University of Southampton

Service-Oriented Grids and Problem Solving Environments

by

Matthew J. Fairman

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the
Faculty of Engineering, Science and Mathematics
School of Engineering Sciences
Computational Engineering and Design Group

September 2004

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ENGINEERING SCIENCES
COMPUTATIONAL ENGINEERING AND DESIGN GROUP

Doctor of Philosophy

by Matthew J. Fairman

The Internet's continued rapid growth is creating an untapped environment containing a large quantity of highly competent computing resources suitable for exploitation in existing capacity-constrained and new innovative capability-driven distributed applications. The Grid is a new computing model that has emerged to harness these resources in a manner that fits the problem solving process needs of the computational engineering design community. Their unique requirements have created specific challenges for Grid technologies to bring interoperability, stability, scalability and flexibility, in addition to, transparent integration and generic access to disparate computing resources within and across institutional boundaries.

The emergence of maturing open standards based service-oriented (SO) technologies has fulfilled the fundamental requirements of interoperability, leaves a flexible framework onto which sophisticated system architectures may be built, and provides a suitable base for the development of future Grid technologies. The work presented in this thesis is motivated by the desire to identify, understand, and resolve important challenges involved in the construction of Grid-enabled Problem Solving Environments (PSE) using SO technologies. The work explains why they are appropriate for Grid computing and successfully demonstrates the application and benefits of applying SO technologies in the scenarios of Computational Micromagnetics and Grid-enabled Engineering Optimisation and Design Search (Geodise) systems. Experiences achieved through the work can also be of referential value to future application of Grid computing in different areas.

Contents

List of Figures

List of Tables

Listings

Declaration Of Authorship

I, *Matthew J. Fairman*, declare that the thesis entitled *Service-Oriented Grids and Problem Solving Environments* and the work presented in it are my own. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published as: journal and conference papers and posters, and technical documentation listed in appendix A.

Signed: .... Date: ....

Acknowledgements

All the work contained in this thesis is my own except for the following sections and contributions. Section 4.3.3 contains shared ideas between Gang Xue and I on legacy system Web Service wrapping. Gang Xue implemented the communication layer of the Optimisation Service, see chapter 5. The ClassAds mapping to XML, section 6.5, of the Computation Web Service, chapter 6, was jointly designed and implemented by myself and Gang Xue. The Geodise system, section 3.3, is a collaborative project[1] between the Universities of Southampton, Oxford and Manchester with several industrial partners, lead by Professor Andy Keane, Professor Simon Cox, Professor Mike Giles, Professor Carole Goble, and Professor Nigel Shadbolt.

[1]Geodise project home page - http://www.geodise.org/

I dedicate this to my parents for the years of patience and support that they have given me during my lengthy stay at school.

Chapter 1

Introduction

The Grid is a global-scale endeavour to make distributed computing resources generically accessible and easily sharable on the Internet. It distinguishes itself from previous distributed computing efforts by its intent to provide global-scale, dynamic and transparent access to computer resources such as PCs, servers, compute clusters, data storage devices, and instrumentation, in a way akin to the consumption of electricity from the power grid. A great number of research projects have been created to study solutions to the challenge of building the Grid, ranging from its high-level application in engineering sciences [1] to Grid architecture and protocols [2, 3]. The Grid will lead to important changes to the usage of computing in general by both enabling access to previously unobtainable resources and facilitating their collaboration in sophisticated large-scale cross-organisational applications.

The concept of the Grid has arisen from the desire to create technologies that improve exploitation of computing resources on the Internet to service the demands of business and engineering science for more powerful distributed systems that can perform larger and further-reaching roles. The continued rapid growth of the Internet is providing a greater capacity of increasingly capable computing resources. Distributed computing technologies developed before the arrival of the Grid do offer access and sharing of Internet computing resources however only the technologies associated with the World Wide Web (WWW) and e-Mail have scaled to the size of the Internet.

The Internet has grown to become a successful and powerful system that is encouraging the sharing of information and data. It has brought new opportunities for novel applications of computer resources and technologies. The WWW and e-Mail are today's biggest applications on the Internet. These distributed applications have reached a global scale and their influences can be seen in almost all corners of human life including education, commerce, business and entertainment. The WWW and e-Mail have enabled information and data to be simply and quickly distributed, exchanged, and accessed despite geographic boundaries.

Commoditisation of computing resources such as, Personal Computers (PCs), and operating ease of the WWW and e-Mail have been the main driving factors in growth in usage of technology and consequent explosion in novel Internet-based applications. This revolution in the capacity and capability of computing resources would not have been achievable without creation of an environment in which people may fully exploit available resources without becoming computer engineering experts in the underlying systems; in a similar way that most drivers of cars should not need to be expert automotive engineers. For example, modern graphical user interface-based operation systems, such as Microsoft Windows and Apple's OS X, have hidden the unnecessary complexity of using computers whilst allowing people to use their machines for sophisticated tasks such as accounting, word processing, and on-line banking. People can simply and quickly upload there pictures from the cameras and share them on the web, e-mail documents to each, and share their printers or Internet connection amongst their home computers.

The desire to share and integrate information and data across the Internet in not only a manner similar to the WWW and e-mail but also a wider selection of computer resources has spawned the research on Grid computing. The Grid will both simplify and ease the task of using distributed computing resources whilst enabling people to exploit them for sophisticated tasks without having to be a computer scientist. Figure 1.1 shows a layered diagram of where the Grid proposes to provide services for applications.

This thesis describes research into the use of service-oriented (SO) based Grid technologies in the fields of computational engineering and science. The rise of the Grid has been driven by the needs of these fields for improvements in the ability to easily share and access a wider variety and greater quantity of computer-based systems, in addition to, knowledge, information and data. Computational engineering and science presents a significant challenge for the Grid in that its demands and level of operation far exceeds the simple-sharing mode of the Internet. Generally, these fields employ computer resources together with software as tools in a wide variety of sophisticated configurations in order to perform high-level design and problem solving processes.

Service-orientation [4] is a new distributed software architecture that promises to facilitate Grid computing and provide significant advantages over classical distributed computing methodologies. The Grid offers features, mainly focused around SO and its associated technologies, which will benefit computational engineering and science. However, as we shall now discuss, starting with the features of the Grid, the computing requirements of these fields requires significant research from many areas of computer science.

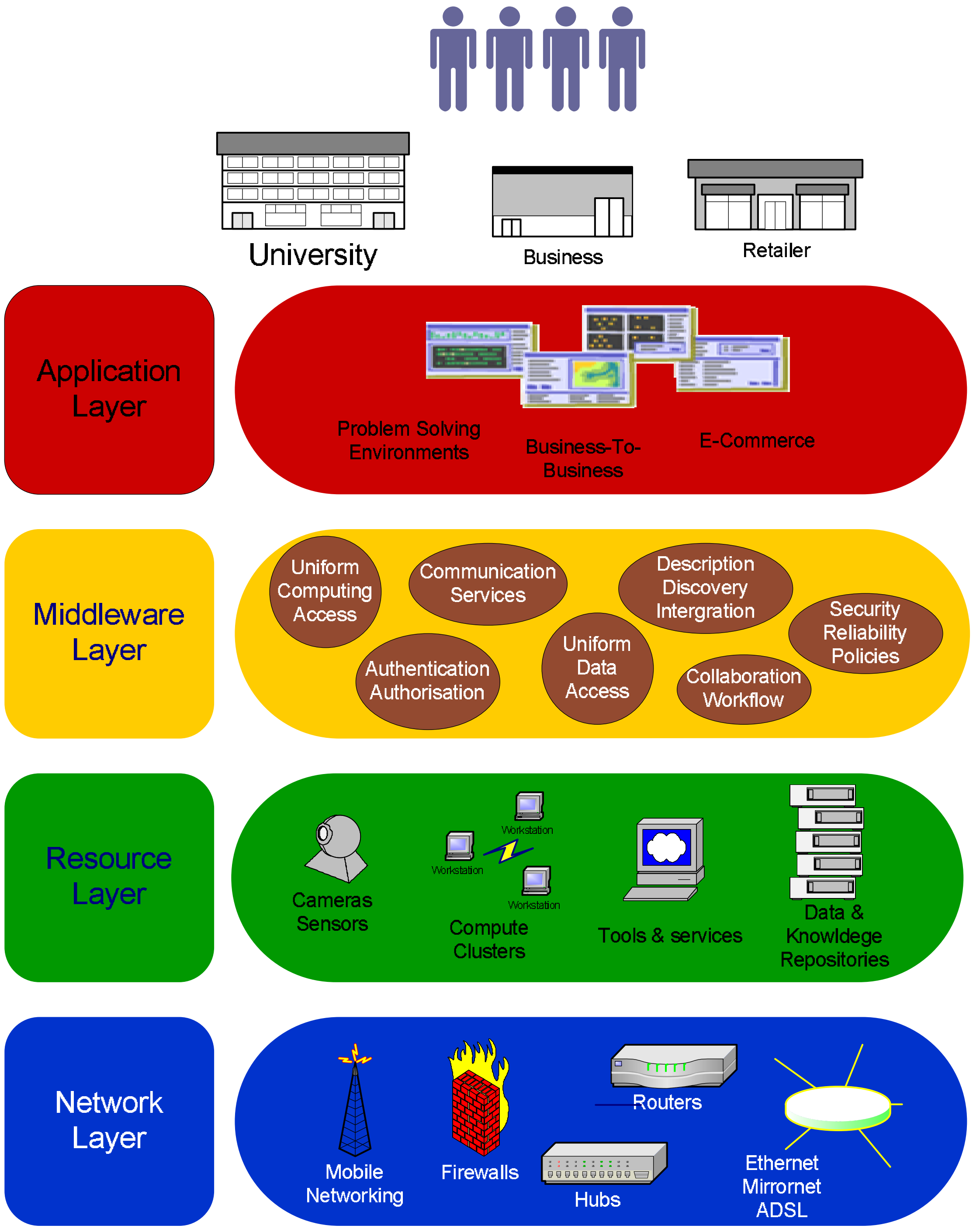


FIGURE 1.1: Layered diagram showing the features of the Grid
The middleware layer contains the Grid technologies that enable generic access and seamless integration of distributed computer resources for the construction of sophisticated applications.

1.1 Features of the Grid

The Grid aims to build on the capabilities of the Internet to provide a global-scale infrastructure which allows generic access to computer resources and their collaboration in sophisticated application systems. The Grid is defined by CERN, the largest consumer of Grid computing in the world, as *"a service for sharing computer power and data storage capacity over the Internet"* [5]. However, it is more than this. The Grid is the next level beyond the Internet. It will commoditise high-level distributed computing by enabling access and seamless integration of distributed computer resources without need for users to understanding how it is done.

Service-Orientation and Common Standards Computer resources will play a much more dynamic service-orientated role, which will allow their collaboration in a multitude of application roles. Standard service-oriented based technologies will provide simple, uniformly accessible, and interoperable computer resources. The Grid aims to provide technologies that, in addition to allowing the sharing of resources in an interoperable manner, also simplify the task of constructing distributed systems. The Grid has benefited from the emergence of the open standard eXtensible Markup Language [6] (XML)-based Web Service Architecture (WSA) [7] technologies. These simple, loosely-coupled Service-Oriented (SO) technologies have provided the elementary building blocks for constructing scalable, easily maintainable and extensible infrastructures for sophisticated distributed environments. The work presented details the SO approaches for Grid computing that facilitate and simplify the integration and exploitation of distributed computing resources.

Emphasis on Data, Information and Knowledge At all levels of the Grid data, information, and knowledge will provide an extremely important role for purposes from aiding in the discovery and integration of resources to providing feedback to users on how to better their results. The Grid provides XML-based technologies that enable definition of common schema essential to providing uniform data access to computer resources. In addition, XML ability to self-describe facilitates data to be more easily cross-referenced, combined, and incorporated into sophisticated information and knowledge-driven applications.

Access and Sharing of Distributed Resources The Grid will allow access and sharing of computer resources across and within institutional and organisational boundaries. Grid computing will feature common authentication and authorisation mechanisms that shall enable resource providers to safely share their computer resource to users inside and outside their administrative domains. In addition, Grid computing gives resource providers common control and management technologies that enable the definition of usage policies, quotas, and qualities of service.

End users will benefit from a broader selection and greater quantity of computer resources.

Dynamic Workflows and Description, Discovery and Integration Finally, the Grid also features support for high-level computer resource sharing operations for the description, discovery, and integration of computer resources, and their collaboration in dynamic workflows. Consequentially, advanced applications will be able to cater for changing requirements by dynamically selecting resources on a per usage basis. Consequently, these features will bring about increased availability through better aggregation and easy location and integration of computer resources.

These are the key features of Grid computing which make the Grid appealing to traditional users of distributed computing. In the following section, we look at the problem solving requirements of the fields of computational science and engineering and how they match with the features provided by the Grid.

1.2 Applying Grid Computing for Problem Solving Environments

The fields of science and engineering have often demanded or benefited from computers. Typically, these fields require access to a heterogeneous range of distributed computer resources such as databases, computing clusters, application servers, and instrumentation. Scientists and engineers often need to couple together a wide range of tools to solve new problems: hardware and software for setting up and solving problems and tools to analyse their results and guide future studies. In a typical scenario of, for example, wing design, an engineer may couple Computer Aided Design (CAD) tools, analysis codes for Computational Fluid Dynamics (CFD), or Finite Element Analysis (FEA) and tools for optimisation.

There is a desire for problem solving environments (PSE) that facilitate the solving of scientific and engineering problems from simulating the effects of folding molecules to find new drugs, to modelling the flow of air over a wing to produce aircraft that are more efficient. PSEs may be described by [8] (see section 3.1 for the full definition) who state *"A PSE is a computer system that provides all the computational facilities necessary to solve a target class of problems."*

Lack of computer resources and the inability to bring available resources together into effective problem solving environments (PSEs) often limits the depth and complexity of research. Typically, scientists need to have a good understanding of the inner workings of the tools they are using in order to effectively use and integrate them with other tools. Here are some of the difficulties that scientist and engineers face:

Proprietary File Formats and Data Models Whilst standards exist they have not been commonly used across all tools and computing resources. In the case of CAD tools, typically each package stores information internally in a format that is best suited for that package. These proprietary formats can only be read by the specific tool (and version of software), and are virtually useless to other tools. Often this forces engineers working on the same problem to all use the same CAD tool in order to exchange files. In addition, in the case of wing design where CAD tools are often required to create the meshes for CFD tools, the file output by the CAD tool must be in a format and have a data model, which is understood by the CFD tool. There is a clear need for common standards and data models across tools to facilitate the exchange of data amongst people and tools and to allow a greater freedom in the choice of tools. The Grid meets the requirement by offering technologies and mechanisms that enable the simple and easy definition of common standards, in addition to, to the ability to self describe data (metadata) to aid in creating composite data models. For instance, the Standard for the Exchange of Product Model Data (STEP) [9] has significantly helped with the exchange of engineering data. This comprehensive standard (ISO 10303) describes ways to represent and exchange digital product information. Currently, most major CAD/CAM system now contain modules to read and write data defined by at least one of the STEP Application Protocols (AP's), in particular AP-203 by USA software vendors and AP-214 in Europe. These protocols are similarly used to exchange data describing designs represented as solid models and assemblies of solid models.

Ineffective Knowledge Capture and Reuse Engineers will often need specific solution methodologies in order to solve or quickly optimise a problem. For instance, in the field of micromagnetics where scientists study the shape and arrangement of particles in magnetic systems to improve areal density of hard drive platters, the conjugate gradient method (CGM) is the preferred method for optimisation. However, CGM sometimes does not converge on a solution for certain arrangements and shapes of particles in the magnetic system in which case the more complex and computationally intensive Landau-Lifshitz-Gilbert (LLG) [10] equations must be employed. The wrong selection of solution methodology may result in incorrect, poor or no results, and inefficient usage of computer resources.

However, knowing which solution methodology is best often comes down to the experience of the scientist or engineer. Whilst engineering books and the WWW provide repositories of information and knowledge on solution methodologies however; they cannot reflect the particular problem the engineer is working or offer the capability to suggest solution methodologies or suitable computing resources for specific instances of the problem. In addition, there is a lack of links between users and PSE. A solution methodology proven in one area often gets lost because there is no common way to describe or capture the gained knowledge and thus reuse it.

There is a requirement here to more effectively capture gained knowledge, provide information on solution methodologies specific to the target problem, and offer means to help decide on solution methodology and aid in their effective usage. The Grid enables this through its aim of interoperable amongst resources such that data and information may be more easily described and related allowing complex work flows to be logged, analysed, and reused.

Lack of Resource Sharing and Accessibility Within the fields of academia and research, the size and complexity of a project sometimes exceeds the capabilities of single institution, consequently it is not uncommon for work on projects to span universities and companies. Typically, computer resources such as compute clusters and databases are shared so that people working together on cross-institutional projects can more easily collaborate and exchange information. However, challenges exist with ensuring that computer resources are protected from malicious users, erroneous software, and accidental damage. The challenge is compounded by the proliferation of different security mechanism and varied types of computer resource. The Grid offers the interoperability fundamentally necessary to facilitate large-scale resource sharing, in addition, to common powerful security mechanisms.

However, further research and studies are required to resolve key challenges that prevents building effective Grid-enabled PSEs. Whilst the Grid provides the building blocks and enabling technologies, each application field presents its own unique and domain-specific problems. In order to cater for application diversity, the Grid needs to encompass many areas of research in computer science. Its development has been inspired by advancements in distributed and parallel computing, software architecture, data management and transmission, middleware, network protocols, and standards for cross-platform interoperability. The number of challenges imposed through the rationalisation of these technologies into global systems puts the Grid at the forefront of applied computer science research.

1.3 Scope of Work

This thesis will study the challenges posed in the creation of effective SO Grid-enabled PSEs. The focus will be placed on typical scenarios of engineering optimisation and design search, and computational micromagnetics because these research fields will be better able to solve their target problems with sophisticated PSEs.

Generally, Grid research can be separated into two categories: the design and development of Grid technologies and the study of their application. The work presented in this thesis serves both purposes. We will explore the application of SO technologies in the field of science and engineering by analysing the domain's requirements and attempting to provide SO based solutions for construction of sophisticated Grid-enabled PSEs. This thesis will draw from the experiences learned from previous efforts on distributed and parallel computing and study current philosophies prompting Grid research. This thesis aims to identify key unresolved issues related to the exploitation of Grid technologies in engineering science and demonstrate the methodologies and work carried out for their resolution. We validate our work by discussing the successful demonstration of its concrete implementations in systems for micromagnetics, and engineering optimisation.

Micromagnetics contains various frontier problems which mean they have no analytical solutions and cannot be solved using trivial computation methods. Typically, the study of micromagnetics problems requires use of advanced modelling and material meshing tools, access to data on materials properties, and large amounts of computation time using parallel and distributed compute clusters for optimisation. Similarly, engineering design search requires access and integration of many types of application tools for modelling and simulation, database containing domain specific information and data, and compute clusters for analysis and optimisations.

It is strongly desired for those who work in both research fields to bring together resources to solve a particular problem and access tools for searching solution methodologies and analysing results to further better study. These two fields represent the key challenges for sophisticated PSE to provide interoperability and generic access for heterogeneous resources and to enable seamless integration of resources and data.

1.3.1 Aims and Contributions of the Research contained in this Thesis

This chapter began with an introduction to the concept of the Grid and highlighted its important features. This was followed by a discussion of how problems exist in the fields of science and engineering thats research would benefit from PSEs which enable better access and integration of computer resources. Key technological issues in problem solving where established that where then matched to the features of the Grid which offered potential solutions. From this we concluded that the Grid provides technologies and concepts that form the basis for building environments that will reduce technological burdens and aid inquiry. However, due to the depth and breadth of science and engineering research and the full range of computing science that it demands, much investigation still remains. We now enumerate the key aims of the research in this thesis.

1. Define the Grid, discuss the reason behind it and what came before it, specify its aims and proposed features. This also includes studying existing distributed technologies to identifying what good practices, architectures and technologies the Grid should adopt or build upon and the poor practices that it must learn from.

2. Expose Web Service technology and its adoption of the concept of loose-coupling as the current best approach to building Grids that will scale to the size of the internet and provide features required to build effective distributed computing environments.

3. Study the problem solving process, identifying its current challenges and solutions provided by the Grid and remaining unresolved issues.

4. Show how Grid-enabled PSEs offer scientists and engineers with a means to compose complex workflows and loosely couple together resources and tools that allow them to focus on the problem at hand rather than intricacies of the underlying technologies.

5. Find methodologies and rules of thumb for how services can and should be built and aggregated in order to better provide a loosely-coupled environment for problem-solving.

6. Give examples of how key components within the micromagnetic and engineering design optimisation problem solving processes can be offered as services and highlight the significant advantages of a Service-Oriented Architecture. Specifically, show how new tools maybe be offered to scientists by building a native service that offers an optimisation algorithm. In addition, show how to leverage, reuse and add additional functionality to legacy technology of a computational cluster by wrapping it as a service.

7. Finally, provide an example that brings together the knowledge that has been gained by integrating the developed services into a Grid-enabled PSE that can aid solve a real-world micromagnetics problem.

The work contained within this thesis focuses on improving the ability of scientists and engineers to do their work by identifying computational challenges that face them and finding the computer science solutions. Specifically this thesis focuses on improving usability of existing software libraries and distributed resources, demonstrated through micromagnetics and engineering design and optimisation. However, the breadth of work required to thoroughly cover Grid-enabled PSE makes this thesis discoveries also beneficial to Grid computing in general. This thesis is foremost a contribution to not to Grid computing but also a good working example of a practical application of computer science in science and engineering.

1.4 Thesis Structure

This thesis is structured as follows: Literature Review, Problem Statement and Solution Methodologies, Description of Work and Results. The first two chapters concentrate on studying the background and purpose of the SO Grid computing and the consequent requirements placed on it by Grid-Enabled PSE. The following chapters represent the work we have carried out to solve the challenges exposed, ending with its validation in the Results.

1.4.1 Part 1: Literature Review

The literature review, chapter 2, explains the ideas behind SO Grid computing. This chapter will introduce the Grid concepts and technologies that will be drawn upon in the following chapters. It discusses the background, advantages and unique features of Grid computing, and goes on to discuss the consequent requirements placed on enabling technologies. Specifically, analysis will begin with a review of the key technologies and concepts that have been applied to the Web and traditional distributed computing systems. The literature review goes on to investigate Web Service technologies and explains why SO fulfils the key requirements of Grid computing. In addition, we discuss important alternate SO approaches.

1.4.2 Part 2: Problem Statement and Solution Methodologies

The problem statement, chapter 3, analyses the specific challenges and requirements placed on SO technologies by Grid-PSEs. We will look at current bespoke and legacy applications and systems used for modelling, simulation and analysis. In particular, we will examine the field of computational micromagnetics, and engineering optimisation and design search. We extract from these the Grid requirements demanded by sophisticated PSEs in addition to computation, such as collaboration and seamless integration of different types of tools and heterogeneous resources to yield improved operational results. From these we will identify specific important challenges, rationalising our choice of these for the basis of our work and subsequent thesis discussion. The chapter will conclude with solution methodologies that will facilitate the simple and seamless integration of new and legacy computer resources which, in addition, will enable previously impossible PSE features, such as knowledge capture, workflows and advanced security. All work presented in the following chapters has been carried out in the context of GEODISE and computational micromagnetics.

1.4.3 Part 3: Description of Work and Results

The last chapter (6) explores different aspects of SO Grid-enabled PSE: the specific computer resources typically required for engineering and science work, and their sharing, integration and collaboration. Chapters 5 and 6 respectively look at the application tools and computational components of PSE; two key elements utilised by both computational micromagnetics, and engineering and design search, specifically in the scenarios of optimisation and design of experiment.

Work on transforming numerical optimisation algorithms into Web Services demonstrates the effects of SO in simplifying collaboration and serves as a reference for construction of different types of services for design optimisation. The proceeding chapter describes work on virtualising and sharing computer resources in a SO manner for Grid computing. The challenges of state management, legacy system integration and security will be explored in this chapter through the discussion of the building of a Computational Web Service. Its design and implementation is illustrated by the successful deployment and integration of the Condor High Throughput Computing system.

In the final work related chapter (7) of this thesis, we describe message passing technologies and concepts for the access and integration of computer resources for computational micromagnetic processes. We discuss the building of a Grid-enabled PSE that adopts an extremely loosely-coupled style to enable a level of interaction amongst resources in a way not previously seen in Grid computing. The micromagnetics PSE demonstrates sophisticated workflow execution possibilities and seamlessly integrates computational and application services using a document-orientated message exchange paradigm. The new Grid technology of WS-Addressing and the capabilities of SOAP message exchange patterns are exploited to route messages through services to implement workflows and to offer the possibility of knowledge capture and reuse. SO methodologies discussed in the previous chapters are drawn upon to construct new services suitable for micromagnetics. The design and approach of the Grid-enabled system is exemplified by the successful integration of micromagnetics tools with the Condor system, and the production of real-world results that otherwise would have been too difficult to achieve.

Chapter 2

Literature Review of Distributed Computing

Service-Oriented Architectures (SOAs) express a perspective of software architecture that defines the use of loosely-coupled, highly interoperable services to support the requirements of software users. The Grid is characterised by its attempt to provide dynamic and coordinated large-scale resource sharing through service-oriented (SO) Web Service technologies. This chapter rationalises the usage of SOA by showing that its promotion of reuse and interconnection of resources facilitates cost-effective and quick adaption of distributed computing environments to changing users requirements. This is in contrast to the more expensive and time-consuming reinvention often required by traditional distributed computing architectures.

SOA is an evolution of the Component Based Architecture, Interface Based Design (Object-Orientation) and Distributed Systems of the 1990s, such as DCOM, CORBA, J2EE and the Internet in general. It is not a revolution because it attempts to capture and build upon the best practices of the architectures that came before it. In addition, the continued rapid growth of Internet has initiated much distributed computing research that aims to improve the ability to exploit and aggregate the vast amount of computer resources connected to the Internet. Therefore, this chapter begins with a review of the most successful technologies and best practices in distributed computing as it provides valuable information for Grid computing research.

Whilst distributed computing research has produced successes and discovered many best practices, typically these exist only in specific areas or are tailored to resolve only particular challenges. The grand ideological goals of Grid computing will require the drawing upon and bringing together of almost all areas of distributed computing which itself will likely require changes or development of new approaches and technologies in order to offer a workable environment. The second section of this chapter will look closely at the idea of SOA and discuss key challenges including: interoperability, security and reuse, and will compare the Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) based messaging style implementations. In addition, it will study Web Services which provide a new technology built on SO principles that has been largely developed and utilised for Grid computing.

Most definitions of SOA identify the use of Web Services using either SOAP or REST based messaging style in its implementation. However, a SOA distributed environment may be built using any service-based technology. The last section of this thesis looks at these alternate SO technologies.

2.1 What is Distributed Computing?

Distributed computing is the study of the coordinated usage of often physically distributed computers. It refers to the achievement of desired computational functionality and results through the coordination and collaboration of components of distributed, networked computers using only message passing. Software of a distributed computing system can be thought of as the high-level glue that provides the links between remotely distributed computing resources of which its components unify and coordinate the remote computer resources into a whole system.

Distributed computing systems serve a variety of purposes from providing low-level network infrastructures to resource sharing, and high-level application services. Andrew S. Tanenbaum [11] states that, *"Distributed systems need radically different software than centralised systems do"*. The software, network topologies, and resource types contained within distributed system vary greatly depending on their purpose. Consequently, there are many different types of distributed computing systems and many challenges to overcome in successfully designing one.

Distributed computing systems offer greater performance, larger capabilities, and more scope for resource sharing than would be possible with individual computers. They can be roughly categorised into two forms: clustered systems usually consisting of a group of computers that share the same software and work closely together, often in parallel, so that they appear as a single computer; and secondly, distributed systems made up of stand-alone computers that provide distinct, independent services.

Distributed systems have a broader range of usages including organisational functions and information distribution. Unlike clustered systems, computer resources within a distributed system often have no links and are generally independent of each other. The Grid is unique in computing in that it aspires to be the computing system that incorporate all other forms of distributed computing systems. It aims to encompass both distributed and clustered systems, offering simple and transparent access to both.

2.2 Key Features required in Distributed Computing Systems

Distributed systems aim to connect users and computer resources in a transparent, open, and scalable way, which we shall now look at in turn. Ideally, this organisation should be considerably more fault tolerant and more powerful than many combinations of stand-alone computer systems, such as workstations, and PCs. The Grid must embrace these characteristics in order for it to achieve generic accessibility and resources sharing on a large-scale.

2.2.1 Transparency

Transparency is the ability of a system to hide its distributed nature from its users so that it appears and functions as a centralised system. Transparency has many forms:

Access transparency: Regardless of how resource access and representation has to be performed on each individual computing entity, the users of a distributed system should always access resources in a single, uniform way.

Location transparency: Users of a distributed system should not have to be aware of where a resource is physically located.

Migration transparency: Users should not be aware of whether a resource or computing entity possesses the ability to move to a different physical or logical location.

Relocation transparency: Should a resource move while in use, this should not be noticeable to the end user.

Replication transparency If a resource is replicated among several locations, it should appear to the user as a single resource.

Concurrency transparency: While multiple users may compete for and share a single resource, this should not be apparent to any of them.

Failure transparency: Always try to hide any failure and recovery of computing entities and resources.

Persistence transparency: Whether a resource lies in volatile or permanent memory should make no difference to the user.

Transparent access is an especially important feature desired of the Grid because it promotes usage by reducing the demands on the users. Transparency enables users to access resources as if they are local to their computer by hiding its distributed nature.

This simplifies the resource's integration into applications and allows users to concentrate on using its functionality. In addition, the Grid is envisioned to contain resources from many different providers that may contain different underlying implementations. Therefore, it is desirable to hide this to facilitate interchange of resources from different providers.

However, the degree to which transparency could or should be achieved may vary widely. Not every system can or should hide everything from its users. In Grid computing, all of these properties will be required but not necessarily for every purpose or for every system built from Grid technologies. For instance, users may actually need to know where a resource is located because as distances between computer resources increase, communication latencies worsen and the cost of network transport of data increases. For user-driven applications such as, Computer Aided Design, which typically requires access to shared design databases and powerful computation resources through graphical user interfaces, high latency can seriously decrease system responsiveness and consequently decrease users' productivity. In addition, users of resources for critical applications may wish to know the type of persistence that a resource uses so that they can target machines that employs robust forms of persistence to reduce the risk of losing data.

2.2.2 Openness

Openness is the property of distributed systems that measures the extensibility and scalability of its standardised interfaces. A system that easily allows connection of more computing resources and features has an advantage over an absolutely closed and self-contained system. Typically, this is achieved by capturing the syntax of the services offered in a system through an Interface Definition Languages (IDL). Consequently, open distributed systems are required to meet the following challenges:

Monotonicity: Once something is published in an open distributed system, it cannot be taken back.

