts can be deal with, and provide a rich and interesting vein of future work. Within a production system, we envisage that one or more Banking agents will be run by an independent, trusted third-party.

6.3.4 Inter-agent Communication

Peer to peer (P2P) systems are recognized as a way of building large, scalable distributed systems. Like distributed e-infrastructure systems, they have evolved as a way to share resources across administrative domains, but do so from a very different starting point and with very different requirements, in terms of security and availability [237]. A key feature of the RAMP system compared to other resource brokering/meta-scheduling systems is that there is no central service in overall control of the system.

In effect, the resource agent and the user agent are peers in a peer to peer system, and connected together by a P2P network infrastructure. This means that RAMP can leverage many of the benefits of P2P systems such as dynamic participation (which may encourage more resource owners to devote some or all of their resource to grid work when the utilization falls below a certain level).

Resilience is a key requirement of the RAMP system. Many distributed scheduling systems rely on a single broker component, which results in a single point of failure which can render the whole system unusable. The JADE development environment allows agents to be distributed across a network of machines and incorporates P2P network features to boost system stability and resilience. The key JADE feature utilized by RAMP is the *Main Replication Service*. All JADE agents run within a *container* which provides basic agent communication and management capabilities. JADE requires a *Main container* to act as the control point for the distributed agent system. The Main Replication Service allows the Main container functionality to be replicated amongst a ring of containers, to which normal containers connect. In the RAMP system, each Resource Agent runs in a replicated version of the system's Main container. User Agents run in normal containers; a User Agent's container can connect directly to any Resource Agent's Main container and, via container replication, have access to all Resource Agents in the system.

In addition, when a new Resource Agent connects to the system, it needs only connect to one of the Main containers to join the whole market place. The *JADE Address Notification Service* runs within all the containers in the system. It monitors agents and therefore containers entering and leaving the system, and reconfigures the network accordingly. In this way, if one of the Main containers crashes or otherwise exits the system, all of the User Agent containers connected to that Main container are reconfigured to automatically connect to another Main container, and the connections in the Main container ring are suitably adjusted.

6.3.4.1 Agent communication ontology

The FIPA Agent Communication Language terms discussed in Section 6.2.1 define the basic semantics of how agents interact in a FIPA compatible multi-agent system. However, these basic actions do not cover the full, rich lexicon which agents need to

possess in order to implement the negotiation protocols described earlier. Within FIPA compliant agents, ontologies are used to represent the set of concepts and symbols that agents need to communicate about. This standard method of inter-agent ‘language’ based on a well defined ontology allows agents implements using different software environments to be mutually intelligible.

In the RAMP system, agents primarily need to communicate about Requests for Quotations. Therefore we have developed an ontology that allows agents to communicated based around the terms of the RFQL syntax described above. Within JADE, this is realized as a series of Java classes that extend the `jade.content.onto.Ontology` class, with each class corresponding to different ontological terms. The key terms in this ontology are depicted in figure 6.9, and the full specification of the ontology is listed, in OWL format, in Appendix B.

6.4 System Integration

As stated previously, AHE and RAMP are independent systems, but are designed to closely interoperate. AHE provides a persistent job launching and execution management service. RAMP provides a market place in which compute cycles can be traded between users and resources. A diagram of the architecture of the system, and relationship between AHE and RAMP, is shown in figure 6.11.

The AHE is pre-configured with details of the applications which the user can run, and the static set of resources on which the applications are installed. This is in contrast to the Condor approach, which stages application binaries to resources before they are run. The parallel MPI applications which the AHE was designed to run are often difficult to compile and need an expert user or system administrator to optimize them for a particular machine architecture, which makes binary staging very difficult.

The sequence of operations required to launch an application using AHE and RAMP is illustrated in figure 6.10. When submitting an application via the RAMP, the user can initiate the submission via the AHE. A list of the candidate resources on which the application that the user wishes to run is installed is retrieved from the AHE server. This lists the subset of e-infrastructure resources with which the user agent needs to conduct the bidding process. The AHE also maintains a list of static resource characteristics, such as CPU architecture, which are used to match the static constraints

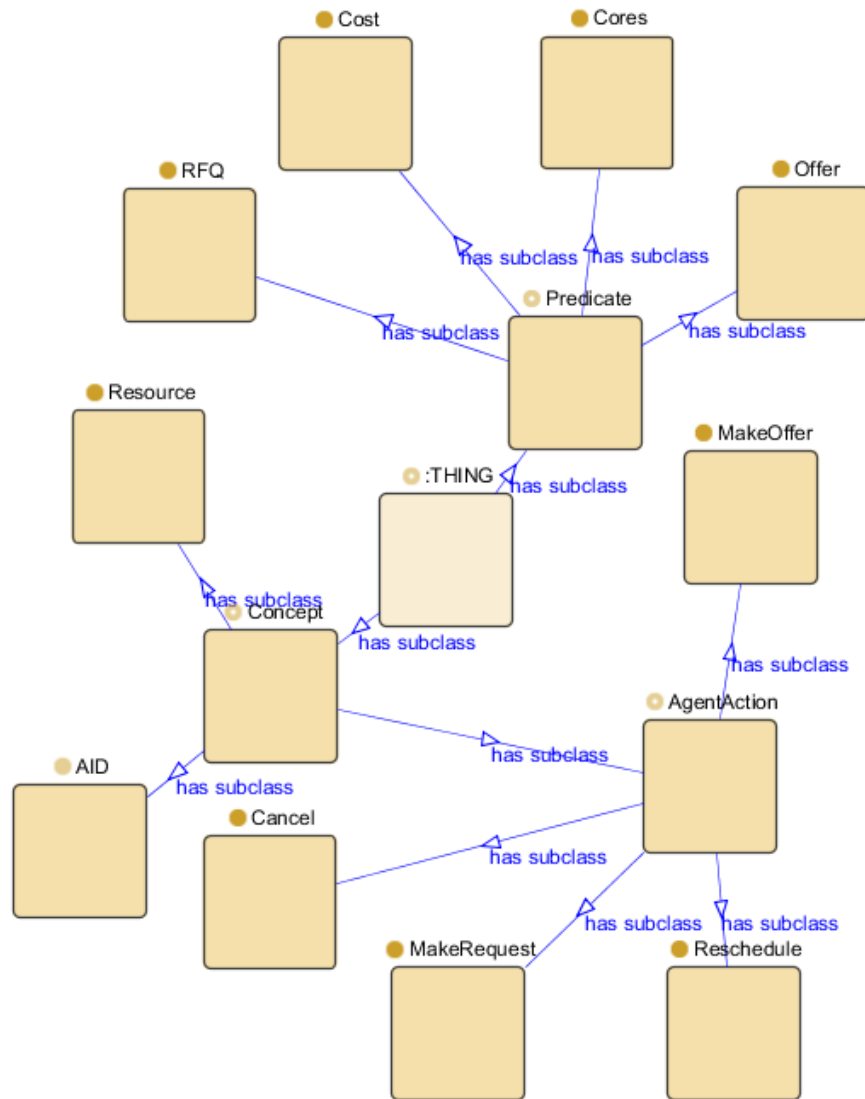


Figure 6.9: A schematic representation of the relationship between concepts in the inter-agent communication ontology.

the user places on the job.

Once a bid from a resource has been accepted by the user agent, the AHE server submits a JSDL job description to the resource, describing the job that is to be run and the location of any input data along with the reservation ID(s) obtained by the RAMP system. The AHE then monitors the progress of the application as it run, and the AHE client can be used to retrieve any output data created.

Since AHE launches applications on a range of different back end middlewares, and can be used to federate resources from administratively distinct grids, the RAMP system can also schedule across federated e-infrastructure resources. However, unlike other meta-scheduling approaches, each of the resource providers does not need to

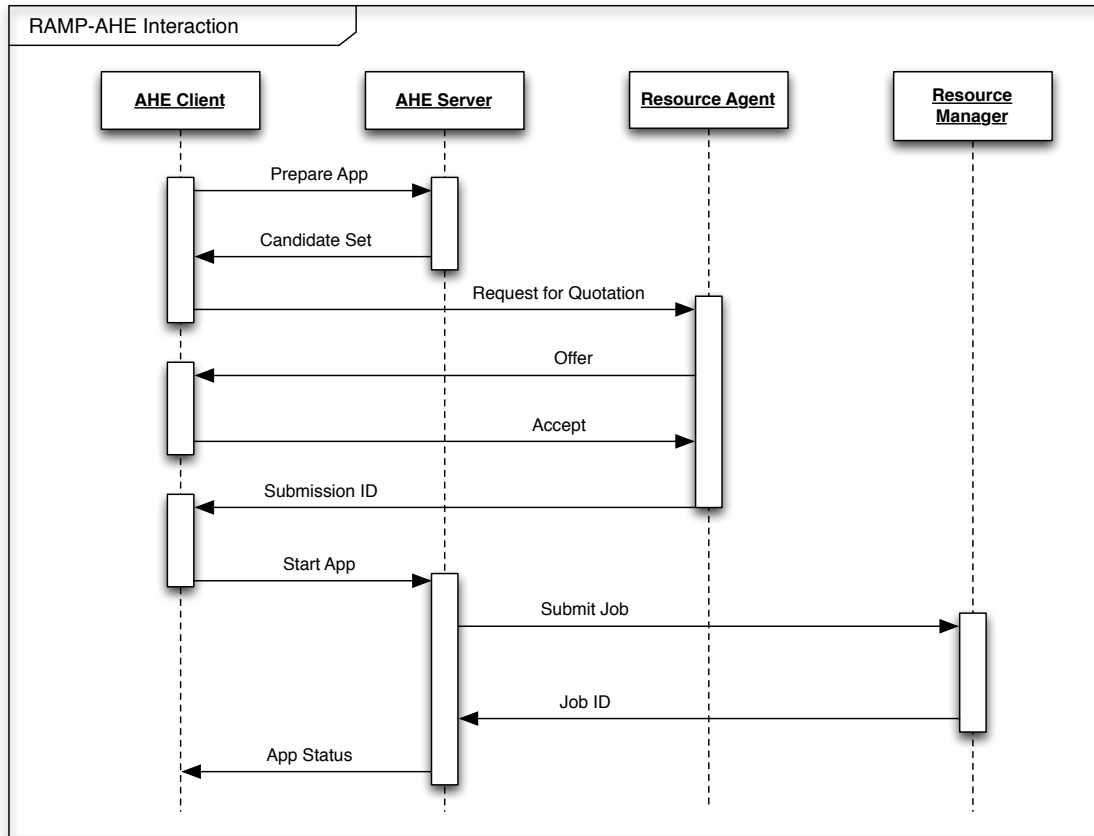


Figure 6.10: The sequence of operations between the AHE client (containing the user agent), the AHE server, and a single resource, required to launch an application on that resource.

subscribe to a centralized scheduling service. They simply need to run the resource management agent on their system.

6.5 Summary

In this section we have reviewed some common grid use cases, and from them derived user requirements that, if satisfied, will greatly improve grid usability. We have described two services, AHE and RAMP, which aim to satisfy these requirements, through elevating the application to a first-class resource with which the user interacts, and developing a computational marketplace, through a combinatorial, multi-attribute reverse auction mechanism which allows users to place jobs on resources based on cost and time to solution minimization. In the next Chapter we put the RAMP system to the test and draw conclusions about its potential utility in a distributed e-infrastructure environment.

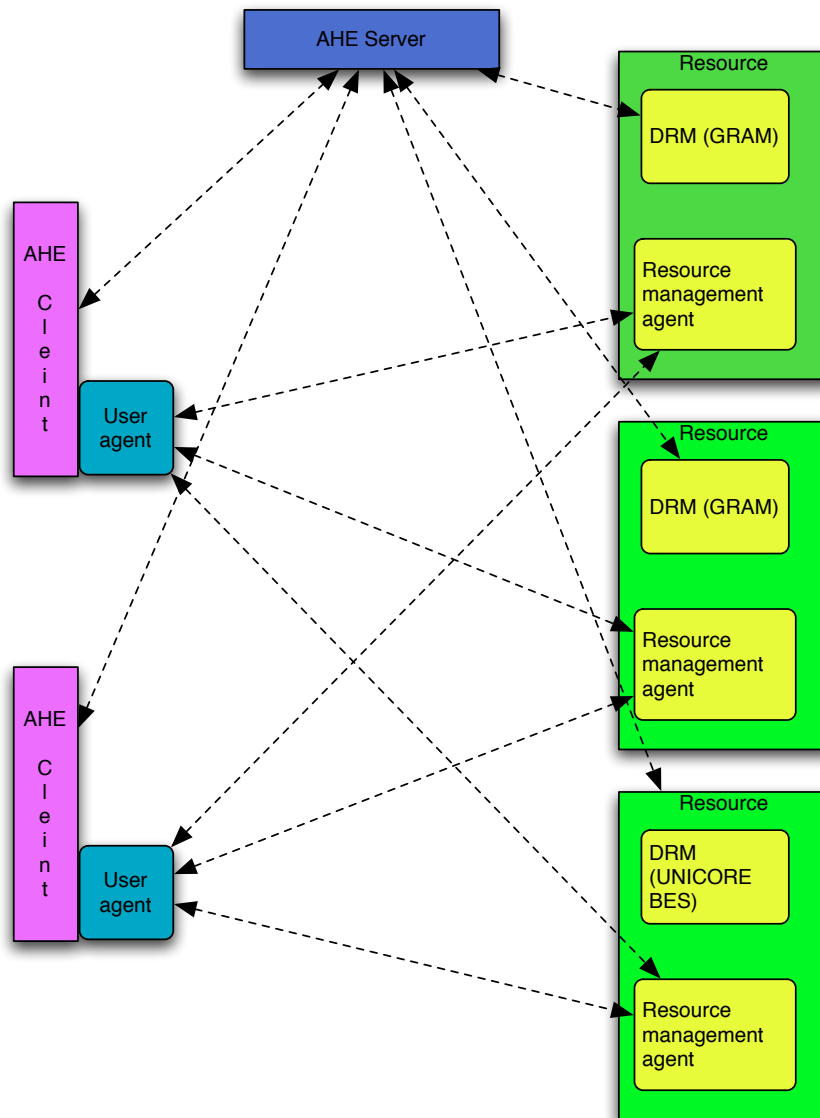


Figure 6.11: Interaction between RAMP system and AHE to schedule application execution on distributed e-infrastructure resources.

Chapter 7

Functional and Performance Testing of the RAMP System

In this section, we evaluate the capabilities of the RAMP platform described in the previous chapter using a simulation environment that we developed to approximate a large scale distributed e-infrastructure environment. We present our finding and draw conclusions on the utility of our system.

7.1 Evaluating the System

The RAMP system is designed to manage resource allocation across a set of high performance compute resources. Deploying the system across such a set of resources in order to evaluate the performance and capabilities is impractical, since root access could be required to install tools that interact with the queuing system, machines could be taken down for maintenance periods and so on. Therefore, we found it practical to develop a simulation environment which would allow us to evaluate RAMP without the external difficulties inherent in using a production HPC e-infrastructure.

In order to perform a realistic evaluation of the RAMP system, our simulation uses

Machine	Cores	Log Start Date	Log End Date	Log Duration (Months)
LLNL Atlas	9216	Nov 2006	Jun 2007	8
LLNL Thunder	4008	Jan 2007	Jun 2007	5
ANL Intrepid	163840	Jan 2009	Sept 2009	8
RICC	8192	May 2010	Sept 2010	5
CEA CURIE	93312	Feb 2011	Oct 2012	20

Table 7.1: Parallel Workload Archive Project log files used in the simulation environment.

historical resource usage data obtained from a number of high performance computing systems, collected and made available by the Parallel Workload Archive Project¹. Archived logs are converted to the Standard Workload Format (SWF) [238] and in some cases cleaned of erroneous data.

The logs chosen to base simulations on are summarized in table 7.1. As can be seen from the table, all of the logs from the archive cover several months of continuous operation. This means that with a relatively few logs we can create a diverse simulated ecosystem with different Resource Agents within the simulated system starting from different time points within the same file. The logs used were chosen to represent a diverse heterogeneous e-infrastructure, with both large petascale machines and smaller clusters.

The simulation environment consists of a resource plug-in for the Resource Agent which allows it to interact with an historic usage log as if it were a live queuing system on a production HPC resource. This in turn is done through two scripts:

- `fake_qstat.pl`: This script mimics to some extent the behaviour of the `qstat` command. `Qstat`, or a differently named variant, is a familiar tool on many HPC systems that allows the user to investigate the status of the queuing system. The script takes as its parameters the path to a configuration file along with the number of CPUs required by the user and the time the job must run. The configuration file lists the full path to the log file to be read, the time offset where the log should be read from, and also the system start time. This last value allows the simulated system to evolve over time. The system start time represent the beginning of the resource log file. Using the difference between the system start time and the current time (plus the time offset), the script can provide a snapshot of the system state at a given time. This snapshot considers the currently running and queued jobs at the given time to calculate whether sufficient cores will be available for the requested job to start at the time specified by the user. If the job can be run, the script returns the percentage load on that machine at the time the job is to be run.
- `fake_qrstat.pl`: This script is almost exactly the same as the

¹<http://www.cs.huji.ac.il/labs/parallel/workload/>

System name	Base system	Time point (seconds)	Start Price	Min Price
atlas1	LLNL Atlas	3370000	33	25
atlas2	LLNL Atlas	1370000	33	26
thunder1	LLNL Thunder	250000	70	40
thunder2	LLNL Thunder	1300000	75	60
thunder3	LLNL Thunder	130000	70	35
thunder4	LLNL Thunder	450000	75	50
intrepid1	ANL Intrepid	50000	55	35
intrepid2	ANL Intrepid	1500000	65	25
intrepid3	ANL Intrepid	15000000	53	25
intrepid4	ANL Intrepid	750000	55	30
intrepid5	ANL Intrepid	2500000	65	28
intrepid6	ANL Intrepid	90000	53	30
ricc1	RICC	50000	40	25
ricc2	RICC	7570000	45	25
ricc3	RICC	500000	45	25
ricc4	RICC	757000	45	30
curie1	CEA CURIE	150000	80	40
curie2	CEA CURIE	1375000	80	65
curie3	CEA CURIE	350000	80	30
curie4	CEA CURIE	2375000	70	65

Table 7.2: Simulation environment machine setup. The time point is the number of seconds within the log at which the test system started. All systems were started at a minimum of 50000 seconds into the machine log file in order to allow the load on the machine to reach a production level (some machine logs commence with the machine being turned on, meaning that initially the queue is empty).

`fake_qstat.pl` script, but instead of returning a utilization percentage it returns a fake reservation ID if the job can be satisfied, and also inserts the job's details into the machine log.

With these two scripts, our Resource Agent plug-in and our historic queue data, we can investigate various aspects of the RAMP system in a way that closely approximates a real deployment.

7.2 Simulation Environment Setup

To evaluate the pure performance of the RAMP system in these initial investigations we deployed all the agents within our simulation environment on a single high power Ubuntu Linux workstation, to eliminate performance problems that could be introduced by network bandwidth limitations between machines. We took initial configurations

based on the machine log files listed in table 7.1, and working from different time points within those logs, configured a experimental system made up of twenty resources.

The configuration of those resources is shown in table 7.2, with a single Resource Agent within the system representing each system. The Resource Agents within the simulation environment logged all transactions for further analysis. The system was left to evolve in real time as the experiments were performed.

The static resource parameters for each Resource Agent were all assigned the same default configuration. We did this so that decisions made in the investigations we performed were governed by machine load and price, rather than static constraints.

7.3 Using the RAMP System

The first tests we performed were designed to assess the RAMP system's ability to successfully schedule a range of heterogeneous workloads. Specifically, we investigated the following two aspects of the RAMP system:

- **Investigation 1:** Initially we want to confirm that the RAMP system works, that jobs can be place with resources at prices favourable to both the user and the resource, and that a majority of jobs submitted will be successful.
- **Investigation 2:** Secondly, we wish to assess how efficiently jobs are placed within the system.

In order to perform these investigations, we submitted a range of different jobs, in terms of system requirements, core counts and deadlines, which are listed in table 7.3, to our RAMP simulation environment. This range of jobs represents the typical tasks a HPC machine will be put to, from small scale, short running jobs to large, long running capability workloads.

Each job listed in table 7.3 was run consecutively, using a separate instance of the User Agent. Each run of all thirteen jobs was repeated three times in order better control for temporal anomalies within the system. In each case the User Agent was configured to conduct three rounds of bidding.