Pluralism: Different subsystems of an open distributed system include heterogeneous, overlapping and possibly conflicting information. Ideally, there is no central arbiter of truth in open distributed systems.

Unbounded nondeterminism: Asynchronously, different subsystems can come up and go down and communication links can come in and go out between subsystems of an open distributed system. Therefore, the time that it will take to complete an operation cannot be bounded in advance.

Openness is a characteristic necessary to the ability of a system to grow in scale, functionality, and capability. As will be discussed in section 2.4.1, the Internet's foremost application, the World Wide Web (WWW), is an extremely open systems by design which has consequently enabled it to scale to a world-wide system, as its name suggest. The Internet itself is perhaps the most open distributed system of all and key to this ability was the standardisation of simple, application-neutral messaging and low-level data transmission formats and protocols. In addition, by design the Internet's structure, largely built and extended from ARPANET (a computer network designed to survive a nuclear attack), has no centralised resources or single points of failure.

2.2.3 Scalability

A scalable system is one that can easily be altered to accommodate changes in the number of users, resources, and computing entities attached to it. Scalability is a highly desirable feature in distributed systems. Scalability can be measured in three different dimensions:

Load scalability: The quality of a distributed system to easily expand and contract its resource pool to accommodate heavier or lighter loads.

Geographic scalability: The quality of a distributed system to remain useful and usable regardless of distances between users or resources.

Administrative scalability: The quality of a distributed system to remain easily reusable and maintainable regardless of how many different organisations share it.

Scaling a system vertically, or scaling up, means to add resources to a single node, such as upgrading a computer with a faster CPU or larger hard drive, whilst horizontal scaling, or scaling out, means to add more resources to the distributed system, such as adding extra nodes to a computer cluster. Vertical scaling is often more expensive than horizontally scaling because the cost of buying, fitting and tuning the upgraded resources is typically close to the cost of just adding an additional node. The performance returns from vertical scaling are often higher than from horizontal scaling because the ability of a system to scale vertically is localised to the node itself where as horizontal scaling is restricted by the ability of the whole system [12, 13, 14]. Nonetheless, improvements in individual computer performance can only go so far and are constrained by the ability of companies to produce faster or better components. Scaling out, whilst the returns are not as good, remains the only practical means for a system to easily expand its capacity to a size demanded by today's enterprise and internet wide applications. In addition, scaling out a distributed system, especially when nodes are loosely-coupled (see section 2.5.2.3), can help with overall system reliability by providing additional redundant resources.

2.3 Architectural Patterns

It is good practice in any software endeavour to adopt an architectural pattern that eases the maintenance and reuse of functionality. Without this, as software grows in size and purpose it quickly becomes very brittle (i.e difficult to change and fix). The principle of good architecture becomes especially important in a distributed system where functional components may exist in different localities consequently amplifying the challenge. Brittle systems do not scale and are hence not capable of reaching an enterprise-level. Component- and Interface-based architectures, which we shall now look at in turn, are common architectures employed in distributed computing which is being built upon by SOA.

2.3.1 Component-based Architecture

In software design, a component-based architecture divides the functionality of the whole into smaller functions, each encapsulated as a independent component. A distributed system may be thought of as an extension of component-based architecture where components may exist in different physical locations. Figure 2.1 shows a simple e-Mail distributed system which employs a component-based architecture. Harry's and Sally's clients, a Domain Name Service (DNS), and e-Mail servers are all interacting components of the system but are often located in different physical locations.

The main advantage of a component-based architecture is that it facilitates reusability and repurposing of components and that it makes maintenance easier; all essential features of a SOA. If these components are distributed it qualifies it as a SOA. The system, shown in figure 2.1, is also an example of a SOA because the components are distributed, loosely-coupled and provide interoperable, independent services. Component reusability and repurposing are primary business drivers for adopting systems with SOAs.

2.3.2 Interface-based Object-Oriented Architectures

Interface-based Architectures extends an object-oriented programming system by allowing components known as objects to be distributed across a heterogeneous network, so that each of these distributed object components interoperate as a unified whole. These objects may be distributed on different computers throughout a network, living outside of an application, and yet appear as though they were local to an application.

A remote procedure call (RPC) is a protocol that allows a computer program running on one host to cause code to be executed on another host without the programmer

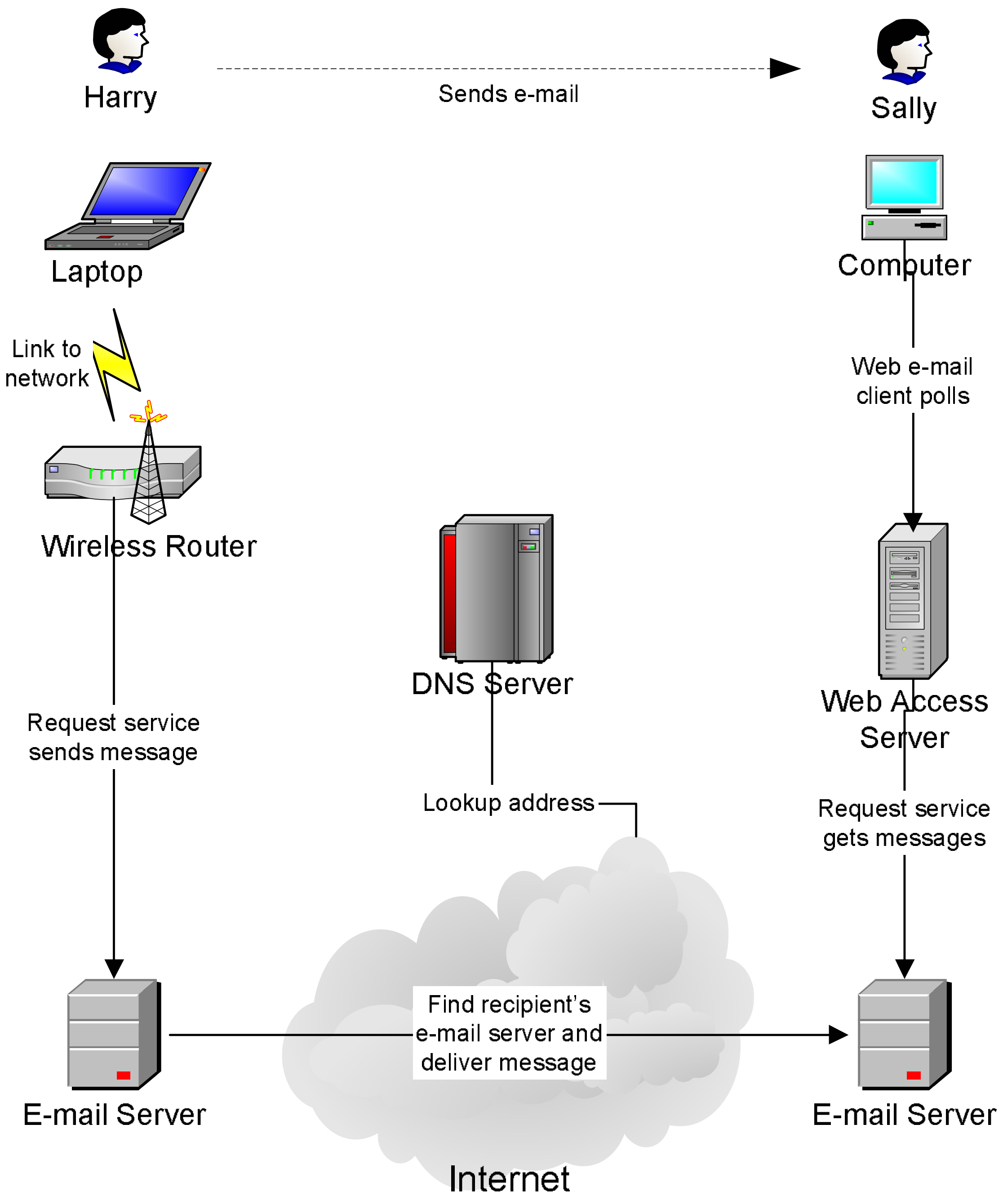


FIGURE 2.1: A simple example of a distributed, component-based architecture of e-Mail.

Distributed Object Technology	**CORBA**	**DCOM**	**Java RMI**
Interface Definition	CORBA - IDL	DCOM - IDL	Java Interface Definition
Remoting Protocol	Internet Inter-ORB Protocol (IIOP)	Object Remote Procedure Call (ORPC)	Java Remote Method Protocol (JRMP)
Platform Support	Any platform with CORBA ORB implementation	Any platform with COM Service implementation	Any platform with Java Virtual machine implementation
Object Name-Implementation Mapping	System (Windows) Registry	Implementation Repository	RMIRegistry
Object Implementation Locating	Service Control Manager	Object Request Broker	Java Virtual Machine

Table 2.1: Comparison of Distributed Object Technology features.

needing to explicitly code for this. When the code in question is written using object-oriented principles, RPC is sometimes referred to as remote invocation or remote method invocation.

RPC is an easy and popular paradigm for implementing the client-server model of distributed computing. An RPC is initiated by the caller (client) sending a request message to a remote system (the server) to execute a certain procedure using arguments supplied. A result message is returned to the caller. There are many variations and subtleties in various implementations, resulting in a variety of different (often incompatible) RPC protocols.

Enterprise computing systems make extensive usage of distributed object technology such as, Distributed Component Object Model (DCOM) [15], Java Remote Method Invocation (Java RMI) [16] and Common Object Request Broker Architecture (CORBA) [17]. Table 2.1 shows a comparison of the various technologies used by each.

Distributed object technology works well in closed environments and on a small scale. However, poor interoperability caused by a lack of common standards, specific platform targets, and programming environments amongst their forms prevents interaction and collaboration between enterprises systems enabled with different types of distributed object technology.

Adoption of common standards such as, XML and SOAP (covered in section 2.5.4), may offer a bridge between enterprise systems with different distributed object technologies. However, little may be done to overcome the inherent poor scalability of Interface-based Architectures. These demand understanding of an objects interface from all interacting sides requiring high levels of coordination across the entire system and close monitoring of each component. Consequently, a change to the interface of an object necessitates modification and redeployment of all of its interacting objects. Whilst manageable inside

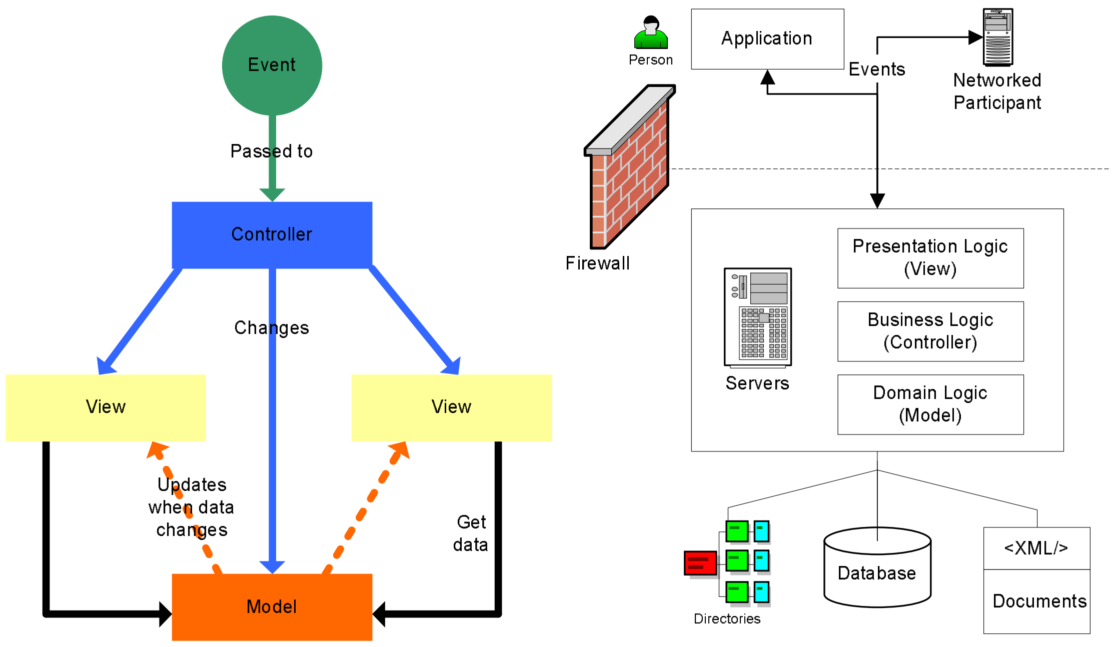


Figure 2.2: Model-View-Controller (MVC) architectural pattern used within enterprise computing. Block diagram shows the communication between the layers started by an external event such as, a stock keeper requesting or updating an inventory.

a closed environment with total control and monitoring of system components, the scale of the Internet and possible anarchy caused by participants makes close administration next to impossible. Thus, distributed object technologies do not provide the adaptability, scalability or openness demanded by Grid computing.

2.3.3 Model-View-Controller Architectures

Enterprise computing has traditionally employed the Model-View-Controller (MVC) architecture [18], shown in figure 2.2, in order to be able to reach the scale and sophistication demanded by today's business processes.

MVC is such an architecture that, in addition, adopts a common layered pattern that characterises business process need for presentation (view), domain-specific representation (model), and controlled access of operations (controller) on information.

Model: The domain-specific representation of the information on which the operations of the business process occur. Domain logic implemented in this layer adds meaning to raw information, such as working out the totals, taxes and shipping charges for online-applications shopping cart items.

View: This layer renders the model into a form suitable for interaction by clients of the system. Which typically is a user interface however it may be for any type of participant. MVC is often seen in web applications, where the view is the HTML page and the code which gathers dynamic data for the page.

Controller: Responds to events, typically user actions, and invokes changes on the model and perhaps the view. Many applications use a persistent storage mechanism, such as a database, to store data however MVC does not specifically mention this data access layer because it is understood to be hidden (i.e encapsulated) within the Model.

Decoupling of business processes into layers allows each to be changed irrespective of the implementation of the other. For instance, a typical implementation of a web application will rely on the view component to layout HTML in response to a user's request. If, however, an XML response is required, only the view component need be change or appended (with an XML serialiser) while the model and controller remain the same.

Although MVC has been implemented by vendors using a host of different distributed object-oriented technologies (see section 2.3.2), control flow generally works as follows:

1. The client interacts with the interface in some way and generates an event, such as a person adding an item to their online shopping trolley by pressing a button.
2. The controller processes the input event, possible through a handler or callback registered to a method exposed in a user interface.
3. The controller accesses the model, updating it according to the client's action (e.g. the item is added to the person's shopping trolley). To simplify implementation of more complex operations, controllers are often structured using a command pattern that encapsulate actions.
4. The view uses the model to generate a interface representation appropriate to the client (e.g. a web page listing the shopping trolley's contents). Whilst the view layer gets the data from the model, the model should have no direct knowledge of the view.
5. The controller waits for further events from the interface presented by the view layer, restarting the whole process.

MVC introduces the idea of a controller object in between the view (the GUI class) and the model (the object) to communicate between the other two objects. In addition the actual implementation of the controller object can vary quite a bit, but the idea of an object to 'transform' events to changes in data and execution of methods is the essence of this pattern.

2.3.4 Service-Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) is an architecture that defines the use of distributed components known as services to support the requirements of software users. Resources are made available to other participants in a distributed system as independent services that the participants access in a standardised way. Unlike traditional component- and interface-based architectures, SOAs comprise loosely coupled, highly interoperable components. Services interoperate based on formal contracts which are independent from the underlying platform and programming language such as, Web Services Description Language see section 2.6.1.1. The interface definition encapsulates the vendor and language-specific implementation. A SOA is independent of development technology (such as Java and .NET). The software components become very reusable because the interface is standards-compliant and is independent from the underlying implementation of the service logic. So, for example, a C# (C Sharp) service could be used by a Java application and vice versa.

SOA can support integration and consolidation activities within complex enterprise systems, but SOA does not specify or provide a methodology or framework for documenting capabilities or services. High-level languages such as the Business Process and Execution Language for Web Services (BPEL4WS) [19, 20] and specifications such as WS-Coordination [21] extend the service concept further by providing a method of defining and supporting orchestration of fine grained services into coarser grained business services, which in turn can be incorporated into workflows and business processes implemented in composite applications or portals.

One area where SOA has been gaining ground is in its power as a mechanism for defining business services and operating models and thus provide a structure for IT to deliver against actual business requirements and adapt in a similar way to the business. The purpose of using SOA as a business mapping tool is to ensure that the services created properly represent the business view and are not just what technologists think the business services should be. Within this area, the first SOA method was announced in Service-oriented Modeling (sic) and Architecture (SOMA) [22]. Since then, efforts have been made to move towards greater standardisation, particularly within the OASIS standards group and specifically the SOA Adoption Blueprints group.

Many e-Commerce companies offer these through web interfaces such as, Amazon and eBay, however recently they have also begun to offer their services through Web Service interfaces. Technologies such as, IBM Web Sphere and Microsoft ASP.Net provide the application platforms to present the interfaces to users, define and control business logics, and to model and persist data.

2.4 Successful Distributed Computing Systems

2.4.1 Hypermedia Resource Sharing on the World Wide Web

The most well known distributed computing system is the World Wide Web (WWW). It provides *"a shared information space through which people and machines could communicate"* [23]. It has become the world's biggest and most influential technological invention since the printing press. The WWW is a globally reaching systems enabling sharing of information and data amongst tens of millions of users and heterogeneous computing systems across institutional and organisational boundaries, and huge distances.

The WWW success can be ascribed to its ability to share multimedia information, such as pictures, text, and videos, through simply hypermedia means regardless of geographic location or type of computer system. The WWW's extreme scalability and openness have brought about new opportunities for novel applications. Not only should the Grid build upon the infrastructure of the Internet but also in addition, it must extend and adopt key approaches and technologies that have made the WWW successful. Important and influential companies and organisations such as, Microsoft and IBM are applying the approaches of Internet to Grid computing [24].

For the WWW to offer an expansively wide information space, it must handle a forever changing, heterogeneous array of information sources, data formats, character encoding schemes, storage mechanisms, and network topologies and infrastructures. On top of this, the WWW intended role as an Internet-scale distributed information system, requires it to provide independent deployment of system components in a disordered system. The inventor of the WWW, Tim Berners-Lee[1], itemises the following design principles for its creation, in [23]:

- An information system must be able to record random associations between any arbitrary objects, unlike most database systems;
- If two sets of users started to use the system independently, to make a link from one system to another should be an incremental effort, not requiring unscalable operations such as the merging of link databases.
- Any attempt to constrain users as a whole to the use of particular languages or operating systems was always doomed to failure;
- Information must be available on all platforms, including future ones;
- Any attempt to constrain the mental model users have of data into a given pattern was always doomed to failure;

[1]http://www.w3.org/People/Berners-Lee/Overview.html

- If information within an organisation is to be accurately represented in the system, entering or correcting it must be trivial for the person directly knowledgeable.

It can be argued that it is these principles that have made the WWW suitable, extensible, adaptable, and scalable. However, the combination of common standards and technologies with simply well defined modes of operation, late bound resource references, and easily transferable information representations has made the WWW a global system. We will now look at the key technologies behind the WWW.

2.4.1.1 Universal Resource Identifiers

The Universal Resource Identifier (URI) [25] provides a simple yet comprehensive, uniform, and consistent naming and addressing scheme for resources. It is the most important technology in not only hypermedia resource sharing but also, Grid and distributed computing in general. Any designatable resource may be mapped uniquely into the URI address space, such as computers, web pages, files, telephone numbers, and email addresses. URI have several schemes and variations in their common syntax components, examples of which are given below:

```
ftp://mail.soton.ac.uk/~mjf/thesis.pdf

http://www.soton.ac.uk/

ldap://[2001:db8::7]/c=GB?objectClass?one

mailto:mjf@soton.ac.uk

news:comp.infosystems.www.servers.unix

tel:+44-1234-111-496

urn:oasis:names:specification:docbook:dtd:xml:4.1.2
```

URI can be further classified as a resource locator, name, or both. Uniform Resource Locator (URL) refers to the subset of URIs that, in addition to identifying a resource, provides a means of locating the resource by describing its primary access mechanism. Within the WWW, URIs are generally used as resource locators or Universal Resource Locators (URLs). URLs have the format shown below:

```
http://www.bbc.co.uk/news/today.html
```

URLS are particularly useful as a means to uniquely identify or reference information, web pages, or files without need to understand the content. Web pages on the WWW use hyperlinked URLs to reference other web pages or information in any other web page regardless of its content. The Uniform Resource Name (URN) refer to both URIs under the “urn” scheme [RFC2141], which are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable, and to any other URI with the properties of a name.

Importantly, URIs provides late binding and opaqueness by decoupling resources' identification from their understanding. A URI may be passed between entities without need to access the resource itself. Indeed, a URI may be printed on to paper, sent by postal mail, and scanned back into a computer. URIs' abilities have been proven on an Internet scale, with its application in the WWW, and therefore must not be ignored as a suitable technology for Grid computing.

2.4.1.2 The Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) [26] is a technology designed for conveyance of information formatted in the Hypertext Mark-up Language (HTML) [27]. Its original purpose was to provide a means to publish and receive Web pages on the WWW. HTTP's development has been coordinated by the World Wide Web Consortium and working groups of the Internet Engineering Task Force, culminating in the publication of a series of RFCs, most notably [RFC2616], which defines HTTP/1.1, the version of HTTP in common use today.

HTTP is a request/response protocol between clients and servers. The originating client, such as a web browser, spider, or other end-user tool, is referred to as the user agent. The destination server, which stores or creates resources such as HTML files and images, is called the origin server. In between the user agent and origin server may be several intermediaries, such as proxies, gateways, and tunnels.

A HTTP client initiates a request, example shown below, by establishing a Transmission Control Protocol (TCP) connection to a particular port on a remote host (port 80 by default). A HTTP server listening on that port waits for the client to send a Request Message. Upon receiving the request, the server sends back a status line, such as “HTTP/1.1 200 OK”, and a message of its own, the body of which is perhaps the requested file, an error message, or some other information. Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs) (or, more specifically, URLs) using the http (or https) URI schemes.

```
GET http://www.futureflight.org/index.html HTTP/1.1
Accept-Language: en
```

In the first line in the above example HTTP request, the client has requested the index.html web page from the web site identified as www.futureflight.org using the HTTP/1.1 transfer protocol. The second line of the request is an optional header directive to inform the Web server that the client prefers that the web page be written in the English language.

HTTP has a simply interface with a well defined set of eight methods:

GET: Requests a representation of the specified resource. By far the most common method used on the Web today.

HEAD: Asks for the response identical to the one that would correspond to a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.

POST: Submits user data (e.g. from a HTML form) to the identified resource. The data is included in the body of the request.

PUT: Uploads a representation of the specified resource.

DELETE: Deletes the specified resource (rarely implemented).

TRACE: Echoes back the received request, so that a client can see what intermediate servers are adding or changing in the request.

OPTIONS: Returns the HTTP methods that the server supports. This can be used to check the functionality of a web server.

CONNECT: For use with a proxy that can change to being an SSL tunnel.

Methods GET, HEAD, PUT and DELETE are defined to be idempotent, meaning that multiple identical requests should have the same effect as a single request. Also, the methods OPTIONS and TRACE should not have side effects, and so are inherently idempotent.

The usefulness of the WWW has gained HTTP/HTML wide support on almost every computing platform and even extends to devices such as, mobile phones, hand-held devices and Internet-enabled consumer devices. However, it has been argued by [26] and proponents of REST that the WWW has scaled because of a few key design principles (see section 2.4.1.3).

In addition to the above features, HTTP also allows close control of its actions on a Web Server and access via network security systems such as, firewalls, and provides support for network facilities such as, caching proxy servers that improve communication performance and reduce network load.

However, HTTP employs a request-response mechanism originally designed for retrieval of short HTML messages that is unsuitable for applications that demand long-lived connection, high-bandwidth transfers, or asynchronous operation. Whilst, HTTP may be adapted it would require changes to fundamental network infrastructure, such as routers and proxies, to support these added capabilities.

2.4.1.3 REST-based Architectural Style

The success of the WWW is best explained by Representational State Transfer (REST) [28, 29]. It is an architectural style, which attempts to describe the software architecture of the Web and explain the success of the Web as a hypermedia system on the Internet.

REST emphasises scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems. For works on distributed computing systems with similar environment to the Web, REST can be a very good reference for creating and evaluating architectures and principle technologies.

Whist REST originally referred to a collection of architectural principles, it is now used in a looser sense to describe any simple web-based interface that uses XML and HTTP without the extra abstractions of Message Exchange Pattern (MEP)-based approaches like the web services SOAP protocol. Strictly speaking, it is possible (though not common) to design web service systems in accordance with Fielding's REST architectural style, and it is possible to design simple XML+HTTP interfaces in accordance with the RPC style, so these two different uses of REST cause some confusion in technical discussions.

REST's proponents argue that the web has enjoyed the scalability and growth that it has as a result of a few key design principles:

- A stateless anonymous client/server protocol: each HTTP message contains all the information necessary to understand the request. HTTP was designed to ease software implementation, simplify operation, and expedite processing by removing the need for continuing communication between user agent and origin server. This means that immediately after the origin server has initiated a response to the user agent's request the server need maintain no record of what the request was or who sent it. Consequently, as each HTTP request is discrete, the user agent must send within a request all information necessary for the origin server to process and send back a response. As a result, neither the client nor the server needs to remember any communication-state between messages. In practice, however, many logic-driven HTTP-based applications use cookies and other devices to maintain session state.

- A simple well-defined interface: HTTP itself defines a small set of methods, the most important of which are POST, GET, PUT and DELETE. People often compare these with the Create, Read, Update, and Delete (CRUD) operations required for data persistence, though POST does not fit cleanly into the comparison. HTTP interface has proven to be extremely adapt and flexible on a global-scale. In addition, to enabling access and sharing of a forever changing and increasing quantity of information and data, it provides software components of a distributed system with a means to interoperate through a ready-made contract whilst providing a comprehensive set of methods that act on all information allowing building of complex business processes.
- A universal syntax for resource-identification: URI see section 2.4.1.1.
- The use of hypermedia both for application information and application state-transitions: representations in a REST system are typically HTML or XML files that contain both information and links to other resources; consequently, it is often possible to navigate from one REST resource to many others, simply by following links, without requiring the use of registries or other additional infrastructure.

An important concept in REST is the existence of resources (in this case information or data), each of which can be referred to using a URI. Control of resources needs clients to communicate via the HTTP interface to exchange representations of these resources. Any number of intermediaries such as caches, tunnels can act between the request however a constraint of REST is that these connectors do not see further than their own request. In REST terms this is referred to as "layering" and is a common principle in many other parts of information and networking architecture. Consequently, an application can interact with a resource by knowing only two things: the resource's identifier, and the action upon the resource however it does not need to be concerned with anything else between it and the server actually holding the information.

A REST web application requires a different design approach than an RPC application. In RPC, the emphasis is on the diversity of protocol operations, or verbs; for example, an RPC application might define operations such as the following:

```
getUser()
addUser()
removeUser()
updateUser()
getLocation()
addLocation()
removeLocation()
updateLocation()
listUsers()
```

```
<?xml version='1.0'?>
<user>
    <name>Matthew Fairman</name>
    <gender>male</gender>
    <location href="http://www.bbc.co.uk/southeast/soton">
        Southampton, Hampshire, UK, Earth, Sol, Milky Way
    </location>
</user>
```

Listing 2.1: Example record for a User.

```
listLocations()
findLocation()
findUser()
```

With REST, on the other hand, the emphasis is on the diversity of resources, or nouns; for example, a REST application might define the following two resource types:

```
User {}
Location {}
```

Each resource would have its own location, such as:

```
http://www.bbc.co.uk/southeast/soton.
```

Clients work with those resources through the standard HTTP operations, such as GET to download a copy of the resource, PUT to upload a changed copy, or DELETE to remove all representations of that resource. POST is generally used for actions with side-effects, such as placing a purchase order, or adding some data to a collection.

For example, the record for a User might look like this:

To update the user's location, a REST client could first download the above XML record using HTTP GET. The client would then modify the file to change the location, then upload it again using HTTP PUT.

Note, however, that the HTTP verbs do not provide any standard method for resource discovery. Instead, REST data applications work around the problem by treating a collection or set of search results as another type of resource, requiring application designers to know additional URLs or URL patterns for listing or searching each type of resource.

For example, an HTTP GET request on the URL http://www.bbc.co.uk/southeast/soton might return a list of links to an XML file for each location in Southampton whilst a HTTP GET request for the URL http://www.example.org/users?surname=Fairman might return a list of links to all users with the surname "Fairman".

REST provides some guidance on how to perform this kind of action as part of its *"hypermedia as the engine of application state"* constraint, which suggests the use of a forms language (such as HTML forms) for specifying parameterised queries.

2.4.2 Enterprise Computing

Enterprise computing is the term used to refer to the large-scale employment and collaboration of distributed computing resources for high-level applications. Enterprise computing is not limited to business and commerce applications; the term enterprise refers to the scale of the industrious activity.

Enterprise-wide distributed computing systems are typically found in organisations such as, universities, businesses and government institutions, with large numbers of staff or customers who are often geographically distributed. These systems play an essential role in providing common communication, data and information management. In addition, many enterprise systems often provide front-line services to customers and business partners such as order processing, order tracking and delivery, and stock control and monitoring. These systems will consist of a variety of distributed resources including, database, clusters, web servers, backup systems, and application servers.

Early enterprise computing system extended only as far as their organisations network boundaries, necessitating only simple architectures and operational modes. However, it made business sense to extend enterprise systems to integrate with the systems of their customers and business partners. However, realistic implementation of large-scale cross-institution enterprises only became possible with the rise of the Internet and middleware technologies that allowed support for multiple-participants and information sharing operations. In addition, common standards for communication amongst systems and data modelling were essential prerequisites for the integration of enterprise systems across organisational boundaries.

2.4.3 Clustering

The compute issue has become increasingly important, especially in the fields of science and engineering. Traditional single processor computers often cannot provide the calculating power nor do they possess the capacity for attached memory which is demanded by computationally challenging problems. Clustering offers a solution whereby multiple distributed computers or processors are brought together to work on a problem.