7.3.1 Results

From the resulting logs generated by the User Agents and Resource Agents, we computed mean prices for each of the bidding rounds and also the winning price. These

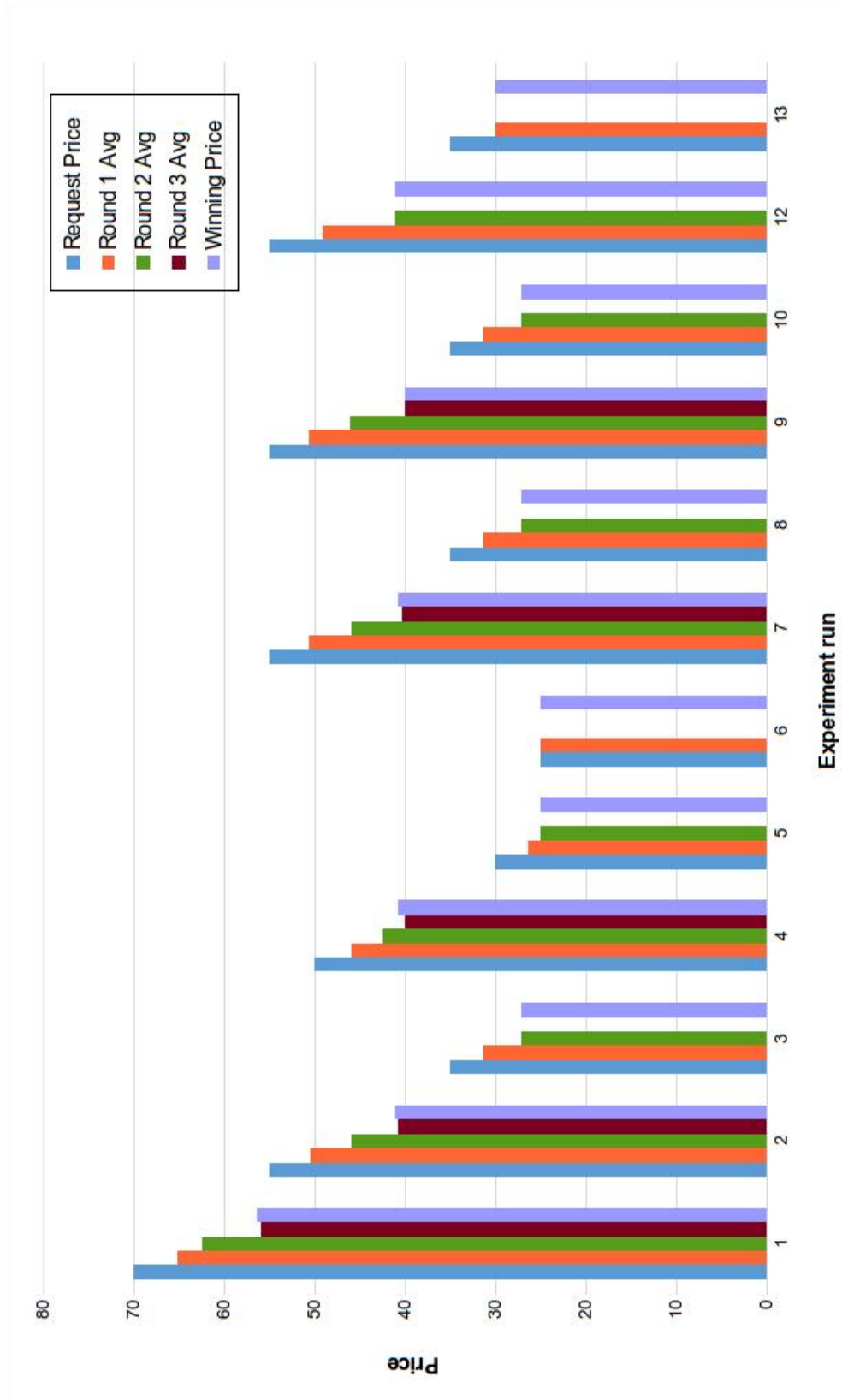


Figure 7.1: Request, offer and winning prices for the requests shown in table 7.3 for three rounds of bidding. Where bidding round values are not shown, no offers were made in that round. Experiment run 11, which failed, is not shown.

Experiment	Cores	Start time	Price
exp1	16	5 min	70
exp2	16	60 min	55
exp3	16	12 hours	35
exp4	256	5 mins	50
exp5	256	60 min	30
exp6	256	12 hours	25
exp7	1024	60 min	55
exp8	1024	12 hours	35
exp9	4096	60 min	55
exp10	4096	12 hours	35
exp11	20480	5 min	80
exp12	20480	60 min	55
exp13	20480	12 hours	35

Table 7.3: Details of experimental workloads run on the RAMP simulation environment.

Experiment	Request Price	Round 1 Avg	Round 2 Avg	Round 3 Avg	Winning Price
1	70	64.78	57.29	54.60	47
2	55	52.17	43.54	39.56	39
3	35	30.61	25.00		25
4	50	47.00	39.00	34.60	34
5	30	28.11	25.00		25
6	25	25.00			25
7	55	51.44	42.94	39.17	39
8	35	30.22	25.00		25
9	55	51.50	43.00	39.25	39
10	35	31.17	26.17	25.00	25
11	80				FAIL
12	55	50.75	42.50	39.50	39
13	35	29.00	25.33		25

Table 7.4: Mean round and winning prices for the auctions of the workloads listed in table 7.3.

values are shown for each of the jobs in table 7.4, along with the request price the user opened the bidding at. Figure 7.1 displays these results as a bar chart, showing how the price offered by each resource fell with each round of bidding.

Where bidding rounds show no result, either the price of the job had fallen below the minimum price threshold of a resource in the system, or the system load on has increased on the resource, meaning that no resource is willing to continue bidding. We found that the median offer price was 71.1% of the request price.

In order to investigate the efficiency of our system mapping jobs onto resources,

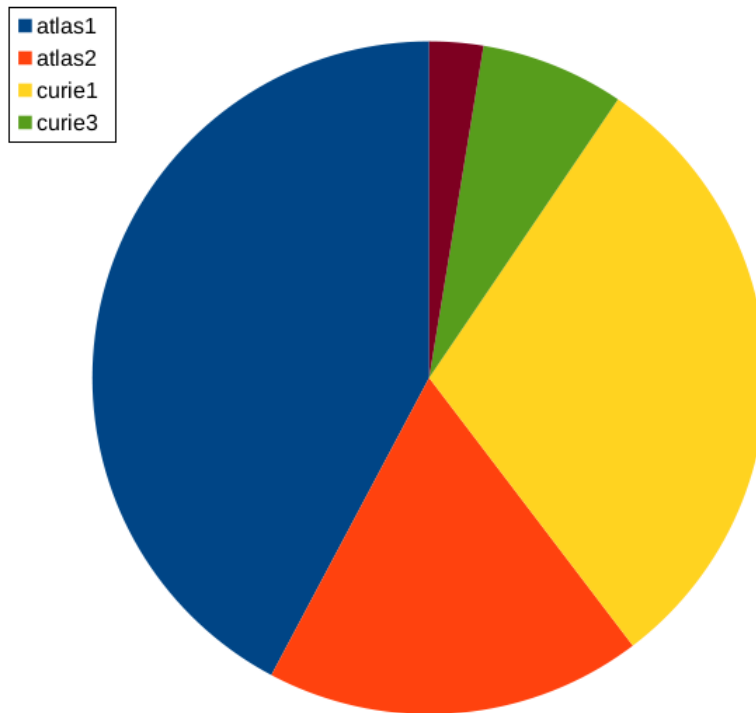


Figure 7.2: The winning resources for the auctions of workloads listed in table 7.3. Five resources were allocated all of the jobs submitted.

we also examined which of the resources in the system won each of the auctions outlined in table 7.3. Five of the resources won all of the jobs submitted in three experimental repetitions, with the share of the jobs distributed as shown in figure 7.2.

To assess whether these resources were allocated to the most appropriate resource, and thus the overall efficiency of the system, we need to consider just more than just the load on each system. Allocation of workloads to resources within the competitive RAMP market place, is based on a function combining both the load on the machine and the price it is willing to offer to get jobs, as outlined in Section 6.3.2.3.

Therefore, to assess whether work is allocated to resources efficiently, we need to consider the *attractiveness* of the resource to the users of the system. A useful measure of the attractiveness of the resource is the price by which it is willing to reduce its offer price while bidding, since this is based on the start and minimum price configuration of the machine, and its load.

In figure 7.3 we plot the attractiveness of the resources in the system over the evolution of the simulation environment. Resources not included in the plot were loaded to such an extent that they did not make offers.

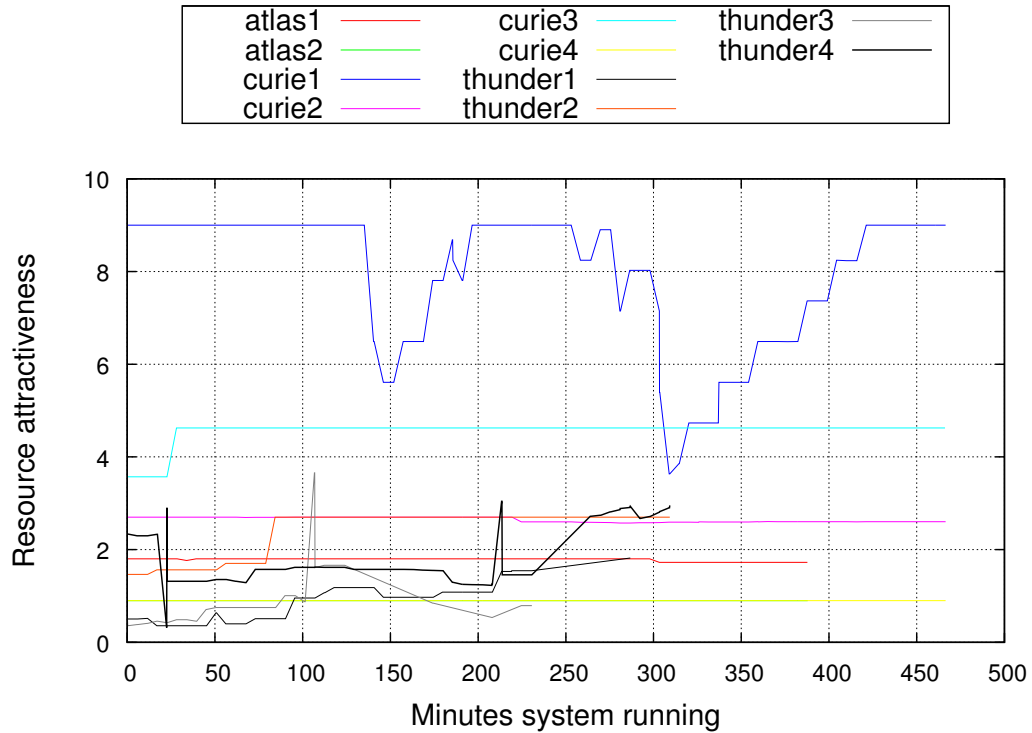


Figure 7.3: Resource *attractiveness* plotted over time. Attractiveness is analogous to the value by which a resource is willing to reduce its prices.

7.3.2 Discussion of Results

The results plotted in figure 7.1 and shown in table 7.4 confirm that our system is capable of successfully allocating jobs to resources at prices lower than the user is willing to pay. Although the majority of requests were allocated during our experimental runs, it is likely that this will not always be the case, especially where the user sets their opening price below the minimum price of all resources in the system, or where resource load across the platform is sufficient that large jobs that need to start very soon cannot be accommodated. The failure of experiment run 13 shows us that very large requests that need to be run soon will likely fail, even when the user is willing to pay a premium to execute the workload.

We found that resource request prices that were very close to the minimum system prices offered by the majority of resources resulted in fewer rounds of bidding, but achieved better prices for the user.

In a production environment, a user would not necessarily know the minimum prices set by the resource providers but, with experience, may well come to learn reasonable estimates of the minimum price various resources were willing to offer, and

would likely be able to price their workloads accordingly.

Our assessment of the efficiency of the system shows that the jobs are allocated to the most attractive resources. As shown in figure 7.2, all of the test workload requests submitted to the system were allocated to just five of the available twenty resources (with the exception of the request that failed). Looking at figure 7.3 we see that in general the most attractive resources won the resource requests made.

Though the curie1 and curie3 resources remained the most attractive over the simulation execution duration, they did not take all of the jobs, and were not even the biggest winner, which was the atlas1 machine. This can be explained by the fact that temporal changes within the attractiveness of resources mean that at different points, one resource will be more attractive than others. Also, the User Agent accepts offers on a first come first serve basis, so that if two resources make the same offer, the offer received first by the Resource Agent will be favoured. This means that a single resource with a low load and favourable pricing structure does not completely dominate the platform and take all requests made.

7.4 Investigating Job Pricing

The Resource Agent provides a bespoke spot price for the resource it is managing, in response to requests from User Agents. As such, the price offered by a resource fluctuates over time. Figure 7.4 shows how the price for various resources varies over time, while the simulation environment is running.

The prices agreed for resource requests are governed by a combination of parameters under the control of both the Resource Agent and the User Agent. The Resource Agent is configured with a minimum price and a start price, which are used, along with the load on the machine, to generate a bid reduction price. Choosing start and minimum prices is therefore a task which the machine administrators need to invest some effort in.

The key is to set start and minimum prices that will actually result in bids that, on average, meet the cost that the resource owners want to sell their cycles for. Of course, the minimum price provides a lower bound; when auctions fall below this minimum, the resource will refuse to participate further in the auction.

In addition to the configuration parameters set for the Resource Agent, the price is

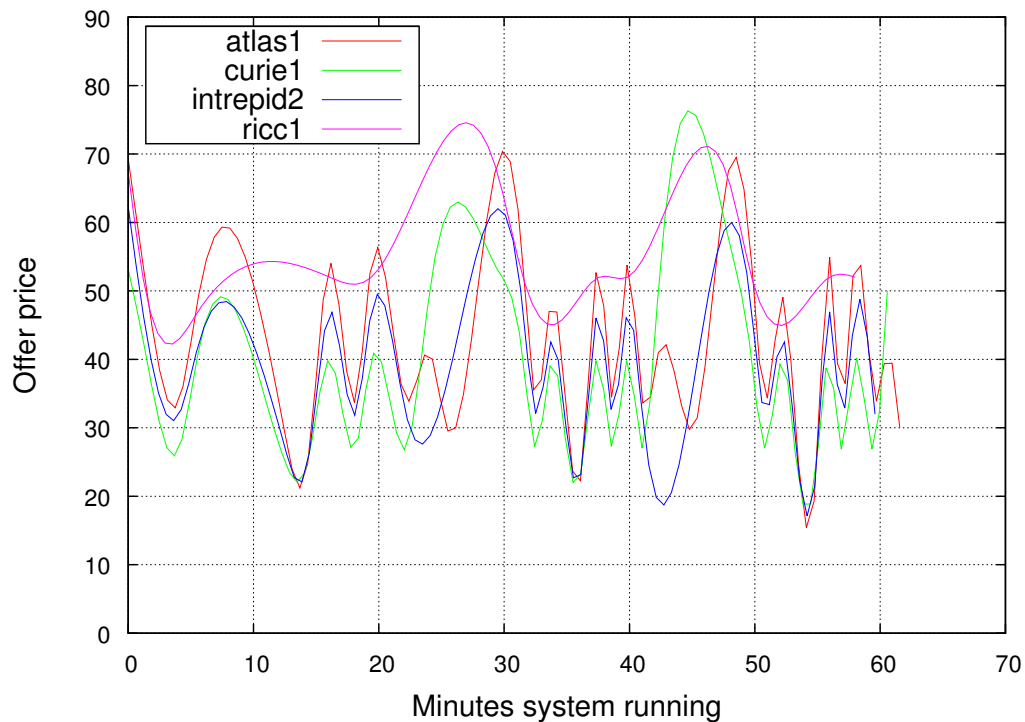


Figure 7.4: The change in resource offer prices over time for four resources in the system.

also governed by how many rounds of bidding the User Agent specifies. To optimize these parameters we performed the following investigations:

- **Investigation 1:** We calculated the average offer price made by a resource over a period of system operation, and compared it to the start and minimum price used to configure the resource.
- **Investigation 2:** We performed an experiment to discover the optimum number of bidding rounds required to minimize the price paid by the user.

The resource offer price is also governed by the price that other resources offer in the system (which is outside the control of any given resource administrator) and the prices that users of the system are willing to pay for the workloads they need to execute.

We analysed the results for the workloads presented in Section 7.3 to discover the mean offer price made by each resource in the system. In figure 7.5 we plot this mean price alongside the start and minimum prices.

To determine the optimum number of bidding rounds required to minimize the price paid by the user, we use the environment simulation described in Section 7.2 to make a user requisition for a single auction unit. The request was run ten times, with

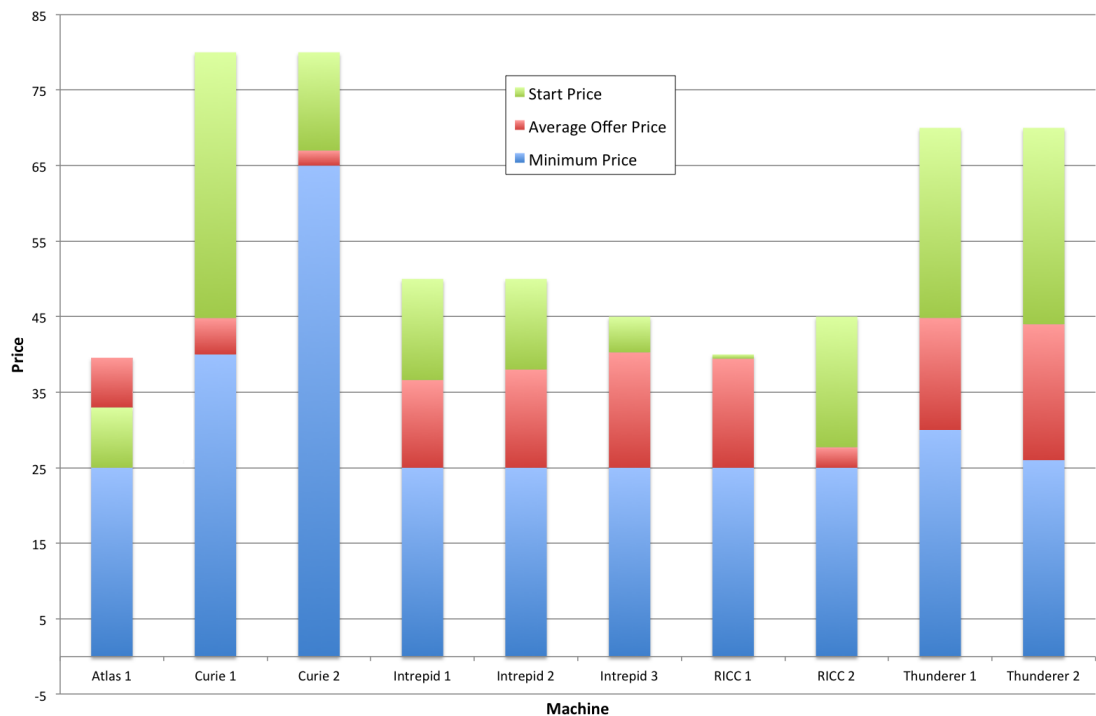


Figure 7.5: Comparison of the average offer price made by each resource to its minimum price and start price.

Number of Bidding Rounds	Mean Sale Price	Mean Duration (sec)
1	66.00	18.81
2	60.00	31.91
3	51.83	46.19
4	41.83	61.15
5	37.40	76.18
6	35.40	91.18
7	37.83	106.21
8	37.83	121.24
9	34.75	136.26
10	36.50	151.26

Table 7.5: Offer price and auction duration as the number of bidding rounds increases.

each run executed with an increasing number of auction rounds, from one round to ten rounds. We repeated each experiment run three times, and calculated mean values for the sale price and duration of the auction, shown in table 7.5.

In figure 7.6 we plot the mean sale price and the mean auction duration the for auctions with one to ten rounds of bidding.

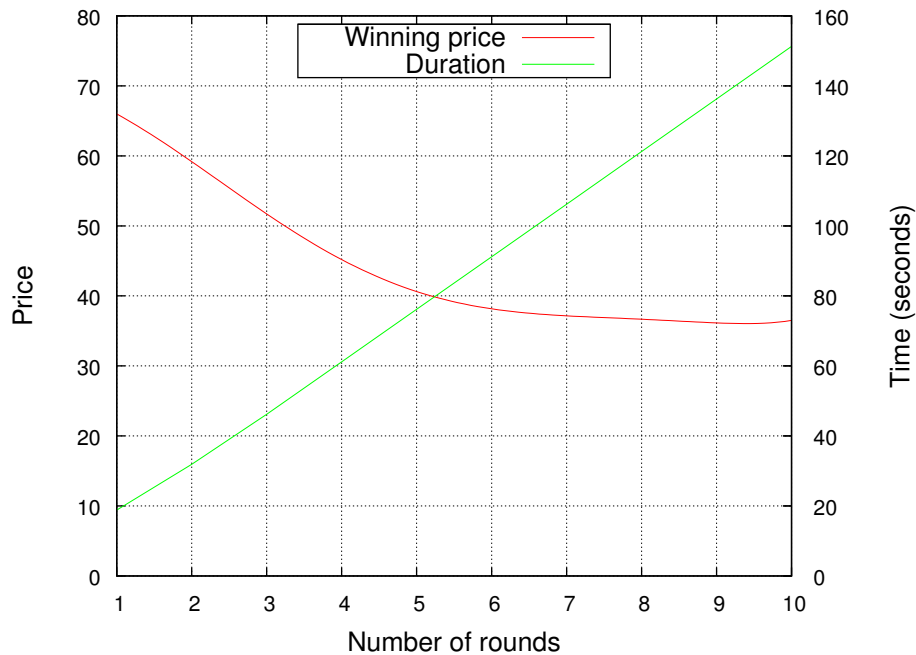


Figure 7.6: Plot showing how the increase in number of bidding rounds affects final cost price and auction completion time.

7.4.1 Discussion of Results

Selecting optimal configuration parameters for a resource is a complex task. As figure 7.5 shows, offers made by a resource will roughly fall between the start and minimum prices set of the resource. However, the minimum price is seemingly the most important parameter; where the mean price is set at a level comparable to other resources in the system, the mean offer price will usually be comparable to those other resources too, and somewhat higher than the minimum price. However, setting a high starting price will increase the attractiveness of the resource by increasing the amount which the resource is willing to reduce its offers by while bidding. In summary, to avoid being outbid and to increase the chance of auction success, a resource owner should try to set a minimum price around the same level to other resources in the system, but a high starting price.

As we see from figure 7.6 and table 7.5, as the number of auction rounds used by the User Agent increases, the final offer price accepted is reduced, with a tail off at five rounds, suggesting that users wanting to optimize the price they pay for auction units could do so by running auctions with five bidding rounds. However, as the number of auction rounds increases, so does the time taken to complete the auction. This scales

linearly with the number of auction units, as is to be expected since auction rounds are of a fixed duration. Users must take into account this trade-off when initiating multi-round auctions.

7.5 Investigating RAMP Performance

Our multi-unit auction system is, we believe, unique, but it is also important that it is usable. A key aspect of usability is the responsiveness of the system. It is important that the RAMP system responds well to user requests, and scales both with the number of units an individual is requesting (in a multi-unit auction) and with the number of simultaneous users of the system. To assess this performance, we conducted two investigations into system scalability:

- **Investigation 1:** We measured the performance of the system in terms of auction duration as the number of request units within a combinatorial reverse auction increase.
- **Investigation 2:** We measured the performance of the system in terms of auction duration and average system response time as the number of User Agents participating in the system increases.
- **Investigation 3:** To assess the impact of network performance on a the responsiveness of a real-world deployment of RAMP, we repeated investigation 2 with our Resource Agents deployed across a network of machines.