Typically, scientists and engineers demand computing power for tasks, such as simulation, modelling and analysis. However, the most demanding problems types, often referred to as "Mega-problems" or "Grand Challenge Applications", typically process huge amounts of data or perform very time-consuming mathematical calculation; or

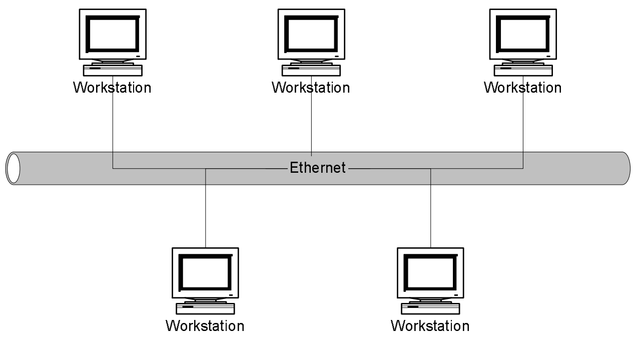


FIGURE 2.3: High Throughput Computing (HTC) network diagram.

both. For example, detectors at the Large Hadron Collider at CERN[2], are currently producing 15 petabytes of data per year [30, 31]. It is estimated that even rudimentary analysis of this data will probably require the sustained application of some twenty tera-flops (trillion floating-point operations per second) of computing power. However, only the fastest supercomputer in the world[3], the Earth Simulator Centre [32], at full capacity could process this much information. Consequently, this is the reason why CERN is leading the development of Grid computing, which aims to link hundreds of major computing centres around the world. It is clear that more sophisticated analyses will need orders of magnitude more power. Other examples of Mega-problems include crystallographies [33], microtomographic structural problems [34], and virtual materials and processing [35].

The clustering paradigm revolves around the principle of division of labour where processors work on different elements of a problem that requires implementation of parallel algorithms. This technique, first explored in the early 1980s, is now standard practice in supercomputer centres, research labs and industry.

Figure 2.3 shows a typical computational clustering system for High Throughput Computing (HTC). It contains the computers that perform the computation that are linked together physically using networking, such as Ethernet, to enable low-level intercommunication. On top of this, each computer runs clustering software that couples them into a unified system. This software would typically provide management, control and utilisation of the cluster's resources. The role of the cluster determines the type, architecture and level of interaction among systems' hardware and software components and technologies.

Other notable forms of clustering environment are High Performance Computing (HPC) and High Availability Computing. HPC environments aim to solve individual compute tasks as quickly as possible. They perform concurrent parallel computation across an array of processors for a large compute task. Unlike HTC tasks, HPC tasks have few or no independent elements and cannot realistically be solved in a timely fashion on a single

[2]CERN public home page - http://public.web.cern.ch/public/
[3]As listed by Top500 SuperComputer Sites - http://www.top500.org/list/2003/11/

computer. Rather than processors working on different jobs independently of each other, they must pass data and control messages amongst each other for correct progression of a task. HPC tasks are written using low-level message passing systems, such as MPI [36]. This provides the intercommunication and synchronisation mechanisms for algorithms needed to run tasks across multiple machines.

2.5 Grid Computing: Large-scale Coordination, Collaboration and Sharing of Resources

Grid computing first emerged as an idea proposed by Larry Smarr [37, 38] with its first definition appearing later on in [39]. The authors describe the Grid as *"a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities"*. Whilst, this roughly points out the target role of the Grid, a more pertinent and less obscure definition of the Grid subsequently appears in [40]. In which it says the the Grid must provide *"coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations (sic)"*. In [41], the author explores potential architectures for the Grid and subsequent enabling technologies. In the paper there is provide a useful checklist, shown below, that helps distinguish the Grid from other forms of distributed computing:

- coordinates resources that are not subject to centralised control;
- using standard, open, general-purpose protocols and interfaces;
- deliver nontrivial qualities of service.

Whilst the definition of the Grid may continue to evolve, the fundamental aims and principles first covered in these ground breaking papers and books will persist. From them it can be discerned that Grid computing must enable dynamic negotiation and allocation of all forms of needed resources from a range of resource providers. In turn, these resources must collaborate as a whole through sharing operations that allow direct harnessing of their capabilities through more the just file exchanges. This means that resources should be able to integrate into the system at a more direct level allowing resources of a system to be organised and integrated together at the level of its hardware, software and data.

The Grid seeks to overcome the need for local ownership of resources by providing access and the sharing of resources on the Internet. This new sharing mode and planned scale for the Grid has created unequalled challenges not matched in other areas of distributed computing. We now summarise the key challenges in Grid computing.

2.5.1 Key Challenges of Grid Computing

Research on Grid computing fits generally into two parts: the establishment of mutual accessibility amongst wide-ranging computer resources, in addition to common connexions between Grid and its users; and the construction of an interconnected network, or Grid, of computer resources that are brought together, dynamically located, aggregated, and allocated. The preponderance of Grid research (and in this thesis) has been focused on part one; creating platform independent interoperability, and providing data and security management, such as single sign-on authentication and authorisation, through the new technologies, toolkits, and architectures. Part one provides the foundation on which a Grid must be constructed thus progress on part two is much less advanced. Identified features required for construction of the Grid include a standardised Grid-wide namespace for all resources involved, mechanisms for resource registration and discovery, resource scheduling and co-allocation, performance and resource usage monitoring.

Ian Foster et al. wrote that Grid computing should focus on "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations" [40]. This means that the Grid should provide access and the ability to share resources globally, so that it would not be necessary to possess all the resources required locally. The unprecedented sharing mode and intended system scale have brought unique characteristics to Grid computing and differentiated it from other distributed computing practises.

2.5.1.1 Heterogeneous Environment and Diverse Application Models

The Grid, like the WWW, targets geographically distributed resource consumers (users) and resource providers whom exist on the Internet in a varied state of conditions. They will want to connect to the Internet through a plethora of devices including, PCs, mobile phones, servers, kiosks and Internet-enabled televisions (as seen in many hotels). These will use a multitude of different operating systems (e.g. Windows, Linux, Solaris, Mac OS-X), file formats (e.g. HTML, XML, and SMTP), and applications (e.g. Web browsing, e-mails, file sharing and news feeds). In addition, these devices will use a variety of transport protocols and mediums for network connectivity including phone lines (e.g. Modems and ADSL/SDSL), wireless networking (e.g. WAP, Wi-Fi, G3 and Bluetooth), and local area networks.

However, the heterogeneity of Grid exceeds that of the WWW. Whilst, the WWW offers only access and sharing of data and information, in addition to this, the Grid will attempt to provide access and sharing of all resource types that can be connected to the Internet. These will range from, for example, traditional computing resource such as, HPC clusters and databases to application specific devices such as, scientific instruments, and machining tools or sensors.

An expected outcome of this will be extremely diverse and unpredictable resource sharing operations on the Grid that will vary in their structure, scale, purpose, duration, and user community. In addition, the Grid will be expected to handle different modes of usage from single to multi-user, cost to performance sensitive, traditional client-server architectures to peer-to-peer, and client to resource orientated.

2.5.1.2 High Unreliability of Internet-Scale Systems

By design, the Grid intends to reach an Internet scale, which is a feat currently only achieved, by the WWW and e-mail. The WWW functions through involvement of components from a multitude of organisations. Typically, deployment and development of components such as, Web and DNS servers, network routers, proxies and firewalls happens independently, with little unified control. The Grid shares this scenario and thus must be capable, like the WWW, of operating effectively within a highly unreliable environment. This goal requires that applications and systems built on the Grid be robust which will require them to be tolerant of system failures, dropped connections, version incompatibilities, and malformed or malicious data.

2.5.1.3 Coordination of Resource Sharing Operations

Resource sharing operations performed on the Grid require layers of coordination. The Grid exceeds the scope of enterprise computing by aiming to cater for a far greater range of heterogeneous resources and their sophisticated applications; and unlike enterprise computing, provide this in an unreliable, disunified environment. Major functions of the Grid include enabling access and sharing of computational power, huge quantities of data, and high-level application processes. However, in an environment of distributed resource ownership and high unreliability, participating components of systems require coordination of security mechanisms, quality of service contracts, fine-grained controls, and clear definitions of component functionality. This need for coordinated sharing differs greatly from the disorganised WWW.

2.5.2 Desirable Characteristics of Grid Technologies

Achievement of the Grid's goals requires careful selection and development of capable enabling technologies. In addition to the characteristics of transparency, openness, scalability, and loose-coupling, technologies must bring to Grid an interoperable and decentralised environment. An essential criterion of these technologies is that they must be easy to deploy and maintain. We covered the latter characteristics in general at the beginning of this chapter as important to distributed computing however; we now look at how they specifically apply to Grid technologies in addition to the other characteristics.

2.5.2.1 Interoperability

Interoperability is key to the success of the Grid and is a direct requirement of the highly heterogeneous environment that systems will run in. Importantly, this characteristic shall make possible interaction amongst differing Grid participants, irrespective, for example, of resources' platform, operating system, and runtime software environment. Interoperability is essential to facilitate dynamic and transparent access of shared resources because of the ad hoc creation and destruction of links between users, resources, and intermediaries. In addition, interoperability is a prerequisite of transparency, openness, and scalability.

Interoperability is important to all levels of the Grid from low-level message passing to high-level semantics. Consequently, interactions throughout the Grid must happen in a consistent manner, which shall require industry-wide agreement on component interfaces, communication protocols, and data formats. Furthermore, interfaces for components such as, services and intermediaries, require definition through common understanding of its types operation and behaviour, which is necessary in order to facilitate their dynamic discovery and integration. This shared perceived understanding will require the introduction of semantically enriched interfaces.

2.5.2.2 Scalability and Openness

The Internet's continued evolution and tremendous growth has increased the quantity and types of resources connected to it and the scale and variety of their application. Philanthropic computing such as, SETI and Folding@Home, along with P2P file sharing are examples of both new and large-scale Internet applications. The Grid's infrastructure and systems built on it must be capable of handling this and thus must be scalable in all directions whilst open to changes and addition to its function.

Due to the high unreliability of resources within the Internet, assumptions cannot be made about their state or existence because they could alter or disappear at any time without notice. In addition, the Grid must be capable of keeping pace with the rapid improvement and addition of new technologies from underlying communication protocols to new Internet-enabled resources such as mobile phones and media-centre PCs. Consequently, the ability to quickly change and evolve to meet new demands is an essential requirement of the Grid.

2.5.2.3 Decentralised and Loosely-coupled Architectures

The scale of the Grid means that no centralised management mechanism is able to provide the required performance, stability and scalability. Even if it were practical, service providers often wish to maintain control of their own systems for political, economic and social reasons. The low uptake of systems such as, UDDI and Microsoft Passport, is attributable to an unwillingness by potential participants to relinquish important aspects of their systems to third parties, especially security and user accounts, or register services with a central authority. To encourage large-scale adoption of the Grid, participants such as, for example resource providers, must have the freedom to decide on usage policies, make changes, or even withdraw service. However, this shall only be possible if the Grid adopts a loosely-coupled structure.

Loose-coupling offers the most suitable architectural style for large-scale systems. It would be significantly more difficult to build or effectively maintain systems using tightly-coupled architectures because the high unreliability and distributed ownership of resources connected on the Grid would make coordination of changes across systems impractical.

Coupling refers to interrelationships and interdependencies between software components of a distributed computing system. Software componentry is a loose term, which refers to the encapsulation of software functionality though adherence to a written specification also known as Interface Definition Languages (IDLs). It allows developers to reuse functionality developed and tested in one program with another. In addition, it allows software to be broken down into smaller more manageable and easily tested parts. The type of coupling between components requires careful considerations as this affects the efficacy and simplicity of their maintenance and reuse. Clemens Szyperski and David Messerschmitt give the following five criteria for what a software component shall be to fulfil the definition:

- Multiple-use,
- non-context-specific,
- composable with other components,
- encapsulated i.e., non-investigable through its interfaces, and
- a unit of independent deployment and versioning.

Object-Orientation and Service-Orientation are formal examples of programming paradigms developed around the idea of software componentry of which the latter is a new concept preferred in Grid computing. Various technologies, including CORBA, DCOM , and Java RMI, encapsulate functionality and provide distributed access to components, known as objects, in an object-oriented manner. On the other hand, technologies, including Web Services and the WS-Resource Framework (WS-RF) [42], encapsulate functionality and provide distributed access to components, known as services, in a service-orientated manner. The abilities and orientation of software componentry technologies have a direct bearing on the transparency, openness and scalability of a distributed system. Nonetheless, componentisation is not enough on its own to guarantee effective reuse (i.e. sharing) of software functionality. Components also need:

- to be fully documented,
- more thorough testing,
- robust input validity checking,
- to pass back useful error messages as appropriate, and
- to be built with an awareness that it will be put to unforeseen uses

Two forms of coupling are prevalent in distributed computing: tight-coupling and loose-coupling. The benefit of loosely-coupled software architectures over tightly-coupled ones lies in its agility and the ability to survive evolutionary changes in the structure and implementation of the internals of each component, that make up the whole system. It foremost relies on componentisation of software and on standardisation of their interfaces and behaviours. This is such that changes to parts of the system will have known consequences and promotes flexibility by focusing design effort on interfaces. Loose-coupling is a concept widely practised in software architecture especially within massively parallel computing systems.

On the other hand, tight-coupling architectures have interfaces between the different components of a system that are tightly interrelated in function and form, thus making them brittle when any form of change is required to parts or the whole of the software. Components within a tightly-coupled software often interact, for instance, in a bespoke way, without standard interfaces, or natively with the hardware. The absence of software abstractions and intermediaries, such as componentisation and tiered system design, can have performance benefits but at the expense of easy extension of software functionality and integration with other software and systems. Tight-coupling makes components artificially dependent on each other's specific implementation thus creating high interdependence and interrelationships among each other which precludes their easy reuse and maintenance.

Whilst the benefits of loosely-coupled architectures abound, software has remained tightly-coupled because of the inability of major software vendors to agree on a universal set of standards to define interfaces across software modules. An example of this is database vendors whose poor adherence to the SQL specification and proprietary componentisation models often results in systems becoming dependent on specific vendors implementations. For instance, stored procedures is the term used to refer the ability of a databases to run at improved performance by encapsulating logic that runs inside the database engine. However, whilst all modern databases offer the ability to store procedure in some manner no standard has been adopted for transferal amongst implementations.

Nonetheless, loose-coupling on its own is not enough to guarantee effective reuse of software. Even when systems employ standardised interfaces and componentisation, incompatibilities between the communication mechanisms and behaviours of competing Object-Oriented technologies make large-scale system integration impossible. Not only must software architectures be loosely-coupled but also components must be accessible in a standard, open and transparent way.

The loose-coupling principle aims to ease design and component maintainability and reuse, by exposing real dependencies through simplification of systems to the very minimum but no further to preserve functionality. This means that systems must not contain components with unnecessary dependencies or relationships. Standardised technologies and components, such as commodity goods, along with common operational and behavioural models aid system builders to meet the loosely-coupled principle. These systems trade efficiency for robustness and scalability that make them especially well suited for roles in large-scale distributed computing, such as High Availability Computing (HAC), HTC, and Grid computing. Service-oriented technologies offer high-level data modelling definition and independent componentisation that make it the ideal paradigm for achieving loosely-coupled architectures.

2.5.2.4 Low-Effort Deployment and Participation

Low-effort deployment and participation in the Grid will encourage its adoption. Features required of the Grid technologies, already mentioned in this chapter, including loose-coupling, decentralisation, transparency, openness, and interoperability all play a part in making the Grid more easily adoptable by both users and resource providers.

Users' applications should not be affected by the adoption of Grid. Details of the efforts that enable operations on the Grid should be transparent from the perspective of high-level applications. In the case of deployment, Grid technologies must not affect the integrity of the local system such that no significant changes are necessary to the underlying workings of the platform. Rather, Grid technologies should work with local systems' mechanisms to reduce deployment effort, ease maintenance, and leverage the skills of system administrators. In addition, side effects, performance overheads, and significant changes should be kept to a minimum.

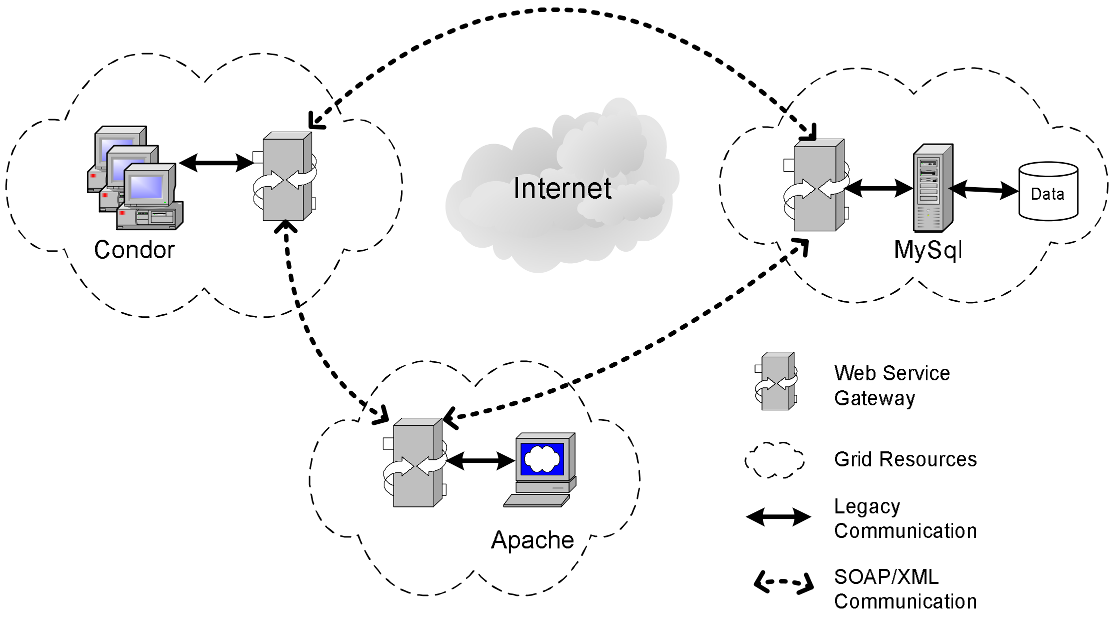


FIGURE 2.4: The usage of legacy distributed computing technologies in Grid computing.

2.5.2.5 Legacy Distributed Computing Technologies in Grid Computing

The research and development of Grid computing have been carried out based on the achievements and experiences from earlier efforts on distributed computing. Although the distinctive features of Grid computing and the consequent requirements for Grid technologies make it clear that legacy distributed computing technologies alone cannot be successful in building the Grid, many of them can still be directly or indirectly applied in enabling Grid computing operations.

For distributed object technologies, the inherent lack of interoperability and scalability makes them undesirable choices for enabling interactions among Grid resources. Yet they are very likely to be applied in constructing Grid resources that are composed of homogeneous, closely administrated distributed systems, in which interoperability is not a major concern and performance is of higher priority. Such resources include computer clusters, High Throughput Computing (HTC) systems and data service providers, as illustrated in Figure 2.4.

The Web system exists in the same environment of the Grid, and has successfully managed to scale to the level of the Internet. While technologies that enable the Web cannot provide a direct solution to Grid computing, as resources and operations on the Grid are far more sophisticated than the sharing of hypermedia resources on the Web, they have actually provided a well established infrastructure on which further extensions and modifications can be made to support Grid computing. Standard Web technologies such as HTTP and URI are deeply involved in the development of standard Grid technologies.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Cluster Name="Southampton E-Science"
    xmlns="http://www.e-science.soton.ac.uk/computation"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Job Name="OOMMF1">
        <MachineHost>e-science04.soton.ac.uk</MachineHost>
        <Status>Running</Status>
    </Job>
    <Job Name="OOMMF2">
        <MachineHost>e-science08.soton.ac.uk</MachineHost>
        <Status>Evicted</Status>
    </Job>
</Cluster>
```

Listing 2.2: Example XML document showing the status and host of jobs running on the Southampton e-Science's Condor cluster.

The impact of the Web on Grid computing has also been made clear by the introduction of Web Services to Grid computing.

2.5.3 Technological Requirements of Grid Computing

Grid technologies must provide simple, standardised loosely-coupled methods for describing, discovering, and integrating distributed resources. Realisation of effective, capable and powerful large-scale distributed systems necessitates a common structural paradigm and modes of interaction. Grid technologies should not replace low-level distributed technologies, such as MPI [43] or Condor [44], but instead provide the high-level glue that binds resources together.

2.5.4 Key Grid Technologies

The most important part of any distributed system and what defines it is messaging. The abilities of technologies used to represent, structure, and convey data directly influence the ability of distributed computing resources to interoperate. The key technology of the Grid is Extensible Markup Language (XML) which, along with XML Schemas and SOAP, provide a standardised, powerful and flexible messaging mechanism for the Grid. In addition, these technologies form the basis for many other Grid technologies.

2.5.4.1 Extensible Markup Language (XML)

The Extensible Markup Language (XML) [6] is an application independent, simple and flexible text format for data. It is the most important technology in Grid computing because it defines standard methods for structuring, self-describing and defining formats for information and data that may be read by almost any computer or software application.

XML exclusively utilises plain text for all document markup, including data structures and data values, an example of XML document is given in Listing 2.2. Summarised below are features of XML that make it well-suited for data transfer:

- its usage of plain text mark-up make it both human and machine readable,
- Unicode support which allows writing of its documents in any human language,
- the ability to represent fundamental computer science data structures such as, records, lists and trees,
- self-documenting formatting that describes structure, field names and specific values, and
- strict syntax and parsing requirements which makes parsing of documents simple, efficient, and consistent

XML is also heavily used as a format for document storage and processing, both online and offline, and offers several benefits:

- its robust, logically-verifiable format is based on international standards,
- the hierarchical structure is suitable for most (but not all) types of documents,
- it manifests as plain text files, unencumbered by licenses or restrictions,
- it is platform-independent, thus relatively immune to changes in technology, and
- it and its predecessor, SGML, have been in use since 1986, so there is extensive experience and software available

However, XML suffers from some of the following weaknesses:

- its syntax's high verbosity and partially redundancy often lowers human readability and application efficiency, and increases storage needs. In addition, the larger size of XML formatted documents makes it unsuitable for bandwidth restricted networks, such as Mobile phones.
- its strict, descriptive and partially redundant syntax requires that all parsers, even for the most basic XML usage, recurse arbitrarily nested data structures and perform additional checks to detect improperly formatted or differently ordered syntax or data. Consequently, XML parsers have significant processing and memory demands that limit its usage on devices with restricted resources, such as embedded devices. In addition, badly or malformed XML documents can cause exhaustion of resources and stack overflows. Therefore, security considerations arise when XML input is fed from untrustworthy sources.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
    targetNamespace="http://www.e-science.soton.ac.uk/computation"
    xmlns:ca="http://www.e-science.soton.ac.uk/computation">
     <xsd:element name="Cluster">
          <xsd:complexType>
               <xsd:sequence>
                    <xsd:element name="Job" type="ca:JobType" maxOccurs="unbounded"/>
               </xsd:sequence>
               <xsd:attribute name="Name" type="xsd:string"/>
          </xsd:complexType>
     </xsd:element>
     <xsd:complexType name="JobType">
          <xsd:sequence>
               <xsd:element name="MachineHost" type="xsd:string" minOccurs="1"
               maxOccurs="1"/>
               <xsd:element name="Status" type="xsd:string" minOccurs="1"
               maxOccurs="1"/>
          </xsd:sequence>
     <xsd:attribute name="Name" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>
```

LISTING 2.3: Example XML Schema that defines data structures for describing the status of jobs running within a computational cluster. Listing 2.2 shows an XML document that employs this schema.

- it does not define a wide array of data types, requiring additional parsing in order to process the desired data from a document. For instance, XML does not specify whether the value "485.12" is an number or a six-character string. In addition, it does not come with out-of-the-box support for rich data types; the scientist must build them.

- Uses the hierarchical model for representation, which is limited compared to the relational model, since it only gives a fixed view of the actual information.

The XML specification includes a simple type and data structure definition model, known as Document Type Definitions (DTDs) [45]. These allow sharing of common document formats and types, and enable document consistency checking. However, it lacks a way to define complex data structures. To address DTDs' failings and to improve XML's usefulness, the XML Schema Language (XSD) [46] was developed as a more advanced, feature-rich and flexible alternative.

XML versatility as a tool for distributed computing, is only fully realised when it is used in conjunction with XML Schemas, see example in Listing 2.3. Standardisation of data formats, essential to the sharing of data. Many important new distributed communication protocols and description languages utilise XML-based documents and the XML Schema Language to specify their data and messaging formats.

XML Schema uses XML itself to define the types and structures of XML data. It provides rich support for basic data types like integer and string as well as common data structures available in computer programming languages. It is also possible to construct user defined data formats, such as postcodes. The flexibility of XML enables the XML Schema to support the sophisticated data structures necessary to define complex user types.

2.5.4.2 Simple Object Application Protocol

The Simple Object Access Protocol (SOAP) [47] provides a simple and extensible framework that defines how an XML message is structured. SOAP is designed to be a lightweight protocol for the exchange of information in a decentralised and distributed environment. In addition, it was designed to be independent of any particular transport mechanisms, however, to facilitate message passing, careful attention was paid to ensuring interoperability with commonly supported transfer protocols, such as HTTP [26] and SMTP [48].

SOAP messages consist of a header and a body. SOAP allows security systems, such as firewalls, to identify XML messages without needing to understand their contents, thus it is possible to prevent the blocking of unknown HTTP requests. SOAP also provides rich semantics for indicating encoding style, array structure, and data types. SOAP is currently under inspection by the W3C consortium and is a prototype of the future XML Protocol (XMLP) [49].

Both SMTP and HTTP are valid application layer protocols for SOAP however HTTP has gained wider acceptance due to its extensive capabilities and Internet infrastructure support. Importantly, HTTP enables SOAP to work with network firewalls which is a major advantage over other distributed protocols like GIOP/IIOP or DCOM which are normally filtered by firewalls. A key issue under discussion is whether or not HTTP is the right transport given its inherent synchronous nature.

As has been discussed in section 2.5.4.1, XML's verbose syntax can be both a benefit and a drawback. Compared with CORBA, GIOP and DCOM (see section 2.3.2) that use much shorter, binary message formats, SOAP's XML messages take much longer to process making SOAP the slower messaging mechanisms. Nonetheless, hardware appliances are available to accelerate processing of XML messages. It has been suggested [50, 51] that Binary XML may offer a solution to improving performance of SOAP messaging, although this creates its own set of problems including the loss of readability and confusing over little or big endian byte representation.

2.6 Service-Oriented Technologies

2.6.1 Web Services

Web Services [52, 53] define Service-Orientated techniques for the description, discovery and integration of remote software components, see Figure 2.5. They have many beneficial characteristics suitable to Grid and distributed applications; such as, programming language and model independence, and platform neutrality. Moreover, Web Services'

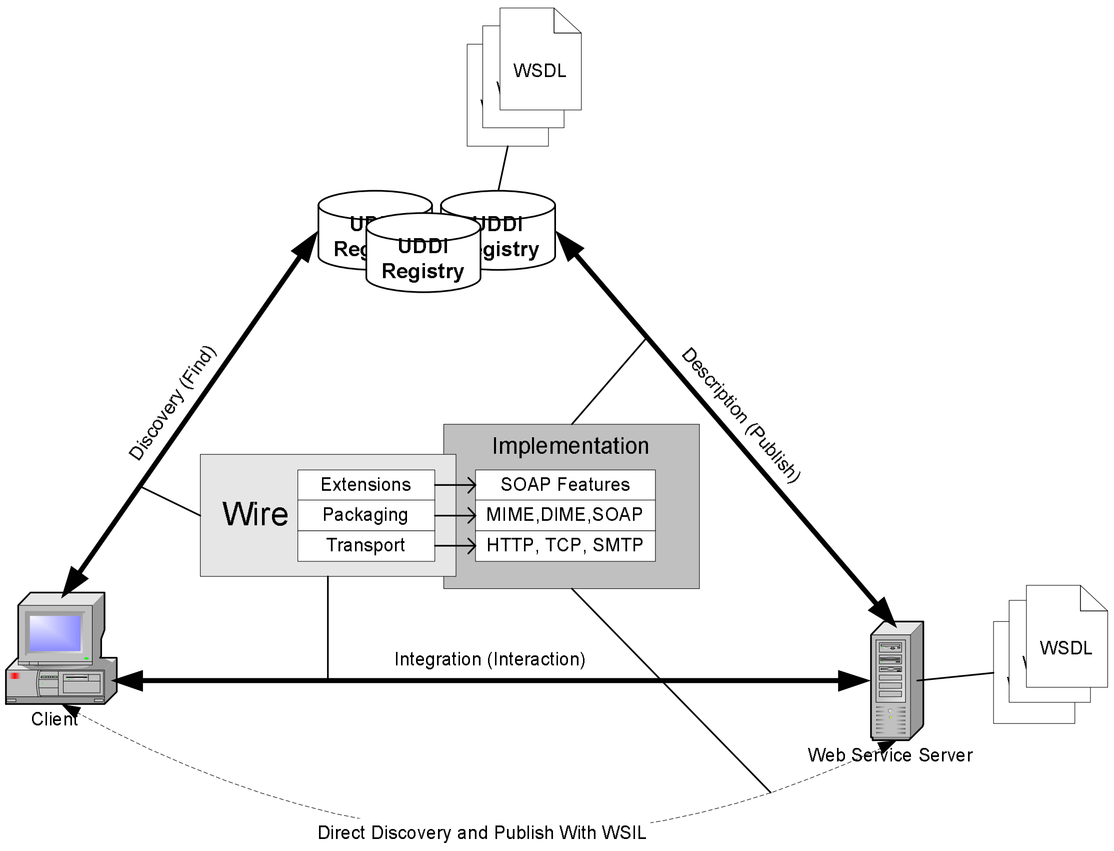


FIGURE 2.5: Web Service usage of XML Technologies.

simplicity are their greatest strength; designed specifically to facilitate creation, deployment, testing and utilisation within client and server software using widely available visual development [54] and client proxy generation tools [55].

2.6.1.1 Web Service Description Language

The Web Service Description Language (WSDL) [56] is an XML Document that provides a standardised means to formally define the interface and the endpoints of a Web Service. It is the IDL for software components (i.e. services) of SOA environments. Unlike IDLs in point-to-point architectures (see section 2.3.2), interfaces of Web Services defined in WSDL are independent from the underlying platform and programming language. This hiding of services vendor specific implementation facilitates the coupling of service which, in addition, to WSDL's standards-compliance makes Web Services very reusable and interopable. Although originally designed to describe Web Services, WSDL is a flexible and extensible language that may be used to describe other service based distributed technologies like Grid Services [57]. WSDL 1.2 is now being adopted by the OGSI Working Group instead of GWSDL.

2.6.1.2 Universal Description, Discovery and Integration

The Universal Description, Discovery and Integration (UDDI) [58] specification defines a way to publish and discover information about web services. It is a collaboration between Ariba, IBM, and Microsoft, who each provides UDDI services. The UDDI project includes a UDDI business registry and a set of operations on it. The UDDI registry, an XML file, identifies a Web Service and provides information about the service. Other programs use the registry to get the information about the Web Service and check compatibility with it. Categorisation of web services in the registry enables location and discovery. UDDI together with WSDL provides the ability to locate and programmatically interface to the Web Service. This allows programmatical access to the Web Service in a similar way that a coder accesses software component, simplifying service collaboration.