7.5.1 Results

Using the simulation environment outlined in table 7.2 we sequentially submitted requests via a single Resource Agent in order to investigate how performance increases with the number of auction units. With each submission the number of units within the request increased, meaning the terms within the combinatorial reverse auction increased.

The simulation environment was run on a single workstation to eliminate disruptions caused by network problems, with a separate Resource Agent for each simulated resource. Runs were performed for auctions with 1 to 25 units (the deadline and price

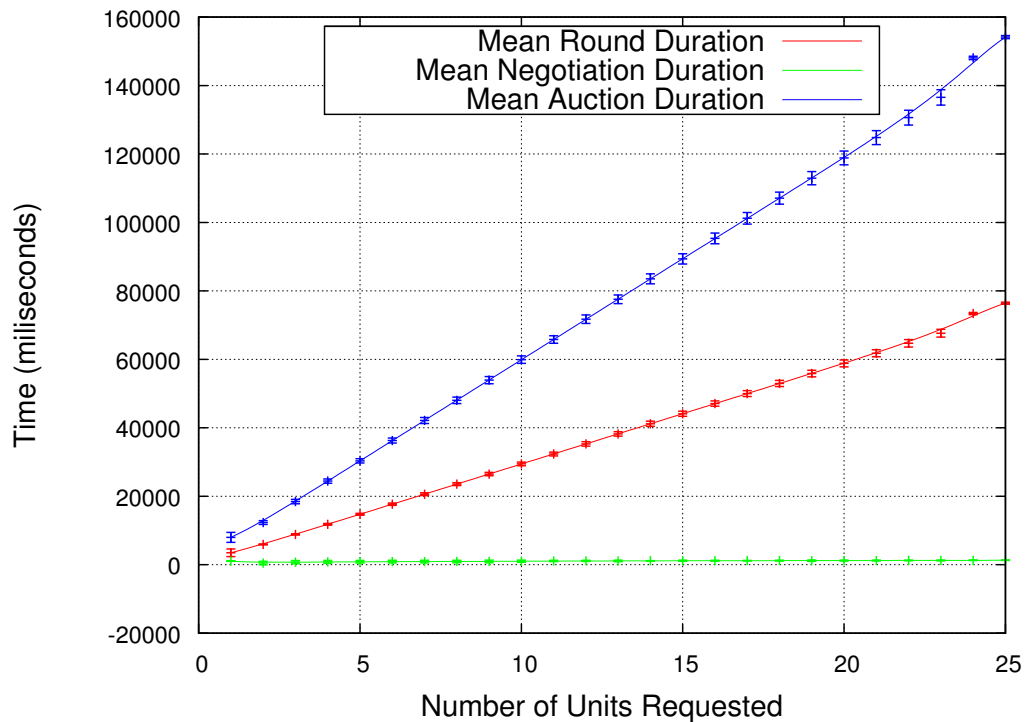


Figure 7.7: Plot showing how the increase in units per auction affect system performance. As the number of units increase, the time taken to complete the auction scales linearly. Bars show the standard deviation of the result mean.

of each unit was randomly generated), and the whole set of runs was repeated one hundred times, and mean response times calculated.

The user configurable auction round duration parameter was set to five minutes, so that we could measure round duration without the auction ending. The times taken to complete bidding rounds, negotiate the final auction agreement, and the total time taken for the auction to complete are displayed in figure 7.7.

To assess how the system performs when multiple individual users are using it, we performed an experiment using the simulation environment outlined above, whereby we ran multiple User Agents simultaneously, each making a single unit request. We ran from 1 to 30 User Agents consecutively, and repeated each run three times then calculated mean response times. Again, the User Agents were configured with a maximum bidding round duration of five minutes, so that auction rounds would not time out before all Resource Agents had been able to respond.

We measured the mean time taken for a Resource Agent to respond to an individual request (shown in figure 7.8) and how the number of competing agents within the system affects the duration of an auction (shown in figure 7.9).

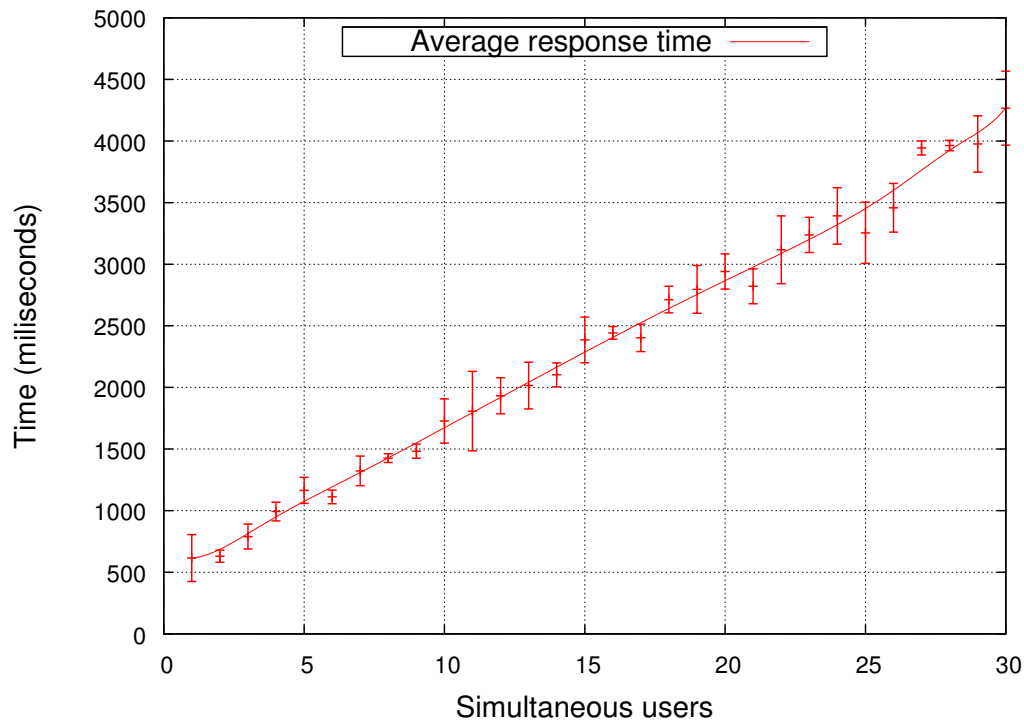


Figure 7.8: Plot showing how the increase in competing agents affects mean system response time. The response time scales linearly with the number of concurrent User Agents. Bars show the standard deviation of the result mean.

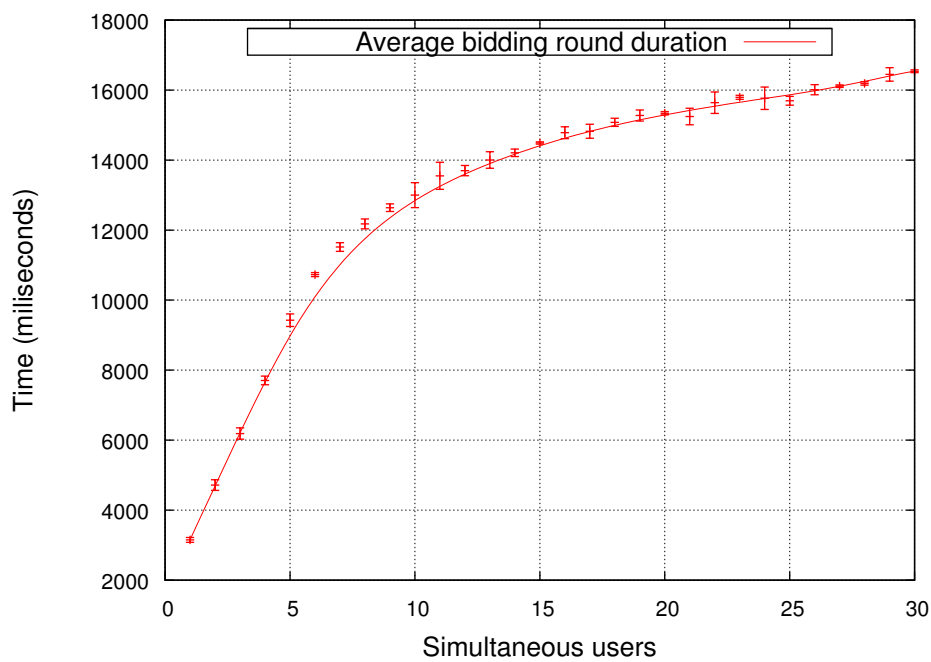


Figure 7.9: Plot showing how the increase in competing agents affects mean auction round duration. The round duration rises sharply up to 10 simultaneous users, then tails off. Bars show the standard deviation of the result mean.

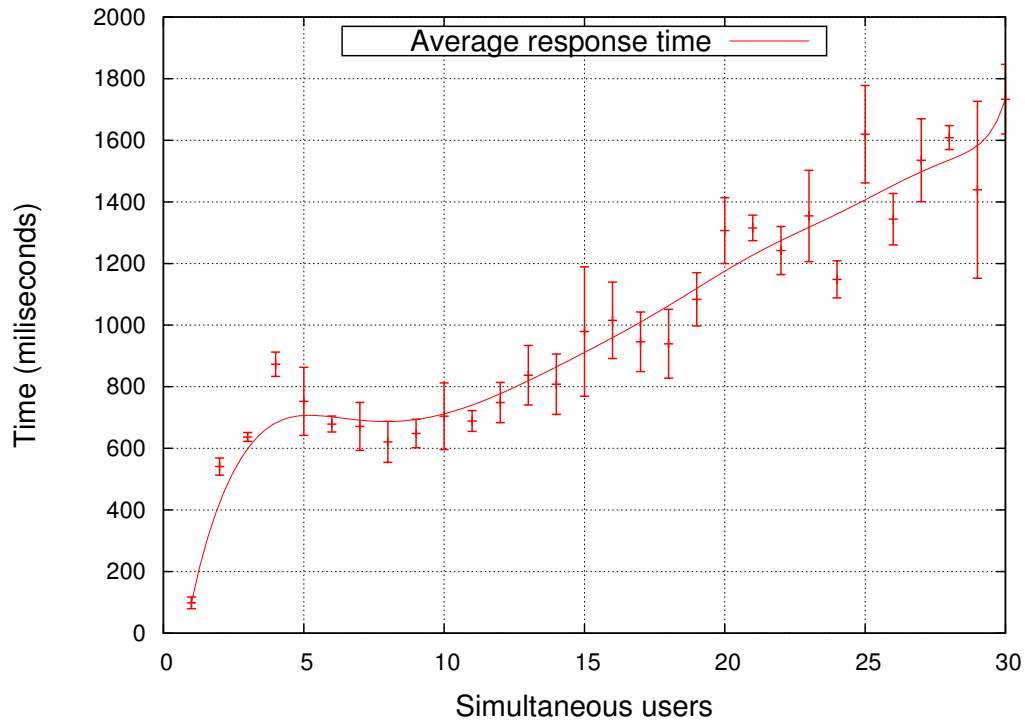


Figure 7.10: Plot showing how the increase in competing agents affects mean system response time with RAMP system deployed across a network of resources. Bars show the standard deviation of the result mean.

In real world applications, the RAMP system is intended to be deployed across a network of HPC class resources. Our performance tests so far have only measured performance with RAMP deployed using our simulation environment on a single machine. To ensure that network effects will not adversely affect the performance of the system, we repeated our tests on network deployment of RAMP.

The system was deployed across 15 networked servers. In a real world deployment we expect that RAMP would be deployed across an Internet wide set of HPC resources. The impracticalities of securing access to such resources in order to carry out our performance tests led to us deploying a system with 10 servers located within the CCS research lab in University College London, with additional Resource Agents deployed at CINECA (Italy), Cyfronet (two agents) and PSNC (both Poland), University of Sheffield (UK). The Resource Agents used the first 15 resource configurations listed in table 7.2.

We repeated the previous investigation, running between 1 and 30 User Agents simultaneously and measuring the impact of doing so on the mean time taken for a Resource Agent to respond to an individual request (shown in figure 7.10) and how the

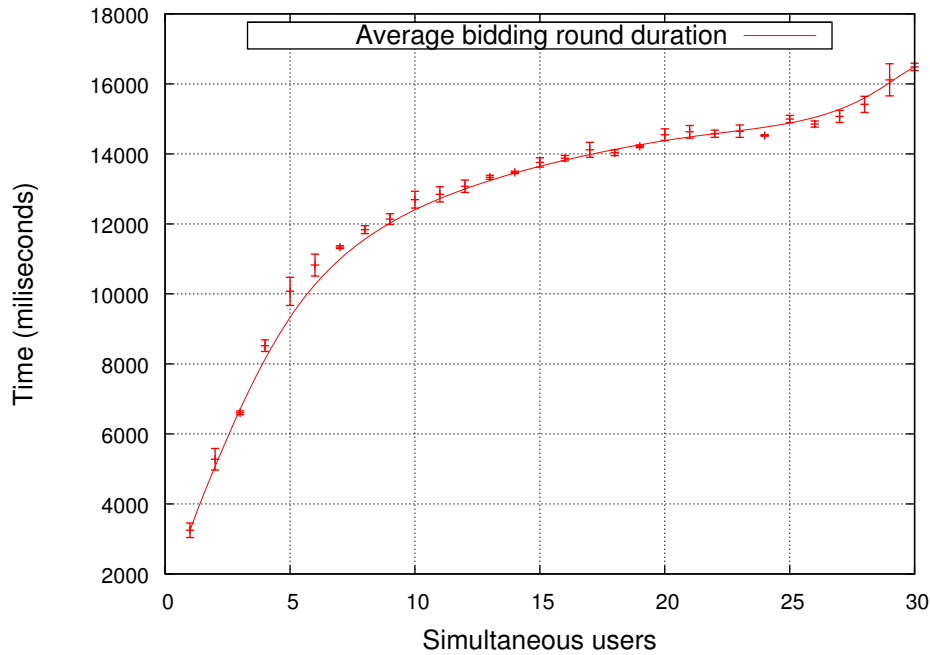


Figure 7.11: Plot showing how the increase in competing agents affects mean auction round duration with RAMP system deployed across a network of resources. Bars show the standard deviation of the result mean.

number of competing agents affects the duration of an auction (shown in figure 7.11).

7.5.2 Discussion of Results

As we see from figure 7.7, the duration of the bidding rounds in an auction scales linearly with the number of units in an auction. This is to be expected, since the duration of the auction is increased by the number of requests the user agent has to make. Surprisingly, the time taken to negotiate the auction does not increase with the number of units. Adding an additional unit only adds a couple of seconds to the overall duration of the auction. so this is unlikely to be of too much concern to the user.

As the number of simultaneous users using the system increases, the responsiveness of the Resource Agents scales linearly, as we see from figure 7.8. However, as shown in figure 7.9, the time taken to complete an auction increases steeply with the first ten simultaneous users of the system, and then tails off as the number of users increases. The tailing off is well below the maximum auction duration we configured in the system, so we are not seeing the effect of this parameter. It is unclear why figure 7.9 is so shaped, and further work is required to understand the observations presented here. On the whole, the results we have obtained show that our system shows good re-

sponsiveness and scalability as both the number of units and number of users increase. When repeated using Resource Agents deployed across a network of hosts (approximating a real-world deployment of RAMP) we found that the effects of the network did not have a significant impact on performance and similar scaling characteristics were obtained, although there was greater variance in the results obtained.

7.6 Summary

In this chapter we have presented our investigations into the performance and capabilities of our RAMP resource allocation platform. We have shown that the system is capable of successfully allocating workloads to computational resources, optimising the price that the user pays and selecting the most attractive resources from the set of available machines. The decentralized nature of the system means that it does this without incurring the overheads and failure points present in a centralized brokering system where a single component is responsible for allocating jobs throughout a grid of machines.

Our system shows good performance and scalability, even when deployed across a wide area network of machines. The ability of users to control the number of and duration of bidding rounds means that they can minimize the price they pay while at the same time placing an upper bound on the time taken to achieve a result. The next step we plan to take is to evaluate our RAMP system formally, by conducting a usability study using real users, on a deployment of RAMP across a production e-infrastructure.

Chapter 8

Overall Conclusions and Future Work

In this chapter we present our overall conclusions, and discuss where we plan to take this work in future.

8.1 Contributions

In this thesis we have presented a comprehensive overview of the field of distributed high performance computing, in order to try to understand the usability problems that are preventing HPC system users from being more creative and ambitious in the way they use resources, and thus push back the boundaries of *in silico* science. We have considered the activities of several ambitious projects which seek to exploit such infrastructures to their fullest potential, and draw conclusions as to how the success of those projects could be more widely replicated, by designing and deploying tools and services to simplify the use of distributed e-infrastructures. As such, we have delivered the contributions outlined in Section 1.4, chiefly the Application Interaction Model, its implementation in the Application Hosting Environment and a decentralized market place for resource allocation.

8.2 Conclusions

HPC resources are a distinct class of system which present their own set of challenges to the user. Alongside the development of distributed HPC e-infrastructures, cloud computing has sought to commoditize access to compute cycles. However, while ideas from the cloud world can assist in addressing distributed HPC usability problems, they are by no means a sufficient solution, due to the inherent overheads introduced by cloud approaches.

Our Application Interaction Model allows complex, multi-component e-infrastructure based applications to be represented as simple Web services, using lightweight client tools. Our usability results have also confirmed the benefit of the Application Interaction Model in that user interaction is reduced to the most essential components: namely a user interacting with his/her application. Users do not need to worry about the details of particular batch queuing systems, or how to stage data back from particular HPC resources; the specifics of how to launch an application are encapsulated within the Application Interaction Model. The approach virtualizes the HPC resources from a user's point of view. Indeed AHE virtualizes the entirety of a grid's HPC resources, and federates resources stemming from multiple different e-infrastructures.

The usability study we reported here focused solely on the 'best case' scenario, where all middleware tools and applications were pre-deployed, and the study participants were used to evaluate and compare the tools' interfaces. In future we plan to extend the study by examining aspects relating to the usability of middleware deployment.

AHE 3.0 provides a number of capabilities not present in earlier versions, including a workflow engine that allows complex simulations to be created, including coupled simulations where data is automatically transferred from one application to another. ACD provides a security suite that includes support for Shibboleth authentication, as well as user auditing. ACD supports virtual organization management and is able to provide access to grid proxy credentials through RESTful web services.

As our performance tests have shown, the redesign of AHE 3.0 has greatly improved performance over older AHE versions. Our usability results have also confirmed the benefit of the Application Interaction Model in that user interaction is reduced to the most essential components: namely a user interacting with his/her application. Users do not need to worry about the details of particular batch queuing systems, or how to stage data back from particular HPC resources; the specifics of how to launch an application are encapsulated within the Application Interaction Model. The approach virtualizes the HPC resources from a user's point of view. Indeed AHE virtualizes the entirety of a grid's HPC resources, and federates resources stemming from multiple different e-infrastructures.

Since its initial release, AHE has been taken up by various user communities. AHE has been used to host a wide variety of computational codes from many different scientific domains, including widely used codes. The key strength of AHE is its flexibility. Since all of the system's complexity resides on the server side, and all of AHE's functionality is exposed as simple services, AHE can be used as a building block for systems of much greater complexity. As such, AHE has been deployed in the infrastructures deployed by a number of international research projects.

A key early aim of grid computing was to make computing power available on demand, by promoting the idea of a resource broker or metascheduler which was able to take a global view of the system, and allocate incoming workloads to the most appropriate resource. However, the capacity to take a global view of the state of a distributed e-infrastructure platform is without doubt of use to the end user, since it obviates the need to manually log into and check the utilisation of the available resources on the system when deciding where to run a job.

While this challenge has been largely met in the high throughput computing world, it has not found much traction in distributed HPC e-infrastructures. This is due to several factors including the access models currently promoted by resource providers, and a lack of suitable infrastructure tools.

Reverse auction based mechanisms are currently finding uptake within the cloud computing world as a means to allocate resources (*cf.* the online service SpotCloud [239] or the work of Roovers *et al.* [240]). This is not surprising, since commercial clouds are based around the idea of paid for access, and the reverse auction mechanism is an elegant means of matching clients to providers in such a system. Our RAMP system is, to the best of our knowledge, the first such system designed to meet the needs of distributed HPC users, and as such includes several novel features including a distributed architecture to increase resilience, the ability to conduct multi-unit auctions in order to schedule workflows and multi-site applications, and the ability for users to control the decision making parameters, including how much they are willing to pay for a job and when they would like it to start and end.

Current HPC allocation practice requires individuals or groups of users to apply for allocations of time on a single HPC resource, or a group of resources made available via a distributed e-infrastructure. Some nominal cost is associated with each CPU/hour,

based on the operating cost of the machine (indeed, industrial users who wish to access the resource are usually charged ‘real money’ for access to the machine based on this core hour cost). However, academic users are not usually charged a monetary fee for access to a machine. Instead their allocation is funded directly by a funding agency (such as EPSRC in the case of the UK’s ARCHER machine). Although the resources allocated to a project have a nominal value, and may be reported in cash terms in a grant application, no money changes hand between user and resource provider.

The RAMP system outlined in this thesis requires a rethink in the way resources are allocated. RAMP brings HPC usage much nearer to a commercial cloud model, whereby users pay for access to resources. Actual implementation of a production system requires buy-in from funding agencies and resource providers, and such policy matters are beyond the scope of this thesis. However, such a system could be realized by funding agencies crediting users or projects with a virtual currency in the RAMP system, which users can then spend with participating resources as they wish.

The RAMP model promotes openness, allowing any resource to join the system and make offers for workloads. As such, it heralds the potential demise of existing distinct closed grids and presages a future where federated e-infrastructure platforms are available to HPC users (both academic and industrial) and commercial providers can sell HPC time following a cloud model. By doing this, it empowers the user by giving her a mechanism by which she can obtain access to resources when convenient to her scientific studies, and not to the resource operator.

In the HPC world this creates problems, since applications often need to be tuned and customized for individual resources. AHE addresses this problem by allowing users to execute shared applications on distributed e-infrastructures without having to worry about the underlying details of code optimization and execution.

8.3 Future Work

Taken together, our AHE and RAMP systems constitute a powerful platform that align distributed high performance computing infrastructures with emerging cloud models. However, many topics remain to be addressed before the system could be deployed in a widespread production environment. We have shown that the platform is technically feasible to implement, but questions remain as to the policies required to use it in

practice. In our future work we plan to address some of these issues, especially relating to the enforcement of one time service level agreements and system wide banking resilience and reconciliation. Our mediate priority therefore is to interest an existing distributed e-infrastructure to deploy and test our RAMP software in a production environment.

We also plan to investigate how our approach can be extended to help solve licensing issues, by allowing users to pay for the core hours their application is used for, with this charge also encompassing compute time, rather than just paying for a licence for the application and for compute time separately.

We will continue to develop AHE, and have ongoing funding to do so within several EU funded projects, which by their nature are increasing the numbers of AHE users. Responding to feature requests from these projects will guide our future AHE development.

Appendix A

Request for Quotations XML Schema

Listing A.1: Request For Quotation XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.↵
  realitygrid.org/AHE/RFQ" xmlns="http://www.realitygrid.org/AHE/RFQ" ↵
  elementFormDefault="qualified">
  <xs:element name="RequestForQuotation">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Request" minOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CPUHourCost" type="xs:integer" maxOccurs="1" ↵
                minOccurs="1"/>
              <xs:element name="EndDate" type="xs:date" maxOccurs="1" minOccurs="1"/↵
                >
              <xs:element name="EndTime" type="xs:time" maxOccurs="1" minOccurs="1"/↵
                >
              <xs:element name="StartDate" type="xs:date" maxOccurs="1" minOccurs="0↵
                "/>
              <xs:element name="StartTime" type="xs:time" maxOccurs="1" minOccurs="0↵
                "/>
              <xs:element name="OperatingSystem" maxOccurs="1" minOccurs="0">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="Unknown"/>
                    <xs:enumeration value="MACOS"/>
                    <xs:enumeration value="ATTUNIX"/>
                    <xs:enumeration value="DGUX"/>
                    <xs:enumeration value="DECNT"/>
                    <xs:enumeration value="Tru64_UNIX"/>
                    <xs:enumeration value="OpenVMS"/>
                    <xs:enumeration value="HPUX"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<xs:enumeration value="AIX" />
<xs:enumeration value="MVS" />
<xs:enumeration value="OS400" />
<xs:enumeration value="OS_2" />
<xs:enumeration value="JavaVM" />
<xs:enumeration value="MSDOS" />
<xs:enumeration value="WIN3x" />
<xs:enumeration value="WIN95" />
<xs:enumeration value="WIN98" />
<xs:enumeration value="WINNT" />
<xs:enumeration value="WINCE" />
<xs:enumeration value="NCR3000" />
<xs:enumeration value="NetWare" />
<xs:enumeration value="OSF" />
<xs:enumeration value="DC_OS" />
<xs:enumeration value="Reliant_UNIX" />
<xs:enumeration value="SCO_UnixWare" />
<xs:enumeration value="SCO_OpenServer" />
<xs:enumeration value="Sequent" />
<xs:enumeration value="IRIX" />
<xs:enumeration value="Solaris" />
<xs:enumeration value="SunOS" />
<xs:enumeration value="U6000" />
<xs:enumeration value="ASERIES" />
<xs:enumeration value="TandemNSK" />
<xs:enumeration value="TandemNT" />
<xs:enumeration value="BS2000" />
<xs:enumeration value="LINUX" />
<xs:enumeration value="Lynx" />
<xs:enumeration value="XENIX" />
<xs:enumeration value="VM" />
<xs:enumeration value="Interactive_UNIX" />
<xs:enumeration value="BSDUNIX" />
<xs:enumeration value="FreeBSD" />
<xs:enumeration value="NetBSD" />
<xs:enumeration value="GNU_Hurd" />
<xs:enumeration value="OS9" />
<xs:enumeration value="MACH_Kernel" />
<xs:enumeration value="Inferno" />
<xs:enumeration value="QNX" />
<xs:enumeration value="EPOC" />
<xs:enumeration value="IxWorks" />
<xs:enumeration value="VxWorks" />
<xs:enumeration value="MiNT" />
<xs:enumeration value="BeOS" />
<xs:enumeration value="HP_MPE" />
<xs:enumeration value="NextStep" />
<xs:enumeration value="PalmPilot" />
```

```

        <xs:enumeration value="Rhapsody" />
        <xs:enumeration value="Windows_2000" />
        <xs:enumeration value="Dedicated" />
        <xs:enumeration value="OS_390" />
        <xs:enumeration value="VSE" />
        <xs:enumeration value="TPF" />
        <xs:enumeration value="Windows_R_Me" />
        <xs:enumeration value="Caldera_Open_UNIX" />
        <xs:enumeration value="OpenBSD" />
        <xs:enumeration value="Not_Applicable" />
        <xs:enumeration value="Windows_XP" />
        <xs:enumeration value="z_OS" />
        <xs:enumeration value="other" />
    </xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="OSVersion" type="xs:string" maxOccurs="1" minOccurs="0" />
<xs:element name="Architecture" maxOccurs="1" minOccurs="0">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="sparc" />
            <xs:enumeration value="powerpc" />
            <xs:enumeration value="x86" />
            <xs:enumeration value="x86_32" />
            <xs:enumeration value="x86_64" />
            <xs:enumeration value="parisc" />
            <xs:enumeration value="mips" />
            <xs:enumeration value="ia64" />
            <xs:enumeration value="arm" />
            <xs:enumeration value="other" />
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<xs:element name="CPUSpeed" type="xs:integer" maxOccurs="1" minOccurs="0" />
<xs:element name="WallTime" type="xs:integer" maxOccurs="1" minOccurs="0" />
<xs:element name="InterNodeBandwidth" type="xs:integer" maxOccurs="1" minOccurs="0" />
<xs:element name="RAMPerCore" type="xs:integer" maxOccurs="1" minOccurs="0" />
<xs:element name="Disk" maxOccurs="1" minOccurs="0">
    <xs:complexType>
        <xs:choice>
            <xs:element name="TotalDiskSpace" type="xs:integer" />
            <xs:element name="NodeDiskSpace" type="xs:integer" />
        </xs:choice>
    </xs:complexType>
</xs:element>

```

```
</xs:complexType>
</xs:element>
<xs:element name="Cores" maxOccurs="1" minOccurs="1">
  <xs:complexType>
    <xs:choice>
      <xs:element name="TotalCores" type="xs:integer"/>
      <xs:element name="Nodes">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="NodeCount" type="xs:integer"/>
            <xs:element name="NodeCores" type="xs:integer"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```


Appendix B

Inter-agent Communication Ontology

Listing B.1: Inter-agent communication ontology in OWL format

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:swrl="http://www.w3.org/2003/11/swrl#"
  xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns="http://www.owl-ontologies.com/Ontology1402054803.owl#"
  xml:base="http://www.owl-ontologies.com/Ontology1402054803.owl">
  <owl:Ontology rdf:about="" />
  <owl:Class rdf:ID="_JADE-SLOT">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#<
      Property" />
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >:JADE-SLOT</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID="_PROJECT-ANNOTATION">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >:PROJECT-ANNOTATION</rdfs:label>
  </owl:Class>
  <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing" />
      <owl:Class rdf:ID="_JADE-CLASS" />
    </owl:unionOf>
  </owl:Class>
  <owl:Class rdf:about="#_JADE-CLASS">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```

>:JADE-CLASS</rdfs:label>
<rdfs:subClassOf>
  <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
</rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="_JADE-INCLUDED-PROPERTIES">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >:JADE-INCLUDED-PROPERTIES</rdfs:label>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="resolvers">
  <_JADE-UNNAMED-SLOT rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
    >false</_JADE-UNNAMED-SLOT>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:ID="AID">
          <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
            >AID</rdfs:label>
          <rdfs:subClassOf>
            <_JADE-CLASS rdf:ID="Concept">
              <_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
                "
              >true</_JADE-IGNORE>
              <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                >The common ancestor for all concepts (i.e. types of entity such as ←
                  Person, Address...) in an ontology</rdfs:comment>
              <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                >Concept</rdfs:label>
            </_JADE-CLASS>
          </rdfs:subClassOf>
          <_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
            >true</_JADE-IGNORE>
        </_JADE-CLASS>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >resolvers</rdfs:label>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="CANCELINSTANCE">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:ID="Cancel">

```

```

<rdfs:subClassOf>
  <_JADE-CLASS rdf:ID="AgentAction">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >AgentAction</rdfs:label>
    <_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
      "
    >true</_JADE-IGNORE>
    <rdfs:subClassOf rdf:resource="#Concept"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >The common ancestor for all actions in an ontology (e.g. Sell, Buy<←
      ...)</rdfs:comment>
  </_JADE-CLASS>
</rdfs:subClassOf>
<_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
>false</_JADE-IGNORE>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Cancel</rdfs:label>
</_JADE-CLASS>
</owl:unionOf>
</owl:Class>
</rdfs:domain>
<rdfs:range>
  <_JADE-CLASS rdf:ID="RFQ">
    <rdfs:subClassOf>
      <_JADE-CLASS rdf:ID="Predicate">
        <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Predicate</rdfs:label>
        <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >The common ancestor for all predicates in an ontology (e.g. FatherOf<←
          ...)</rdfs:comment>
        <_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
        >true</_JADE-IGNORE>
      </_JADE-CLASS>
    </rdfs:subClassOf>
    <_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
    >false</_JADE-IGNORE>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >RFQ</rdfs:label>
  </_JADE-CLASS>
</rdfs:range>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>CANCELINSTANCE</rdfs:label>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="RSCHEDINSTANCE">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >RSCHEDINSTANCE</rdfs:label>
  <rdfs:domain>
    <owl:Class>

```

```

<owl:unionOf rdf:parseType="Collection">
  <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
  <_JADE-CLASS rdf:ID="Reschedule">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Reschedule</rdfs:label>
    <rdfs:subClassOf rdf:resource="#AgentAction"/>
    <_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
    >false</_JADE-IGNORE>
  </_JADE-CLASS>
</owl:unionOf>
</owl:Class>
</rdfs:domain>
<rdfs:range rdf:resource="#RFQ"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="OFFERINSTANCE">
  <rdfs:range>
    <_JADE-CLASS rdf:ID="Offer">
      <_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
      >false</_JADE-IGNORE>
      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Offer</rdfs:label>
      <rdfs:subClassOf rdf:resource="#Predicate"/>
    </_JADE-CLASS>
  </rdfs:range>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:ID="MakeOffer">
          <rdfs:subClassOf rdf:resource="#AgentAction"/>
          <_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >false</_JADE-IGNORE>
          <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >MakeOffer</rdfs:label>
        </_JADE-CLASS>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >OFFERINSTANCE</rdfs:label>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="RFQINSTANCE">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >RFQINSTANCE</rdfs:label>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>

```

```

    <_JADE-CLASS rdf:ID="MakeRequest">
      <_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
      >false</_JADE-IGNORE>
      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >MakeRequest</rdfs:label>
      <rdfs:subClassOf rdf:resource="#AgentAction"/>
    </_JADE-CLASS>
  </owl:unionOf>
</owl:Class>
</rdfs:domain>
<rdfs:range rdf:resource="#RFQ"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="addresses">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdfs:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:about="#AID"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <_JADE-UNNAMED-SLOT rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
  >false</_JADE-UNNAMED-SLOT>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >addresses</rdfs:label>
</owl:DatatypeProperty>
<owl:FunctionalProperty rdf:ID="_JADE-NAME">
  <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdfs:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <owl:Class rdf:about="#_JADE-CLASS"/>
        <owl:Class rdf:about="#_JADE-SLOT"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >:_JADE-NAME</rdfs:label>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="CPUHOURCOST">
  <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >CPUHOURCOST</rdfs:label>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>

```

```

<owl:Class>
  <owl:unionOf rdf:parseType="Collection">
    <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
    <_JADE-CLASS rdf:ID="Cost">
      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Cost</rdfs:label>
      <rdfs:subClassOf rdf:resource="#Predicate"/>
      <_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
        >false</_JADE-IGNORE>
    </_JADE-CLASS>
  </owl:unionOf>
</owl:Class>
</rdfs:domain>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_JADE-PROPERTIES-NAME">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >:_JADE-PROPERTIES-NAME</rdfs:label>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_JADE-IGNORE">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <owl:Class rdf:about="#_JADE-CLASS"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >:_JADE-IGNORE</rdfs:label>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="OPERATINGSYSTEM">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:ID="Resource">
          <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >Resource</rdfs:label>
          <rdfs:subClassOf rdf:resource="#Concept"/>
          <_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
            >false</_JADE-IGNORE>
        </_JADE-CLASS>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>

```

```

</rdfs:domain>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>OPERATINGSYSTEM</rdfs:label>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_JADE-UNNAMED-SLOT">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
<rdfs:domain>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
<owl:Class rdf:about="#_JADE-SLOT"/>
</owl:unionOf>
</owl:Class>
</rdfs:domain>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>:_JADE-UNNAMED-SLOT</rdfs:label>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_NODECORES">
<rdfs:domain>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
<_JADE-CLASS rdf:ID="Cores">
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Cores</rdfs:label>
<_JADE-IGNORE rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
>false</_JADE-IGNORE>
<rdfs:subClassOf rdf:resource="#_Predicate"/>
</_JADE-CLASS>
</owl:unionOf>
</owl:Class>
</rdfs:domain>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>NODECORES</rdfs:label>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_LOCATION">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:domain>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
<owl:Class rdf:about="#_PROJECT-ANNOTATION"/>
</owl:unionOf>

```

```

    </owl:Class>
  </rdfs:domain>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The location to store files into</rdfs:comment>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >:LOCATION</rdfs:label>
  <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="REQUESTID">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >REQUESTID</rdfs:label>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdfs:Description rdf:about="http://www.w3.org/2002/07/owl#Thing">
          <JADE-CLASS rdf:about="#RFQ"/>
        </rdfs:Description>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="CPUSPEED">
  <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >CPUSPEED</rdfs:label>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdfs:Description rdf:about="http://www.w3.org/2002/07/owl#Thing">
          <JADE-CLASS rdf:about="#Resource"/>
        </rdfs:Description>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_JADE-PROPERTIES-COMPATIBILITY">
  <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >:JADE-PROPERTIES-COMPATIBILITY</rdfs:label>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_JADE-JAVA-BASE-CLASS">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >:JADE-JAVA-BASE-CLASS</rdfs:label>
  <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain>
    <owl:Class>

```



```

    <owl:unionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
      <owl:Class rdf:about="#_JADE-CLASS"/>
    </owl:unionOf>
  </owl:Class>
</rdfs:domain>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_JADE-PROPERTIES-PACKAGE">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >:JADE-PROPERTIES-PACKAGE</rdfs:label>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="CORES">
  <rdfs:range rdf:resource="#Cores"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:about="#Cost"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >CORES</rdfs:label>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="DURATION">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:about="#Cores"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >DURATION</rdfs:label>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="COST">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >COST</rdfs:label>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Cost"/>
  <rdfs:domain>
    <owl:Class>

```

```

    <owl:unionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
      <_JADE-CLASS rdf:about="#RFQ"/>
    </owl:unionOf>
  </owl:Class>
</rdfs:domain>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="TOTALDISKSPACE">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >TOTALDISKSPACE</rdfs:label>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:about="#Resource"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="RESOURCE">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >RESOURCE</rdfs:label>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:about="#Cores"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Resource"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="TOTALCORES">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >TOTALCORES</rdfs:label>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:about="#Cores"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>

```

```

</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_ONTOLOGYNAME">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <owl:Class rdf:about="#_PROJECT-ANNOTATION"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >:ONTOLOGYNAME</rdfs:label>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >name of the ontology</rdfs:comment>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_PACKAGE">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <owl:Class rdf:about="#_PROJECT-ANNOTATION"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >packagename</rdfs:comment>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >:PACKAGE</rdfs:label>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_OSVERSION">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:about="#Resource"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >OSVERSION</rdfs:label>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_SUPPORT">

```

```

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>:SUPPORT</rdfs:label>
<rdfs:domain>
  <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
      <owl:Class rdf:about="#PROJECT-ANNOTATION"/>
    </owl:unionOf>
  </owl:Class>
</rdfs:domain>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="RAMPERCORE">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>RAMPERCORE</rdfs:label>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <JADE-CLASS rdf:about="#Resource"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="name">
  <JADE-UNNAMED-SLOT rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
>false</JADE-UNNAMED-SLOT>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <JADE-CLASS rdf:about="#AID"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>name</rdfs:label>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="DEADLINE">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">

```

```

        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <_JADE-CLASS rdf:about="#Cost"/>
    </owl:unionOf>
</owl:Class>
</rdfs:domain>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>DEADLINE</rdfs:label>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="NOTBEFORE">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>NOTBEFORE</rdfs:label>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
                <_JADE-CLASS rdf:about="#Cost"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:domain>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="OFFERCOST">
    <rdfs:domain>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
                <_JADE-CLASS rdf:about="#Offer"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:domain>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>OFFERCOST</rdfs:label>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:range rdf:resource="#Cost"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_JADE-PROPERTIES-GENERATE-ONTOLOGY">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>:_JADE-PROPERTIES-GENERATE-ONTOLOGY</rdfs:label>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_JADE-JAVA-CODE">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>:_JADE-JAVA-CODE</rdfs:label>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>

```

```

<rdfs:domain>
  <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
      <owl:Class rdf:about="#JADE-CLASS"/>
    </owl:unionOf>
  </owl:Class>
</rdfs:domain>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="NODEDISKSPACE">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <JADE-CLASS rdf:about="#Resource"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >NODEDISKSPACE</rdfs:label>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="RFQOntology_Class7">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >RFQOntology_Class7</rdfs:label>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="OFFERID">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >OFFERID</rdfs:label>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <JADE-CLASS rdf:about="#Offer"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="INTERNODEBANDWIDTH">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">

```

```

        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
        <JADE-CLASS rdf:about="#Resource"/>
    </owl:unionOf>
</owl:Class>
</rdfs:domain>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>INTERNODEBANDWIDTH</rdfs:label>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_JADE-PROPERTIES-GENERATE-BEANS">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>:JADE-PROPERTIES-GENERATE-BEANS</rdfs:label>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="_JADE-PROPERTIES-DIRECTORY">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>:JADE-PROPERTIES-DIRECTORY</rdfs:label>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="ARCHITECTURE">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>ARCHITECTURE</rdfs:label>
    <rdfs:domain>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
                <JADE-CLASS rdf:about="#Resource"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:domain>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="NODECOUNT">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>NODECOUNT</rdfs:label>
    <rdfs:domain>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
                <JADE-CLASS rdf:about="#Cores"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:domain>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:FunctionalProperty>

```