2.6.1.3 WS-Inspection and Web Service Integration Language

WS-Inspection [59] consists of a simple XML language and conventions for locating service descriptions published by service providers. The language, Web Service Integration Language (WSIL), may contain a list of service descriptions and links to other service descriptions. These are links to WSDL documents through URLs and may reference an entry within a Universal, Description, Discovery, and Integration (UDDI) registry. Service providers make there WSIL documents accessible through normal Internet protocol such as HTTP GET enabling document retrieval and discover of the advertised providers services.

2.6.1.4 Security

Efforts on ensuring security of Web Services have been focused on the standardisation of the WS-Security [60] protocol. WS-Security is a message-level security mechanism, which identifies enhancements to SOAP messaging to provide security through message integrity, message confidentiality, and single message authentication. WS-Security is designed to support a wide variety of security tokens, security models and encryption technologies. The specification has already been accepted as an OASIS standard. In addition, in order to enable the service to publish its security requirements, the WS-SecurityPolicy [61] specification is drafted as a supplement to WS-Policy.

2.6.1.5 Data Delivery

The XML format is not suitable for delivering large sections of data, especially binaries, due to its redundancy. To facilitate data transfer with Web Services and remain human-interpretable, an abstract model called SOAP Message Transmission Optimization (sic) Mechanism (SOAP MTOM) [62] is proposed to define how to encapsulate SOAP messages and its associated attachments. This model has already been supported in Web Service tools from major software vendors.

2.6.1.6 Business/Application Process Orchestration

A series of proposals have been made to provide standard methods for composing multiple Web Services for sophisticated business/application processes. These include WSFL [63], and BPEL4WS [20]. At the current stage, BPEL4WS has received wide support from the Web Service community. It defines a model and a grammar for describing the behaviour of a business process based on interactions between the process and its partners. It also defines how multiple service interactions are coordinated to complete an operation, how business exceptions and processing faults should be dealt with, and how compensation to participants in the process should be made in cases of exceptions and service cancellation.

2.6.1.7 Message Routing

To support asynchronous delivery of SOAP messages over a variety of transport methods such as TCP, UDP, and HTTP, the WS-Routing specification is proposed to make it possible to describe directly in the SOAP header the entire message path for a SOAP message. It is therefore possible to carry out SOAP messaging in different modes such as request/response and peer-to-peer conversations over extended period.

2.6.1.8 Reliable Messaging

The WS-ReliableMessaging [64] specification is proposed to ensure quality in service communications. It defines mechanisms to guarantee the delivery of SOAP messages, and to make sure that messages are only received once, and are processed in the right order. WS-ReliableMessaging is extensible to allowing integration of additional functionality such as security. The current specification is used with other Web Services specifications such as WS-Security and WS-Policy to provide secured and reliable service communications.

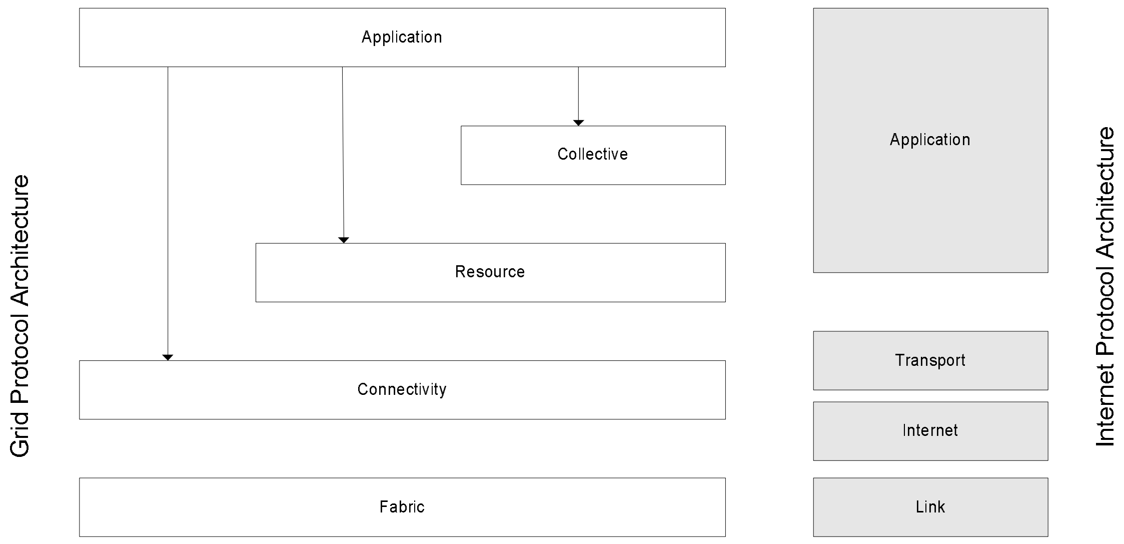


FIGURE 2.6: Shows the relationship between the Grid architecture and the Internet protocol architecture. There is a mapping from Grid layers to Internet layers because the Internet protocol layer extends from network to application.

2.7 Grid Projects and Architectures

There are a number of Grid projects currently running such as those from the UK e-Science Grid projects including GEODISE [65, 66], MyGrid [67], GridPP [68] and the AstroGrid [69]. These systems have used existing technologies (especially Globus[70]); however, they have all developed specialised services that provided layers above or in between standard services like discovery, security, and remote job execution. This has resulted in a vast spread of varying protocols, non-interoperable standards and implementation that are difficult to reuse.

2.7.1 Open Grid Service Architecture

Open Grid Service Architecture (OGSA) represents a long overdue initiative to define Grid architecture and is a milestone in the evolution of the Grid. Before this, Grids have been defined as a collection of distributed services implemented using a collection of toolkits, like Globus [39, 71] or through distributed operating systems like Legion [72], or distributed resource management systems, such as Condor.

Their architecture is one that is essentially a protocol architecture that defines mechanism through protocols for users and resources negotiation, establishment, manageability and exploitation of sharing relationships. Their primarily concern is with the creation of an open architectural structure within which are placed solutions to user's and Virtual Organisational (VO) requirements.

Shown in figure 2.6 is the representation of the proposed Grid architecture as component layers, that is common with many distributed protocols and architectures. There is no attempt made to enumerate all the protocols and it was their goal to discuss only the requirements necessary for general component classes in each layer. Components within each layer share common characteristic that can build upon the capabilities and behaviours by the lower layers.

2.7.1.1 Fabric

The Grid Fabric is at the bottom of the protocol stack and contains the resources for which sharing is mediated by Grid protocols. Resources could typically be computational resources, storage systems and network resources such as proxies. These resources may be logical entities such as a computational cluster or a distributed file system; in which case they may utilise protocols and systems internal to their operation that are not part of the Grid architecture. It is the Grid systems concern to provide access above this point. The role of the fabric layer is to offer the ability to support higher-level operation, for example resource scheduling, that can increase the complexity and therefore the cost of adding a resource to the Grid. It is required that resources must have a minimum of an enquiry mechanisms that allow higher level components to ascertain the facilities, structure and state of a resource, together with resource management mechanism to guarantee a certain level of quality of service.

2.7.1.2 Connectivity

This layer defines the core authentication and communication protocols required for Grid-specific network transactions. The communication protocols are essential to the exchange of data among fabric resources. Built upon these are the authentication protocols, that provide the secure mechanism for the verification of users and resources. For the communication layer, it is necessary to have transport, routing and naming. The assumption is that the TCP/IP protocol stack will form the basis for these, that is a good idea because of the current TCP/IP support by the majority of vendors.

So for instance, the Internet layered protocol architecture [73, 74] defines protocols for the Internet (IP and ICMP), transport (TCP, UDP) and application (DNS, OSPF, and RSVP). Due to the complexity of security issues in the Connectivity layer, where possible it is essential to use existing standard. Authentication solutions for environments should have the following characteristics [75]:

Single Sign on One log on (authentication) should be all that is necessary for users to access multiple Grid resources. No further user invention is necessary. An example of a similar system is Microsoft Passport [76].

Delegation [77] To enable programs to access resources on which the user is authorised, a user must be able to endow a program with the ability to run on the users' behalf. Services such as Microsoft's ASP.NET technology provide such a facility which is part of the Microsoft.NET Framework and importantly provides a container platform for hosting Web Services on the IIS Web server.

Integration with various local security solutions Interoperation with various local security systems is necessary in any Grid solution. It is not feasible to require replacement of local security systems but instead must allow mapping into the local environments such as Kerberos [78] or UNIX security. It may be possible to employ the security models of runtime environments of for instance .Net framework [79] of Java JRE [80].

User-based trust relationships It is required that for users jointly access multiple resources from different providers, no interaction or cooperation is necessary among the security system for the configuration of the security environment.

2.7.1.3 Resource

The Resource layer defines protocols, APIs and SDKs for secure negotiation, initiation, monitoring, control, accounting, and payment sharing operations. This layer builds upon the Connectivity layers communication and authentication protocols and calls protocols from the Fabric layer functions to access and control local resources. The Resource layer does not address global state and atomic actions as its protocols are only concerned with individual resources. The main classes of Resource Layer protocols are:

Information protocols allow information retrieval of the structure and state of the resource. Example protocols include LDAP [81] used by GRIP [82] in Globus.

Management protocols negotiate access to a shared resource e.g. resource requirements, and operations to be performed. As the responsibility of management protocol is to instantiate sharing relationships, they must apply policy of the underlying shared resource. The GRAM protocol [83] is used for allocation of computational resources, monitoring, and control of computation.

2.7.1.4 Collective

The Collective layer is primarily concerned with the coordination of multiple resources. Built upon the Resource layer this layer may implement a wide variety of sharing behaviours without the need for new resource requirements, these include, Directory services, Co-allocation, scheduling and brokering services, Monitoring and diagnostics, Software Discovery services and many more. The Collective layer, unlike the Resource layer, is not concerned with any single resource but the global state and interactions among resources.

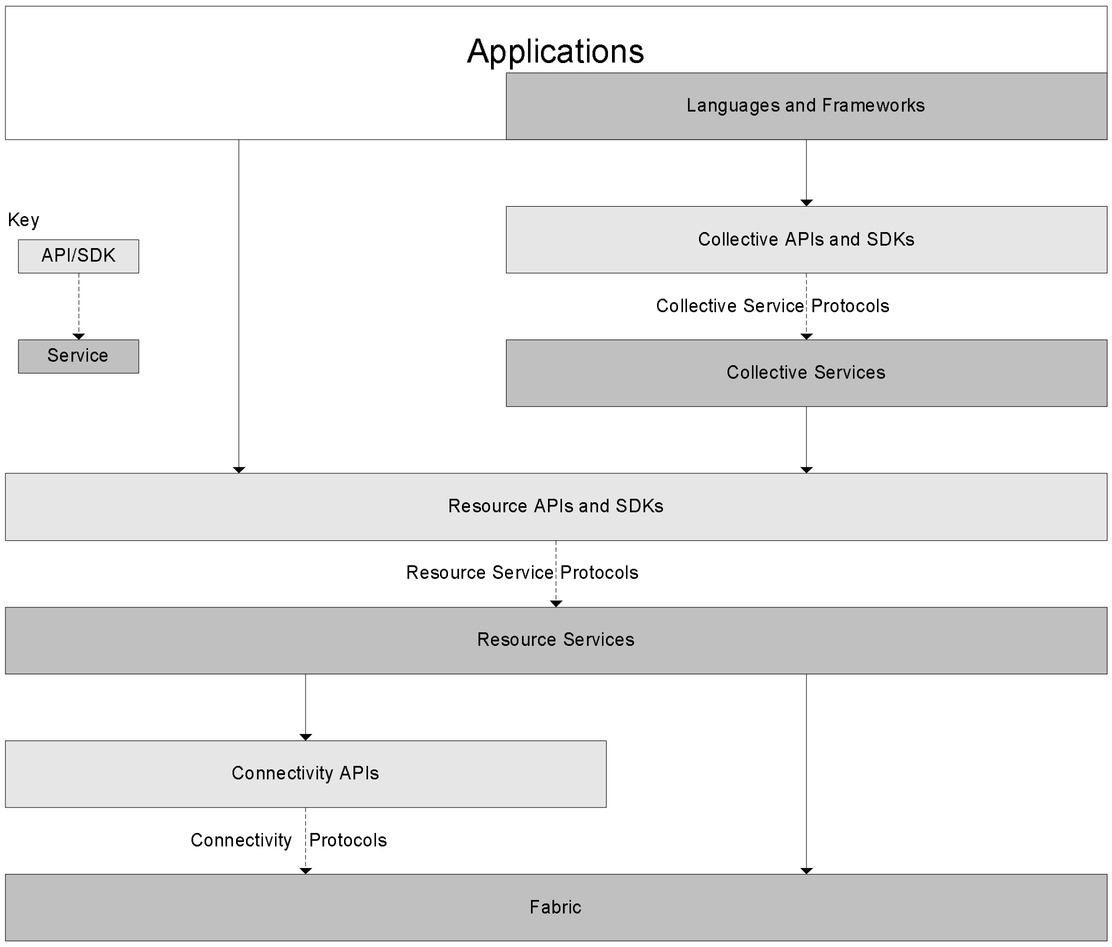


FIGURE 2.7: Programmers view of the Grid Architecture.

2.7.1.5 Application

At the top most level of the Grid architecture is the application layer. The user interacts with the system within the application environment. Shown in figure 2.7 is the programmers view of the Grid Architecture.

We can see in figure 2.7 that the application layer is emphasised and contains Languages and Frameworks, that themselves may contain many APIs and protocols. At each layer of the architecture we have APIs and SDKs interfacing with the underlying services.

2.8 Summary

In the first part of this chapter we studied existing distributed technologies and identified that transparency, openness and scalability where fundamental characteristics which the Grid should adopt. In addition, we identified which key technologies, architectures and practices the Grid must build upon to reach an Internet-level. We conclude that because of the success, power and pervasiveness of technologies present in existing distributed technologies, such as the WWW, the Grid success is dependent upon its ability to harness these technologies and bring them together with new technologies in order to enable greater access and integration of resources.

We have characterised the Grid by its attempt to provide dynamic and coordinated large-scale resource sharing through service-oriented Web Service technologies. The second part of the chapter rationalised the usage of SOA by showing that its promotion of reuse and interconnection of resources facilitates cost-effective and quick adaption of distributed computing environments to changing users requirements. The challenges of Grid computing have been explored and the subsequent requirements, of which specifically, interoperability, encapsulation and loose-coupling have been identified as key desirable characteristics of its technologies. Consequently, justification was given for Web Service as the best technology for implementing Grids because of its usage of key existing distributed technologies and its adoption of a highly interoperable loosely-coupled SOA approach.

Compared with distributed object-based architectures, such as used by Globus (GT2) which employs RMI/RPC based distributed message passing, SOA has significant advantages because, unlike the latter, the SO concept was specifically crafted from conception to cater for the needs of distributed systems for interoperability and easy maintenance, component reuse and integration. In addition, distributed object-based architectures have been extensively employed for many years and systems which employ this architecture have failed to reach a scale desired by the Grid. SOA does not intend to replace existing distributed object-based systems however, it does offer a technological bridge that will enable them to link together at a higher level into larger systems. This is very similar to TCP/IP stack which itself was initially developed to bridge and provide interoperability between computer networks that employed incompatible or poorly scalable network technologies. Eventually, as networks were upgraded, TCP/IP became the defacto standard employed in nearly all computer networks. Therefore, this work focuses on employing SOA and Web Service technologies to link together legacy technologies and as the native architecture to build and integrate new service components.

Nevertheless, the current rapid rate of change and addition of new Web Service standards presents a challenge for distributed software developers. The newness of the Grid means that whilst standards exist they have not yet reached a maturity where developers can be sure how stable they are (i.e how quickly they may be deprecated) or if they will be widely supported. It is for this reason that the work presented in this concentrates on the core Web Service standards XML, SOAP, WS-Security and DIME.

The assumption is that the Grid should be built using the principles of SOA and that Web Services currently provide the most suitable technological platform for constructing sophisticated Grid applications. However, in the next chapter we examine in depth the challenges faced when sharing and aggregating computation and other resources types within scientific and engineering Problem Solving Environments. We will demonstrate the challenges faced using existing distributed technologies and approaches, whilst exposing additional challenges not yet resolved by Grid technologies.

Chapter 3

Challenges in Problem Solving Environments

The research work on service-oriented (SO) Grid computing presented in this thesis has been carried out in the context of the Grid Enabled Optimisation and Design Search for Engineering (Geodise) project. In addition, the research work has lead to the building of an experimental Grid-Enabled Problem Solving Environment (PSE) for Computational Micromagnetics.

The first section of this chapter concentrates on the problem solving challenges of computational micromagnetics. It is the field associated with the study of magnetic characteristics of materials at a sub-micron scale. Its research has direct application in improving the data density of hard drives used in PCs, servers, and data repositories. It represents a field of computational science that's problem solving process would benefit greatly from Grid computing in providing greater access and sharing of resources and enabling technologies for seamless integration of tools and data. It serves as a suitable scenario for study as it has a problem solving process often hindered by simulation tools, which have issues that include non-standard operation, application-centric data formats, tightly-coupled resource requirements, and poor interoperability with other tools. We highlight from these specific challenges that Grid Computing must resolve in order to facilitate the problem solving process. In addition, our discussion serves as an introduction to the work on development of an experimental Computational Micromagnetics Grid-enabled PSE in chapter 7.

The second section of this chapter looks at Geodise, a current state-of-the-art Grid-enable PSE. Geodiseis "a Grid-based generic integration framework for computation and data intensive multidisciplinary design optimisation tasks while maintaining the autonomy of each individual domain expert". Its service-oriented style makes it an ideal research topic on the application and behaviour of Web Services in Grid computing from the perspective of computation, data management, software applications, intelligent knowledge repository, and service integration.

In this chapter we will compare the Grid requirements of Computational micromagnetics with the solutions provided by GEODISE's SO style identifying key unresolved issues and presenting solution methodologies, which form the premise for work contained in the rest of the thesis.

3.1 Problem Solving Environments (PSEs)

Describing a problem solving environment (PSE) is difficult because of the relatively wide diversity of research topics that it covers. Consequently, like other computer science terms, it has resisted a universally agreed definition. Nonetheless, understanding of what a PSE is essential before we can begin to think about what it should be and subsequently its technical requirements: in particular, its capabilities, who the users are and their needs, fundamental architectures, and resource/component reuse. One of the best attempts at a high-level definition of the term PSE comes from [8] whom write:

"A PSE is a computer system that provides all the computational facilities necessary to solve a target class of problems. These features include advanced solution methods, automatic or semiautomatic selection of solution methods, and ways to easily incorporate novel solution methods. Moreover, PSEs use the language of the target class of problems, so users can run them without specialized knowledge of the underlying computer hardware or software. By exploiting modern technologies such as interactive color graphics, powerful processors, and networks of specialized services, PSEs can track extended problem-solving tasks and allow users to review them easily. Overall, they create a framework that is all things to all people: they solve simple or complex problems, support rapid prototyping or detailed analysis, and can be used in introductory education or at the frontiers of science."

Whilst this definition gives an overview of PSEs and what it may offer to improve the problem solving process, its realisation will require significant effort therefore further definition is required to fully understand the benefits that PSE can bring. What follows is an attempt to give a more specific description of what a problem-solving process is, the motivating factors behind PSE, and the development of the software infrastructure for the environment.

3.1.1 Problem Solving Processes and Motivation for PSE

A problem solving process is the course of actions taken to discover the answer to a question. The characteristics of the problem solving process need to be known to determine what the users wants to do and how the PSE can assist in the course of actions. Therefore it is important to have a good understanding of the problem solving process itself. There are many different types of problem-solving of which some common scenarios are listed below:

Manufacturing A plant manager or technical officer must monitor and adjust some process to respond to current conditions, e.g., properties of input materials, the current operating environment. Possible adjustments are evaluated for their predicted effect on the manufacturing process. The goal is to produce a product in the most timely, cost-effective manner, while maintaining quality. Speed may be especially critical in this setting, where shutting down the manufacturing line is virtually unacceptable. Previous solutions and expert help are extremely valuable.

Research A scientist wants to compare models or algorithms or implementations. A small number of test cases are used. Accuracy or computational performance are of primary interest. Comparison with experimental data may be required. The goal is to model some physical phenomenon more accurately, or to develop a better solution algorithm.

Design and development An engineer is trying to design a better system of some kind. The mathematical models and numerical methods used are relatively fixed, although a few key parameters (e.g., the geometry) may be varied. The goal is to find a better design, according to some criteria. This optimisation process may be automatic, but is most likely to involve human-in-the-loop steps. Previous "nearby" solutions are of considerable help.

The problem solving process is often a multidisciplinary affair and it is in this scenario that PSE offer the most. Demand exists for PSEs that offer software infrastructures to support many, geographically located engineers, collaborating on a design. In particular, multidisciplinary design optimisation (MDO), such as used in aerospace engineering is a good example of a problem domain that benefit from PSEs. In this scenario of large-scale MDO, design of aircraft requires input from a combination of fields from fluid dynamics to controls and structural mechanics. Consequently, design processes can be extremely heterogeneous involving many models, codes [1], and people and computer resources possible in different locations. MDO is unfeasible without capable and usable software environments.

The requirement for a PSE rises with the complexity and heterogeneity of the problem solving processes i.e. more realistic and larger quantity of models, more people, codes, resources and paths of investigation. The set of tools available to the computational

[1]Codes (short for source code) is a commonly used term in the fields of science and engineering for a purpose- or problem-specific single software program often user-written and typically used in the processes of design of experiments, optimisations and analysis

scientist and engineer has increased significantly. A few decades ago, choice was limited to FORTRAN-based codes, calculation on mainframe computers, and output as tabular data or simple plots. Now scientists may choose to write their codes in visual programming environments using a variety of languages from the general purpose (e.g. C++, Matlab, Python, and Java) to the performance-oriented (e.g. C, Fortran) and problem-oriented; they employ complex and rapidly evolving set of parallel computer architectures; they use three dimensional visualisation, with animation, and analyse the output in immersive interactive environments; and they often need to collaborate on projects with people dispersed across the world. What is needed is a usable software infrastructure to aid scientists to manage and coordinate all these tools to allow more effective problem solving.

Work presented in this thesis has been carried out in the context of the field computational micromagnetics, and engineering optimisation and design search. These two scenarios where chosen as examples to work on because they share the following key problem-solving characteristics arising in important areas of manufacturing, engineering research and development, and the natural sciences:

- They are based on sophisticated mathematical models.
- They require sophisticated numerical solution methods.
- They require large scale high performance computing resources.
- They are often multi-disciplinary.
- They are often solved by groups rather than individuals.

3.1.2 Characteristics of the Environment

Already covered in section 2.5, is a high-level overview of the many challenges faced in Grid computing that are generally pertinent to the challenges faced in science and engineering. This is no coincidence as one of the driving forces behind research into the Grid, other than business, has been in the better allocation and sharing of resources in environments for problem solving processes. In addition, in the introduction, section 1.2, we also looked at some of the technical challenges faced by scientists and engineers and the key areas of Grid computing which aim to provide a solution. The adjectives below represent a list of characteristics that are desirable, to one degree or another, in engineering and scientific PSEs:

Problem-oriented The PSE should allow the specialist to concentrate on their discipline without having to become a self-taught computer scientist.

Collaborative It is very common that to see collaborative science and engineering often with geographically remote participants.

Persistent Problem solving sessions often occur in different locations over a periods of time. Consequently persistence is necessary for the continuance of these sessions but more impermanently to maintain a record of the work actions that may be reused in some form of intelligence.

Powerful Without sufficiently powerful hardware and software resources it will difficult to solve problems in a timely fashion or perform an serious research.

Integrated Many problems and solution strategies are extremely heterogeneous. One of the most challenging aspects of good PSE design will be to manage all this heterogeneity in an integrated way so that the user may operate in the system in predictable and consistent environment.

Open, flexible, adaptive In many settings it is important that PSE-builders and sophisticated users be able to tailor or add to the functionality of a PSE.

Graphical, visual Most large scale applications require visualisation of results; many rely on graphical input as well.

Intelligent In certain settings it is likely that a PSE could supply some expert "advice" in choosing among several numerical methods, for example, or in advising a machine operator on the factory floor about process control.

PSEs must bring together state-of-the-art research and technology from almost all areas of computer science in order to facilitate the creation of powerful and importantly easy-to-use computing environments. Emphasis must be paid particularly to the latter feature because, whilst drawing from computer science, it must not require a computer scientist to operate or it loses its power and certainly its appeal. It makes sense that scientists' productivity would increase if they where able to concentrate on solving their target problem rather than digress through the challenge of problem solving the tools. Relevant subdisciplines, which its research will need to be brought into PSE, include artificial intelligence, collaborative computing, graphics and visualisation, human-computer interaction, networks and the World Wide Web, numerical analysis, object-oriented computing, parallel and distributed computing, and software engineering.

3.1.3 Existing Problem Solving Environments

A number of PSE applications have been developed that bring together the necessary hardware and software resources to operate in a particular domain without the need for specialised knowledge of the underlying infrastructure. Examples of these include BioSoftLab [84] a scientific laboratory environment, Geodes Elements [85] an engineering-scientific workspace, and *'Parrallel (///) ELLPACK'* [86] a problem solving and development environment for partial differential equation based applications. Whilst powerful, they are neither open, flexible or adaptable because they couple together resources in a domain-specific manner, often employ tightly-coupled bespoke enabling technologies and provide only limited distributed infrastructures. This makes integration and exploitation of computer resources, and collective collaboration at a global scale an infeasible task. On the other hand, technical computing environments such as Matlab[2] provide more flexible general purpose PSE that enable bringing together a broader range of scientific libraries and computational resources. However, they do not provide the underlying distributed infrastructure to enable seamless integration and sharing of resources. Nonetheless, as will discussed in section 3.3 they do provide a suitable working environment within which scientists and engineers will be familiar that can employed as the front-end or "portal" into a distributed PSE environment.

3.2 Study of the Computational Micromagnetics Problem-Solving Process

Micromagnetics is the field associated with the study of magnetic characteristics of materials at a sub-micron scale. Its research has direct application in improving the data density of hard drives used in PCs, servers, and data repositories that store and retrieve large amounts (Gigabytes to Terabytes) of digital data. It represents a field of computational science in which the collaborative problem solving process would benefit greatly from Grid-enable PSE in providing greater access and sharing of resources and enabling technologies for seamless integration of tools and data. It serves as a suitable scenario for study as it has a problem solving process often hindered by simulation tools, which have issues that include non-standard operation, application-centric data formats, tightly-coupled resource requirements, and poor interoperability with other tools.

3.2.1 Motivation and Problem of Interest

The technology behind high-capacity data storage media used in hard disk drives is rapidly approaching fundamental physical limits. However, as hard drives capacities creep up new ways are found to consume ever-greater amounts of space. People are demanding larger hard drive capacities to store more of their high-quality multimedia data and games. Businesses demand larger repositories to store more information on customers and partners. Moreover, hard drive technology, once exclusive to the PC, is increasingly being found in commodity goods from personal video recorders such as, the Humax Television Hard Disk Recorder series, to fridge freezers such as LG Electronics's Digital Multimedia Fridge Freezers.

[2]Homepage of MatLab http://www.mathworks.com/

3.2.2 Overview of Problem-solving Process

To overcome the data storage limits of current hard drives, new research is required to examine methods that improve the areal density (data bits per unit area) of the magnetic media used to store data on a hard drive. Computational micromagnetic tools OOMMF [87] and Magpar [88], based on the Finite Difference (FD) method [89] and the Finite element/Boundary Element (FE/BE) method [90] respectively, have been developed to accurately simulate the magnetic characteristics of very small particles. Computational micromagnetics is a frontier problem. Analytical methods are inadequate for complex geometries, being limited to the most primitive cases; therefore numerical modelling and simulation using FD and FE/BE methodologies is the only viable solution.

3.2.3 Specific Environmental Requirements

Computational analysis using OOMMF and Magpar requires a significant amount of resources, such as those provided by cluster computing environments. Most problems require powerful tools (i.e. OOMMF and Magpar); hundreds of megabytes of RAM and many gigabytes of data storage to run simulations and persist the results; and plotting and visualisation tools to analyse the results in three-dimensions. In addition, the problem solving of micromagnetic simulation is complex even outside the actual core modelling code.

Within the field of micromagnetics research often involves close collaboration between experimental and computational scientists. Experimental research in micromagnetics is generally costly and imprecise because of the sub-micron scale of the particles that form the basis for hard drive media. On the other hand, computational scientists can get a greater amount of detailed high-resolution data on the magnetic characteristics of differently shaped and arranged particles than would otherwise be possible in the laboratory and at a fraction of the cost. Nonetheless, collaboration of both parties is necessary in that experimental scientists play an essential role in verifying computational results and models, in addition, to discovering new avenues of research and vice-a-versa.

3.2.4 Brief Overview of Hard Drive Technology

A hard drive, shown in figure 3.1, consist of a number of platters (disks). Each platter has a planar magnetic surface to which digital data maybe stored and retrieved. Data is written to the platter by transmitting an electromagnetic flux through a read-write head that is very close to a magnetic material, which in turn changes its polarisation due to the flux. The magnetic fields on the platter cause electrical changes in the read-write head as it passes over a platter, which allows the reading of data.

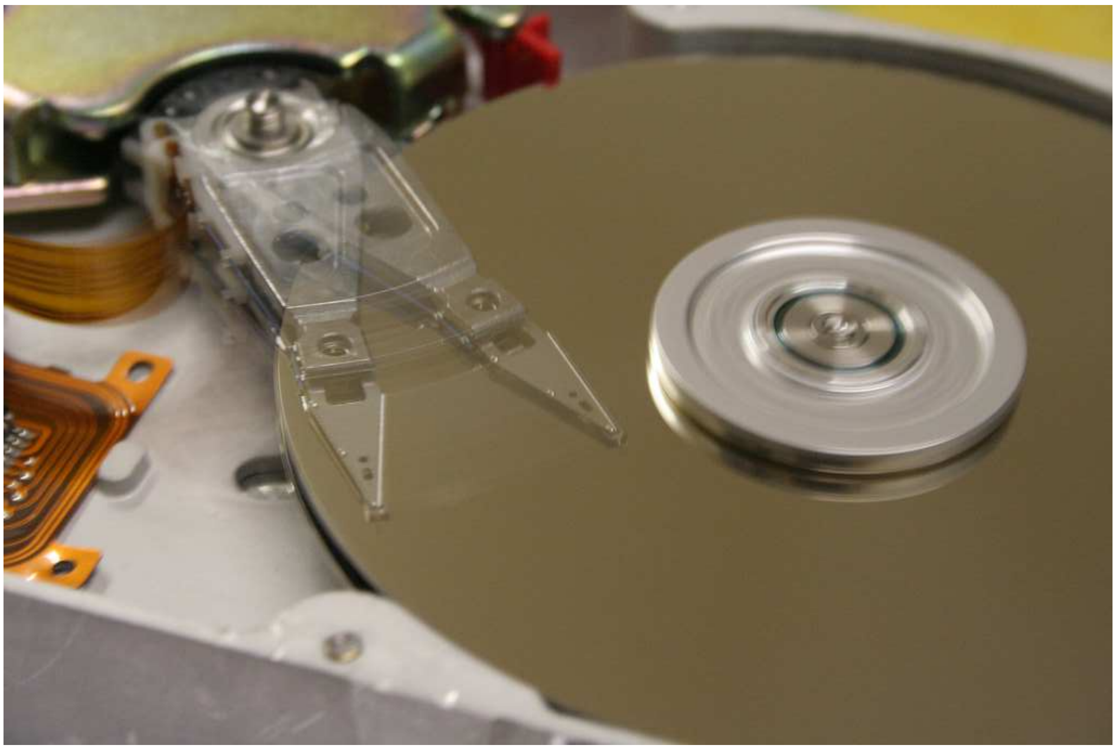


FIGURE 3.1: Image[4] of the inside of a hard disk (without cover) showing the read/write head moving across the spinning platters.

In order to cover the whole platter in data, it is spun at a constant velocity and heads move backwards and forwards across the platter reading and writing data. High data transfer rates to and from the hard drive are achieved by spinning the platters faster, typically from 5,000 to 15,000 revolutions per minute in modern PC hard drives. This allows data transfer rates in access of 50MB/s. Consequently, the heads must be more sensitive and accurate so they can read and write data faster.

Several platters may be stacked on top of each other to increase the data storage capacity of the hard drive. However, the high rotational velocity of the platters puts large mechanical stresses on the driving motor and housing components, which limits the number of platters that may be stacked in a single hard drive. On the other hand, platters can be made with larger radii to increase their storage capacity. However, this is no longer popular with hard drive manufacturers because it places greater strains on the mechanical components of a hard drive requiring a reduction in spin speed to maintain reliability whilst increasing the read/write latency as the head must travel further across the platter to reach the data. In addition, large platters are unsuitable for small form factor devices, such as cameras, laptops and hand held music/video players.

The head must float very close to the surface of the platter in order to detect the small magnetic fields used to represent the data. Consequently, platters must be very smooth

[4]This image is the unmodified work of Alpha Six (http://www.flickr.com/photos/alphasix/). It is licensed under the Creative Commons Attribution ShareAlike License version 2.0: http://creativecommons.org/licenses/by-sa/2.0/

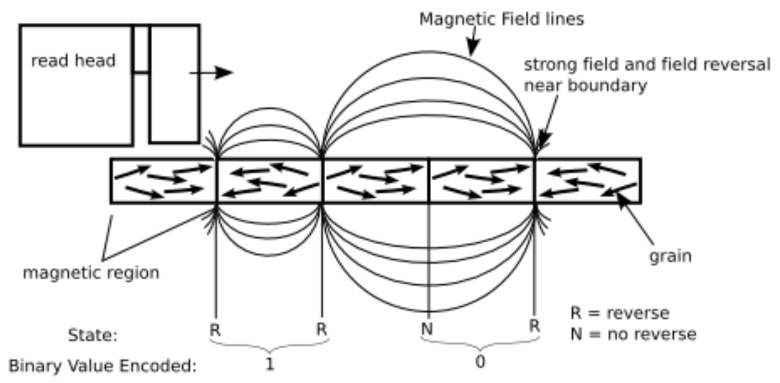


FIGURE 3.2: Illustration[6] of the magnetic surface of a hard disk platter showing its operation, in the case of, binary data encoding using frequency modulation.

and flat so that it can spin at high velocities without vibrating and allow the head to move freely without striking the surface and destroying the data. In addition, the platter must be very rigid so that in does not deform under the high rotational velocities. Sophisticated manufacturing processes are necessary in order to produce platters that meet the high data-density and data-transfer requirements of modern hard drives.

3.2.4.1 New Manufacturing Approaches

Traditionally, in order to produce platter shaped magnetic media suitable for high-speed, high-capacity hard drives, manufacturers use a sputtered non-patterned media fabrication process. A piece of glass or aluminium that provides the base rigidity of the platter is cut into a flat round disk, coated with a substrate then 'sputtered' (i.e. sprayed) with very small particles (also known as grains, see figure 3.2) of ferromagnetic material, such as cobalt-chromium-platinum-tantalum. The substrate provides the glue that bonds the particles to the base platter material. The sputtering process enables the application of a sub-micron thick layer, which is required for the platter to store a high magnetic density of data and ensures an even flat layer of the magnetic material faces the head.

However, the sputtered non-patterned media fabrication process does not, as its name suggests, control the shape or arrangement of magnetic particles on the platter. First developed in the 1970s, the process was initially designed to produce hard drive media with capacities of a few megabytes. In order to squeeze more out of the process, manufacturers have developed read-write heads capable of detecting magnetic flux at close to

[6]This illustration is the unmodified work of Allan Haldane and License under GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. Subject to disclaimers.

Acronyms	Definitions
gcc	Gnu Compiler Collection
g77	GNU Fortran 77 compiler
zlib	File compression library
ParMETIS	Parallel Graph Partitioning and Fill-reducing Matrix Ordering
SUNDIALS	SUite of Nonlinear and DIfferential/ALgebraic equation Solvers
ATLAS	Automatically Tuned Linear Algebra Software
Python	Programming language
netgen	Automatic 3D tetrahedral mesh generator
libpng	Image format reference library
MPICH	Message-Passing Interface
LAPACK	Linear Algebra PACKage
sh	Bourne shell scripting language
PETSc	Portable, Extensible Toolkit for Scientific Computation
AVS UCD	Meshing file format
mpmaker	Script to generate material input files
mifmaker	Script to generate micromagnetic input files
magpar	Parallel Finite Element Micromagnetics Package
TAO	Open source implementation of Common Object Request Broker Architecture
OOMMF	Object Oriented MicroMagnetic Framework
Tcl/Tk	Programming language and graphical user interface toolkit
xmgrace	Plotting tool
VTK	Visualization ToolKit
MayaVi	Scientific data visualiser

Table 3.1: Key for micromagnetic process in figure 3.3.

the fundamental limits of the media created by the sputtered non-patterned media manufacturing process. However, research has shown [91] that this manufacturing process is rapidly reaching fundamental limits.

It has been conceptually demonstrated [92, 93] that a patterned media manufacturing process - controlling the shape and arrangement of the particles that cover the hard drive platters - can increase areal densities by two orders of magnitude; increasing a single platter's capacity from a 100 Gigabytes to Terabytes. Research is necessary to determine what shape and arrangement of particles produce the greatest areal densities whilst being cost effective to produce.

3.2.5 Computational Research Tools

Computational Micromagnetics makes use of tools such as OOMMF and Magpar to simulate the magnetic properties of particles that form the media for hard drive platters. There are two core strategies for simulating the shape and arrangement of small magnetic particles. The fundamental difference between the FD and the FE/BE methods, in this case, is that where FD problems have a fixed simulation cell size to describe the physical geometry of the particle system, FE/BE problems have a variable-sized mesh for the same.

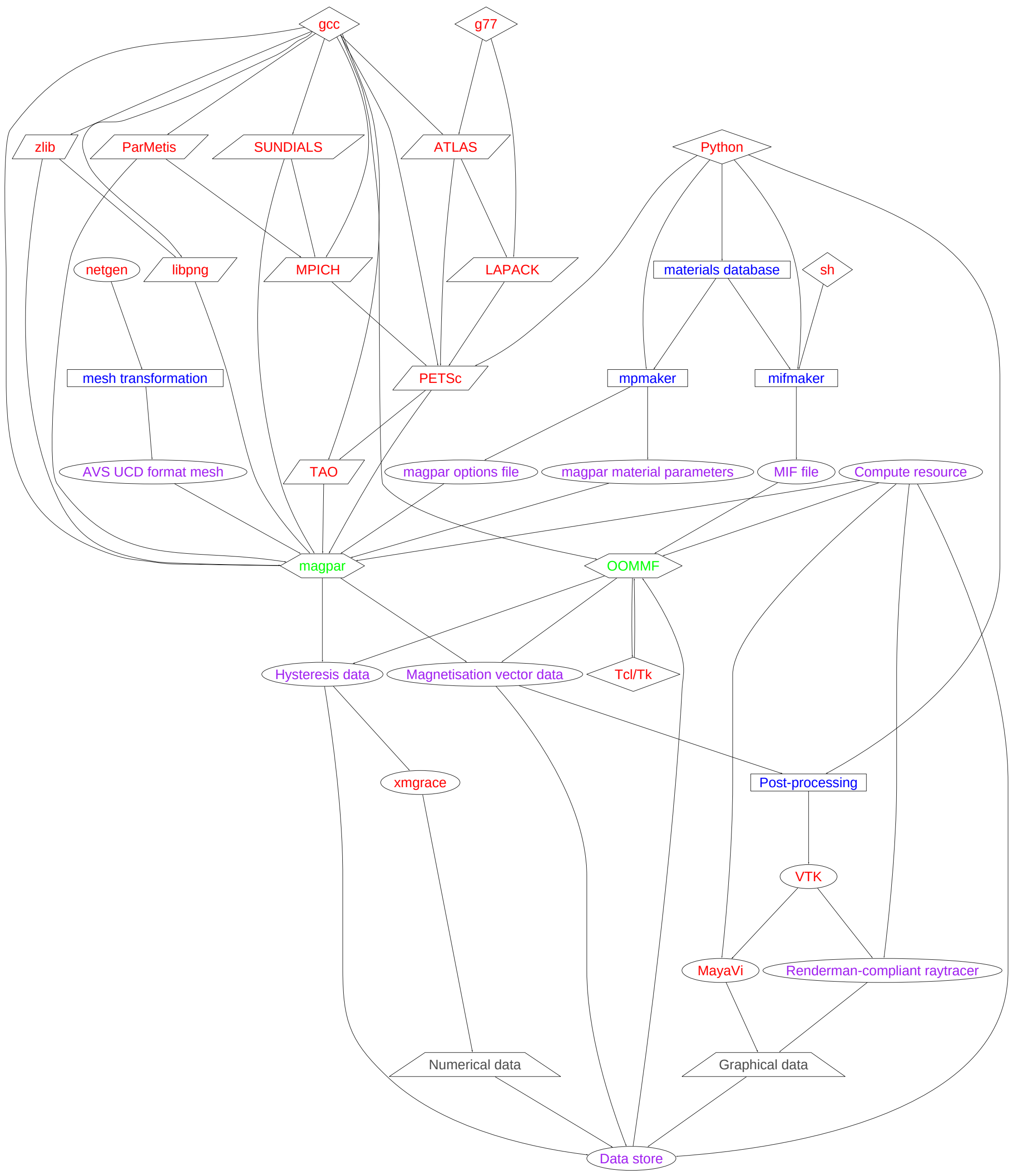


FIGURE 3.3: Interrelationships between packages and tools in micromagnetic modelling, simulation and analysis process as used in the research work by Dr. Richard P. Boardman [94]. Items shaded purple indicate stored data, red and blue indicate dependent packages. See table 3.1 for the key.

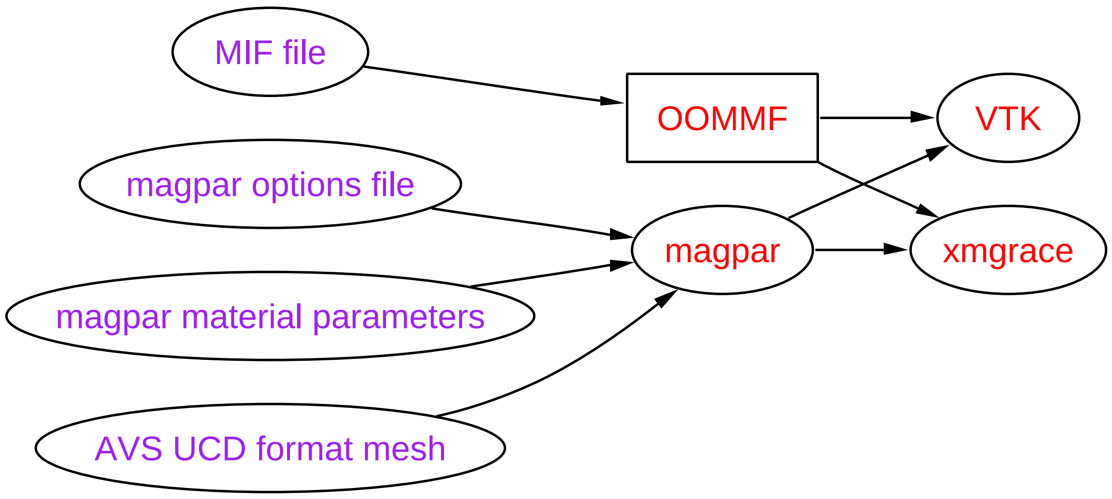


Figure 3.4: An example usage of the software tools in a micromagnetic simulation and visualisation. Figure shows the input files in purple and tools in red. Arrows indicate the flow of data between tools.

Packages, such as OOMMF and Magpar, respectively offer FD and FE/BE numerical modelling for micromagnetic research. Whilst open source, each package has its own specific problem setup strategies, execution requirements, and results post-processing that require a loose framework in order to bring a degree of unification and aid effective tool usage. Figure 3.3 shows the interrelationships and interdependencies of other packages and tools with OOMMF and Magpar whilst figure 3.4 shows an example of the tools being used together to perform a simulation and visualisation.

3.2.5.1 Object-Oriented Micromagnetic Framework (OOMMF)

The Object-Oriented Micromagnetic Framework (OOMMF) is a collection of applications for solving micromagnetics problems. Its applications are designed to be portable, flexible, and extensible, with user-friendly graphical interface. The applications are written in C++ and Tcl/Tk. OOMMF targets a wide range of heterogonous computing platforms including Linux and Windows NT.

OOMMF version 2.1 includes an application called the OOMMF eXtensible Solver (Oxs). This is an extensible micromagnetic computation engine capable of solving problems defined on three-dimensional grids of rectangular cells holding three-dimensional spins. Oxs provides the ability to simulate and study in three dimensions materials' magnetic properties at scales and of geometries not currently possible in the laboratory. Oxs is particularly useful for scientists who are studying the problem of improving data-density by altering the shape, size, and arrangement of very small ferromagnetic particles on patterned media-based hard drives.

Problem definition for Oxs is accomplished using input files in the MIF 2.1 format. An example of a MIF file is shown in listing 3.1. The MIF file specifies, using the

Extension	Lines	Definitions
Oxs_BoxAtlas	8-11	Specifies the atlas of geometric volume of spaces
Oxs_RectangularMesh	14-17	Specifies the rectangular mesh to use
Oxs_UniformExchange	19-21	Specifies the exchange coefficient across all spaces
Oxs_UZeeman	23-28	Specifies the uniform (homogeneous) applied field energy
Oxs_Demag	30-30	The standard demagnetisation energy term
Oxs_EulerEvolve	32-35	The evolver responsible for updating the configuration from one step to the next
Oxs_TimeDriver	37-55	The driver for controlling time evolvers
Cone	57-67	Specifies the shape of the magnetic particles

TABLE 3.2: Table describing the Oxs extensions employed in listings 3.2. See OOMMF User's Guide [87] for full description of the extensions.

TCL/TK language, how Oxs should carry out the micromagnetic simulation. Contained within the MIF file are parameters and algorithms that define the materials magnetic characteristics, the arrangement of particles, and the shape, size and number of cells used to represent each particle as a FD mesh. Scientists can adapt the values of these properties to study, for instance, how changing the shape of a small piece of iron e.g. from a cube to a cone, affects its magnetic characteristics.

Scientists can also define in the MIF file what solution methodology and how Oxs should use it to simulate the magnetic particles. This is done through selection of code libraries and modules that contain solution methodologies necessary to solve the target problem. Methodologies such as Landau-Lifshitz-Gilbert (LLG) [10], produce more realistic simulations, whilst others produce less realistic results but converge on a solution more rapidly. The LLG must be iterated to within a certain stability tolerance to ensure the correctness of results. The Conjugate Gradient method, whilst suitable for the general case, with certain system geometries, such as tori, it cannot converge on a solution, which consequently causes perpetual computation. Therefore, more complex solution methodologies such as LLG are required to reach a convergence. Typically, depending on the complexity of the system and solution methodology defined in the MIF problem statement, the calculation of the systems magnetic characteristics may take from hours to weeks to produce.

Oxs interprets the information in the MIF and simulates the magnetic system. Oxs outputs a set of result files containing the magnetic characteristics of the system's cells in vector format (OMF) and the overall system in tabular format (ODT), examples of which are shown in listings 3.3 and 3.4 respectively. Contained within these files are the sizes and directions of the magnetisation vectors for the cells, and holistic system attributes. Typically, scientists then take these results and post-process those to produce hysteresis graphs and three-dimensional visualisations of the particles to aid in their analysis. The latter of which requires significant computational processing as discussed in section 3.2.6.

```
# MIF 2.1

set pi [expr 4*atan(1.0)]
set mu0 [expr 4*$pi*1e-7]

set TIMEDRIVER 0

Specify Oxs_BoxAtlas:atlas {
  xrange { 0 5E-08 }
  yrange { 0 5E-08 }
  zrange { 0 8E-08 }
}

Specify Oxs_RectangularMesh:mesh {
  cellsize { 5e-009 5e-009 5e-009 }
  atlas :atlas
}

Specify Oxs_UniformExchange {
  A    1.3e-011
}

Specify Oxs_UZeeman "
  multiplier [expr 0.001/$mu0]
  Hrange {
    { 500.0 0.0 0 -500.0 0.0 0 500 }
  }
"

Specify Oxs_Demag {}

Specify Oxs_EulerEvolve {
  alpha 0.5
  start_dm 0.01
}

Specify Oxs_TimeDriver {
  basename 5E-08-8E-08.mif_0008
  evolver Oxs_EulerEvolve
  stopping_dm_dt 0.01
  mesh :mesh
  stage_count 0
  stage_iteration_limit 0
  total_iteration_limit 0
  Ms { Oxs_ScriptScalarField {
    atlas :atlas
    script { Cone 795774.715459 }
   }
  }
  m0 { Oxs_UniformVectorField {
    norm 1
    vector { 1.0 0.0 0}
  }
 }
}

proc Cone { Ms x_in y_in z_in} {
    set x [expr 2.*$x_in - 1.]
    set y [expr 2.*$y_in - 1.]
    set z [expr 2.*$z_in - 1.]
    set left_side [expr ($x*$x) + ($y*$y)]
    set right_side [expr (($z+1)/2) * (($z+1)/2)]
    if { $left_side <= $right_side } {
        return $Ms
    }
    return 0
}

# set some outputs

Destination archive mmArchive

Schedule    DataTable archive                      Stage 1
Schedule    Oxs_TimeDriver::Magnetization archive  Stage 1
```

Listing 3.1: Example MIF task code to model cone shaped particles. MIF files are written in the TCL language. To reduce development time the code was auto-generated using a Python script which is run from the command line as shown in listing 3.2. Table 3.2 gives a description of the extensions employed in this MIF example.

```
X:\mm\scripts\mifmaker.py --cone --x=5E-08 --y=5E-08 --z=8E-08 --high=500
    --low=-500 --name=5E-08-8E-08.mif --stages=500 --cell=5E-09
    --direction=down --exchange=1.3E-11 --magnetisation=795774.715459
```

Listing 3.2: MIF code in listing 3.1 was created with a Python script called MIF-Maker which was written by Richard Boardman and Hans Fangohr at University of Southampton. It was run from the command line with the above design variable input arguments.

```
# OOMMF: irregular mesh v0.0
## File: sample.ovf
## Boundary-XY: 0.0 0.0 1.0 0.0 1.0 2.0 0.0 2.0 0.0 0.0
## Grid step: .25 .5 0
#   x       y       z         m_x       m_y       m_z
   0.01    0.01    0.01    -0.35537   0.93472  -0.00000
   0.01    1.00    0.01    -0.18936   0.98191  -0.00000
   0.01    1.99    0.01    -0.08112   0.99670  -0.00000
   0.50    0.50    0.01    -0.03302   0.99945  -0.00001
   0.99    0.05    0.01    -0.08141   0.99668  -0.00001
   0.75    1.50    0.01    -0.18981   0.98182  -0.00000
   0.99    1.99    0.01    -0.35652   0.93429  -0.00000
}
```

Listing 3.3: Example vector output of Oxs.

```
# ODT 1.0
# Table Start
# Title: This is a small sample ODT file.
#
## This is a sample comment. You can put anything you want
## on comment lines.
#
# Columns: Iteration "Applied Field"  {Total Energy}     Mx
# Units:         {}           "mT"            "J/m^3"         "A/m"
                103           50             0.00636        787840
               1000           32             0.00603        781120
              10300        -5000             0.00640       -800e3
# Table End
}
```

Listing 3.4: Example tabular output of Oxs.

3.2.5.2 Application of OOMMF for Finite Dimensional analysis of Patterned Media

The FD problem solving process (the OOMMF path in figure 3.3) starts with problem initialisation stage where, for example, data and information is drawn from the scientist, databases and files and collated into the MIF problem specification file.

Once a MIF file has been created OOMMF must be configured for execution on computational resource such as a cluster of PCs. This will involve the selection of Oxs binary, which matches the platform of the available computational resources and creation of scripts to initiate the environment, which involves, for instance, specifying the location to modules and working directory such that Oxs may execute. Unfortunately, OOMMF only supports sequential execution on single processor computers and therefore cannot take advantage of parallel HPC clusters. However, different MIF scripts may be executed concurrently in a batch mode or HTC environment that, for example, ideally suits simulation of micromagnetic systems of different sizes to explore parameter sets and identify system trends.

OOMMF produces dataset results in the form of ASCII or binary encoded magnetisation vector data suitable only for computer consumption. Effective analysis of these results inevitably requires visualisation however, first OOMMF's output data must be translated into formats suitable for three-dimensional rendering packages such as, VTK [95] and graphing tools such as, Gnuplot [96].

3.2.5.3 Application of Magpar for Finite Element/Boundary Element analysis of Patterned Media

Modelling and simulation of micromagnetic systems using the FE/BE method involves a more complex solving process than the FD method. Meshes passed to Magpar must be in one of two application-specific file formats, see figure 3.4, that it understands. In addition, Magpar requires a file containing the simulation parameters and another file containing the material parameters for the target micromagnetic system. The Magpar software supports simulation of huge micromagnetic systems over parallel HPC clusters. The datasets that Magpar generates can be combined with the input geometry to generate a complete data file that can be then translated into an appropriate format for visualisation.

3.2.6 Performance Challenges

Micromagnetic research into the shape of particles is severely computationally bound. The Oxs application of OOMMF models geometrically-static systems i.e. where the size, geometry, and arrangement of particles remains constant throughout the simulation. In order to do parametric studies or optimisation where, for instance, a scientist is aiming for high remanence (magnetic "memory") by varying the height and diameter of cone shaped particles, Oxs must run once for each step change in the particle's physical size.

In the case of cuboidal particle geometries, there are three alterable dimensions: width, height, and depth. So for example, with a edge length range of 100nm to 200nm and a parameter step change of 5nm then to do a full parameter study Oxs must be run eight thousand times. If a single simulation takes at best an hour to compute in the example we have just given, sequentially running eight thousand simulations would take almost a year to complete. A dedicated compute cluster of reasonable size, for example 100, working exclusively and uninterrupted would take just over 3 days to complete the task. However, cuboids are the simplest of the three-dimensional geometries; more complex geometries, such as particles with curved surfaces, would require greater numbers of simulation cells to accurately model and would take significantly longer to study.

The only reasonable way to perform these forms of micromagnetic parameter studies or optimisations is to employ the computational power of massive clusters of computers.

Unfortunately, clusters of these sizes do not exist in large enough quantities. Consequently, micromagnetic research into particle shapes has been held back by lack of compute resources.

3.2.7 Selection and Usage of Code libraries in OOMMF and Magpar

The greater maturity of OOMMF means there is more extension code contributed by the computational physics community than for Magpar, and as such OOMMF supports a larger number of simulation points, for example thermal effect modelling, and its comparative ease-of-use means it enjoys a wider following.

Specific extensions to individual packages often require specific configuration of the package and in some cases a complete rebuild. In addition, they often have specific dependencies on other libraries requiring complex build processes to ensure correct functionality. Semi-automatic tools to aid in the update (to fix bugs or add functionality) and addition of new code libraries would provide a solution to complex configuration or rebuilds. However, particular code libraries offer specialised functionality that has its own peculiar mode of operation and behaviour. Knowledge of the abilities of package comes from experience and study. Knowing what code libraries offer might be missed resulting in replication of already well-optimised code or misuse of libraries resulting in poor performance or incorrect results.

Parameterisable tool types, such as optimisers [97], require the experience and knowledge of the user to facilitate the production of results. Many class problem types have common parameters and options that speed up their solution process or produce better results. In addition, selection of solution methodologies also requires the experience and knowledge of the user in order to gain the best results. However, the majority of scientists carry out their work in isolation without the usage of shared knowledge repositories or knowledge gathering tools. Therefore, anyone inexperienced with particular problem types may only discover which solution methodologies, parameters or options are optimal through trial and error. Tools exist, such as OPTIONS[7] [98], to aid in the selection of parameters and options. Unfortunately, these tools are often proprietary, lack common standard support and modes of operation. Accordingly, these require complex methods, such as scripting, to enable their interaction with other tools.

3.2.8 Specific Challenges of driving the Workflow

The major challenge with working with bespoke simulation tools such as OOMMF and Magpar is the complexity involved in the setup process. Data and information necessary

[7]Software developed by Prof. Andy Keane, University of Southampton - http://www.soton.ac.uk/ ajk/options/welcome.html

to solve the problem must be collated and represented to the tools in a form understandable by OOMMF or Magpar. In a typical scenario of computational micromagnetics, a research scientist will do the following things:

1. Problem Initialisation,
2. Simulation Execution, then
3. Results Post-processing

All three stages are time consuming and complex exercises for the scientist. This is not including time taken to compute the simulation. To compound the issue, for parameter studies, in the case of OOMMF, the three stages must be repeated for each step change in a particle's geometry.

Problem Initialisation can broken down further into the following stages:

1. Decide the number, size, arrangement and material of the particles,
2. Select the overall solution methodology (FD or FD/BE),
3. Create or decide on the 3D mesh representation of the particles,
4. Choose the simulation tool (only OOMMF or Magpar),
5. Code or select appropriate solution methodologies for the magnetic system, then
6. Write the simulator specific problem statement file providing links to the location or explicitly specifying the required particle data, meshes and solution methodologies.

Difficult choices must be made on a number of points including which modules, solution methodologies, meshes, and tools should be employed. An error in choice often ends in poor results and no guides exist to lead the scientist through the problem initialisation. A portal or front end application could aid scientist through automation of much of the initialisation process. Many of the tedious error prone setup tasks could be automated such as the creation of the MIF file and selection of meshing approaches.

In addition, semi-automatic and automatic selection of solution methodologies could be provided through a portal with knowledge capabilities. A repository of gathered meta-data built from previous problem initialisations could be utilised by a portal to make suggestions on which solution methodology and optimisation parameters produced high quality results for types of micromagnetic systems. This would require gathering information from the post-processing stage and scientist whom must make the judgement on the quality of results produced. The knowledge repository must be feed information from all three stages in the form of meta-data such that the portal can easily cross-reference problem initialisation, execution configuration, and post-processing results in order to make possible intelligent suggestions to the scientist.

Simulation execution has a shorter but equally complex process as follows:

1. Select or compile the simulator binary for the target execution platform,
2. Write the script to setup the environment for the simulator on the target execution platform,
3. Specify the resource requirements to run the simulation job, then
4. Schedule the simulation job and monitor its progress

The choice of execution platform and resource requirements may slow or prevent running of OOMMF or Magpar. In the case of Magpar, which requires compilation of its core simulator and its dependent modules for its target execution environment, if an execution platform is chosen which does not have support for any required software dependency or solution methodology module, Magpar will fail. For both OOMMF and Magpar, if the scientist request too low resource requirements then its is possible that the simulation will run out of memory or hard-disk space before completion and fail. This is a serious issue because simulation jobs may run for days or weeks before failing and the scientist would lose valuable time or their computer resource allocation. Again, a portal or similar front-end application could help select suitable execution platforms that match the simulation resource requirements, automate the process of creating environment setup scripts, and schedule the job.

Post-processing of results is a potentially more complex and resource hungry task than the simulation itself. Effective analysis of simulation results requires their collation, translation, and rendering in order to produce graphs and visualisation suitable for human study. The raw data produced by OOMMF and Magpar is only computer readable because of its format, encoding and often great size. ODT files of OOMMF are easily convertible to formats suitable for graphing because of their tabular nature and small data size. However, OMF vector based files require not only translation but also significant amounts of rendering time using tools such as Povray, to process the Gigabytes of vector data to produce three dimension representations of parameter studies. Rendering tools such as Povray, have similar complex problem initialisation and execution requirements as the simulation tool; requiring the problem specification and data to be built and specified in an application-centric file format. In addition, compute clusters are often necessary to reduce the processing time for 3D visualisations. Consequently, in the field of computational micromagnetics 3D visualisation is rare. We now analyse the specific challenges of the computational micromagnetic problem solving process.

3.2.8.1 Understanding, Operation and Extension of OOMMF and Magpar

OOMMF's and Magpar's open object-oriented (OO) extension and application framework is ideal for maintenance, addition of improvements and new code libraries. However, OO development paradigm and its three-tired API structure tightly-couples the functionality to the resource requirements. Indeed, its whole architecture as a standalone executable tool makes impossible the separation of compute from functionality, which is necessary in order to facilitate efficient resource usage an sharing, and platform interoperability.

OOMMF's and Magpar's operation and behaviour are domain specific which is not unreasonable as they are designed as tools that allow FD and FE/BE micromagnetic simulation studies. However, their operation and behaviour of the simulation tools differs significantly at runtime. Whilst the Oxs simulation tool of OOMMF is standalone, Magpar depends on external software packages for its execution. Oxs exhibits features of SO by being independent of outside dependences allowing it to be execute on a greater number of computational resources types. However, Magpar's tight-coupling to external dependencies means its execution is restricted to only the type of computational resources supported by all the required software dependencies. In chapter 7 we employ OOMMF instead of Magpar because not enough computational resources existed with the required platform or operating system for Magpar for the timely completion of parametric studies.