```
<_PROJECT-ANNOTATION rdf:about="project_annotation">
  <_ONTOLOGYNAME rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Market</_ONTOLOGYNAME>
  <_PACKAGE rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >uk.ac.ucl.chem.ccs.ramp.rfq.onto</_PACKAGE>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >project_annotation</rdfs:label>
</_PROJECT-ANNOTATION>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.4.6, Build 613) http://protege.↳
stanford.edu -->
```


Appendix C

AHE Usability Handouts

This appendix contains the data collection screens and instructions presented to participants in our AHE usability study.

Intro Text

Thank you for agreeing to take part in our usability study. The idea behind grid computing is to hide the complexity of using high performance computers distributed across the globe. The purpose of the study is to assess the usability of several different grid computing client tools, designed to allow you to run simulations on remote high performance (super) computers. For the purposes of this study we will be running a test code that sorts a list of words into alphabetical order.

This study is broken into two different sections. Firstly we will ask you to try out four different grid computing client tools to launch and monitor the sorting application on a remote system. Secondly, we will ask you to evaluate two different grid security solutions.

Please refer to the documentation provided in order to complete each task. When you are ready to begin a task, click Start, and when you have completed the task click Stop. You will be asked to answer various questions relating to the tasks you have completed.

You will be observed on your performance during each task, but the observer is not allowed to help you with the tasks, or answer any questions. Please remember, we are not assessing your individual performance, rather we are looking at the usability of each tool in the trial. Please answer each question truthfully and to the best of your ability.

Your feedback is very important because as it will be fed back to those responsible for developing these clients and thus help them determine which client is easiest to use and how the clients could be improved.

Please enter your Participant ID and click Next to proceed.

Section 1: Previous Experience

About you. Please help us to gain an insight into your background by answering the following eight questions.

Questions

- 1) What is the highest level degree that you currently hold (Bsc, PhD etc)?
- 2) What subject is your degree in (Computer Science, Chemistry etc)?
- 3) Have you ever used a high performance computing cluster before? If Yes, please give details
- 4) Have you ever used Grid computing before? If Yes, please give details
- 5) I am familiar with the Unix operating system – strongly agree – strongly disagree
- 6) I am familiar with command line interfaces – strongly agree – strongly disagree
- 7) I am familiar with graphical user interfaces – strongly agree – strongly disagree
- 8) I understand the concept of distribute computing – strongly agree – strongly disagree

Section 2: Using the Globus toolkit

Tasks

Task 1: Run an application

Upload the following files from your local machine:

`$HOME/sortapp-input/input.txt`

`$HOME/sortapp-input/config.txt`

To the directory `$GRIDDIR` on the machine `bunsen.chem.ucl.ac.uk`, then run the application `/usr/local/bin/ahe_sort.pl` on the machine `bunsen.chem.ucl.ac.uk` with the parameter `$GRIDDIR/config.txt`

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Task 2: Monitor the status of the application that you launched on `bunsen.chem.ucl.ac.uk` until it completes

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Task 3: Once the job is complete, download the output file `$GRIDDIR/output.txt` from the machine `bunsen.chem.ucl.ac.uk` to your home directory (`$HOME`)

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Questions

1. I think that I would like to use this system frequently - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
2. I found the system unnecessarily complex - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.

3. I thought the system was easy to use - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
4. I think that I would need the support of a technical person to be able to use this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
5. I found the various functions in this system were well integrated - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
6. I thought there was too much inconsistency in this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
7. I would imagine that most people would learn to use this system very quickly - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
8. I found the system very cumbersome to use - - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
9. I felt very confident using the system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
10. I needed to learn a lot of things before I could get going with this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.

Section 3: Using the Unicore Graphical User Interface

Tasks

Task 1: Run an application

Using the Unicore client, create a new Generic grid bean to run the Perl 5.8.8 script /usr/local/bin/ahe_sort.pl installed on the machine crick.chem.ucl.ac.uk (DEMO-SITE) with the parameter config.txt

Configure the grid bean to stage the following input files from your local machine:

/Users/ccs/study/sortapp-input/input.txt

/Users/ccs/study/sortapp-input/config.txt

Configure the grid bean to stage back the output file output.txt.

When the grid bean is configured, submit the job.

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Task 2: Monitor the status of the application that you launched on crick.chem.ucl.ac.uk until it completes

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Task 3: Download the output file (output.txt) from the job you just ran to the directory /Users/ccs/study

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Questions

1. I think that I would like to use this system frequently - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
2. I found the system unnecessarily complex - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
3. I thought the system was easy to use - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.

4. I think that I would need the support of a technical person to be able to use this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
5. I found the various functions in this system were well integrated - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
6. I thought there was too much inconsistency in this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
7. I would imagine that most people would learn to use this system very quickly - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
8. I found the system very cumbersome to use - - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
9. I felt very confident using the system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
10. I needed to learn a lot of things before I could get going with this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.

Section 4: Using the Application Hosting Environment Graphical User Interface

Tasks

Task 1: Run sort application

Using the AHE graphical client, run the sort application on the machine crick.chem.ucl.ac.uk using the configuration file in /Users/ccs/study/sortapp-input/config.txt to automatically configure the input and output data that needs to be staged. Use a single CPU.

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Task 2: Monitor the status of the application that you launched on crick.chem.ucl.ac.uk until it completes

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Task 3: Download the output files from the job you just ran to the directory /Users/ccs/study

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Questions

1. I think that I would like to use this system frequently - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
2. I found the system unnecessarily complex - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
3. I thought the system was easy to use - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.

4. I think that I would need the support of a technical person to be able to use this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
5. I found the various functions in this system were well integrated - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
6. I thought there was too much inconsistency in this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
7. I would imagine that most people would learn to use this system very quickly - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
8. I found the system very cumbersome to use - - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
9. I felt very confident using the system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
10. I needed to learn a lot of things before I could get going with this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.

Section 5: Using the Application Hosting Environment Command Line Interface

Tasks

Task 1: Run an application

Using the AHE command line client, run the sort application on the machine crick.chem.ucl.ac.uk using the configuration file in /Users/ccs/study/sortapp-input/config.txt to automatically configure the input and output data that needs to be staged. Use a single CPU.

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Task 2: Monitor the application that you ran bunsen.chem.ucl.ac.uk until it completes

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Task 3: Download the output files from the job you just ran to the directory /Users/ccs/study

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Questions

1. I think that I would like to use this system frequently - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
2. I found the system unnecessarily complex - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
3. I thought the system was easy to use - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.

4. I think that I would need the support of a technical person to be able to use this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
5. I found the various functions in this system were well integrated - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
6. I thought there was too much inconsistency in this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
7. I would imagine that most people would learn to use this system very quickly - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
8. I found the system very cumbersome to use - - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
9. I felt very confident using the system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
10. I needed to learn a lot of things before I could get going with this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.

Section 6: Any comments on tasks 2-5

Please provide any comments or feedback on the tasks you have just completed in sections 2-5.

We will now move on to test two different aspects of AHE security, using a system based on certificates, and also a system based on usernames/passwords. Click **Next** to continue.

Section 7: Configure the AHE client to use a grid certificate and submit a job

Create a new AHE keystore using the certificate file available at:

/Users/ccs/study/cert.p12

The password for the certificate is **tmpstore**

Once you have created the certificate, open the AHE client, upload a new proxy certificate and run the sort application again on the machine crick.chem.ucl.ac.uk

The configuration file is available at /Users/ccs/study/sortapp-input/config.txt

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Questions

1. I think that I would like to use this system frequently - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
2. I found the system unnecessarily complex - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
3. I thought the system was easy to use - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
4. I think that I would need the support of a technical person to be able to use this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
5. I found the various functions in this system were well integrated - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
6. I thought there was too much inconsistency in this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
7. I would imagine that most people would learn to use this system very quickly - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
8. I found the system very cumbersome to use - - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
9. I felt very confident using the system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
10. I needed to learn a lot of things before I could get going with this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.

Section 8: Submit a job using the ACD client

Log in to the AHE client using username and password and run the sort application again on the machine crick.chem.ucl.ac.uk

The username is test and the password is tmpstore

The configuration file is available at /Users/ccs/study/sortapp-input/config.txt

Q1: This task was – Very Easy, Easy, Neutral, Difficult, Very Difficult

Q2: How satisfied were you with the software used to complete this task – Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied

Questions

1. I think that I would like to use this system frequently - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
2. I found the system unnecessarily complex - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
3. I thought the system was easy to use - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
4. I think that I would need the support of a technical person to be able to use this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
5. I found the various functions in this system were well integrated - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
6. I thought there was too much inconsistency in this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
7. I would imagine that most people would learn to use this system very quickly - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
8. I found the system very cumbersome to use - - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
9. I felt very confident using the system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.
10. I needed to learn a lot of things before I could get going with this system - Strongly Disagree, disagree, Neutral, Agree, Strongly Agree.

Section 9: Any comments on tasks 7-8

Please provide any comments or feedback on the tasks you have just completed in sections 7-8.

Usability Study Observer Record Sheet

Participant ID:				
Section 2: Using the Globus toolkit				
	Success	Failed	False completion	Failed due to previous
Task 1				
Task 2				
Task 3				
Section 3: Using the Unicore Graphical User Interface				
	Success	Failed	False completion	Failed due to previous
Task 1				
Task 2				
Task 3				
Section 4: Using the Application Hosting Environment Graphical User Interface				
	Success	Failed	False completion	Failed due to previous
Task 1				
Task 2				
Task 3				
Section 5: Using the Application Hosting Environment Command Line Interface				
	Success	Failed	False completion	Failed due to previous
Task 1				
Task 2				
Task 3				
Section 7: Configure the AHE client to use a grid certificate and submit a job				
	Success	Failed	False completion	Failed due to previous
Task 1				
Task 2				
Task 3				
Section 8: Submit a job using the ACD client				
	Success	Failed	False completion	Failed due to previous
Task 1				
Task 2				
Task 3				

- **Successful:** The participant completed the task without difficulty.
- **Failed:** The participant gave up on the task without completing it.
- **False completion:** The participant failed to complete the task but erroneously believed that they had in fact been successful.
- **Failed due to previous:** The participant could not complete the task because they had incorrectly completed the preceding task, e.g. couldn't download data due to problem completing the job submission task

Appendix D

AHE Usability User Manuals

This appendix contains all of the client documentation guides presented to participants in our AHE usability study. Specifically, this section includes:

- Globus Toolkit Manual
- Unicore Manual
- AHE Graphical Client Manual
- AHE Command Line Tools Manual
- ACD Manual
- Configuring AHE Security Guide

Globus Toolkit Manual

The Globus Toolkit provides command line clients that allow you to interact with grid resources running the Globus middleware.

Preparation

Prior to using any of the Globus toolkit components, you need to create a proxy certificate. This is a short lived security credential that can be used on the grid. To create a proxy use the command:

```
grid-proxy-init
```

You will be prompted for your certificate password. The password used in all of these exercises is tmpstore

Submitting a job

Prior to launching a job, you need to stage any input files required by the job to the grid machine you are going to use. Files are staged with the globus-url-copy command, e.g.:

```
globus-url-copy file:///tmp/foo gsiftp://remote.machine.my.edu/tmp/bar
```

The command takes two arguments: the path to the source file, and the path to the destination it is to be copied to. On some systems environment variables can be used in place of the full path name.

The simplest command for job submission is globus-job-submit. The minimum parameters used by this command are where to send the job and what the program to run is. Submit your first job with the command:

```
globus-job-submit hostname application parameters
```

where hostname is the name of the machine you want to use, application is the full path to the application, and parameters are the parameters passed to the application. For example,

```
globus-job-submit ngs.oerc.ox.ac.uk /bin/hostname -f
```

will run the application /bin/hostname on the UK National Grid Service machine at Oxford with the parameter -f. The globus-job-submit command returns a reference handle to the job that allows it to be monitored:

```
https://ngs.oerc.ox.ac.uk:64001/1415/1110129853/
```

Monitoring a job

Jobs are monitored with the `globus-job-status` command followed by the reference handle of the job to monitor, e.g.:

```
globus-job-status https://ngs.oerc.ox.ac.uk:64001/1415/1110129853/
```

The command will report the status of the job. When the job is complete, the status will be reported as `DONE`.

Retrieving output

Once a job is complete, the output files generated can again be retrieved using the `globus-url-copy` command, with the local and remote components reversed, e.g.:

```
globus-url-copy gsiftp://ng2.auckland.ac.nz/home/grid-  
bestgrid/output.txt file:///home/yhal003/output.txt
```

Will copy the file `output.txt` from the directory `/home/grid-bestgrid/` on the machine `ng2.auckland.ac.nz` to the user's home directory.

Unicore Manual

Preparation

When the client first starts, select the Unicore grid registry under the Registries heading. A list of target systems will be shown under the Target Systems heading. Right-click on the target system that you want to submit your job to and click Connect.

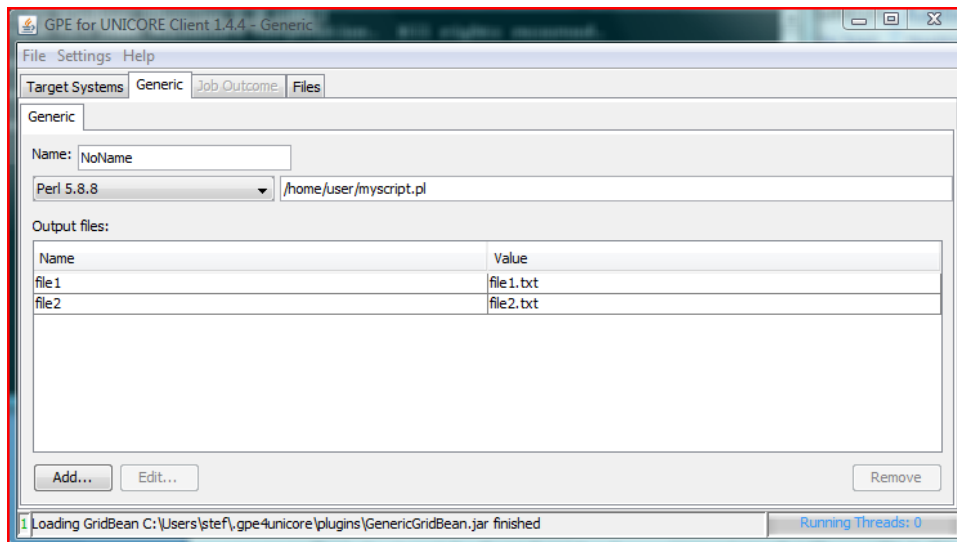
Submitting a job

To submit a job, first download an appropriate gridbean. Click File > Download Gridbeans. The Generic gridbean can be used to run any application on a grid system. Select your gridbean and click OK.

Once the gridbean has loaded, the second tab in the main window will change to the name of the bean. Click on this tab (e.g. Generic) to configure your job. Enter a name to refer to your job in the Name field, then select the type of application you want to run from the drop-down list (e.g. Perl 5.8.8 to run a Perl script). Enter any parameters to the application (such as the name of the input file) on the same line after the application's name.

Next, enter the full path to the application that you want to run on the remote system.

With the application configured, you next need to specify the output files that will be generated. To add an output file, click Add, then add a name to refer to the file, and the actual name of the file that will be generated, then click OK. Repeat this step for each output file that will be created.



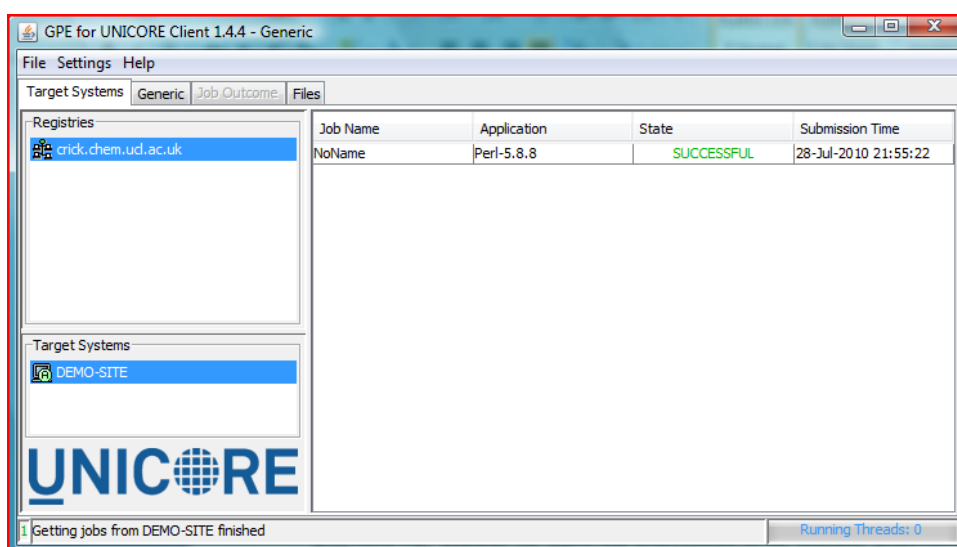
Finally, you need to configure the input files that will be staged to the remote grid resource in order to run the job. Click on the Files tab, select InputFileSet then click Edit.

For each input file that needs to be staged to the remote machine, click on the Add File button, then click Browse and select the file that needs to be staged, and click OK. Repeat for each file that needs to be staged, then click OK.

To run your job click File > Submit Job. Enter the parameters for your job (the default will allow the job to run for up to one hour) then click OK.

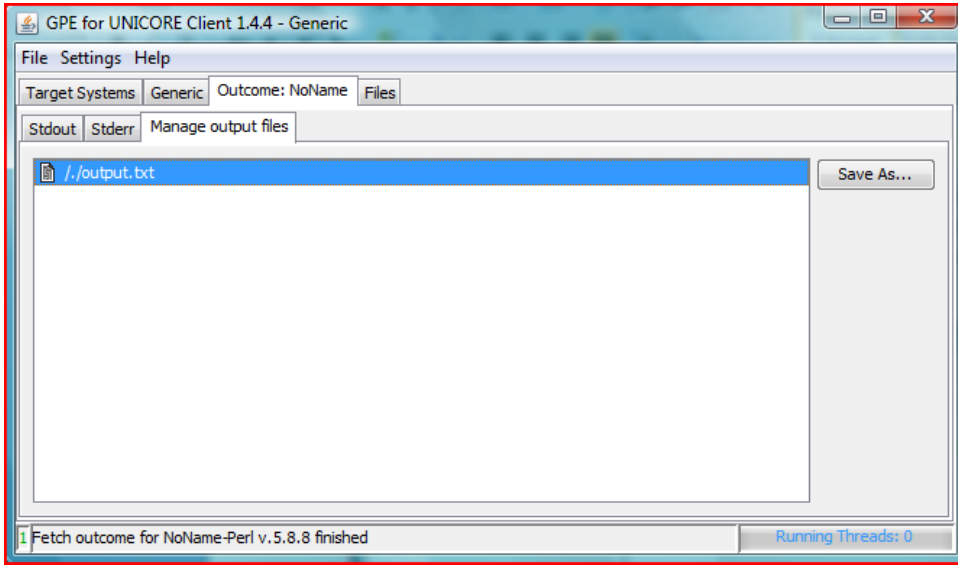
Monitoring a job

To monitor a submitted application, click on the Target Systems tab. The job you have submitted will be displayed in the main panel. Right-click on the job name and choose Refresh to update the status. Once the status is shown as **SUCCESSFUL** the job is complete.



Retrieving output

Once a job is complete, you retrieve the output by right-clicking on the job name and choosing Fetch Outcome. Once the output has been downloaded, it will be available on the Job outcome (third) tab of the main window. In addition to any output files specified, the standard output and standard error will also be retrieved. To save an output file, click on the Managed Output tab, select the file then click Save As. Choose a location for the output file and click Save. Repeat this step for every output file you need to download.



AHE Graphical Client

Submitting a job

In order to submit a job using the AHE GUI client, double-click on **Prepare a new job**.

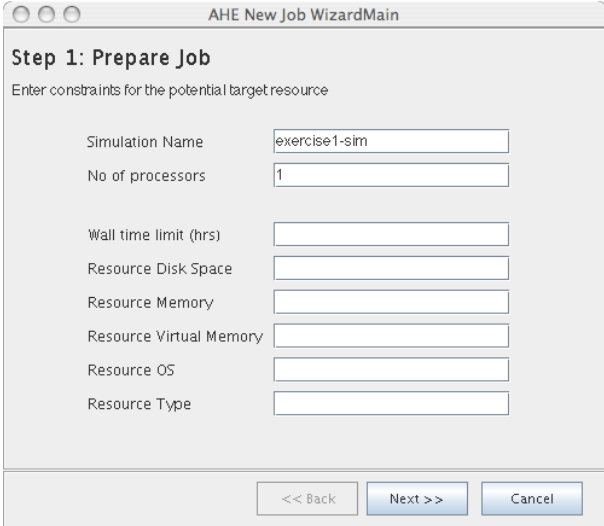
From the **Select an application to run** drop down list, choose the application to run then click on the button **Find Job Factories**. After a short time you will see a the factory endpoint for your application appear in the box beneath.

Click on the sort factory endpoint to select it (it should turn blue), then click **Launch Wizard**.

The AHE job launching wizard will open. Enter a name for the job, for example **job1** (this name is used to give the user a convenient way to refer to the job in future).

Enter a number of processors for the job – for example enter **1** so that the application is run on a single processor.

Click **Next**.

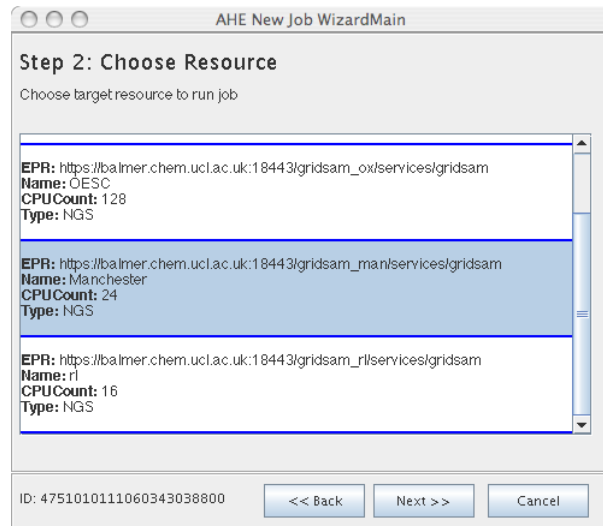


The screenshot shows a window titled "AHE New Job WizardMain" with a sub-header "Step 1: Prepare Job". Below the sub-header is the instruction "Enter constraints for the potential target resource". The form contains several input fields:

Simulation Name	<input type="text" value="exercise1-sim"/>
No of processors	<input type="text" value="1"/>
Wall time limit (hrs)	<input type="text"/>
Resource Disk Space	<input type="text"/>
Resource Memory	<input type="text"/>
Resource Virtual Memory	<input type="text"/>
Resource OS	<input type="text"/>
Resource Type	<input type="text"/>

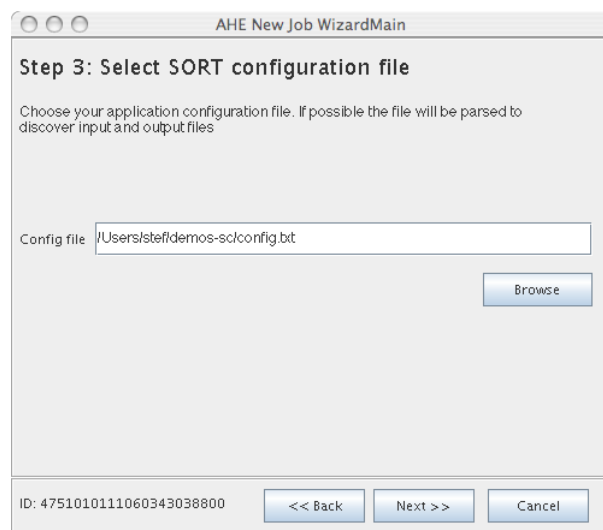
At the bottom of the window are three buttons: "<< Back", "Next >>", and "Cancel".

You are now prompted to choose a machine to run the application on – for this example choose **crick.chem.ucl.ac.uk** (the UCL cluster), then click **Next**.



The AHE client will parse your application's input file to discover the data files that need to be moved from the local machine to the grid machine, and the data that will be created on the cluster that needs to be moved back to your local machine after the job has finished.

Click on the **Browse** button, select your sort application configuration file (e.g. **config.txt**). Click **Open**, then click **Next**.



The AHE client will parse the config.txt file and discover the input and output files associated with the job. Clicking on the **Stage** button will move the input files over to the AHE file staging area from where it will be moved over to the grid machine.

Once the files have been staged click **Next**.

AHE New Job WizardMain

Step 4: Stage Files

The following input files will be staged to the resource

Description	File Name
conf-file	/Users/stef/demos-sc/config.bt
inputfile	/Users/stef/demos-sc/input.bt
argument	/Users/stef/demos-sc/config.bt.ahe

The following output files will be staged from the resource

Description	File Name
outputfile	/Users/stef/demos-sc/output.bt
stdout	/Users/stef/demos-sc/stdout.bt
stderr	/Users/stef/demos-sc/stderr.bt

Review the details of your sort job and click the **Finish** button to launch the job.

AHE New Job WizardMain

Step 5: Check job details and submit

Job arguments:
config.bt.ahe

Job standard in:
[Dropdown menu]

Job configuration file:
config.txt

Job standard out:
stdout.txt

Job standard error:
stderr.txt

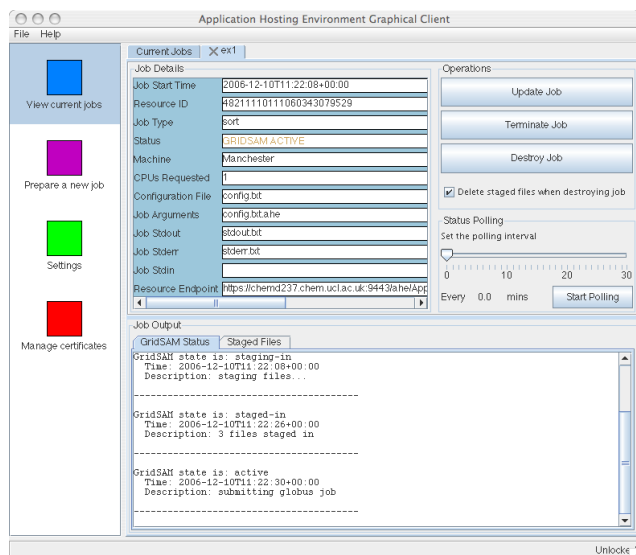
Start job running now

Job built ok

Monitoring a job

To check on the status of your application once launched, double-click on **View current jobs**. You will see the job that you have just launched at the top of the list. Double-click on the job entry to open its monitoring window.

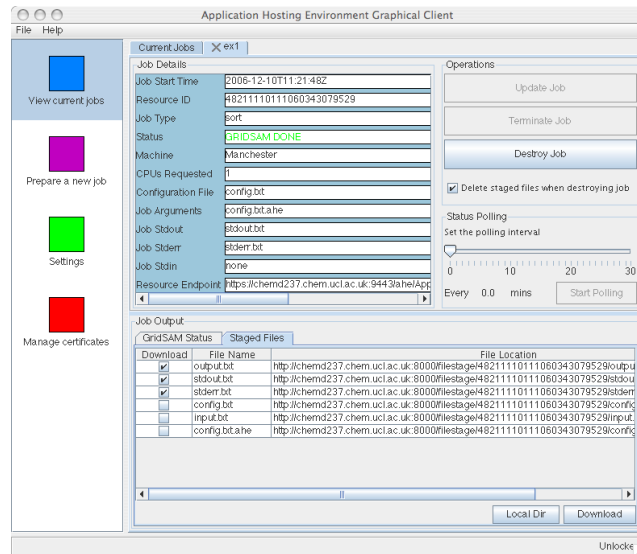
To check up on the status of a job, click on the **Update Job** button. This will poll the AHE server and update the status of the job.



Once a job is complete, the status will be set to **AHE DONE**.

Retrieving output

When the job has finished (signalled by a status of **AHE DONE**) the output files will be left on the AHE file staging server for you to download. To do so, click on the **Staged Files** tab under job output. Here you will see all of the output that has been generated.



Click on the **Local Dir** button. This allows you to change the directory where the output files will be saved (by default they will be saved to the same directory as the input files). Create a new output directory in your home directory and choose **Open**.

Click on the **Download** button. The output files will be saved to the directory you created. Browse to this folder and check the output of the sort job.

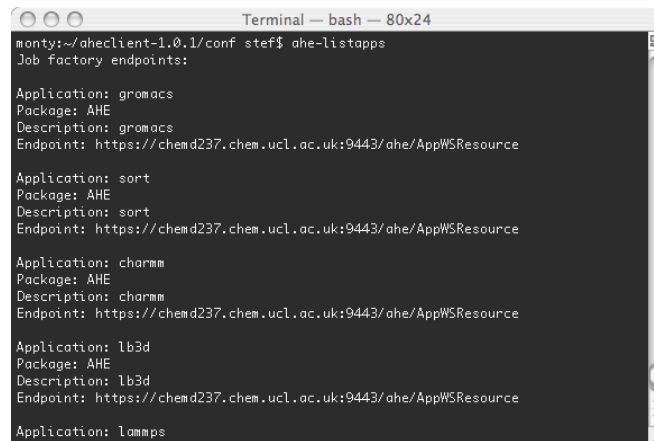
AHE Command Line Tools

Submitting a job

Prior to launching an application you need to discover which applications are available. At the terminal type:

```
ahc-listapps
```

This will list all of the applications installed in the AHE, along with the factory endpoints needed to start them. From this list find the endpoint of the application you want to run.



```
Terminal -- bash -- 80x24
monty:~/ahcclient-1.0.1/conf stef$ ahc-listapps
Job factory endpoints:

Application: gromacs
Package: AHE
Description: gromacs
Endpoint: https://chemd237.chem.ucl.ac.uk:9443/ahc/AppWSResource

Application: sort
Package: AHE
Description: sort
Endpoint: https://chemd237.chem.ucl.ac.uk:9443/ahc/AppWSResource

Application: charmm
Package: AHE
Description: charmm
Endpoint: https://chemd237.chem.ucl.ac.uk:9443/ahc/AppWSResource

Application: lb3d
Package: AHE
Description: lb3d
Endpoint: https://chemd237.chem.ucl.ac.uk:9443/ahc/AppWSResource

Application: lammmps
```

The first step to launch an application using the AHE command line clients is to issue the `ahc-prepare` command. The command takes the following parameters:

```
ahc-prepare -e endpoint -app application -s name -RMCPUCount cpucount
```

Where:

- *endpoint* = the endpoint of the application discovered using the `ahc-list` command.
- *application* = the name of the application to run, discovered using the `ahc-list` command.
- *name* = a user defined name to easily refer to the application
- *cpucount* = the number of CPUs to use, usually 1.

For example, to run the application `charmm` with one CPU use the following command (all on one line):

```
ahe-prepare -e
https://chemd237.chem.ucl.ac.uk:9443/ahe/AppWSResource -app charmm
-s job1 -RMCPUCount 1
```

The ahe-prepare command will return a list of the machines that are able to run the application.

To set application running, use the ahe-start command as follows:

```
ahe-start -s name -config conf-file -RM machine -n cpucount
```

Where:

- *name* = the name of the application set using the ahe-prepare command.
- *conf-file* = the full path to the job configuration file
- *machine* = the name of the machine to use, returned by the ahe-prepare command.
- *cpucount* = the number of CPUs to use, usually 1.

For example, type:

```
ahe-start -s job1 -config $HOME/charmm/config.txt -RM
mavrino.chem.ucl.ac.uk -n 1
```

Would run the charmm application using the input file \$HOME/charmm/config.txt on the machine mavrino.chem.ucl.ac.uk with one CPU. The command will stage the necessary files to the AHE file staging area and start the job running.

Monitoring a job

The ahe-list command allows you to view a list of the jobs you have previously started. At the terminal type:

```
ahe-list
```

The ahe-monitor command allows you to monitor an individual job. To check the status of the job type:

```
ahe-monitor -s name
```

where *name* is name of the job previously configured. For example, to monitor the charmm example from earlier:

```
ahe-monitor -s job1
```

The ahe-monitor command will return the status of the job.

```
Terminal — bash — 80x24
Time: 2006-12-10T12:03:30+00:00
Description: 3 files staged in

GridSAM state is: active
Time: 2006-12-10T12:03:30+00:00
Description: submitting globus job

GridSAM state is: executed
Time: 2006-12-10T12:03:51+00:00
Description: globus job completed

GridSAM state is: staging-out
Time: 2006-12-10T12:04:11+00:00
Description: staging files out...

GridSAM state is: staged-out
Time: 2006-12-10T12:04:17+00:00
Description: 3 files staged out

GridSAM state is: done
Time: 2006-12-10T12:04:17+00:00
Description: Job completed

monty:~/aheclient-1.0.1/conf stef$
```

When a job is completed, the status will be done

Retrieving output

Once the ahe-monitor command reports the status of the job as complete, you can retrieve the output files from the AHE file staging area using the ahe-getoutput command. The -l parameter allows you to specify the path to the directory where you would like the output to be placed. For example, to create a new folder in your home directory and download the output data from the charm job example type:

```
mkdir $HOME/output-dir
```

```
ahe-getoutput -s job1 -l $HOME/output-dir
```

Task: Using ACD-AHE Client to run an Application on the grid

A Virtual Organisation (VO) is intended to offer simplified end-user access to and use of high performance computing (HPC) resources shared across of number of different institutions with different administrative domains. A typical example of a VO is the computational grid, which aims to provide control over distributed resources consisting of enormous computational power (parallel processing machines), data storage (hard disks, memory) and visualisation on high speed networks. Examples of currently operating grids include: the UK National Grid Service (NGS) and US TeraGrid.

The sharing of these resources is intended to support academic research and industrial development. A computational grid environment may consist of a mixture of several kinds of organisations including academic, governmental, industrial and commercial institutions.

Middleware tools exist in order to smooth the connection between the administrative domains and their associated resources, but these have not always resulted in the envisaged simplicity and ease of use. The Application Hosting Environment¹ (AHE) is a tool that allows scientists to run computational applications on grid resources in a quick, transparent manner. Version 2.5 of the AHE allows end-users to acquire credentials from their local site administrators to access grid resources. This means removing X509 digital certificate from end-users' experience.

Aims and objectives

The principal objective of this trial is to introduce you to the state-of-the-art grid computing where you will have access to some of the largest supercomputers in the UK. The trial is designed as follows:

1. The first section shows you how to install and configure light client software, ACD-AHE Client, which allows you to run applications on Grid resources using username-password credentials.
2. In the second section you will use the client ACD-AHE client to run the sort application run on the UK National Grid service resources.

Introduction

This ACD-AHE client does not require the end user to obtain a personal X.509 certificate in order to run applications at grid. The authentication is based on "username/password", which most users have already been familiar with. The user logs in to ACD-AHE client with a username and password given to them by the local administrator in the VO and submits jobs to named grid resources that are provided by the VO.

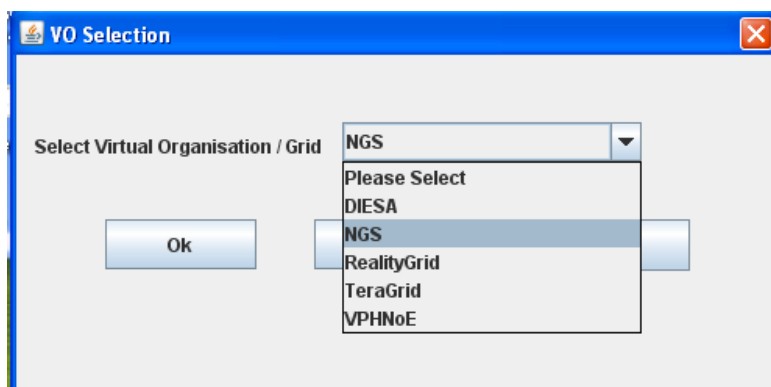
Stage 1: Install the ACD-AHE client on your systems

1. Download ACD-AHE certificate client file: name – This has already been done for you and you can find the file under your home directory
 - a. `[user@servername] cd ~`
 - b. `[user@servername] tar xvfz aheacdclient.tgz`

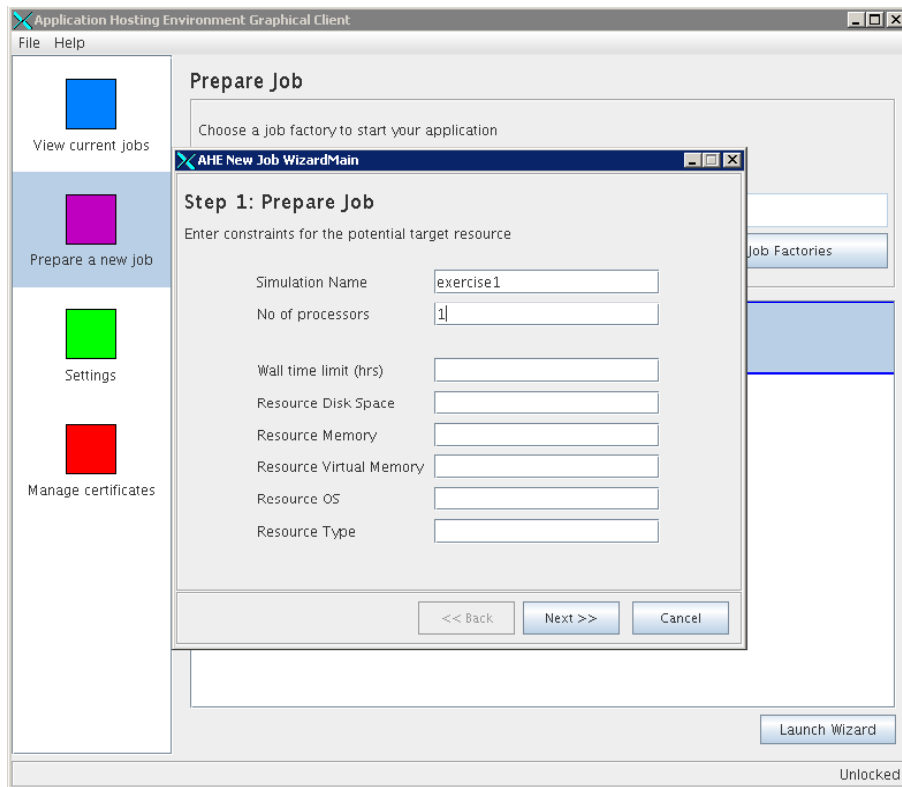
2. Login with the username/password given to you.



3. Select the Virtual organization that you want to use for running your tasks on its resources. In this case it is NGS.



4. Follow the same steps as with the normal AHE client (described below)
5. Click on Prepare a new job
6. From the Select an application to run drop down list, choose sort then click on the button Find Job Factories.
7. After a short time you will see a Sort factory endpoint appear in the box beneath.
8. Click on the sort factory endpoint to select it (it should turn blue), then click Launch Wizard
9. The AHE job launching wizard will open. Enter a name for the job, for example exercise1 (this name is used to give the user a convenient way to refer to the job in future).
10. Enter a number of processors for the job – in this case enter “1”, so that the sort application is run on a single processor.
11. Click Next.



12. You are now prompted to choose an NGS machine to run the application on – for this example choose Oxford-NGS2 (the Oxford NGS node), then click Next
13. The AHE client will parse your sort job's input file to discover the data files that need to be moved from the local machine to the NGS node at Oxford, and the data that will be created at Oxford that needs to be moved back to your local machine after the job has finished.

Application Hosting Environment Graphical Client

File Help

View current jobs

Prepare a new job

Settings

Manage certificates

Prepare Job

Choose a job factory to start your application

Job Factories

AHE New Job WizardMain

Step 2: Choose Resource

Choose target resource to run job

EPR: https://mavrino.chem.ucl.ac.uk:18443/gridsam/services/gridsam
Name: Mavrino
CPUCount: 96
Type: Local

EPR: https://schroedinger.chem.ucl.ac.uk:18443/gridsam_oesc/services/gridsam
Name: Oxford-NGS2
CPUCount: 128
Type: NGS

EPR: https://schroedinger.chem.ucl.ac.uk:18443/gridsam_man/services/gridsam
Name: Manchester-NGS2

ID: 82416180108517053359

<< Back Next >> Cancel

Launch Wizard

Unlocked

Configuring AHE Security

A certificate is a way of identifying you to remote computers. It is simply a way of letting a resource know you are who you say you are (i.e. authenticate yourself). A Certificate can best be imagined as a "Digital Passport" which cannot be forged.

Prior to using the AHE client, you need to create a keystore for your client, containing your grid certificate. This will be used to authenticate you to the AHE server, and generate the proxy credentials required to access grid resources. A script is provided to allow you to import your grid certificate in p12 format into a Java keystore for use by the AHE client.

At the command prompt type:

kssetup path_to_certificate

where *path_to_certificate* is the full path to your grid certificate (for example /home/ahel/nescuser00.p12).

When prompted to set up a local configuration directory, answer **N**

You will be prompted to enter the password for your p12 format certificate. Enter the password (**tmpstore**) and press [**Return**].

Next, you will be prompted to enter a password for the keystore that you are setting up. This will be the password that you enter to unlock the AHE client.

You will see a message telling you that the certificate has been imported, and then you will be prompted for your keystore password again (the second password you had to enter).

From here, you can use the AHE client to submit jobs as before. Launch the AHE GUI client by typing **ahе-guіclient** followed by [**Enter**]. You will be prompted to enter the password you created when you generated your keystore to open the client.

Prior to submitting a job, you need to create a proxy credential to use on the grid and upload it to a MyProxy server. This is a short lived certificate which the AHE can use to submit the job to the grid machine on behalf of the user. There are many different tools that will allow you to generate and upload a proxy certificate to a MyProxy server (e.g. Gloubs's myproxy-init). The AHE GUI client features the ability to generate and upload proxies, which we will use in this section.

To generate and upload a proxy certificate using the AHE, double-click on **Manage certificates**, and, in the Upload Proxy box, enter and confirm a password for your proxy certificate, then click **Create and Upload**.

When the proxy certificate has been successfully created and uploaded, the details of the proxy will be shown in the Current Proxy Credentials box. Clicking

on the **Delete Credential from Server** button will delete the proxy credential. The proxy credential tool allows you to specify the lifetime of the proxy on the proxy server. After a proxy expires a new one will need to be created before you can submit jobs again.

Once the proxy certificate has been uploaded and created, you can submit a job.

Glossary

ACL Agent Communication Language, a proposed standard language for agent communications.

Agent a computer program that acts for a user or other program in a relationship of agency.

Beowulf a computer cluster of what are normally identical, commodity-grade computers networked into a small local area network with libraries and programs installed which allow processing to be shared among them.

BPEL an OASIS standard executable language for specifying actions within business processes with web services.

Cluster a set of loosely or tightly connected computers that work together so that they can be viewed as a single system.

Condor an open-source high-throughput computing software framework for coarse-grained distributed parallelization of computationally intensive tasks.

DEISA an e-infrastructure consortium of eleven national supercomputing centres from seven European countries that promoted pan-European research on European high-performance computing systems.

FIPA Foundation for Intelligent Physical Agents, a body for developing and setting computer software standards for heterogeneous and interacting agents and agent-based systems.

gLite a middleware software toolkit for grid computing used by the CERN LHC experiments and other scientific domains.

Globus an open source grid computing middleware toolkit developed and supported by the Globus Alliance.

GPGPU general-purpose computing on graphics processing units, the utilization of a graphics processing unit (GPU) to perform computation in applications traditionally handled by the central processing unit.

GRAM a Globus component which enable users to locate, submit, monitor and cancel remote jobs on grid based compute resources.

Grid the aggregation of computer resources from multiple locations to reach a common goal.

GridFTP an extension of the standard File Transfer Protocol (FTP) for high-speed, reliable, and secure data transfer.

GSI a specification for secret, tamper-proof, delegatable communication between software in a grid computing environment.

HPC refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation, in order to solve large problems.

HTTP an application protocol used to build distributed, collaborative, hypermedia information systems.

IaaS a provision model in which an organization outsources the equipment used to support operations, including storage, hardware, servers and networking components.

InfiniBand a computer network communications link used in high-performance computing featuring very high throughput and very low latency.

iRODS a logical distributed file system based on a client-server architecture which presents users with a single global logical namespace or file hierarchy.

JSDL an extensible XML specification used to described simple tasks to be run on non-interactive computer execution systems.

LoadLeveler a family of IBM Tivoli workload automation products that plan, execute and track jobs on several platforms and environments.

LSF a workload management platform, job scheduler, for distributed HPC environments, developed by Platform Computing.

MapReduce a programming model for processing and generating large data sets using a parallel, distributed algorithm on a cluster.

MAS multi-agent system, a computerized system composed of multiple interacting intelligent agents within an environment.

MDS the information services component of the Globus toolkit and provides information about the available resources on the grid and their status.

MPI a standardized and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computers.

MPICH-G2 a distributed version of MPI that allows a single application to be distributed across multiple grid resources, the forerunner to MPIg.

MPIg a distributed version of MPI that allows a single application to be distributed across multiple grid resources.

MyProxy open source software for managing X.509 Public Key Infrastructure (PKI) security credentials.

Myrinet a high-speed local area networking system designed by Myricom to be used as an interconnect between multiple machines to form computer clusters.

NAMD a freeware molecular dynamics simulation package noted for its parallel efficiency and often used to simulate large systems.

NAT a method[1] of modifying network address information in Internet Protocol (IP) datagram packet headers while they are in transit across a traffic routing device for the purpose of remapping one IP address space into another.

OGSA a distributed interaction and computing architecture based around services, assuring interoperability on heterogeneous systems so that different types of resources can communicate and share information.

OGSA-BES a Web services based interface for creating, monitoring, and controlling computational entities such as UNIX or Windows processes, Web Services, or parallel programs.

OGSI a proposed recommendation intended to provide an infrastructure layer for the Open Grid Services Architecture (OGSA) by extending Web services to accommodate grid computing resources that are both transient and stateful..

OpenMP a portable, scalable API that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from a desktop workstation to a supercomputer.

OSI open systems interconnection model, a conceptual model that characterizes and standardizes the internal functions of a communication system by partitioning it into abstraction layers.

PBS a job scheduling application used to allocate computational tasks, originally developed by NASA.

POP3 an application-layer protocol used by local e-mail clients to retrieve e-mail from a remote server.

PRACE a persistent high performance computing e-infrastructure providing services to scientists and researchers from academia and industry in Europe.

PVM parallel virtual machine, a software tool for parallel networking of computers.

QCG QosCosGrid, a middleware toolkit offering advanced job and resource management capabilities to deliver to end-users supercomputer-like performance and structure.

REST an architecture style used as a set of guidelines for creating web services which allow nodes connected to a network to communicate with one another via the HTTP communication protocol.

RFQ request for quotation, a standard business process whose purpose is to invite suppliers into a bidding process to bid on specific products or services.

SaaS a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted.

SAGA a family of related standards specified by the Open Grid Forum to define an API for common distributed computing functionality.

SGE a compute cluster batch-queuing system, developed and supported by Sun Microsystems and Oracle.

SMP symmetric multiprocessing, a multiprocessor system with centralized shared memory and a single operating system, with two or more homogeneous processors.

SMTP an Internet standard for e-mail transmission.

SOA a set of principles and methodologies for designing and developing software in the form of interoperable services.

SOAP a protocol specification for exchanging structured information in the implementation of web services in computer networks.

SUS system usability scale, a simple, ten-item attitude Likert scale giving a global view of subjective assessments of usability.

Unicore an open source grid computing middleware toolkit, the development of which is supported by UNICORE Forum e.V..

VPH Virtual Physiological Human, a methodological and technological framework, and associated research initiative, designed to enable collaborative investigation of the human body as a single complex system.

WebDAV an extension of the Hypertext Transfer Protocol (HTTP) that allows clients to perform remote Web content authoring operations.

WSDL an XML-based interface definition language that is used to describe the functionality offered by a web service.

WSRF a set of operations that web services may implement to become stateful.

X.509 an ITU-T standard for a public key infrastructure (PKI) and Privilege Management Infrastructure (PMI).

XSEDE a US based e-infrastructure that scientists can use to interactively share computing resources, data and expertise.

XSLT a language for transforming XML documents into other XML documents or other formats.

Bibliography

- [1] S. J. Zasada, D. C. W. Chang, A. N. Haidar, and P. V. Coveney, Flexible composition and execution of large scale applications on distributed e-infrastructures, *Journal of Computational Science*, 5(1):51–62, 2014.
- [2] S. J. Zasada and P. V. Coveney, Distributed biomedical computing, in P. V. Coveney, V. Daz-Zuccarini, P. Hunter, and M. Viceconti, editors, *Computational Biomedicine*, chapter 9, Oxford University Press, Oxford, 2014.
- [3] S. J. Zasada, D. Chang, A. N. Haidar, and P. V. Coveney, A lightweight platform for managing biomedical simulation, in *Proceedings of the 3rd international workshop on Emerging computational methods for the life sciences*, pages 75–78, ACM, 2012.
- [4] S. J. Zasada, M. Mamonski, D. Groen, J. Borgdorff, I. Saverchenko, T. Piontek, K. Kurowski, and P. V. Coveney, Distributed infrastructure for multiscale computing, in *Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, pages 65–74, IEEE Computer Society, 2012.
- [5] S. J. Zasada, T. Wang, A. Haidar, E. Liu, N. Graf, G. Clapworthy, S. Manos, and P. V. Coveney, IMENSE: An e-infrastructure environment for patient specific multiscale data integration, modelling and clinical treatment, *Journal of Computational Science*, 3(5):314–327, 2012.
- [6] S. J. Zasada, A. N. Haidar, and P. V. Coveney, On the usability of grid middleware and security mechanisms, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1949):3413–3428, 2011.

- [7] A. N. Haidar, S. J. Zasada, P. V. Coveney, A. E. Abdallah, B. Beckles, and M. A. S. Jones, Audited credential delegation: a usable security solution for the virtual physiological human toolkit, *Interface Focus*, 1(3):462–473, 2011.
- [8] S. J. Zasada and P. V. Coveney, From campus resources to federated international grids: bridging the gap with the application hosting environment, in *Proceedings of the 5th Grid Computing Environments Workshop, GCE '09*, pages 10:1–10:10, ACM, 2009.
- [9] S. J. Zasada and P. V. Coveney, Virtualizing access to scientific applications with the application hosting environment, *Computer Physics Communications*, 180(12):2513–2525, 2009.
- [10] S. Manos, S. J. Zasada, and P. V. Coveney, Life or death decision-making: The medical case for large-scale, on-demand grid computing, *CTWatch Quarterly Journal*, 4(2):35–45, 2008.
- [11] S. Manos, S. Zasada, M. Mazzeo, R. Haines, G. Doctors, S. Brew, R. Pinning, J. Brooke, and P. Coveney, Patient specific whole cerebral blood flow simulation: A future role in surgical treatment for neurovascular pathologies, in *Proceedings of the 3rd TeraGrid Conference*, 2008.
- [12] P. V. Coveney, R. S. Saksena, S. J. Zasada, M. M. Keown, and S. Pickles, The application hosting environment: Lightweight middleware for grid-based computational science, *Computer Physics Communications*, 176(6):406–418, 2007.
- [13] S. J. Zasada, R. S. Saksena, P. V. Coveney, M. M. Keown, and S. M. Pickles, Facilitating User Access to the Grid: A Lightweight Application Hosting Environment for Grid Enabled Computational Science, *Proceedings of the International Conference on e-Science and Grid Computing*, pages 50–58, 2006.
- [14] D. Turek, High performance computing and the implications of multi-core architectures, *CTWatch Quarterly*, 3:31–33, 2007.
- [15] J. Dongarra, D. Gannon, G. Fox, and K. Kennedy, The impact of multicore on computational science software, *CTWatch Quarterly*, 3(11), 2007.

- [16] P. V. Coveney (ed.), Scientific Grid Computing, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 363(1833):1707–1713, 2005.
- [17] I. Foster, C. Kesselman, and S. Tuecke, The anatomy of the grid: Enabling scalable virtual organizations, *International Journal of Supercomputer Applications*, 15:3–23, 2001.
- [18] J. Chin and P. V. Coveney, Towards Tractable Toolkits for the Grid: a Plea for Lightweight, Useable Middleware, Technical report, 2004, http://nesc.ac.uk/technical_papers/UKeS-2004-01.pdf.
- [19] M. Halling-Brown, D. Moss, C. Sansom, and A. Shepherd, A computational grid framework for immunological applications, *Philosophical Transactions of the Royal Society A*, 367:2705–2716, 2009.
- [20] I. Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems, *Journal of Computer Science and Technology*, 21(4):513–520, 2006.
- [21] The UNICORE Project, <http://www.unicore.org>.
- [22] gLite Middleware, <http://glite.web.cern.ch/glite/>.
- [23] S. K. Sadiq, S. J. Zasada, and P. V. Coveney, Grid assisted ensemble molecular dynamics simulations of hiv-1 proteases reveal novel conformations of the inhibitor saquinavir, in M. Berthold, R. Glen, and I. Fischer, editors, *Computational Life Sciences II*, volume 4216 of *Lecture Notes in Computer Science*, pages 150–161, Springer Berlin / Heidelberg, 2006.
- [24] B. Boghosian, P. V. Coveney, S. Dong, L. Finn, S. Jha, G. Karniadakis, and N. Karonis, Nektar, spice and vortonic: using federated grids for large scale scientific applications, *Challenges of Large Applications in Distributed Environments, 2006 IEEE*, pages 34–42, 2006.
- [25] P. Chakraborty, S. Jha, and D. S. Katz, Novel submission modes for tightly coupled jobs across distributed resources for reduced time-to-solution, *Philosophi-*

- cal Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1897):2545–2556, 2009.
- [26] M. Zelkowitz and D. Wallace, Experimental models for validating technology, *Computer*, 31(5):23–31, 1998.
- [27] M. Zelkowitz and D. Wallace, Experimental validation in software engineering, *Information and Software Technology*, 39(11):735–743, 1997.
- [28] D. Ridge, D. Becker, P. Merkey, and T. Sterling, Beowulf: harnessing the power of parallelism in a pile-of-pcs, *Aerospace Conference, 1997. Proceedings., IEEE*, 2:79–91 vol.2, Feb 1997.
- [29] L. Branscomb, *From desktop to teraflop: Exploiting the US lead in high performance computing*, National Science Foundation, 1993.
- [30] S. K. Sadiq, M. D. Mazzeo, S. J. Zasada, S. Manos, I. Stoica, C. V. Gale, S. J. Watson, P. Kellam, S. Brew, and P. V. Coveney, Patient-specific simulation as a basis for clinical decision-making, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1878):3199–3219, 2008.
- [31] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Computing*, 22(6):789–828, 1996.
- [32] V. Sunderam, Pvm: A framework for parallel distributed computing, *Concurrency: practice and experience*, 2(4), 1990.
- [33] Altair pbs professional, <http://www.altair.com/software/pbspro.htm>.
- [34] Sun Grid Engine, <http://gridengine.sunsource.net>.
- [35] Platform lsf, <http://www.ibm.com/systems/platformcomputing/products/lsf>.
- [36] K. Rycerz, E. Ciepiela, G. Dyk, D. Groen, T. Gubala, D. Harezlak, M. Pawlik, J. Suter, S. Zasada, P. Coveney, et al., Support for multiscale simulations with molecular dynamics, *Procedia Computer Science*, 18:1116–1125, 2013.

- [37] J. Day and H. Zimmermann, The osi reference model, *Proceedings of the IEEE*, 71(12):1334–1340, 1983.
- [38] Infiniband, <http://www.intel.com/technology/infiniband/>.
- [39] Myrinet, <http://www.myri.com/myrinet/overview/>.
- [40] L. Dagum and R. Menon, Openmp: an industry standard api for shared-memory programming, *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.
- [41] J. Dean and S. Ghemawat, Mapreduce: simplified data processing on large clusters, *Communications of the ACM*, 51(1):107–113, 2008.
- [42] P. V. Coveney, G. Giupponi, S. Jha, S. Manos, J. MacLaren, S. M. Pickles, R. S. Saksena, T. Soddemann, J. L. Suter, M. Thyveetil, and S. J. Zasada, Large scale computational science on federated international grids: The role of switched optical networks, *Future Generation Computer Systems*, 26(1):99–110, 2010.
- [43] I. Foster and C. Kesselman, Computational grids, *Cern European Organization for Nuclear Research-Reports-Cern*, pages 87–114, 1998.
- [44] S. Jha, P. V. Coveney, and M. Harvey, Spice: Simulated pore interactive computing environment, in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, IEEE Computer Society Washington, DC, USA, 2005.
- [45] R. Schmelzer, T. Vandersypen, J. Bloomberg, M. Siddalingaiah, S. Hunting, M. Qualls, C. Darby, D. Houlding, and D. Kennedy, *XML and Web Services Unleashed*, 2002.
- [46] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. Frystyk, Soap version 1.2 part 1: Messaging framework, Technical report, Jun 2003, <http://www.w3.org/TR/soap12-part1/>.
- [47] C. Goble, S. Pettifer, R. Stevens, and C. Greenhalgh, Knowledge integration: In silico experiments in bioinformatics, *The Grid: Blueprint for a New Computing Infrastructure*, pages 121–134, 2003.

- [48] M. Colan, Service-oriented architectures expands the vision of web services, part 1, Technical report, Apr 2004, <http://www-106.ibm.com/developerworks/library/ws-soaintro.html>.
- [49] M. MacKenzie, K. Laskey, F. McCabe, P. Brown, and M. Metz, Reference model for service oriented architecture 1.0, Technical report, 2006, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.
- [50] I. T. Foster and S. Tuecke, Describing the elephant: The different faces of it as service, *ACM Queue*, 3(6), 2005.
- [51] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, Colombo: Lightweight middleware for service-oriented computing, *IBM Systems Journal*, 44(4):799–820, 2005.
- [52] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke, From open grid services infrastructure to ws-resource framework: Refactoring and evolution, Technical report, May 2004, https://www.globus.org/wsrf/specs/ogsi_to_wsrf_1.0.pdf.
- [53] S. Graham, A. Karmarkar, J. Mischkin, I. Robinson, and I. Sedukin, Web Services Resource Framework, Technical report, 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf.
- [54] S. Tuecke, K. Czajkowski, I. Foster, I. Frey, S. Graham, C. Kesselman, T. Maquire, T. Snadholm, D. Snelling, and P. Vanderbilt, Open grid services infrastructure, Technical report, Global Grid Forum, Jun 2003.
- [55] R. T. Fielding, Architectural styles and the design of network-based software architectures, Ph.D. thesis, University of California, Irvine, 2000.
- [56] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Technical report, 2002, <http://www.globus.org/research/papers/ogsa.pdf>.
- [57] A. Grimshaw, S. Newhouse, D. Pulsipher, and M. Morgan, Ogsa basic execution service version 1.0, in *Open Grid Forum*, 2006.

- [58] Job Submission Description Language Specification, <http://forge.gridforum.org/projects/jsdl-wg/document/draft-ggf-jsdl-spec/en/21>.
- [59] B. Dillaway, M. Humphrey, C. Smith, M. Theimer, and G. Wasson, Hpc basic profile, version 1.0, in *Grid Forum Document GFD-RP*, volume 114, page 28, 2007.
- [60] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, et al., The open grid services architecture, *Global Grid Forum Draft, draft-ggf-ogsa-ogsa-011*, September, 23, 2003.
- [61] RFC 2459 - Internet X.509 Public Key Infrastructure Certificate and CRL Profile, <http://www.faqs.org/rfcs/rfc2459.html>.
- [62] The Partnership for Advanced Computing in Europe, <http://www.prace-project.eu/>.
- [63] Enabling grids for e-science, <http://www.eu-egee.org/>.
- [64] K. Kurowski, W. de Back, W. Dubitzky, L. Gulyás, G. Kampis, M. Mamon-ski, G. Szemes, and M. Swain, Complex system simulations with qoscosgrid, *Computational Science–ICCS 2009*, pages 387–396, 2009.
- [65] B. Bosak, J. Komasa, P. Kopta, K. Kurowski, M. Mamoński, and T. Piontek, New capabilities in QosCosGrid middleware for advanced job management, advance reservation and co-allocation of computing resources—quantum chemistry application use case, *Building a National Distributed e-Infrastructure–PL-Grid*, pages 40–55, 2012.
- [66] P. Troger, H. Rajic, A. Haas, and P. Domagalski, Standardization of an API for Distributed Resource Management Systems, in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, CCGRID '07*, pages 619–626, IEEE Computer Society, Washington, DC, USA, 2007.
- [67] D. Thain, T. Tannenbaum, and M. Livny, Condor and the grid, *Grid computing: Making the global infrastructure a reality*, pages 299–335, 2003.

- [68] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. Von Laszewski, C. Lee, A. Merzky, H. Rajic, and J. Shalf, SAGA: A Simple API for Grid Applications. High-level application programming on the Grid, *Computational Methods in Science and Technology*, 12(1):7–20, 2006.