3.2.8.2 Meshing of Particles in OOMMF and Magpar

The relatively simpler FD approach employed in OOMMF, as compared to FE/BE approach employed in Magpar, uniquely enables OOMMF to automate the particle meshing process for the scientist. This removes one difficult step enabling the scientist to concentrate on other parts of the problem process solving. Nonetheless, scientists often use the powerful yet more complex tool Magpar for particle geometries with curved surfaces. This is because the FD approach relies on fixed sized cells, which is less accurate then than the variable cell size of the FE/BE approach in describing particle geometries with curved surfaces such as cones and spheres.

However, because of the potential complexity variable cell size of cells of the FE/BE approach, Magpar cannot automate the process of meshing. The mesh geometry of a micromagnetic system must be passed to Magpar. Meshes are usually created graphically with a computer-aided design program capable of creating FE/BE meshes however; its level of quality affects the ability to produce correct simulation results. A low quality mesh of, for example, too few, or badly positioned or arranged cells will adversely affect the quality of the results from the numerical model. Consequently, scientists often choose to use OOMMF over Magpar because of the formers automatic mesh generation.

3.2.8.3 Data Representations and File Formats in OOMMF and Magpar

Generally, tools will have no understanding of the interface, operation or behaviour of any of the other tools. In addition, many of these tools lack support for common data formats and transferral standards. Typically, the scientist would create custom scripts or macros to coordinate execution of tools and, if necessary, handle the translation and transferral of data between tools.

The problem initialisation for OOMMF requires knowledge of a number languages including Tcl/Tk, and requires detailed knowledge of the operation and interface of OOMMF in order for its most effective usage. The process of initialisation can be eased through automation of the MIF's geometry and description and material through scripting languages such as Python or sophisticated operating system shells. However, again it requires detailed knowledge of the inner working of OOMMF and its MIF file format, often requiring the user on a need basis to add or modify the scripts to handle new geometries or platter materials. Whilst effective for simple cases, no such tools exist for the general case or Magpar, which requires the further step of mesh generation.

3.2.8.4 Integration of OOMMF and Magpar with Compute Clusters

OOMMF is available in either a pre-compiled binary format or as source code that can be optimised for specific architectures. No out-of-the-box binary executable exists for Magpar, since its dependent libraries are very heavily optimised for specific computer architecture and operating system platform.

The overhead in actually getting Magpar to execute is significantly greater than for OOMMF as high level optimisations within the numerous dependent libraries of Magpar make production of a generic binary impractical. Consequently, a long difficult build process must be repeated for each architecture within a heterogeneous cluster; even the subtle differences between Intel Pentium III and IV processors need a different build of Magpar.

Scientists unfamiliar with building large software packages would benefit from a catalogue of pre-compiled binaries however; they would still be responsible for selecting the correct version of the package for the target computer. With the multitude of combinations of architectures and platforms possible on a heterogeneous cluster the task of software deployment and consequent effective resource harnessing becomes unrealistic as scale of the task increases.

3.2.9 Summary of Computational Micromagnetics

To summarise, scientists aspire to produce results quickly, correctly and of a high quality. However, the problem-solving environment often gets in the way of this. In the computational micromagnetic problem-solving process, the scientists have to contend with coupling of heterogeneous hardware and platform specific software, which each have their own environment requirements, file formats, data models, operation, and APIs. Moreover, the scientist is often forced to manually select or create the solution methodologies, mesh models, and material data. Much of the processes described necessitate that the researcher combine the necessary hardware and software resource in an ad hoc manner to achieve a result.

3.3 Geodise: A Service-Oriented Engineering Optimisation and Design Search PSE

Geodise [99, 100] is a current state-of-the-art Grid-enabled PSE that provides a generic integration framework for computation and data intensive MDO tasks which maintains the autonomy of each individual domain expert. Its service-oriented style makes it an ideal research topic on the application and behaviour of Web Services in Grid computing from the perspective of computation, data management, software applications, intelligent knowledge repository, and service integration.

Engineering optimisation and design search is a process whereby existing engineering modelling and analysis capabilities are exploited to yield improved designs. In the next 2-5 years intelligent search tools will become a vital component of all engineering design systems. Such facilities will steer the user through the process of setting up, executing and post-processing design search and optimisation activities in a variety of disciplines. A major driving force in these developments is the need to allow distributed design teams from multiple enterprises to share design and analysis capabilities over the Internet. Such capabilities may include various specific engineering design and optimisation applications, reference data or job processing power, with each implemented using different technologies, and deployed in a different computational environment. In a typical scenario of, for example, wing design, an engineer may couple together Computer Aided Design (CAD) tools, analysis codes for Computational Fluid Dynamics (CFD), or Finite Element Analysis (FEA), and tools for optimisation. The resulting sequence of computations may be performed on local or national machines, or on pay-per-use Internet cluster resources. And the resulting data sets may be stored temporarily or permanently on a remote storage service.

3.3.1 Vision and Design of Geodise

Geodise holds a service-oriented vision of the Grid, in which all resources are regarded as services. The following services involved in engineering optimisation and design search operations are identified by Geodise:

3.3.1.1 The Computation Service

The computation service is concerned with sharing and management of computational resources that are either excessive or not generally available. Complicated business/application processing can therefore be outsourced through this service to resource providers. This allows more reasonable allocations and more efficient usage of resources. The most important feature that the computation service needs to provide is a generic representation that is independent of the underlying resource providing mechanisms, so that it can be deployed over various kinds of computation systems, and be consumed by different clients. An execution environment in the style of a 'sandbox' is also valuable as it creates a generic model for computation jobs and enforces security for the underlying systems.

3.3.1.2 The Data Service

Data in Grid computing can generally be divided into two categories: data resources located on the often distributed storage devices, and meta-data that is used to describe the stored data. In Geodise, data service components provided to higher level applications manage data resources on the Grid through XML based meta-data, which provides information not only on the location of the data, but also its characteristics [101]. This can greatly improve the efficiency of data management and data search. Data storage supported by the data services can be in the form of database, file store, or other storage mechanisms depending on purpose and performance.

3.3.1.3 The Application Services

Application services in Geodise provide access to design and analysis functionalities required for engineering design optimisation, including design of experiment, geometry generation, and response surface modelling. The use of service-orientation makes it possible to integrate best available technologies from tools such as ProEngineer [8], OPTIONS [102], and various CFD analysis codes. Whilst some tools may run adequately on the local server, the application services often need to make use of the computation service and the data service to ensure quality of service (QoS) for compute and data intensive operations.

[8]Home page of ProEngineer http://www.proengineer.com/.

3.3.1.4 The Optimisation Service

Optimisation technologies such as those provided by OPTIONS are also provided as services in GEODISE. At the start of the optimisation process, the engineer needs to provide initialisation information- in particular the analysis codes to be coupled together, the permitted methods by which a design may be modified, and an objective function by which each design may be evaluated. The Optimisation Service will then be called to coordinate the maximisation (or minimisation) of the objective function to improve the design. Many optimisation operations can be divided into several stages, which require single or multiple evaluations of the objective functions. Our Optimisation Service therefore provides strong support for state management so that after successful completion of each transactional stage, the optimisation's related state and current data is automatically maintained. When necessary, the data service can also be used to store and archive the optimisation state information.

3.3.1.5 The Knowledge Service

The knowledge service in GEODISE provides intelligence that guides the user through the process of combining the different components required to perform sophisticated operations, such as efficient engineering optimisation and design search. The service also has built-in knowledge repositories based on the operation record data of each service. For example, by querying the records of past optimisation operations, new heuristics about optimisation can be deduced and used to improve future design processes. This may also be achieved using formal knowledge elicitation methods and held in rule bases. User activities may also create domain specific knowledge that they may wish to archive and reuse in various ways to enhance their capabilities in the future. Various knowledge management and re-use tools are therefore called for to underpin this process.

The design of the GEODISE system is based on the services described above is shown in figure 3.5. An important component apart from the resource services is the portal, which is the point of access to the entire system. The portal provides a problem solving environment (PSE) to the engineers to locate and combine the resources they require with Grid-based, secured and seamless access to all the services. Currently, Matlab is selected as the environment in which the GEODISE PSE is built. Matlab package provides a language for numerical computation, built-in math and graphics functions and numerous specialised toolboxes for advanced mathematics, signal processing and control design. It is widely used in academia and industry for algorithm prototyping, and for data visualisation and analysis. From version 6.5 Matlab also provided a number of Just-In-Time (JIT) acceleration technologies to improve the performance of native Matlab code, while since the latest version 7 the performance of native code can even match that of the compiled code which involves translating Matlab code to C and recompiling natively.

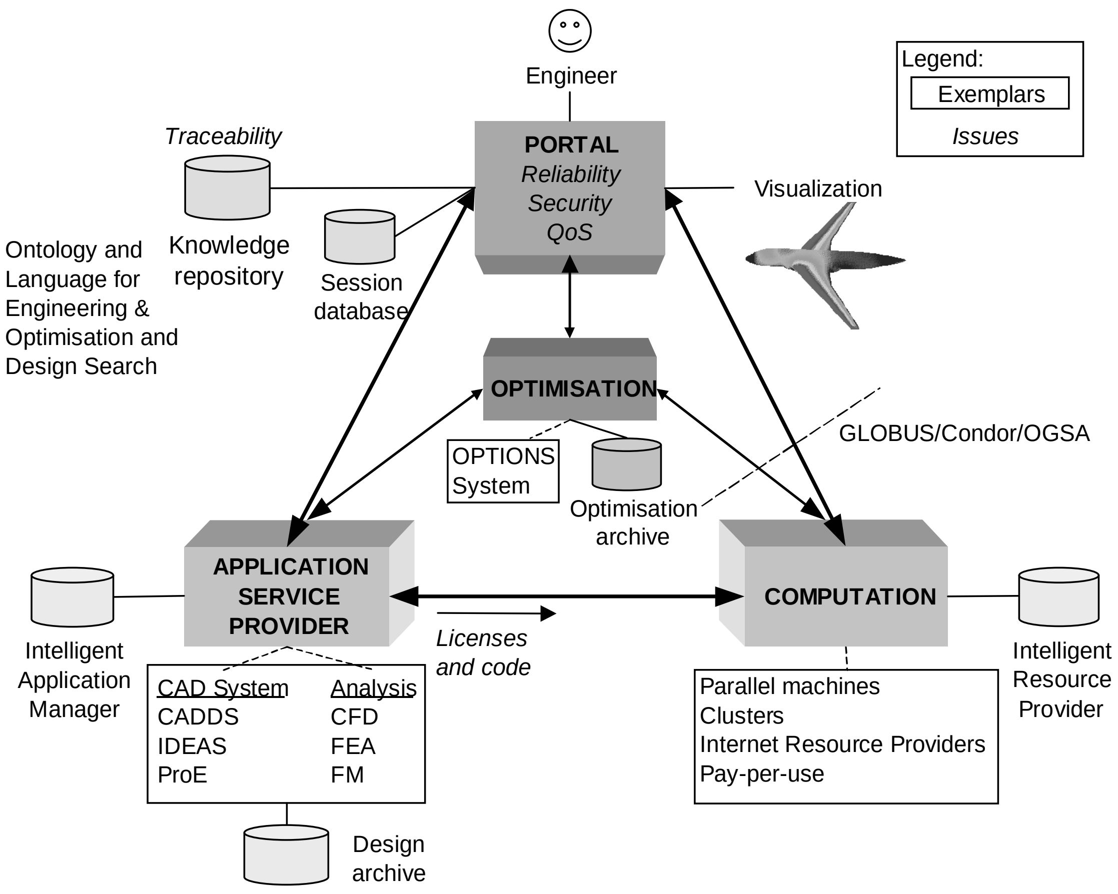


FIGURE 3.5: Grid Enabled Optimisation and Design Search for Engineering

3.3.2 Implementation of Geodise

To make the resources involved in engineering design optimisation appear on the Grid, a service layer is introduced which contains middleware components that handle the service communications and interact with the actual resource systems. Middleware technologies most commonly used in GEODISE are based either on the Microsoft .NET Framework [9], or the J2EE [103] (Java 2 Enterprise Edition) platform, including Web Service hosting systems such as the Microsoft Internet Information Services [10] (IIS), and binary compatibility technologies such as JNI [16] (Java Native Interface). As described before, the portal to the GEODISE system is embedded in Matlab. To make services in GEODISE accessible to the Matlab scripting environment, clients for the services are all built using Java so that they can run natively in Matlab's own JVM (Java Virtual Machine) and can therefore be used in the same way as normal Matlab commands. Based on that, toolkits are made containing customised commands that perform transactional operations on the services [104, 100]. The service clients can also be used in other scripting environments that support Java, for example, Jython.

[9]Home page of Microsoft .Net Framework http://msdn.microsoft.com/netframework/.
[10]Home page of Microsoft IIS http://www.microsoft.com/WindowsServer2003/iis/default.mspx

3.4 Summary

This chapter has studied the problem solving process, identifying its current challenges, and presented current solutions. Grid-enabled PSEs offer scientists and engineers with a means to compose complex workflows and loosely couple together resources and tools that allow them to focus on the problem at hand rather than intricacies of the underlying technologies. However lack of standards and poor interoperability and integration of tools remain as a barrier to creating environments suitable for problem solving.

In order to prove the feasibility of service-oriented Grid computing and to better understand how Web Services technologies should be applied in practical Grid problems, the research work introduced by this thesis has been based on the GEODISE project, which aims to apply and develop Grid technologies to solve concrete engineering design optimisation problems. Chapters 5 and 6 present in detail two of the GEODISE services representative of the work on service-oriented Grid computing, namely, the computation service and the Optimisation Service. These services where developed outside the main GEODISE project development as proof-of-concept however the Computation Web Service was later integrated into the computational toolkit to provide remote access to Condor through Matlab.

Chapter 4

Solution Methodology

Here we discuss various solution methodologies for solving the challenges faced by Grid-enabled PSEs. We attempt to with the methodologies and rules of thumb described here to show how services can and should be built and aggregated in order to better provide a loosely-coupled environment for problem-solving.

4.1 General Methodologies

4.1.1 Comprehensive Definition for Grids

Because of the increased popularity of Grid-related projects and ideas, the need for a comprehensive definition has arisen. Ian Foster, in his paper "What is the Grid? A Three Point Checklist" [105], mentions the existence of many "types" of Grid - computational Grids, biological Grids, knowledge Grids, campus Grids and many more. He makes three valid points - coordination of resources through decentralisation, open-standards general purpose protocols and interfaces to address fundamental issues, and quality of service - and then points out that Web Services with Open Grid Service Architecture (OGSA) "InterGrid" protocols [106] should be the interoperability standard. However, we wish to expand on this and not just limit the Grid to Web Service interoperability - for reasons of performance or legacy a standard mechanism for interoperability requires implementation in addition to web services. This could be used to provide, for example, translation, firewall tunnelling and common description of services through OGSA wrapping, such as the Condor Web Service. This addresses the important issues of transparency.

4.1.2 Transparency, Discovery and Integration

Transparency provides the ability to work with the Grid; it improves usability. The second point is adaptability - because of user transparency; changes to the system will not affect existing users. Finally, transparency facilitates scalability - there are no issues with expanding the Grid as are usually present in opaque systems; the reliance on static resources is lifted.

Another problem is that of the discovery of resources. The existing model for resource discovery is the client-server model, which is often neither scalable nor inherently reliable. We wish to abandon this in favour of a more flexible system, unlike peer-to-peer systems, making the latter the more preferable candidate as the underlying architectural model for the Grid. There are two main P2P approaches to this: layered networks, involving organisation by hardware (nodes are organised by location) or by search layer (nodes are organised by metadata - services). The second approach could utilise fuzzy heuristic and returns a true closest match via WSDL-based descriptions. Here we define what we understand to be the Grid: it is transient and attempts to supply resources transparently to users (in the broadest sense). Additionally, there should be transparent and seamless access, through tools for searching, understanding of description and communication. The Grid should be intrinsically reliable; i.e., no central point of failure.

4.1.3 Common Operational and Behaviour

Sharing and collaboration of multiple different resources in an effective manner demands a simple, common, flexible, extensive and application-unspecific architecture. Web Services, see section 2.6.1, provide technologies that ease achievement of architectures with these characteristics. However, these technologies only provide the building blocks for Grid systems and do not address structural and organisational challenges nor the behaviour or operation of resources themselves. Although, common ontologies would benefit the later extending interoperability to the service level, the application of Web Service technologies demands careful examination.

4.2 Simple Grid Architectures

OGSA attempted to create a layered architecture that described the role, behaviour and interaction of components types in a Grid systems. Each layer of this Grid architecture provided functionality for the layers above. However, this model of the Grid never got off the ground as it did not treat all components of a Grid system as resources. Indeed, the layered model itself mirrors the successful concept of the TCP/IP stack however, TCP/IP is a message based system whilst OGSA assumed that Grid components would communicate through RPC style interfaces (e.g. Web Methods). The only successful systems and technologies to reach the scale of the Internet have all been message-based or had REST architectural-styles.

We believe it is not feasible or flexible to develop Grid-system in a OGSA manner with layer types only communicating across and directly below the immediate level. This model creates systems that are overly complex to integrate with other systems as both its architecture and technologies have to be understood. Grid architectures should be simple such that the particular organisation and structure of one system does not limit the organisation and structure of other collaborating systems. Components in a Grid system should be modelled on whether they are resource consumers or producers and the structure and organisation of systems should be done at the message-level not at the service or resource level.

4.2.1 Modelling of Resources

We define a resource as a source of wealth that contains a supply or reserve that can be drawn upon such as, for example, computational clusters, data repositories, or scientific instrumentation. In a top down view of the Grid resources sit at the base with services and applications sitting on top. They provide the raw untapped power for the Grid offering basic functionality pertaining only to the processing, storage, release and production of data. For example, Computational resources both process data and/produce data while, data repositories store and release data.

Resources typically have properties that identify information about the capability and type of the resources, such as the processing power and computing architecture of a computational resource or the amount of storage available in a data repository. In addition, resources contain knowledge in the form of application logic that drives its operation and behaviour.

4.2.2 Modelling of Services as Resources

We model services as a particular kind of resource with a standard, well-understood communication model. Services should be thought of as distributed software components that may be combined in many different ways to create sophisticated systems. Access to raw resources should be achieved though service gateways that offer transparent access to the underlying resource. From the view point of the resource consumer the underlying raw resource does not exist, only the gateway that for all intents and purposes is treated as resource provider.

4.2.2.1 Virtualisation of Resources

Resources of the same type should be virtualised (abstracted) to provide consistent behaviour and operation. This particular applies to resources where interaction occurs at the machine level. Unlike for instance information resources, such as Web Servers, that offer multimedia data in a human interactive and readable form. These have intrinsic interoperability with their target consumer, the human user, who understands a picture no matter what file format it was stored in.

Virtualisation overcomes application level misunderstanding and inoperability. Whilst open standard technologies, such as Web Services, provide wire and service level interoperability, consumption a resource need understanding of its behaviour and operation. For example, two optimisation service might exist however their solving manner differs: the first operates a forward communication model whilst the second employ a reverse communication model. Application of these services requires two different modes of interaction. Migration between various optimisation system becomes difficult as each implementations operation and behaviour must be understood.

Virtualisations requires constructing of common, generic interfaces and interaction models for the target resource, XML Schema and WSDL being a suitable technology. Further more this may be further refined with the study of common sematic with language ontologies.

4.2.3 Modelling systems

Systems contain components that consume and produce data. Data flows from producers to consumers. Components of systems may both consume and produce data. The flow of data starts from a particular component, passes through other components and ends up at an ultimate receiver. These components must be identified and classified and the dependencies discovered. Once the desired flow of data is understood and the consumer and producers have been identified the system can then be modelled as a Grid system with components becoming services, resources or applications.

4.3 Service Interaction

4.3.1 Client-driven Architectures

Client-driven architectures offer a method to avoid artificial interdependence between services and clients. This concept requires that clients drive the operation of services and precludes asynchronous calls from services that require a synchronous response. Services should only return information directly back to the client as a synchronous response to a method call request or as a one-way asynchronous method call.

Client-driven architectures allow for flexible, simple and light-weight clients and reduces the complexity of services implementation. For instance, a user may effectively access and operate a service through a Web browser using web forms [107] or a small Java-based Applets[1] [108] if they wish to receive asynchronous calls and have greater functionality. Stand-alone programs or other services, acting as clients, require only a client-side stub[2] of the service they wish to invoke.

4.3.2 One-way Asynchronous Event Notification

Asynchronous notification is a powerful mechanism that allows a service to inform the client of an event without need for polling to determine when a circumstance occurs. However, it requires a listening service on the client-side and support for it should not be a prerequisite to the operation of the service. This precludes service behaviour that needs a synchronous client response to an asynchronous call otherwise services would be dependent on their clients.

Another advantage of an asynchronous notification mechanism is that it reduces the load on the service and client. Along with reducing server CPU load and network bandwidth caused by polling, client sessions may reside on disk longer rather than in memory because they are called less frequently. Thus it allows more efficient usage of server resources and improves scalability.

4.3.3 Coupling of Legacy Systems

SOA-based technologies, see 2.5.4, provide the glue that enables the coupling of otherwise incompatible legacy systems with other legacy systems or resources; extending their usefulness and interoperability. We propose a process that we call wrapping, that involves constructing a service that lies between the system and its users, see figure 4.1.

This involves construction of Application Programmer Interfaces (APIs) and a WSDL document that represent the functionality and data formats of the underlying system. These would then be implemented with Web Service technologies to replace the standard methods of interaction. The Web Service would provide transparency from the legacy system's particular mode of operation and behaviour.

A simple approach is to construct a service that simple acts as a interpreter by converting protocols and data formats to the internal systems representations. However,

[1]Sun Microsystem Technology - http://java.sun.com/applets/

[2]Generated by tools such as WSDL.exe [109]

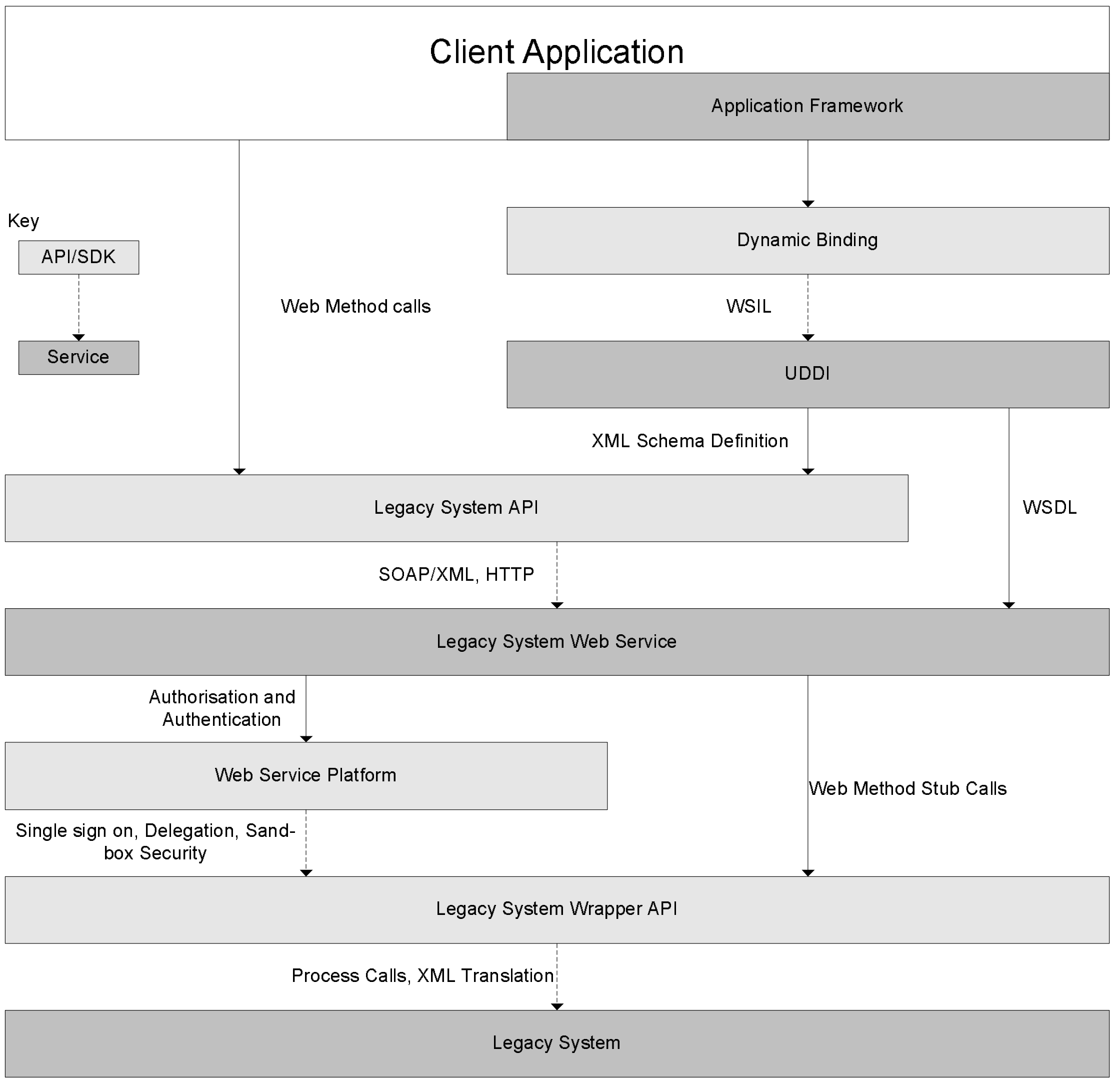


FIGURE 4.1: Example of legacy system wrapping with Web Service technologies.

this is only suitable for legacy system with no external dependencies. Service wrapping offers the opportunity to enhance its functionality, removes implementation specific artificial dependencies and provides a layer of abstraction (e.g. using an ontology) and encapsulation that eases interoperability. In addition, implementation abstraction enables preservation of the service if the legacy service were to be retired. It would be able to transparently alter its underlying logic to accommodate another implementation without affecting its operation or behaviour.

4.3.4 Dispersed Infrastructure

We propose that distribution of independent and vital system components will aid the usefulness, scalability and flexibility of Grid-based environments. This differs from monolithic approaches that attempt to provide all resources and functionality from a single provider or service. Alternatively, environments should utilise dispersed infrastructures with its key components running as independent services that can then be more easily replicated or exchanged.

For instance, an infrastructure for HPC may contain a resource management service that performs computation. However, this ties both users and the computational resource to a single specific service. Loose-coupling and dispersion of these as independent services would aid flexibility by allowing the computation element to be employed by other forms of services or applications. In addition, different service implementations may be more easily exchanged, for example, where a user may wish to employ Maui [110] instead of Condor.

4.3.5 Context within and across Web Services

State is typically employed within a service for maintaining consistency across Web methods calls, providing insulated user sessions and storing global environmental information. However, not all types of Web Service roles call for state.

Sharing operations between resources demands proper handling of the context in which the operations are called. This demands information about the state of the system to be stored across sharing operations over potentially long periods of time. For example, users should not be able to execute a job before they have uploaded the files necessary for its execution. A potential loss of this state information could have disastrous results possible resulting in a malfunction system, damage to files or even a breach in security. In addition, the service may receive multiple concurrent operation request for internal resources and multiple users, each with their own state. Consequently, services must employ state management scheme that are efficient, robust, and tolerant of service failures.

The simplest type of Web Service applications consists of collections of isolated Web methods implementations that require no state or lifetime management components. These services roles do not require shared data. This has the advantage of intrinsically providing isolated and thread-safe Web methods; enabling dependable multiple client access to a single service without requiring any additional code or infrastructure.

If we want to add state to this Web Service, things start to become more complicated. In order to maintain state across methods calls the server must be capable of remembering the internal state of the service in memory or on disk and be able to associate that state with the correct service being called.

A Web Service may wish to offer session state for users, such that each user accessing it has their own service session with associated data and state, that is isolated from modification by other users by the creation of service instances and service handles.

This requires mechanisms for creation, destruction, and lifetime management however none of the above features is defined in the Web Service Specification requiring the developer to either create their own mechanisms or work with those provided by the service's runtime environment, that could vary greatly across platforms.

4.3.6 Common Operation and Behaviour

Common functionality, such as lifetime management, ideally should be described, deployed, and behave in the same way across all Web Services wishing to offer this. To draw comparison with object oriented paradigms, an inheritance mechanism is required for WSDL description. If we now look at Grid Services as defined by the OGSI Specification, they have created a mechanism for extending and describing service logic using the concept of Port Types. Grid service functionality may be extended by the inclusion of additional Port Types into a WSDL descriptor for each service and the linking of Port Types in the service logic through either multiple inheritance or attributes. Common port types are defined by the OGSI specification, such as lifetime management, and notification mechanisms. Grid services with defined behaviour and mechanism or interaction and operation are extensions of Web Services.

Grid Services employ the paradigm of persistent and transient services, Traditional stateless and simply state-full Web Services are a form of persistent Grid Services because these Web Services allow multiple users simultaneous access to it through a stats address or URL and exist for as long as the server is up and running. However, Web Services have no equivalent model for transient services. State-full session based Web Services are similar to transient services in that a new state instance is required for each new user however, Grid Services require factories to create transient service to which Web Services have no similar defined mechanism. Web Service developers are free to implement this state instance model as they see fit.

Something completely missing for Web Services is the concept of service data of the Grid Service model. Again Web Service developers can implement such a feature however the mechanism for accessing, modifying, and querying such data is not defined.

4.4 Summary

We have explored concepts that we believe have attempted to cover many of the challenges discussed in the previous chapter 3. In particular, we have covered methods that facilitate the construction of interoperable services and provide means for adding knowledge into the problem solving process. In the following chapters we put these in to practice.

Chapter 5

Service-Oriented Numerical Optimisation

This chapter discusses research into service-oriented based numerical optimisation. The aim of the work was to find service models that eased its integration with other problem solving tools and enable better computational resource allocation. In addition, the work serves as a proof of concept which shows the suitability of offering optimisation facilities in a service-oriented architecture.

This chapter describes a new technique that de-couples the processing of the objective function from the optimisation algorithm using a reverse communication model which allows for the creation of an extremely flexible, robust, loosely-coupled service. To check the validity of the technique and examine its potential, a reverse communication-based service was built which uses Grid technologies to provide hill climbing optimisation. We demonstrate that an Optimisation Service provides not only the possibility of aiding in the selection of appropriate methodologies but also facilitates integration with other services and resources employed in problem solving workflows.

This chapter gives examples and rules for how an Optimisation Service should be built and used. In addition, it will look at the idea of check-pointing and removal of internal state from within services to enhance their reliability and facilitate their transparent integration with other services and resources. Whilst this chapter uses optimisation because of its importance to science and engineering, the ideas expressed in this chapter have benefits to all types of application services.

5.1 Challenges of Optimisation

Typically, optimisation forms part of an engineering or scientific problem solving workflow for research, modelling, iterative adjustment and re-design of 'objects' which can be anything such as, buildings, products, mechanical or electrical components, or manufacturing operations. Optimisation plays a key role in, for example, the process of bettering 'objects' characteristics, reducing their manufacturing costs, and finding solutions to meet requirements.

Numerical optimisation is an essential tool required by scientists and engineers for systems that have extremely difficult or impossible analytical solutions. Typically, optimisation is employed to find a combination of system parameters that best meets the goals of the target environment or desired constraints. A simple example of the usage of optimisation would be for finding the highest attainable speed for a car on a motorway. System optimisations become more tricky with two or more parameters. Unlike the car example, these may have one or more results that meet the target criteria however, finding all or any suitable results quickly and efficiently is not straightforward demanding particular optimisation algorithms for certain system or mathematical models.

Traditional execution of these workflows typically involves the interaction of software tools including professional design tools, such as CAD packages, custom written software to model the problem, optimisation tools, such as OPTIONS, and analysis tools, such as, visualisation and graphing software. Effective usage and aggregation of this software requires high computer science abilities in addition to the computer resource requirements of the software. Therefore, problem-solving workflows involving optimisation would keenly benefit from implementation into PSEs. However, there are many challenges to overcome in order for a PSE to ease the process of optimisation.

Numerical optimisation is often a computationally intensive process. Numerical methodologies are often employed as a means to converge on optimal solutions to scientific and engineering challenges that have no possible or trivial analytical solution. Depending on the size and complexity of the system being optimised, it may take from days to months to compute, typically requiring the processing power of HPC clusters. There exist many general purpose optimisation algorithms that have been developed to derive high quality and fast converging solutions. However, systems with unusually large or complex numerical models often require specific optimisation algorithms to successfully or quickly converge on a high quality optimal solution.

Consequently, careful selection of optimisation algorithms is essential to ensure successful and fast convergence on a solution. Despite its name, optimisation does not necessarily mean finding the optimum solution to a problem. Often this is not possible, and heuristic algorithms must be used instead. Two fundamental goals in computer science are finding algorithms with provably good run times and with provably good or optimal solution quality. A heuristic is an algorithm that gives up one or both of these goals; for example, it usually finds pretty good solutions, but there is no proof the solutions could not get arbitrarily bad; or it usually runs reasonably quickly, but there is no argument that this will always be the case.

The research described here started with the idea that numerical optimisation tools could be offered using service-orientated principles and technologies. The aim of which was to create a service model for optimisation that enables flexibility in the choice of compute resources and provides an architecture to which tools, for example, knowledge facilities could be easily integrated to aid in the selection of optimisation algorithms in Grid PSEs.

5.2 Optimisation Algorithms

An optimisation algorithm is a numerical method or algorithm for finding a value x such that the result of $f(x)$ is as small (minima) or as large (maxima) as possible, for a given objective function f, possibly with some constraints on x. Here, x can be a scalar or vector of continuous or discrete values. If x is continuous, then the study of the algorithm is part of numerical analysis.

Optimisations practical application often involves writing software (codes) that contains the functional representation (i.e. $f(x)$) of the problem. This is then linked to the optimisation algorithm, often supplied in a software library, to create a executable which is typically run on a computer or cluster. The optimisation algorithm during its execution invokes the objective function, often many thousands of times, each time adjusting the input parameters x which define the problem subject to any constraints placed on them. Typically, the optimisation algorithm uses the results of prior objective function instantiations to determine if the result is converging or diverging from the optimal, adjusting the parameters accordingly. In a successful run the optimisation finishes once it finds a combination of input parameters to the objective function which produces optimal results to within a specified tolerance. Often optimisations fail because the problem subject is extremely complex making it difficult to determine appropriate constraints and tolerances that often results in the algorithm not converging or converging on a less than optimum result. In numerical optimisation programming terms the mathematical model is referred to and handled as the objective function because this provides a suitable model for development of optimisation algorithm in isolation from the specifics of the mathematical model.

The aim of optimisation is to find results that best meets a specified criteria. For example, within the field of wing design, engineers may wish to find the optimal air flow over a wing that creates the most lift but, with the minimal drag to reduce fuel consumption. The engineer would select a set of parameters for a particular wing model, such as the wings geometry, mesh cell size, material weight and air resistance. The engineer would then produce a mathematical model (objective function) that calculates the air flow. This takes as input information regarding the placement of the particular mesh cell and the applied parameters. Then to simulate the whole wing system this object function must be passed to a selected optimisation algorithm along with a set of optimisation boundary parameters (constraints) that control the ranges of boundary parameters to try with the optimisation function and the desired tolerances for finding the minimum or maximum optimal result.

The way in which the optimisation algorithm finds the optimal point depends on its type and constraints. The selection of all these variables affects the quality of produced results. Many optimisation algorithms take a non-linear discovery approach whereby it may need to try several different boundary parameters before it can determine if it is headed in the desired direction. This is especially true in multidimensional problems where for instance, in a wing design its geometry and construction material all effect the flow of air.

Optimisation can be a very computationally challenging process as the algorithm may have to instantiate the objective function many thousands of times. This is often sped up by the employment of multiple computers with each machine processing a different objective function in parallel. In addition, objective function themselves can can often be parallelised benefitting both non-linear and linear optimisation.

5.2.1 Nelder-Mead algorithm: Amoeba

The description of the Amoeba algorithm contained in this section is an adaption of the work presented in [111]. Consider the unconstrained optimisation problem of maximising a nonlinear function $f(x)$ for $x \in \Re^n$. A well-known class of methods for solving this problem is direct search, which does not rely on derivative information (either explicitly or implicitly), but employs only function evaluations. One of the most widely used direct search methods for nonlinear unconstrained optimisation problems is the Nelder-Mead downhill simplex algorithm [112]. It is extremely economical in the number of function evaluations per iteration, and is often able to find reasonably good solutions quickly. On the other hand, the theoretical underpinnings of the algorithm, such as its convergence properties, are less than satisfactory. Nonetheless, its geometric naturalness, working simplicity makes it an idea candidate for study. In this chapter, we focus on one implementation of the Nelder-Mead algorithm as described in the popular handbook Numerical Recipes [113], where it is called the amoeba algorithm.

The amoeba algorithm maintains at each iteration a non-degenerate simplex, a geometric figure in n dimensions of non-zero volume that is the convex hull of $n + 1$ vertices, $x_0, x_1, \ldots, x_n$, and their respective function values. In each iteration, new points are computed, along with their function values, to form a new simplex. The algorithm terminates when the function values at the vertices of the simplex satisfy a predetermined condition.

One iteration of the amoeba algorithm consists of the following steps:

1. **Order**: Order and re-label the $n+1$ vertices as $x_0, x_1, \ldots, x_n$, such that $f(x_o) \geq f(x_1) \geq \ldots \geq f(x_n)$. Since we want to maximise, we refer to x_o as the best vertex or point, to x_n as the worst point, and to x_{n-1} as the next-worst point. Let $\overline{x}$ refer to the centroid of the n best points in the vertex (i.e., all vertices except for x_n): $\overline{x} = (\sum_{i=0}^{n-1} x_i)/n$

2. **Reflect**: Compute the reflection point x_r,

$$x_r = \overline{x} + \alpha(\overline{x} - x_n) \tag{5.1}$$

   Evaluate $f(x_r)$. If $f(x_o) \geq f(x_r) > f(x_n)$, accept the reflected point x_r and terminate the iteration.

3. **Expand**: If $f(x_r) > f(x_0)$, then compute the expansion point x_e,

$$x_e = x_r + \beta(x_r - \overline{x}) \tag{5.2}$$

   If $f(x_e) > f(x_r)$ accept x_e and terminate the iteration; otherwise (i.e., if $f(x_r) \geq f(x_e)$) accept x_e and terminate the iteration.

4. **Contract**: If $f(x_r) < f(x_{n-1})$, perform a contraction between $\overline{x}$ and x_n

$$x_e = \overline{x} + \varsigma(\overline{x} - x_n) \tag{5.3}$$

   If $f(x_e) \geq f(x_n)$ accept x_e and terminate the iteration.

5. **Shrink Simplex**: Evaluate f at the n new vertices for $i = 1, \ldots, n$.

$$x_i = x_0 + \eta(x_i - x_0) \tag{5.4}$$

For the four coefficients, the standard values reported in the literature are: $\alpha = 1, \beta = 2, \varsigma = 0.5, \eta = 0.5$.

5.3 Challenges with Current Optimisation Systems and Software

As has been discussed in more detail in 3.1 and 5.1, problem solving, in this case optimisation, often requires the integration of an array heterogeneous resources and tool within a distributed design environment. There is a desire for easy-to-use optimisation systems that enable the creation of different solution strategies that may be tailored to specific design problem. These requirements have prompted research and development of dedicated systems for design optimisation, such as SPINEware [114], iSIGHT [115], and ModelCenter [116], which attempt to provide an integrated environment for engineering design optimisation. These provide packages of software tools which all share the common need for users to provide programmatic access to the modelling and analysis capabilities of the their mathematical models (i.e objective functions).

However, these software tools often differ in implementation and interface technology. For example, there exist different computer aided design (CAD), finite element analysis (FEA) and computational fluid dynamics (CFD) codes. It is therefore necessary to develop wrappers for these packages to be used in integrated environments mentioned above although a variety of technologies and methods have been developed to help overcome incompatibilities between the software tools. The most commonly used method is to communicate via data files, which relies on shared data types and formats, or specially developed parsers to interpret the input/output files. However, there now exists Standard for the Exchange of Product Model Data (STEP) [9] which has been developed to facilitate product data exchanges. Another approach to integration is based on common object interface technologies, such as CORBA, in which function calls to the tools are carried out as standard remote procedure calls (RPC). Although the use of exchange data files can be seen as a generic approach, the lack of standard formats in native data description and semantic descriptions of the file content means that extra layers of processing are required almost every time a new component is introduced.

The idea of presenting numerical optimisation technologies as Grid services arose from our efforts to adopt service-oriented Grid technologies for engineering design optimisation, see section 3.3. It offers a generic and extensible framework to address the integration issues by decoupling the optimisation tools from the other software components. The optimisation codes, regardless of what programming language they are written in or what the platform they run on, are encapsulated into standard Grid services that are universally accessible. The tightly-coupled programmatic links between the optimisation modules and the modelling codes that used to be required for integration are replaced with loosely-coupled, standards based message level interactions. It therefore becomes easier to adopt and exploit in one particular engineering design system a number of different optimisation technologies, or to apply one optimisation method to a variety of design problems.

A number of technologies have been applied to enable distributed numerical optimisation, foremost amongst these is the NEOS project [117]. However, it requires that the design problems be formulated in the AMPL languages [118] for submission to the server for execution. Consequently, it is not suitable in the case where objective function contain proprietary or commercial sensitive code.

Other technologies such as iSight, ModelCenter and Nimrod/G [119] adopt a different approach where by optimisation is often an inherent part of an integrated environment and the modelling and analysis tools are often integrated using CORBA, RMI or other RPC technologies. In these systems the optimisation logic is tightly-coupled to job submission and scheduling facilities. For instance, the evaluation of the objective function evaluations is controlled by the optimisers and submitted to distributed computing resources through job submission systems such as Netsolve [120], Globus [70] and GridRPC [121]. The disadvantage of this approach is that it is difficult to for users to employ or add new optimisation algorithms from outside the integrated system nor can they choose their own job submission system.

Users are required to wrap their objective functions in a prescribed format and language for the optimisation system to submit to computing resources. This type of operation is can be termed "forward communication" because the optimisers decides when to run users' problem codes. It lacks the flexibility that allows accessing optimisation algorithms from outside the integrated environments, which makes it infeasible to share optimisation methods among multiple heterogeneous, distributed design environments, across multiple administrative domains. Moreover, the proprietary interfaces used by these systems also mean that users will have to develop individual interface for each package.

Our vision in this aspect is to embrace recent developments on Web Services and Grid technologies to deliver highly scalable and flexible optimisation services to heterogeneous environments using a generic interface, in loosely-coupled manner. This approach allows various programming languages, PSEs, and middleware technologies to be applied without delving into implementation details of the Optimisation Service. Figure 35 illustrates the architecture of our proposed system. Users communicate with the Optimisation Service via SOAP messages using client tools in a design environment of their own choice such as Matlab [104].

5.4 Aims and Methodologies of offering Optimisation as a Service

As has been discussed in section 5.1 and 3, challenges of optimisation for the user include: appropriate selection of optimisation methodology and constraints on the model and boundary conditions, and integration with other tools and resources. This chapter focuses primarily on the latter however, the first aim in creating the Optimisation Service was create a loosely coupled component of a PSE that eases its integration with a range of heterogenous tools, including knowledge services. Traditional optimisation practises, as discussed in 3, do not have the openness nor flexibility to easily allow the integration of heterogenous tools and resources. There are four identifiable stages to the optimisation process:

1. Selection of optimisation algorithm,

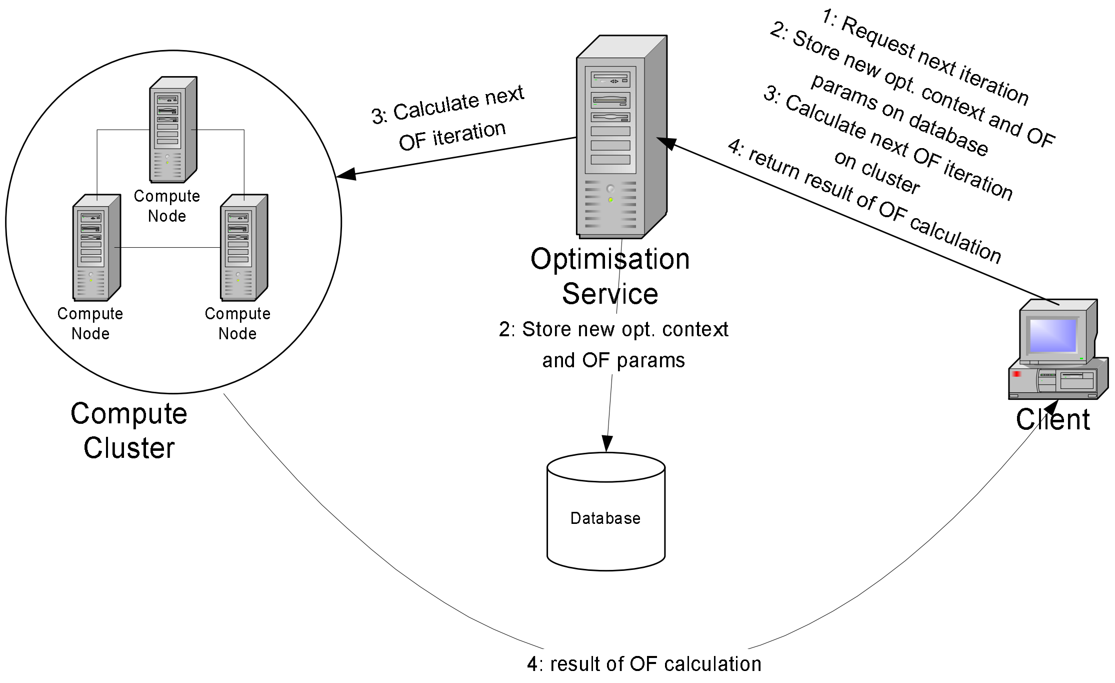

Compute Node
Compute Node
Compute Node
Compute Cluster
3: Calculate next OF iteration
Optimisation Service
2: Store new opt. context and OF params
Database
1: Request next iteration
2: Store new opt. context and OF params on database
3: Calculate next OF iteration on cluster
4: return result of OF calculation
Client
4: result of OF calculation

FIGURE 5.1: Example integration of Optimisation service with a compute cluster and persistence database using a document-oriented message architecture.

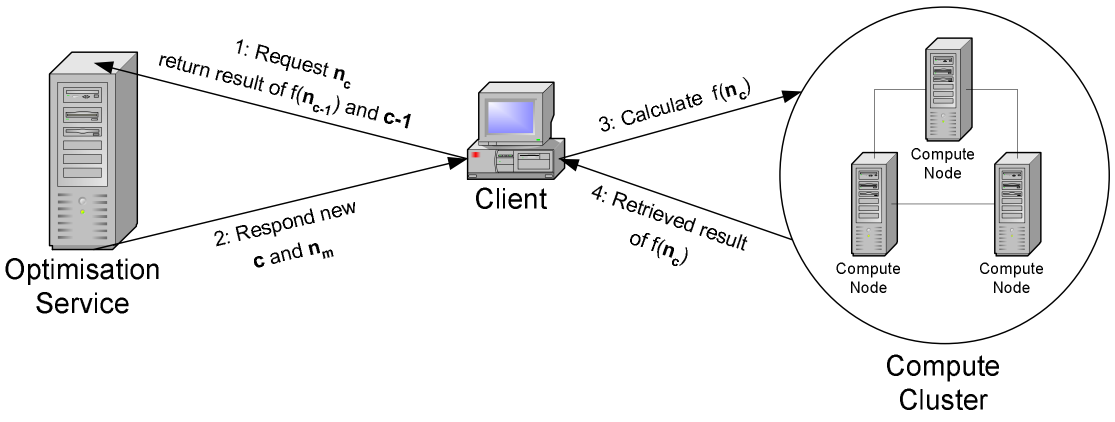

1: Request n_c return result of $f(n_{c-1})$ and $c-1$
2: Respond new c and n_m
Optimisation Service
Client
3: Calculate $f(n_c)$
4: Retrieved result of $f(n_c)$
Compute Node
Compute Node
Compute Node
Compute Cluster

FIGURE 5.2: Example integration of Optimisation service with a compute cluster with the client acting as the controller of the workflow.

2. Linking of objective function to optimisation algorithm,
3. Configuration of input parameters such as boundary conditions, then
4. Execution of optimisation algorithm

The Optimisation Service takes a fundamentally different approach to traditional optimisation by loosely-coupling the optimisation algorithm and objective function. Service-based optimisation differs in that it reverses the roles of the optimisation and objective function such that the objective function calls the optimisation function. This reverse communication model simplifies the coupling between the functions and frees the computational aspect of optimisation from its functionality. This is essential if we are to provide a loosely-coupled service that offers flexible integration with other resources, as shown in figures 5.1 and 5.2.

The Optimisation Service does not perform any calculation of the objective function component. Instead, it offers functionality pertaining to the state and operation of the optimisation algorithm: providing bootstrapping (initialisation), stateful management of the process, issuing of boundary parameters for calculation and multi-client support for and session management. It operates a pull model architecture that demands that the client must send commands, containing instructions or results, to the server in order to control and cause the progression of the optimisation.

Separation of objective function from the optimisation algorithm not only allows provision of common optimisation functionality, but, as a consequence, enables the user to concentrate on their objective function. In addition, the adoption of a service-based paradigm and reverse communication model simplifies the integration of the optimisation process with other associated PSE tools and computational resources.

5.4.1 Bootstrapping

Bootstrapping allows knowledge and information to be included at the initialisation stage of the optimisation process. We define this as the procedure that the optimisation server employs to guide the user through the initiation of an optimisation.

The service contains no knowledge only information about the available optimisation algorithms and their capabilities. Responsibility for selection of algorithms and boundary parameters is left to the client. This mirrors the traditional optimisation process whereby the engineer had to decided which optimisation algorithm and boundary parameters to select. However, this gave the engineer greater flexibility. If the service provided a knowledge repository it would reduce its flexibility and go against the principle of simple services. We just want the service to provide optimisation and nothing more.

A scenario of this would be at the start of the optimisation process, the client (user) sends general information to an intermediary optimisation knowledge service, such as the target-problem and its dimensions. The knowledge service will then access its database so that it can suggest suitable algorithms. For instance, the service could offer extended search capabilities to enable refinement of parameter and algorithm selection or allow manually selection and specific configuration information.

5.4.2 Stateful Code and User Sessions

Stateful code allows optimisation algorithm code to fit into the framework of web oriented application architectures, such as Sun Microsystems's Java Enterprise Server

(J2EE), where each optimisation algorithm is represented by an Enterprise Java Bean (EJB) [103]. In addition, each step of the optimisation becomes a transactional unit allowing consistency and durability. Both recoding approaches require no changes to the way in which the objective function is traditionally written. They are also independent of the operating system and programming language to which the objective function uses.

Upon receiving and accepting a new request from the user, the server initialises a new optimisation session by generating a unique session identification (ID) and creating a new associated session EJB. We have created a class structure with different implementations of EJBs that may be employed by other optimisation algorithms. This allows the easy addition of more optimisation algorithms. In addition, each contains all state and initialisation data associated with the optimisations progress, such as parameter values, previous results and awaiting calculation points.

The service employs dissemination methods described in section 4.3.4 and implements the EJB using checkpoint strategy mentioned in section 5.5.1. After successfully completion, the optimisation's related state and current data could be stored in an external database, using the EJB created at the bootstrap stage providing robustness.

This strategy makes the service flexible enough to meet the requirements of different types of client and enables the server to deal with multiple service requests at the same time. Stateful code allows the possibility of parallel multiple clients calculating independent points for a single function optimisation.

5.4.3 Client Access and Service Communication

As a means to provide a cross-platform user interface to the Optimisation Service, we choose to employ the commonly supported Web-based technology, HTTP, as the basis for the services submission system. Currently, an interface to the HTTP methods are accessible throw Web page forms that exist as static files on the service-side however, in the context of our future design objectives, we would like to dynamically generate these at runtime using optimisation schemas as the template. With the initial aim of simplifying the process of adding new services but, also enabling the possibility, with the addition of a knowledge repository, of on-the-fly changes to the user's interface that reflects their particular optimisation and environment requirements.

The emerging XForm [122] technology will, in the future, provide an effective dynamic solution with new features like platform-device independence, XML integration, and the separation between user interface and data. At the moment we support standalone client application through XML and Web Forms for browsers. We employ Servlet technology [123] to generate and process XML and Web forms. However, we shall drop support for Web forms in favour of just XML because providing the later does not fit into a our philosophy of simple services.

5.4.4 Checkpointing for Robustness, Monitoring and Control

We describe here a method, called checkpointing, that not only provides robust service execution but, in addition, provides users of traditional optimisation practices with a means to monitor and control (or steer) the progress of an optimisation. Checkpointing is an interesting side-effect that became apparent from the redevelopment of the Amoeba algorithm to support loose-coupling. Checkpointing is the ability of an optimisation algorithm to journal, or in other words, serialise its complete state to disk or database, allowing it to be resumed from any journalled point that occurred during its execution. This provides robustness such that in the case of a computer crash it may be restarted from its latest serialised instance. In addition, this also allows optimisations to be executed on compute resources in a distributed ownership environment, such as Condor (see section 6.5), where compute resources availability cannot be guaranteed requiring that executing jobs support the ability to be migrated from a resource that has become unavailable to an available resource. To support migration the job must have the ability to shut itself down and resume from where it left off on a new computer.

Another ability provided by checkpointing is too allow the possibility of steering an optimisation. Checkpointing in effect allows an optimisation to be paused or rolled back to a previous checkpoint, modified and then resumed however the optimisation must offer an interface to support modification. One last application of checkpointing would be to speed up repetitive workflows where for instance multiple optimisations are instantiated. The configuration of these optimisations may only differ slightly such that the initial parts are exactly the same and only diverge at known stages within the execution of the algorithm. Checkpointing at this stage would allow a short cut such that this stage need only be executed once and then started from that checkpoint.

A checkpoint may contain workable results however, some algorithms may only sporadically produce useful data or only upon completion. In which case, time can be saved by employing resumable algorithms. Saved state within a checkpoint alleviates the service from recalculation of variables redeemable from stored data. In addition, it allows retrieval of irrecoverable variables that would otherwise prevent the algorithm's continued execution.

Viewing of checkpoints and resumable programs provide an uncomplicated way to respectively monitor and control an algorithms progression. For example, a user may view and compare checkpoints to determine if an algorithm was advancing correctly. If not then they may simple the restart the algorithm from scratch or, if recoverable, alter its configuration and resume execution from a checkpoint with correct data.

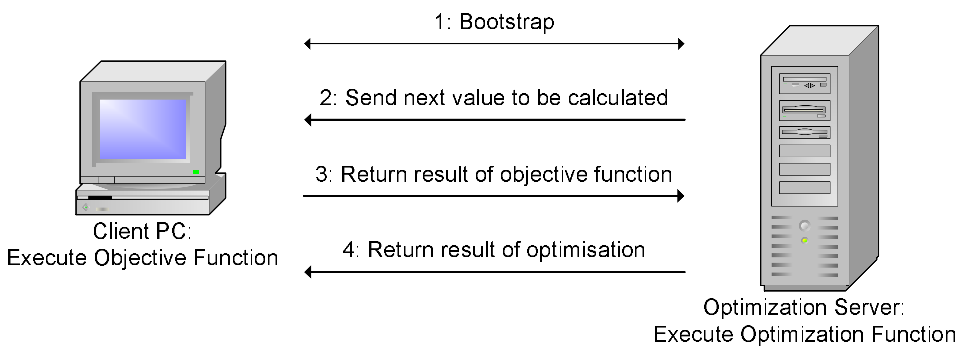

1: Bootstrap
2: Send next value to be calculated
3: Return result of objective function
4: Return result of optimisation
Client PC:
Execute Objective Function
Optimization Server:
Execute Optimization Function

FIGURE 5.3: Client driven optimisation.

5.5 Methodologies used in Optimisation Service

We will describe methods that make Optimisation Services suitable for disseminated infrastructures and reverse communication. In addition, we will propose methods to add knowledge into the optimisation process and provide automatic checkpointing functionality for the client.

5.5.1 Reverse Communication

Dissemination of objective function calculation from Optimisation Services allows the client to select alternate means of computation. This frees the service to concentrate on functionality pertaining to the state and operation of the optimisation algorithm. In addition, it allows a method to enable client-driven execution of an Optimisation Service. This involves the application of stateless logic that allows distributed execution between the optimisation algorithm and the objective function. This method is based upon dividing the optimisation process into several “stages” using checkpointing.

Once the bootstrap phase has been completed, the client is then in charge of the optimisation process that involves retrieving the points x to be evaluated, initiating the next stage of the optimisation, and retrieving the final results. Figure 5.3 shows how the optimisation process starts at 1 with the client bootstrapping the server through to stage 4 with the server returning the result of the optimisation. In order to complete the optimisation, stages 2 and 3 must be repeated consecutively for the number of times configured in the bootstrap stage. Depending on the nature of the optimisation algorithm, stage 2 may generate multiple points. These points may be calculated in any order, however all their results must be returned in stage 3 for any new points to be generated. Figure 5.4 shows this process as a flow diagram.

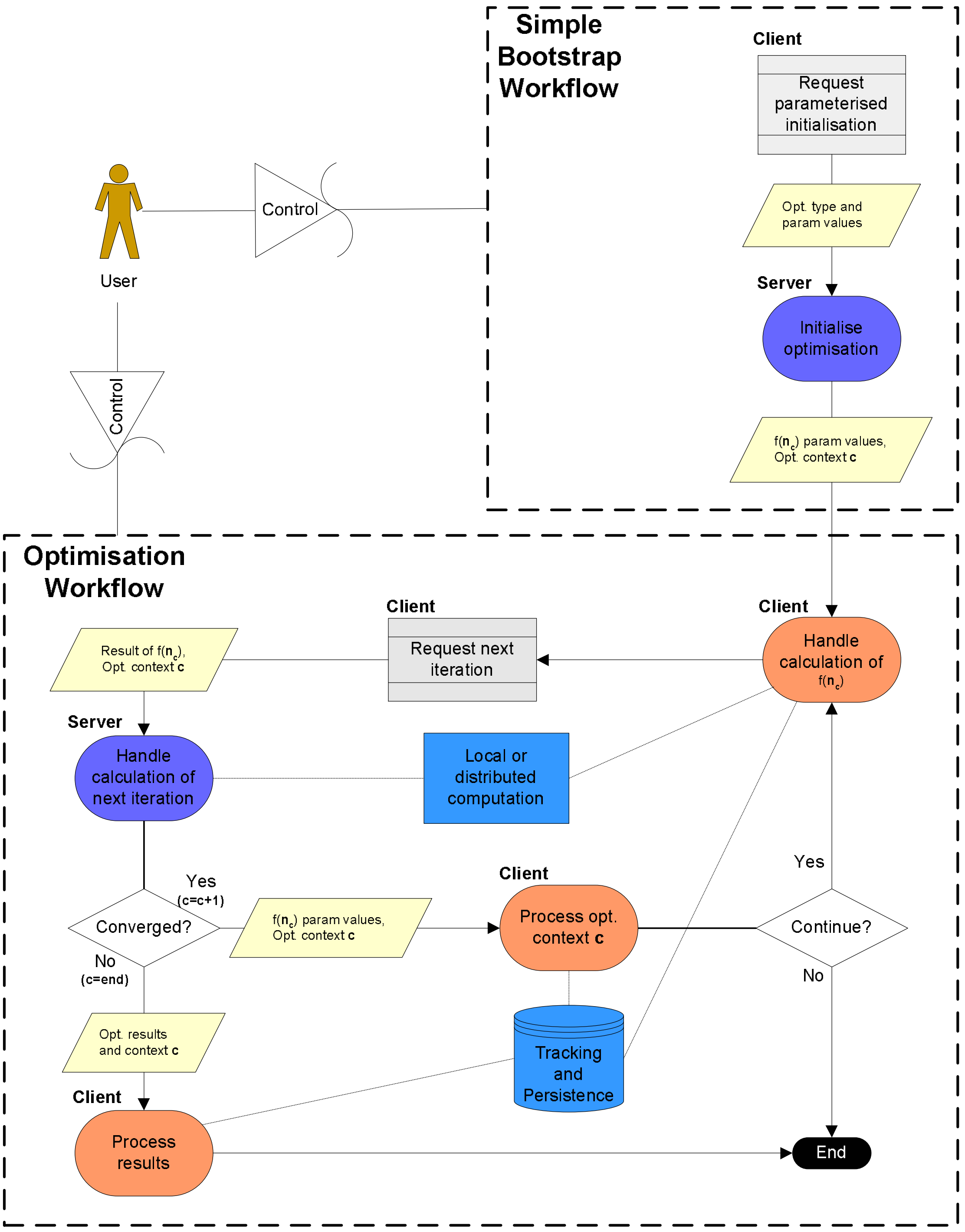

Simple Bootstrap Workflow
Client
Request parameterised initialisation
Opt. type and param values
Server
Initialise optimisation
f(n_c) param values, Opt. context c
User
Control
Control
Optimisation Workflow
Client
Request next iteration
Client
Handle calculation of f(n_c)
Result of f(n_c), Opt. context c
Server
Handle calculation of next iteration
Local or distributed computation
Yes (c=c+1)
Converged?
No (c=end)
f(n_c) param values, Opt. context c
Client
Process opt. context c
Yes
Continue?
No
Tracking and Persistence
Opt. results and context c
Client
Process results
End

Figure 5.4: Flow diagram of optimisation process.