- [69] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, GridFTP: Protocol extensions to FTP for the grid, *Global Grid Forum GFD-RP*, 20, 2003, <http://www.ggf.org/documents/GWD-R/GFD-R.020.pdf>.
- [70] The EUDAT Project, <http://www.eudat.eu>.
- [71] IGTF: Interoperable Global Trust Federation, <https://www.igtf.net/>.
- [72] R. J. Blake, P. V. Coveney, P. Clarke, and S. M. Pickles, The TeraGyroid experiment – SuperComputing 2003, *Scientific Programming*, 13(1):1–17, 2005.
- [73] B. Beckles, User Requirements for UK e-Science grid environments, in *Proceedings of UK All Hands e-Science Meeting, Nottingham*, pages 763–767, 2005, <http://www.allhands.org.uk/2004/submissions/papers/251.pdf>.
- [74] P. V. Coveney, J. Suter, R. Saksena, L. Pedesseau, J. Blower, and E. Auden, Usable middleware for grid based computational science, in *e-Science Usability Meeting at NeSC*, NeSC, Jan 2006.
- [75] C. M. Pancake, Usability issues in developing tools for the grid - and how visual representations can help, *Parallel Processing Letters*, 13(2):189–206, 2003.
- [76] B. Beckles, V. Welch, and J. Basney, Mechanisms for increasing the usability of grid security, *International Journal of Human-Computer Studies*, 63(1/2):74–101, 2005.
- [77] B. Kernighan and R. Pike, *The UNIX Programming Environment*, 1984.
- [78] WSRF::Lite, <http://www.sve.man.ac.uk/research/AtoZ/ILCT>.
- [79] SOAP::Lite Web Services Toolkit, <http://www.soaplite.com>.
- [80] J. Brooke, S. Zasada, and M. Computing, Implementing WS-Security in Perl, in *Proceedings of the UK e-Science All Hands Conference, Nottingham*, 2005.

- [81] M. Hayes, L. Morris, R. Crouchley, D. Grose, T. Van Ark, R. Allan, and J. Kewley, GROWL: A lightweight grid services toolkit and applications, in *4th UK e-Science All Hands Meeting, Nottingham, UK, 2005*.
- [82] J. Yu and R. Buyya, A taxonomy of scientific workflow systems for grid computing, *SIGMOD Record*, 34(3):44–49, 2005.
- [83] G. Sipos and P. Kacsuk, Classification and Implementations of Workflow-Oriented Grid Portals, *Lecture notes in computer science*, 3726:684–693, 2005.
- [84] E. Ciepiela, L. Zaraska, and G. D. Sulka, Gridspace2 virtual laboratory case study: implementation of algorithms for quantitative analysis of grain morphology in self-assembled hexagonal lattices according to the hillebrand method, in *Building a National Distributed e-Infrastructure–PL-Grid*, pages 240–251, Springer, 2012.
- [85] K. Amin, G. Von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi, Gridant: A client-controllable grid workflow system, in *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 10–pp, IEEE, 2004.
- [86] K. Gor, D. Ra, S. Ali, L. Alves, N. Arurkar, I. Gupta, A. Chakrabarti, A. Sharma, and S. Sengupta, Scalable enterprise level workflow and infrastructure management in a grid computing environment, in *IEEE International Symposium on Cluster Computing and the Grid, 2005. CCGrid 2005*, volume 2, pages 661–667, 2005.
- [87] I. Taylor, M. Shields, I. Wang, and A. Harrison, Visual grid workflow in triana, *Journal of Grid Computing*, 3(3-4):153–169, 2005.
- [88] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. J. Taylor, and I. Wang, Programming scientific and distributed workflow with triana services, *Concurrency Computat.: Pract. Exper.*, 18(10):1021–1037, 2006.
- [89] T. M. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, Taverna: a tool for

- the composition and enactment of bioinformatics workflows, *Bioinformatics*, 20(17):3045–3054, 2004.
- [90] Business process execution language for web services, 2005, <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>.
- [91] A. Slomiski, On using bpel extensibility to implement ogssi and wsrfl grid workflows, *Concurrency and Computation: Practice and Experience*, 18(10):1229–1241, 2006.
- [92] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S. L. Price, Grid service orchestration using the business process execution language (bpel), *J. Grid Comput*, 3(3-4):283–304, 2005.
- [93] D. Nurmi, R. Wolski, C. Grzegorzczak, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, The eucalyptus open-source cloud-computing system, in *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 124–131, IEEE, 2009.
- [94] O. Sefraoui, M. Aissaoui, and M. Eleuldj, Openstack: toward an open-source solution for cloud computing, *International Journal of Computer Applications*, 55(3):38–42, 2012.
- [95] D. Milojičić, I. M. Llorente, and R. S. Montero, Opennebula: A cloud management tool, *IEEE Internet Computing*, 15(2):0011–14, 2011.
- [96] US TeraGrid, <http://www.teragrid.org>.
- [97] The Extreme Science and Engineering Discovery Environment (XSEDE), <http://www.xsede.org/>.
- [98] Distributed European Infrastructure for Supercomputing Applications, <http://www.deisa.org/>.
- [99] IBM LoadLeveler, <http://www.ibm.com/systems/clusters/software/loadleveler.html>.
- [100] The Globus project, <http://www.globus.org>.

- [101] N. Karonis, B. Toonen, and I. Foster, MPICH-G2: a Grid-enabled implementation of the Message Passing Interface, *Journal of Parallel and Distributed Computing*, 63(5):551–563, 2003.
- [102] J. Brooke, P. Coveney, J. Harting, S. Jha, S. Pickles, R. Pinning, and A. Porter, Computational steering in realitygrid, in *Proceedings of the UK e-Science All Hands Meeting*, volume 2003, 2003.
- [103] A. S. McGough, A. Afzal, J. Darlington, N. Furmento, A. Mayer, and L. Young, Making the grid predictable through reservations and performance modelling, *Comput. J*, 48(3):358–368, 2005.
- [104] J. MacLaren, M. Mc Keown, and S. Pickles, Co-allocation, fault tolerance and grid computing, in *Proceedings of the UK e-Science All Hands Meeting*, pages 155–162, 2006.
- [105] L. Lamport, Paxos made simple, *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 32, 2001.
- [106] B. W. Lamson, How to build a highly available system using consensus, *Lecture Notes in Computer Science*, 1151:1–17, 1996.
- [107] P. Beckman, S. Nadella, N. Trebon, and I. Beschastnikh, SPRUCE: A system for supporting urgent high-performance computing, *Grid-Based Problem Solving Environments*, pages 295–311, 2007.
- [108] M. Alfano and G. Poli, The HIV life cycle: multiple targets for antiretroviral agents, *Drug Design Reviews-Online*, 1(1):83–92, 2004.
- [109] P. M. A. Slood, A. V. Boukhanovsky, W. Keulen, A. Tirado-Ramos, and C. A. Boucher, A grid-based hiv expert system, *J Clin Monit Comput*, 19(4-5):263–278, Oct 2005.
- [110] D. W. Wright, B. A. Hall, O. A. Kenway, S. Jha, and P. V. Coveney, Computing clinically relevant binding free energies of hiv-1 protease inhibitors, *Journal of Chemical Theory and Computation*, 10(3):1228–1241, 2014.

- [111] S. K. Sadiq, D. Wright, S. J. Watson, S. J. Zasada, I. Stoica, and P. V. Coveney, Automated molecular simulation based binding affinity calculator for ligand-bound HIV-1 proteases, *J. Chem. Inf. Model*, 48(9):1909–1919, 2008.
- [112] S. Wan and P. V. Coveney, Molecular dynamics simulation reveals structural and thermodynamic features of kinase activation by cancer mutations within the epidermal growth factor receptor, *Journal of Computational Chemistry*, 32(13):2843–2852, 2011.
- [113] J. Guilbert, The world health report 2002-reducing risks, promoting healthy life., *Educ Health (Abingdon)*, 16(2):230, 2003.
- [114] M. Thubrikar and F. Robicsek, Pressure-induced arterial wall stress and atherosclerosis, *The Annals of Thoracic Surgery*, 59(6):1594–1603, 1995.
- [115] M. Goyen, G. Laub, M. Ladd, J. Debatin, J. Barkhausen, K. Truemmler, S. Bosk, and S. Ruehm, Dynamic 3 d mr angiography of the pulmonary arteries in under four seconds, *Journal of Magnetic Resonance Imaging*, 13(3):372–377, 2001.
- [116] A. C. Taylor, M. T. Draney, J. P. Ku, D. Parker, B. N. Steele, K. Wang, and C. K. Zarins, Predictive medicine: Computational techniques in therapeutic decision-making, *Computer Aided Surgery*, 4(5):231–247, 1999.
- [117] D. R. Noble, J. G. Georgiadis, and R. O. Buchius, Comparison of accuracy and performance for lattice boltzmann and finite difference simulations of steady viscous flow, *Int. J. Num. Meth. Fluids*, 23(1), 1996.
- [118] M. D. Mazzeo and P. V. Coveney, Hemelb: a high performance parallel lattice-boltzmann code for large scale fluid flow in complex geometries, *Computer Physics Communications*, 178:894–914, 2008.
- [119] M. Riedel, B. Schuller, M. Rambadt, M. S. Memon, A. S. Memon, A. Streit, T. Lippert, S. J. Zasada, S. Manos, P. V. Coveney, et al., Exploring the potential of using multiple e-science infrastructures with emerging open standards-based e-health research tools, in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 341–348, IEEE, 2010.

- [120] UK Collaborative Computational Projects, <http://www.ccp.ac.uk/>.
- [121] L. Kale, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulte, NAMD2: Greater scalability for parallel molecular dynamics, *Journal of Computational Physics*, pages 283–312, 1999.
- [122] T. Borden, J. Hennessy, and J. Rymarczyk, Multiple operating systems on one processor complex, *IBM Systems Journal*, 28(1):104–123, Jan 1989.
- [123] The IP network address translator (NAT), <http://www.faqs.org/rfcs/rfc1631.html>.
- [124] The TLS protocol version 1.0, <http://www.faqs.org/rfcs/rfc2246.html>.
- [125] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist, X. 509 Proxy Certificates for Dynamic Delegation, in *3rd annual PKI R&D workshop*, volume 14, 2004.
- [126] I. R. Holmes and R. S. Kalawsky, The RealityGrid PDA and Smartphone clients: Developing Effective Handheld User Interfaces for e-Science, in *Proceedings of the UK e-Science All Hands Conference, Nottingham, 2006*.
- [127] I. Stoica, S. K. Sadiq, and P. V. Coveney, Rapid and accurate prediction of binding free energies for saquinavir-bound HIV-1 proteases, *Journal of the American Chemical Society*, 130(8):2639–2648, 2008.
- [128] J. L. Suter, P. V. Coveney, H. C. Greenwell, and M.-A. Thyveetil, Large-scale molecular dynamics study of montmorillonite clay: Emergence of undulatory fluctuations and determination of material properties, *The Journal of Physical Chemistry C*, 111(23):8248–8259, 2007.
- [129] Globus GRAM 4 Job Description Definition Schema, http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html.
- [130] J. Clark et al., XSL transformations (XSLT) version 1.0, *W3C Recommendation*, 16(11), 1999.

- [131] E. Gamma, R. Helm, R. Johnson, and V. J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1995.
- [132] Grid Job Submission and Monitoring Web Service, <http://gridsam.sourceforge.net>.
- [133] M. Gudgin and M. Hadley, *Web Services Addressing*, 2005, <http://www.w3c.org/TR/2005/WD-ws-addr-core-20050331>.
- [134] J. Treadwell and S. Graham, *Web Services Resource Properties*, 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf.
- [135] L. Srinivasan and T. Banks, *Web Services Resource Lifetime*, 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf.
- [136] T. Maguire, D. Snelling, and T. Banks, *Web Services Service Group*, 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf.
- [137] L. Liu and S. Meder, *Web Services Base Faults*, 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf.
- [138] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, *Web services security: SOAP message security 1.1 (ws-security 2004)*, Technical report, 2004.
- [139] The Apache Web Server, <http://httpd.apache.org/>.
- [140] The Apache Tomcat Servlet Container, <http://tomcat.apache.org>.
- [141] PostgreSQL, <http://www.postgresql.org/>.
- [142] The Open Middleware Infrastructure Institute, <http://www.omii.ac.uk>.
- [143] VirtualBox, <http://www.virtualbox.org/>.
- [144] CentOS Linux Distribution, <http://www.centos.org>.
- [145] JBPM - JBoss Community , <http://www.jboss.org/jbpm>.
- [146] Hibernate - JBoss Community, <http://www.hibernate.org/>.
- [147] Restlet - RESTful web services framework for Java, <http://www.restlet.org/>.

- [148] S. Pickles, R. Haines, R. Pinning, and A. Porter, A Practical Toolkit for Computational Steering, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 363(1833):1843–1853, 2005.
- [149] C. Cavaness, *Quartz Job Scheduling Framework: Building Open Source Enterprise Applications*, Prentice Hall, 2006.
- [150] R. O. Sinnott, J. Jiang, J. Watt, and O. Ajayi, Shibboleth-based Access to and Usage of Grid Resources, in *IEEE International Conference on Grid Computing, Barcelona, Spain*, pages 28–29, 2006.
- [151] The EU FP6 ViroLab Project, <http://www.virolab.org>.
- [152] P. V. Coveney, G. De Fabritiis, M. J. Harvey, S. M. Pickles, and A. R. Porter, Coupled applications on distributed resources, *Computer Physics Communications*, 175(6):389–396, 2006.
- [153] R. Keller, E. Gabriel, B. Krammer, M. S. Mueller, and M. M. Resch, Towards efficient execution of MPI applications on the grid: porting and optimization issues, *Journal of Grid Computing*, 1(2):133–149, 2003.
- [154] The Open Grid Forum, <http://www.ogf.org>.
- [155] S. Zasada, B. Cheney, R. Saksena, J. Suter, P. Coveney, and J. Essex, Production Level Scientific Simulation Management on International Federated Grids, in *Proceedings of the 2nd TeraGrid Conference*, 2007.
- [156] B. Beckles, P. V. Coveney, P. Y. A. Ryan, A. E. Abdallah, S. M. Pickles, J. M. Brooke, and M. Mc Keown, A user-friendly approach to computational grid security, in *Proceedings of the UK e-Science All Hands Meeting, Nottingham*, pages 473–480, 2006.
- [157] A. Haidar, S. Zasada, P. Coveney, A. Abdallah, and B. Beckles, Audited credential delegation - a user-centric identity management solution for computational grid environments, in *Information Assurance and Security (IAS), 2010 Sixth International Conference on*, pages 222–227, August 2010.

- [158] J. Nielsen, Usability inspection methods, in *Conference companion on Human factors in computing systems*, pages 413–414, ACM, 1994.
- [159] J. Cohen, *Statistical power analysis for the behavioral sciences*, Lawrence Erlbaum, 1988.
- [160] R. Likert, A technique for the measurement of attitudes., *Archives of Psychology*, 22(140):1–55, 1932.
- [161] J. Brooke, Sus-a quick and dirty usability scale, in *Usability Evaluation in Industry*, pages 189–194, CRC Press, 1996.
- [162] S. Chiasson, P. C. van Oorschot, and R. Biddle, A usability study and critique of two password managers, in *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, USENIX Association, Berkeley, CA, USA, 2006.
- [163] The realitygrid project, <http://www.realitygrid.org>.
- [164] S. Plimpton, Fast parallel algorithms for short-range molecular dynamics, *J. of Comp. Phys.*, 117:1–19, 1995.
- [165] J. W. Fenner, B. Brook, G. Clapworthy, P. V. Coveney, V. Feipel, H. Gregersen, D. R. Hose, P. Kohl, P. Lawford, K. M. McCormack, et al., The EuroPhysiome, STEP and a roadmap for the virtual physiological human, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1878):2979–2999, 2008.
- [166] The Virtual Physiological Human ToolKit, <http://toolkit.vph-noe.eu/>.
- [167] R. Magjarevic, K. Marias, V. Sakkalis, A. Roniotis, C. Farmaki, G. Stamatakos, D. Dionysiou, S. Giatili, N. Uzunoglou, N. Graf, R. Bohle, E. Messe, P. V. Coveney, S. Manos, S. Wan, A. Folarin, S. Nagl, P. Bchler, T. Bardyn, M. Reyes, G. Clapworthy, N. Mcfarlane, E. Liu, T. Bily, M. Balek, M. Karasek, V. Bednar, J. Sabczynski, R. Opfer, S. Renisch, and I. C. Carlsen, Clinically oriented translational cancer multilevel modeling: The ContraCancrum project, in O. Dssel and W. C. Schlegel, editors, *World Congress on Medical Physics and Biomedical*

- Engineering, September 7 - 12, 2009, Munich, Germany*, volume 25/4 of *IFMBE Proceedings*, pages 2124–2127, Springer Berlin Heidelberg, 2010, 10.1007/978-3-642-03882-2_564.
- [168] M. Bubak, T. Gubala, M. Kasztelnik, M. Malawski, P. Nowakowski, and P. Sloot, Collaborative virtual laboratory for e-Health, in *Expanding the Knowledge Economy: Issues, Applications, Case Studies, eChallenges e-2007 Conference Proceedings*, pages 537–544, Citeseer.
- [169] D. Groen, J. Borgdorff, C. Bona-Casas, J. Hetherington, R. W. Nash, S. J. Zasadada, I. Saverchenko, M. Mamonski, K. Kurowski, M. O. Bernabeu, A. Hoekstra, and P. V. Coveney, Flexible composition and execution of high performance, high fidelity multiscale biomedical simulations, *Interface focus*, 3(2):20120087, 2013.
- [170] D. Groen, S. J. Zasadada, and P. V. Coveney, Survey of multiscale and multiphysics applications and communities, *Computing in Science & Engineering*, 16(2):34–43, 2014.
- [171] The MAPPER Project, <http://www.mapper.eu>.
- [172] The European Grid Infrastructure, <http://www.egi.eu/>.
- [173] The VPH-Share Project, <http://www.vph-share.eu>.
- [174] P. Hunter, P. V. Coveney, B. de Bono, V. Diaz, J. Fenner, A. Frangi, P. Harris, R. Hose, P. . Kohl, P. Lawford, et al., A vision and strategy for the virtual physiological human in 2010 and beyond, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1920):2595–2614, 2010.
- [175] J. Pitt-Francis, P. Pathmanathan, M. Bernabeu, R. Bordas, J. Cooper, A. Fletcher, G. Mirams, P. Murray, J. Osborne, A. Walter, et al., Chaste: a test-driven approach to software development for biological modelling, *Computer Physics Communications*, 180(12):2452–2471, 2009.

- [176] K. Czajkowski, I. Foster, and C. Kesselman, *Grid 2: Blueprint for a New Computing Infrastructure*, chapter 18, 2004.
- [177] E. Huedo, R. S. Montero, and I. M. Llorente, Adaptive grid scheduling of a high-throughput bioinformatics application, in *Parallel Processing and Applied Mathematics*, pages 840–847, Springer, 2004.
- [178] E. Elmroth and J. Tordsson, A grid resource broker supporting advance reservations and benchmark-based resource selection, in *Applied Parallel Computing. State of the Art in Scientific Computing*, pages 1061–1070, Springer, 2006.
- [179] S. Venugopal, K. Nadiminti, H. Gibbins, and R. Buyya, Designing a resource broker for heterogeneous grids, Technical report, 2007, Grid Computing and Distributed Systems Laboratory, The University of Melbourne.
- [180] S. Venugopal, R. Buyya, and L. J. Winton, A grid service broker for scheduling e-science applications on global data grids, *Concurrency Computat.: Pract. Exper.*, 18(6):685–699, 2006.
- [181] L. Adzigogov, J. Soldatos, and L. Polymenakos, Emperor: An ogsa grid meta-scheduler based on dynamic resource predictions, *Journal of Grid Computing*, 3(1-2):19–37, 2005.
- [182] E. Elmroth and J. Tordsson, An interoperable, standards-based grid resource broker and job submission service, in *e-Science and Grid Computing, 2005. First International Conference on*, pages 9–pp, IEEE, 2005.
- [183] J. H. Abawajy, Robust parallel job scheduling infrastructure for service-oriented grid computing systems, in *Computational Science and Its Applications–ICCSA 2005*, pages 1272–1281, Springer, 2005.
- [184] A. Messer and T. Wilkinson, A market model for resource allocation in distributed operating systems, Technical report, 1995, City University, London.
- [185] M. Chen, G. Yang, and X. Liu, Gridmarket: A practical, efficient market balancing resource for grid and p2p computing, in *Grid and Cooperative Computing*, pages 612–619, Springer, 2004.

- [186] Q. Fu, S. Yang, M. Li, and J. Zhun, Decentralized computational market model for grid resource management, *Lecture Notes in Computer Science: Grid and Cooperative Computing*, 3033:227–230, 2003.
- [187] D. Abramson, R. Buyya, and J. Giddy, A computational economy for grid computing and its implementation in the Nimrod-G resource broker, *Future Generation Computer Systems*, 18(8):1061–1074, 2002.
- [188] J. Frey, T. Tannenbaum, M. Livny, I. T. Foster, and S. Tuecke, Condor-g: A computation management agent for multi-institutional grids, *Cluster Computing*, 5(3):237–246, 2002.
- [189] R. Buyya, D. Abramson, and J. Giddy, Nimrod/g: An architecture of a resource management and scheduling system in a global computational grid, *Proceedings of HPC Asia'2000*, 2000.
- [190] B. Sundaram and B. M. Chapman, Xml-based policy engine framework for usage policy management in grids, *Lecture Notes in Computer Science*, 2536:194–198, 2002.
- [191] C. Smith, Open source metascheduling for virtual organizations with the community scheduler framework (csf), Technical report, 2003.
- [192] C. Chapman, M. Musolesi, W. Emmerich, and C. Mascolo, Predictive resource scheduling in computational grids, in *IEEE International Parallel and Distributed Processing Symposium, 2007. IPDPS 2007*, pages 1–10, 2007.
- [193] L. Yang, J. M. Schopf, and I. Foster, Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments, in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 31, ACM, 2003.
- [194] W. Smith, V. Taylor, and I. Foster, Using run-time predictions to estimate queue wait times and improve scheduler performance, *Job Scheduling Strategies for Parallel Processing*, pages 202–219, 1999.

- [195] K. G. Binmore, *Fun and Games: A Text on Game Theory*, Houghton Mifflin, 1992.
- [196] W. Vickrey, Counterspeculation, auctions, and competitive sealed tenders, *Journal of finance*, pages 8–37, 1961.
- [197] P. Klemperer, Auction theory: a guide to the literature, *Journal of Economic Surveys*, 13(3):227–285, 1999.
- [198] K. Chatterjee and W. Samuelson, Bargaining under incomplete information, *Operations Research*, 31(5):835–851.
- [199] S. Beall, The role of reverse auctions in strategic sourcing, Technical report, 2003.
- [200] S. Jap, Online reverse auctions: Issues, themes, and prospects for the future, *Journal of the Academy of Marketing Science*, 30(4):506–525, 2002.
- [201] T. Matsuo and T. Ito, A designated bid reverse auction for agent-based electronic commerce, *Lecture notes in computer science*, pages 460–469, 2002.
- [202] D. Pardoe and P. Stone, Agent-based supply chain management: Bidding for customer orders, *Autonomous Agents and Multiagent Systems, International Joint Conference on*, 3:1442–1443, 2004.
- [203] Y. Che, S. S. R. Institute, and U. of Wisconsin-Madison, Design competition through multidimensional auctions, *The RAND Journal of Economics*, 24(4):668–680, 1993.
- [204] S. Thiel, Multidimensional auctions, *Economics Letters*, 28(1):37–40, 1988.
- [205] P. Cramton, Y. Shoham, and R. Steinberg, *Introduction to Combinatorial Auctions*, chapter 1, pages 1–14, The MIT Press, Cambridge, Massachusetts, 2006.
- [206] R. Wilson, Auctions of shares, *The Quarterly Journal of Economics*, pages 675–689, 1979.

- [207] A. Meisels and E. Kaplansky, Scheduling agents—distributed timetabling problems, in *Practice and Theory of Automated Timetabling IV*, pages 166–177, Springer, 2003.
- [208] W. Zhang and S. Xie, Agent technology for collaborative process planning: a review, *The International Journal of Advanced Manufacturing Technology*, 32(3):315–325, 2007.
- [209] F. Raimondi, Model checking multi-agent systems, Ph.D. thesis, University College London, 2006.
- [210] D. Weyns, T. Holvoet, and K. Schelfhout, Multiagent systems as software architecture: another perspective on software engineering with multiagent systems, in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1314–1316, ACM, 2006.
- [211] C. Petrie, Agent-based engineering, the web, and intelligence, *IEEE Expert*, 11(6):24–29, 1996.
- [212] M. Wooldridge and N. Jennings, Intelligent agents: Theory and practice, *Knowledge Engineering Review*, 10(2):115–152, 1995.
- [213] M. Wooldridge and A. Rao, *Foundations of Rational Agency*, Kluwer, 1999.
- [214] P. Novák and J. Dix, Modular BDI architecture, in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1009–1015, ACM, 2006.
- [215] K. Su, X. Luo, A. Sattar, and M. A. Orgun, The interpreted system model of knowledge, belief, desire and intention, in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 220–222, ACM, 2006.
- [216] G. I. Simari and S. Parsons, On the relationship between mdps and the bdi architecture, in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1041–1048, ACM, 2006.

- [217] M. Dastani, M. B. Van Riemsdijk, and J.-J. C. Meyer, Goal types in agent programming, in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1285–1287, ACM, 2006.
- [218] J. M. Bradshaw, *Software Agents*, MIT Press Cambridge, MA, USA, 1997.
- [219] M. Nikraz, G. Caire, and P. A. Bahri, A methodology for the development of multi-agent systems using the jade platform, *Comput. Syst. Sci. Eng.*, 21(2), 2006.
- [220] N. R. Jennings, E. H. Mamdani, J. M. Corera, I. Laresgoiti, F. Perriollat, P. Skarek, and L. Z. Varga, Using archon to develop real-world dai applications, part 1, *IEEE Expert*, 11(6):64–70, 1996.
- [221] M. Gierke, J. Himmelspach, M. Roehl, and A. M. Uhrmacher, Modeling and simulation of tests for agents, in Fischer, K and Timm, IJ and Andre, E and Zhong, N, editor, *MULTIAGENT SYSTEM TECHNOLOGIES, PROCEEDINGS*, volume 4196 of *LECTURE NOTES IN COMPUTER SCIENCE*, GI German SIG-DAI, SPRINGER-VERLAG BERLIN, 2006.
- [222] M. Wooldridge, N. R. Jennings, and D. Kinny, The gaia methodology for agent-oriented analysis and design, *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.
- [223] M. Wooldridge, *An introduction to multiagent systems*, John Wiley & Sons, 2009.
- [224] N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O’Brien, and M. E. Wiegand, Agent-based business process management, *International Journal of Cooperative Information Systems*, 5:105–130, 1996.
- [225] I. Ashlagi, D. Monderer, and M. Tennenholtz, Resource selection games with unknown number of players, in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 819–825, ACM, 2006.

- [226] G. James, D. Cohen, R. Dodier, G. Platt, and D. Palmer, A deployed multi-agent framework for distributed energy applications, in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 676–678, ACM, 2006.
- [227] J.-D. Kant and S. Thiriot, Modeling one human decision maker with a multi-agent system: the codage approach, in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 50–57, ACM, 2006.
- [228] S. Vanhastel, F. De Turck, and P. Demeester, Design of a generic platform for efficient and scalable clustercomputing based on middleware technology, in *First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001. Proceedings*, pages 40–47, 2001.
- [229] Z. Zhao, A. Belloum, P. Sloot, and B. Hertzberger, Agent technology and generic workflow management in an e-science environment, in *Grid and Cooperative Computing-GCC 2005*, pages 480–485, Springer, 2005.
- [230] I. Foster, N. R. Jennings, and C. Kesselman, Brain meets brawn: Why grid and agents need each other, in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 8–15, IEEE Computer Society, 2004.
- [231] T. Stockheim, M. Schwind, and O. Gujo, Agents bidding strategies in a combinatorial auction, in *Multiagent System Technologies*, pages 37–48, Springer, 2006.
- [232] G. Confessore, S. Giordani, and S. Rismondo, A market-based multi-agent system model for decentralized multi-project scheduling, *Annals of Operations Research*, 150(1):115–135, 2007.
- [233] G. Cabri, L. Leonardi, and F. Zambonelli, Implementing agent auctions using mars, Technical report, 2000, <http://sirio.dsi.unimo.it/MOON/papers/papers.html#Paper18>.

- [234] R. Dash, N. Jennings, and D. Parkes, Computational-mechanism design: A call to arms, *IEEE Intelligent Systems*, 18(6):40–47, 2003.
- [235] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing multi-agent systems with JADE*, John Wiley & Sons, 2007.
- [236] D. Nurmi, J. Brevik, and R. Wolski, QBETS: Queue bounds estimation from time series, *Lecture Notes in Computer Science*, 4942:76, 2008.
- [237] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Penanen, K. Popov, V. Vlassov, and S. Haridi, Peer-to-Peer resource discovery in Grids: Models and systems, *Future Generation Computer Systems*, 23(7):864–878, 2007.
- [238] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, Benchmarks and standards for the evaluation of parallel job schedulers, in *Job Scheduling Strategies for Parallel Processing*, pages 67–90, Springer, 1999.
- [239] SpotCloud, <http://www.spotcloud.com>.
- [240] J. Roovers, K. Vanmechelen, and J. Broeckhove, A reverse auction market for cloud resources, in *Economics of Grids, Clouds, Systems, and Services*, pages 32–45, Springer, 2012.