5.5.2 Checkpointing Strategies for Distributed Optimisation Algorithms

Where machines perform simultaneous computation of a single optimisation, the task of checkpointing and resuming becomes increasingly complex. However, the optimisation routine itself may require large amounts of time between processing the results from the objective function and generating the next optimisation point. Therefore, it seems sensible to checkpoint before and after the evaluation of the objective function, shown on the right in Figure 5.5.

Where the objective function is running on a separate machine from the optimisation function, a two stage check pointing scheme becomes all the more important because the chance of failure has doubled from having two machines.

The flowcharts on the left and right show single and double stage check pointing respectively. Choosing where to place a checkpoint stage and what data to checkpoint depends on the complexity of optimisation and objective functions. Figure 5.5 show a very simplified model for check pointing of optimisation functions however; it illustrates how to split an optimisation function into "stages" shown by the process boxes in the diagram.

Optimisation functions share common key stages as follows:

1. Processing of the input parameters and data.
2. Generation of optimisation points.
3. Execution of the objective function.
4. Processing of the objective function results.
5. Finalisation of optimisation and returning results.

Stages 1 and 5 happen only once each and are the initialisation and finalisation of the optimisation function. Stages 2, 3, and 4 repeat depending on the input parameters and data, the type of optimisation function, and the results returned by the objective function. In some optimisations functions, stages 2 and 4 combine in to one because the result of the last objective function effects what the next optimisation point will be. However, we have chosen to separate out stages 2 and 4 to aid in modelling the optimisation function for check pointing purposes, as shown in Figure 5.5.

It is possible to subdivide the optimisation function further into additional stages, that allows for extra checkpoints however, this requires greater code complexity, decreases performance due to more database accesses, and produces more stage data. Each check pointing stage must save the optimisation functions current state and data enabling users to restart the function correctly and from the right place. The state data is possibly

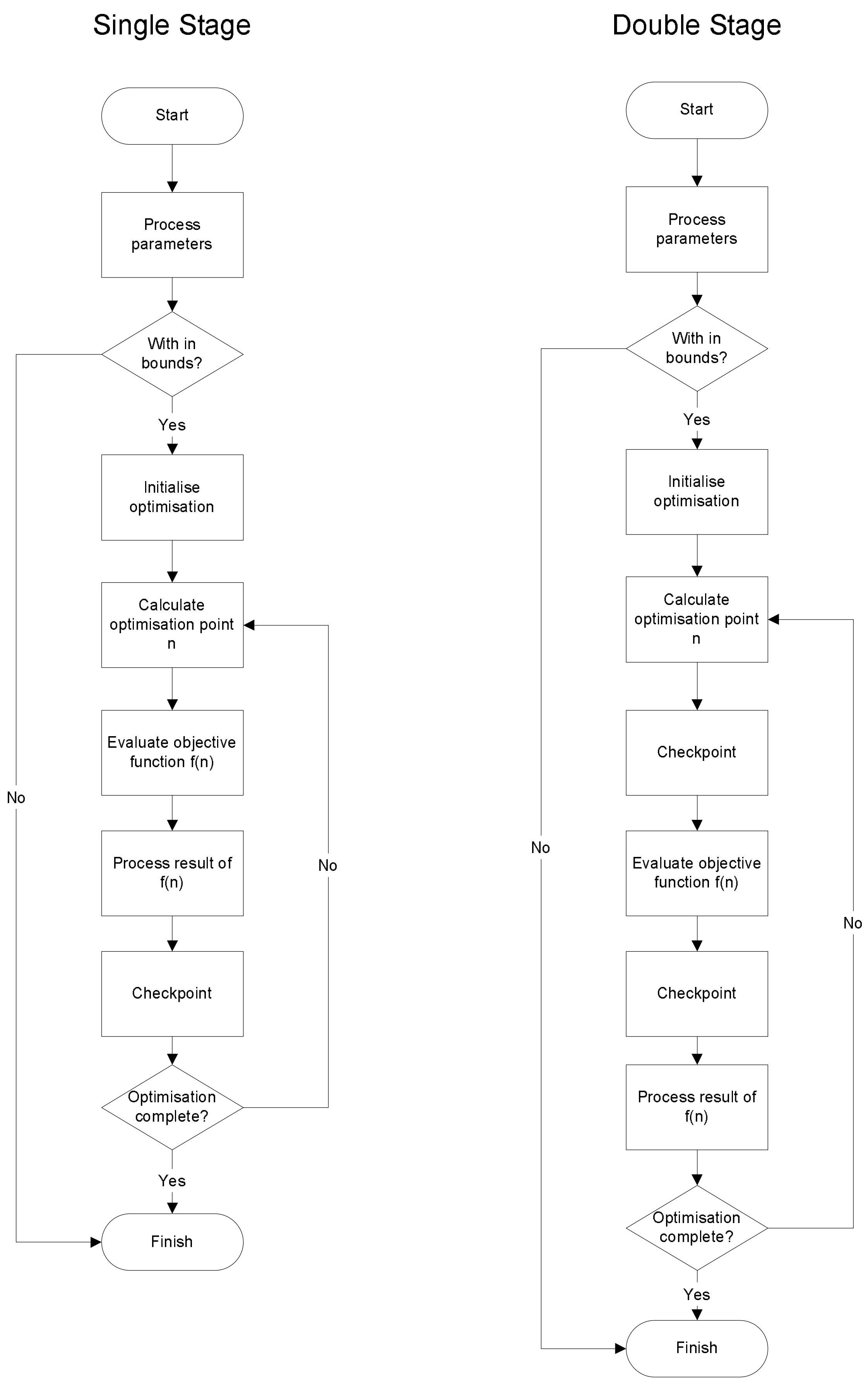


FIGURE 5.5: Single and double stage checkpointing schemes for a typical optimisation routines

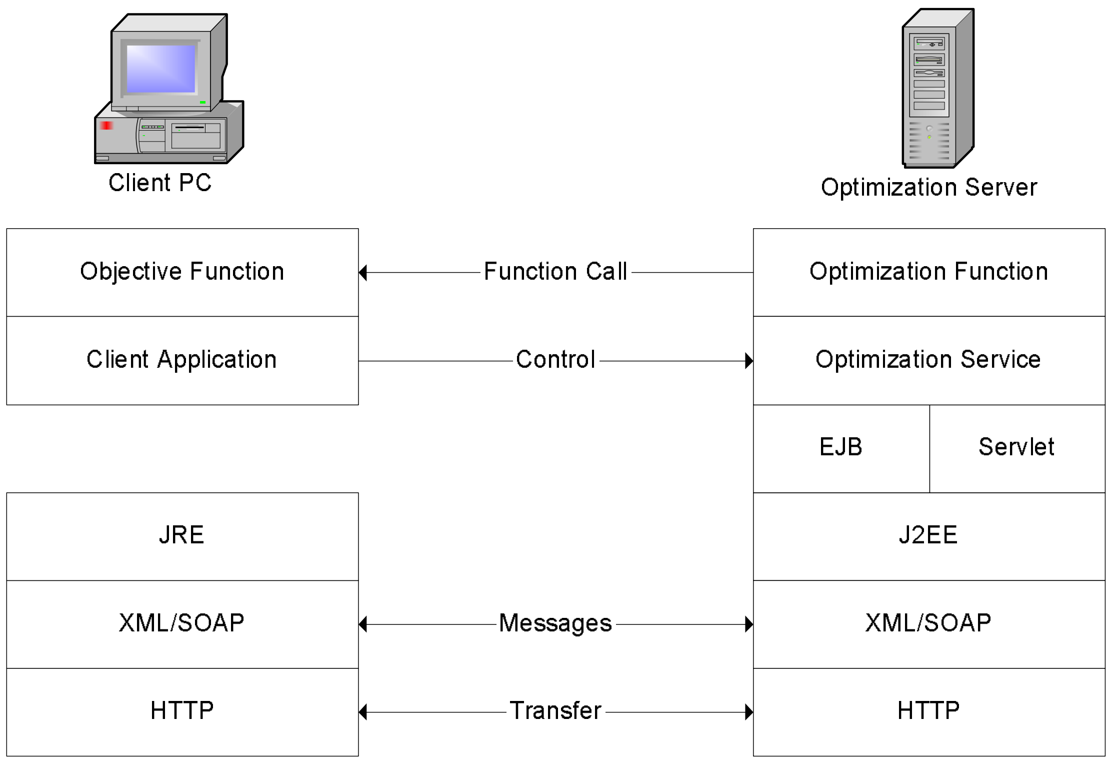


Figure 5.6: Optimisation Service Layered Technologies

saved to either a file or a database and must contain the user, the current optimisation name and its internal data, and the current session. Restarting of the optimisation requires coding the function to support restarts from any check pointed stage without re-computation of any prior stages.

5.6 Implementation

At the time of the services development it was considered that Sun Microsystems J2EE platform provided the most suitable service hosting environment. Overall, it was chosen because it provides a mature and platform-independent development and hosting environment based on Java and offers a well-defined yet flexible framework that allowed the usage of architectural patterns to separate the interface, business logic, and data layers of the service. The latter was considered most important because it allowed us to both encapsulate the implementation of the optimisation algorithm within the business logic layer whilst allowing us to explore different interface approaches. In addition, J2EE hosting environment as it is based on popular Java language had greater selection of XML serialisation tools which subsequently aided us develop the XML interface to the service as show in figure 5.6.

Two of the three components of the J2EE hosting environment were employed to create the service: Java Servlets to provide the XML, HTTP and Web form interfaces, and

```
void amoeba(
    \* The matrix p[1..ndim+1][1..ndim] is input. Its ndim+1 rows are ndim-dimensional vectors
      which are the vertices of the starting simplex.*/
    float **p,
    \*The matrix vector y[1..ndim+1] is input, whose components must be preinitialised to the
     values of funk evaluated at the ndim+1 vertices (rows) of p.*/
    float y[],
    \*Matrices dimensions input.*\
    int ndim,
    \*ftol input is the fractional convergence tolerance to be achieved in the function value
     (n.b.!)*/
    float ftol,
    \*Function pointer input to the objective function*/
    float (*funk)(float []),
    \*nfunk output gives the number of function evaluations taken*/
    int *nfunk);
```

Listing 5.1: Functional prototype of original C Language Amoeba optimisation.

Enterprise Java Beans (EJB) to provide the logic for the optimisation algorithm. A key design decision of the service was that it would be stateless consequently the data access object layer of the J2EE was unused.

5.6.1 Techniques for creating Stateless, Reverse Communication Optimisation Algorithms

The code for the Amoeba optimisation algorithm was written in the C language of which the function prototype is shown in listing 5.1 and the full listing of the source code is found in Appendix B.

A function pointer to the objective function must be supplied to the amoeba function which is then statically linked after compilation into a executable (main entry point not shown). In order to do this, source code or a library containing an implementation of the objective function must be available.

If we look at the source code for the Amoeba C source code we see that it synchronously invokes the objective function (funk) locally (within the helper function amotry) by passing in input parameters (psum) and waits for its results. Consequently, execution of the objection function and optimisation algorithm are tightly-coupled to each other. However we wish to able to perform the execution of the objection function on different computer to the execution of the optimisation algorithm. A number of solutions where explored in order to do this.

The simplest solution which requires no recoding of the optimisation function is too remove the objection function implementation within funk and replace it with a remote procedure call (RPC) to another machine that has an executable containing the actual implementation of the objective function, shown in listing 5.2. The usage of RPC is just an example, it could be replaced in the same manner with a call to a method of a Web Service.

The main issue with this solution is that the optimisation algorithm must remain in memory on the computer it is running whilst it synchronously waits for the objective

```
float funk(float [] p)
{
    /* example of how we might call the object function on a machine called whiskey */
    return remotefunk("rpc:\\whiskey.soton.ac.uk\myfunk", p)
}
```

Listing 5.2: Example pseudocode for objective function replacing implementation with remote procedure call to enable execution of code on another machine.

function to complete. If either the remote machine or the local machine running the optimisation crashes all previous work is lost. In addition, synchronous waiting ties up valuable resources of the local machine whilst it idles. What is required is that the optimisation algorithm save its state somehow during these idle times then terminate itself so it use no resources and resume from where it left off once an objective function execution completes. As discussed in section 5.5.1, reverse communication gives us an approach that enables us to achieve this. Implementation of the optimisation algorithm to support reverse communication approach involved coding the algorithm as a state machine.

Java-based source code found in Appendix C contains our implementation of the downhill simplex algorithm. Its implementation is encapsulated within a stateless Session EJB which exposes its interface locally to the Servlet layer of the J2EE hosting environment. There are two methods of interest: bootstrap method, which must be called first, initialises the optimisation; and the next method, which must be called successively by the client through the interface exposed by Servlet, which performs the optimisation itself.

Each client call to the next method represents the completion of the execution of the objective function by the client. The input parameters to the next method are its state, shown in listing 5.3 containing the context of the optimisation prior execution and the results of the optimisation function. The next function returns when it either reaches a point in the algorithm where it requires that the client compute the result of a new matrix of input parameters on the objective function, it has found the optimum result or it exceeds the maximum number of objective function iterations. The next method returns to the client via the servlet interface (e.g. HTTP response) a representation (e.g. XML) of the optimisation state data which the client can check the value of the entry point into the state machine to see which condition has happened. The client extracts from the state data the input parameters for the objective function (x[]), executes the objective function, updates the result (res) of the state data, and call next method again (e.g. with a http request) at the same time passing the updated state data to the service.

```java
package optimisation;

import java.io.Serializable;

public class AmoebaState
    implements Serializable
{
    /*initial input parameters supplied by client and bootstrap*/
    public int ndim;
    public float p[][], y[], ftol;

    /*contextual state of optimisation*/
    public float rtol, summ, swap, ysave, ytry, psum[];
    public int i, ihi, ilo, inhi, j, mpts;

    /*entry point into the state machine*/
    public int state;

    /*result of the objective function*/
    public float res;

    /*input parameters to the objective function*/
    public float x[];

    /*number of objective function iterations*/
    public int nfunk;

    public Amotry amotry; /* additional state of amotry helper function */

    public AmoebaState() {
        super();
        /*public parameterless constructor required by Java serialiser*/
    }

    public Object clone() throws CloneNotSupportedException
    {
        /*helper function to make a copy of the state*/
        /*implementation not shown for conciseness*/
    }
}
```

Listing 5.3: Implementation of the state object for the Session EJB downhill simplex algorithm.

5.7 Performance Analysis

As has been covered in section 2.5.4.1 in more detail, whilst the advantages of XML abound, it is not well suited to applications which are latency sensitive, require lengthy message exchanges or have large data sets. Here we analyse the potential performance issues of using XML-based messaging for the Optimisation Service and present required solution methodologies.

The Optimisation Service can operate and be used successfully within a very high latency network because the reverse communication protocols are time independent and service control interaction are infrequent. High latency caused by overheads of XML parsing and transfer do not impact on the period of execution of the objective function. This is because the reverse communication protocol is designed to have coarse-grained message exchanges that occur only at the beginning and end of an objective functions execution. The reverse communication protocol assumes encapsulation of the execution details of the objective function therefore clients can choose to implement there own message schemes most appropriate to speedy execution.

The period of execution of an objective function will typically be minutes or greater. Anything substantially less and other more traditional methods should be explored as being more appropriate as the service is aimed at long running optimisations. The average measured message passing time, including parsing and transfer, for the messages on the Amoeba optimisation function was in the region of 100ms. As will be discussed in section 5.7.1 this time will increase for optimisation functions that require more state data. A simple test of the downhill simplex based Optimisation Service was carried out to optimise De Jong's function 1 [124], which is defined as follows:

$$f(x) = \sum_{i=1}^{n} x_i^2 \tag{5.5}$$

where

$$-5.12 \leq x_i < 5.12 \tag{5.6}$$

and

$$n = 2 \tag{5.7}$$

An optimisation will possible require 100s to 10,000s of message exchanges between client and server. In addition, the Optimisation Service supports multiple concurrent users. Therefore, as latency is not an issue, the performance goal of the Optimisation Service should be to maximise message throughput. To partly achieve this goal, the schema of the XML message should be concise and simple to promote scalability by reducing the resource requirements on both the client and the server. In the following sections we discuss methodologies to achieve this.

5.7.1 Highly Stateful Optimisation Algorithms

The Optimisation service is designed such that all state data required for it to continue an optimisation process must be carried within the request/response message passed to and from the client and server. The data structures and consequently the size of the message are dependent on the amount and type of state data required by each optimisation algorithm. A highly stateful optimisation algorithm may require more complex and larger XML message which will require extra CPU, memory, and network bandwidth resources. For the client this will result in greater messaging overheads from longer parsing and transfer times thus increasing the optimisation period. For the server the extra resource overheads will reduce the maximum number of messages that can be concurrently processed. Consequently, message schemas must be concise and simple, and only essential optimisation state data should be conveyed in the message. Information, such as debugging, friendly messages, and logging must be included only during development and testing stages of the problem-solving process and must be strictly optional. Furthermore, during the run of an optimisations, intermediate stages within the process will not always require the full set of state data e.g. if a variable is no longer needed or not required until later on. These variables can defined in the state data schema as optional and then may be left out of message until required thus further reducing the size of the message.

5.7.2 Large Objective Function Results Set

Similar to the above section, larger result sets returned from the completion of a objective function will have bigger message sizes and thus greater resource requirements. In addition, the data structure, format and encoding of these results sets will differ according to the objective function prototype. However, unlike state data in which the Optimisation Service understands which data is required, as the implementation of the optimisation algorithm is encapsulated, the client must supply the complete result set to the server. Furthermore, only a partial result set may be needed at a particular time by the optimisation algorithm thus wasting valuable resources. The issue is worsened with large data sets, complex or multiple file result sets, and binary formatted data which are costly to include in XML messages. These issues can be overcome by usage of a data/file server that both sides share accessibility and which supports partial file/data transfers. Upon completion of the objective function the client can arrange for the results to be made available on the shared file/data server to the optimisation server and then instead supply it with an URI reference to the data. The optimisation server upon receiving the message may then read as much or as little of the results set as it requires. It is important to note that results sets supplied to the Optimisation Service in this fashion must be in a non-relationsional and non-hierarchical format in order for the optimisation server to partially read yet fully understand the data. This unfortunately rules out using XML however this is an advantage because data formats such as flat files, arrays, and lists are significantly quicker and easier to generate and parse, and are generally much more concise.

5.8 Summary

The Optimisation Service demonstrates the successful usage of key technologies that we discussed in section 2.5.4, including XML and HTTP, and has shown how it is possible to create stateless loosely-coupled numerical Optimisation Service. In addition, we have implemented the concepts described in chapter 4, and can show, although they have only been utilised in a functionally simple example, that the analysis described therein is valid and has great potential in future projects. This chapter has given an example of how a key component of the problem solving process can be offered as services and highlight the significant advantages of a Service-Oriented Architecture. We now summarise important conclusions from work in this chapter:

1. The new technique of decoupling the objective function from the optimisation algorithm using a reverse communication model has allowed us to build an extremely flexible, loosely-coupled service. Its adoption was a vital design decision which gives much greater flexibility in the choice of services which may be coupled together with it in to environments for problem solving; specifically computation, persistence and knowledge services of which it is independent of any specific implementation.

2. Reverse communications provides users with a programming style to build robustness, monitoring and control into their codes.

3. Adoption of a stateless service model was another key design decision that reduced the complexity of service's implementation, allowed the service to better scale and in combination with reverse communication provides additional reliability through checkpointing. In addition, reverse communication made it possible to easily create a stateless service.

4. The numerical Optimisation Service is an example of how a purely message-driven service provides flexibility by allowing its integration with resources such as databases and compute cluster.

5. Encapsulation of optimisation algorithm within the service removed the implementation details of the optimisation algorithm from the client which further promotes interoperability.

However, whilst we have shown that its is feasible to use some of the basic Grid technologies in a single service it is apparent that in order to make sophisticated PSE it requires examination of the newer Grid concepts of Web Services. The following chapter will examine Web Service technology in detail.

Chapter 6

Service-Oriented Computation

This chapter discusses the sharing and management of high throughput cluster resources using service-oriented methodologies discussed in chapter 4. We demonstrate the usage of Web Service technologies to provide meta-based computing system that virtualises standard cluster operations; including submission, management, monitoring of compute jobs and the discovery of compute resources and their matchmaking with job requirements.

We will show how a legacy system, the Condor High Throughput Computing (HTC) system, may be modelled as a Grid Resource and consequently adapted and virtualised to provide the necessary interoperability for inclusion in Grid-enabled Problem Solving Environments (PSEs). We will demonstrate the success of our work by showing the deployment of the Service-Oriented HTC resource, the Computation Web Service, by discussing its deployment in the Geodise project for engineering design simulations. In addition, in the next chapter 7 we will demonstrate its employment in a micromagnetics PSE. The work presented in this chapter has been published in [66, 65, 125, 126].

6.1 Virtualisation of Compute Clusters

Resource virtualisation of compute clusters provides a common operation and behaviour abstraction that allows easy migration between particular implementations and improves application level interoperability.

6.1.1 Issues with existing virtualised Compute Resource Systems

The Gateway system [127] and the Globus Resource Allocation Manager (GRAM) [128] attempted to provide virtualised access to computation resources. However, whilst they

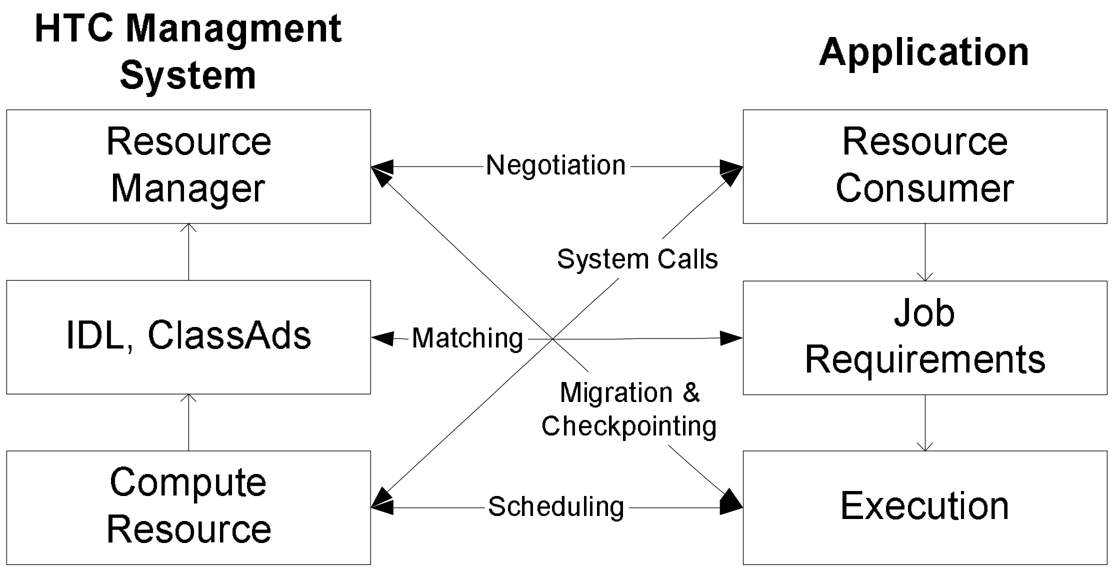


FIGURE 6.1: Common submission of Jobs in a HTC management system.

intended to provide a generic representation, their success has been limited by their employment of proprietary interaction and resource description methods. With Gateway employing CORBA for its middleware component model and GRAM defining its own resource description language (RDL) [83]. In addition, these systems bind to a limited number of resource management system and software platforms, all said, making migration to alternative resource providers difficult.

6.1.2 Common Compute Operations

HTC systems base their mode of operation around jobs and matchmaking of their requirements with machine capabilities, see section 2.4.3. They all share common facilities for the submission, control and monitoring of jobs, and the discovery and description or compute resources. This forms the basis for the virtualised interface to our Computation Web Service.

6.1.3 Interoperability and Portable Infrastructure through Common Open Standards

We have identified a common set of attributes and behaviours used for job submission in a variety of HTC systems; describing and classifying them as service properties and operations using XML Schema and WSDL. The Computation Web Service provides an open standards based, and portable infrastructure for representing compute resources. All interaction with it can only be carried out via SOAP encapsulated XML messages therefor, hiding details of the underlying system, in this case Condor.

6.2 Analysis of a Cluster Management System: Condor

The Cluster Resource Management Web Service exploits the capabilities of a well established HTC computing system called Condor [44]. Condor provides a HTC environment by detecting and harnessing idle computers in a distributed ownership context. Our selection of Condor was motivated by the desire to take a powerful yet closed system, with limited interoperability and make it suitable for Grid computing.

6.2.1 Large-scale Cycle-stealing

The opportunity represented by idle computers has been recognised for some time [129]. Condor was proposed as a system to harness idle cycles for useful work. It exploits the multitasking possibilities of popular Unix operating system and the connectivity provided by the Internet. The Condor system is now used extensively within academia to harness idle processors in workgroups or departments. It is used regularly for routine data analysis as well as for solving open problems in mathematics. At the University of Wisconsin, for example, Condor regularly delivers 400 CPU days per day of essentially free computing to academics at the university and elsewhere: more than many supercomputer centres.

Condor is a freely available software system that creates a High-Throughput Computing environment by harnessing the power of computer clusters and workstations. It is capable of managing dedicated clusters however, Condor's most appealing feature is its ability to use non-dedicated, pre-existing computational resources in a distributed ownership context and highly heterogenous environment.

Condor employs the concept of organisational pools each consisting of a network of computational resources controlled by a central manager, shown in Figure 6.2. It supports a wide range of computing platforms, including Linux and Windows NT. Different types of supported platform may coexist within a pool as a computational. Condor enables submission, remote execution checkpointing of users' programs (jobs) on computational resources within a pool. Pools may be flocked together allowing jobs submitted from within a pool to be executed on another pool. Computational resources may take the form of dedicated or non-dedicated pre-existing computers, of which the latter would typically be an office-based PC or shared workstation that intermittently perform short tasks.

Condor's power comes from its ability to effectively harness unutilised processing power. It is capable of detecting when a resource should be considered idle by evaluating a user configurable algorithmic expression based on discernable factors (e.g. keyboard and processor activity) and user specified constraints (e.g. period of idleness before availability). Jobs may run only run on an idle computer but, Condor has the ability to

```
MyType = "Machine"
TargetType = "Job"
Name = "node01.condor.soton.ac.uk"
StartdIpAddr = "<1.2.3.4:32780>"
Arch = "INTEL"
OpSys = "WINNT51"
UidDomain = "condor.soton.ac.uk"
FileSystemDomain = "condor.soton.ac.uk"
State = "Unclaimed"
EnteredCurrentState = 8987459837
Activity = "Idle"
EnteredCurrentActivity = 8987459836
VirtualMemory = 256000
Disk = 80192
KFlops = 24034
Mips = 2403
LoadAvg = 0.0234233
CondorLoadAvg = 0.0000000
KeyboardIdle = 1987
ConsoleIdle = 123232
Cpus = 2
Memory = 2048
Start = LoadAvg - CondorLoadAvg <= 0.300000 && KeyboardIdel > 15 * 60
Requirements = TRUE
Rank = Owner == "mrsmith" || Ownre == "mrssmith"
CurrentRank = -1.0000000
LastHeardFrom = 8987459837
```

Listing 6.1: Example Condor Classads for an execution node. Sample output returned from *"condor_status -l node01"* command.

checkpoint and migrate a job to another idle computer if the owner wishes to use their computer for another task.

Condor simplifies job submission by acting as a matchmaker [130] between job requirements and machine attributes. To facilitate this, Condor employs a powerful framework founded on an extremely flexible expressive text-based language called Classads [130] of which an example is shown in listings 6.1. This language enables computer resources of any type to advertise their complex attributes. In addition, Classads flexibility allows users to both easily run jobs with complex needs and define sophisticated job execution instructions, such as number of runs, arguments, and redirection of standard input and output streams. Each pooled computational resource advertises its own attributes in the form of a Classads. This contains specific information regarding the machine's platform type, available hardware resources and its runtime environment capabilities. In turn, users must submit their jobs to Condor in the form of Classad-based job submission files that contains their job's requirements, desires and file location information.

Condor has the following strengths that are desirable in Grid computing:

- its computational nodes (i.e. computers) are loosely-coupled: they may be added, removed or crash without effecting the overall operation of the system. In addition, adding of nodes to it requires no changes to the configuration of central manager.
- its adopt a efficient point-to-point file transfer mechanism where by files required to run jobs are transferred directly from the submission node to the execution node. Consequently, it avoids bandwidth bottlenecks. In addition, this mechanism takes advantage of the extra bandwidth brought by nodes as they are added to the system and effectively distributes overall transfer loads across the network making Condor very scalable.

- it provides dynamic discovery, aggregation and allocation of resources. All execution nodes advertise (offer to sell) their own capabilities. And likewise all submission nodes advertise (offer to buy) nodes that fit their jobs requirements. Condor acts as a simple broker by periodically reading all nodes' advertisements, matching and ranking job classads with machine classads. This approach requires no centralised repository, information across the system is kept up to date and is dynamically updated as and when machines and jobs are added and removed from the pool.
- it provides sharing and access of a heterogenous network of machines which is made possible by the adoption of a simple common language (Classads) for expressing all information.

Nonetheless, Condor suffers from the following weakness:

- access to it requires heavy-weight bespoke client software. Submission nodes (i.e. the computer from which users wish to access Condor's resources) must run a set of Condor daemon services in order to submit, run and monitor jobs, as shown in Figure 6.3. The remote interfaces of the daemons services are unpublished and closed consequently making it impossible to natively access Condor from computer platforms not supported by its software. Consequently, integration of Condor with other systems is difficult although specific support for Globus access has been added in recent revisions of the software.
- enabling access across network boundaries poses significant security risks. Its daemon services communicate via remote procedure call mechanism and a large number of incoming and outgoing TCP/IP sockets that due to Condor's architecture can originate from any Condor node to any other Condor node. Consequently, large ranges of ports must be opened up between network firewalls from any machine to any machine.

6.2.2 Condor and the Grid

There exists fundamental differences between Condor and the Grid. As a project, it began development before the concept of the Grid came into existence. Whilst its powerful architecture, communication and data model suit a closed network and environment, its monolithic closed nature and employment of proprietary technologies and single sharing mode precludes easy integration within PSEs and Grid systems.

Figure 6.2 shows the interacting components of the Condor System within a pool. Condor refers to the cluster of condor-capable submission and execution computers in its

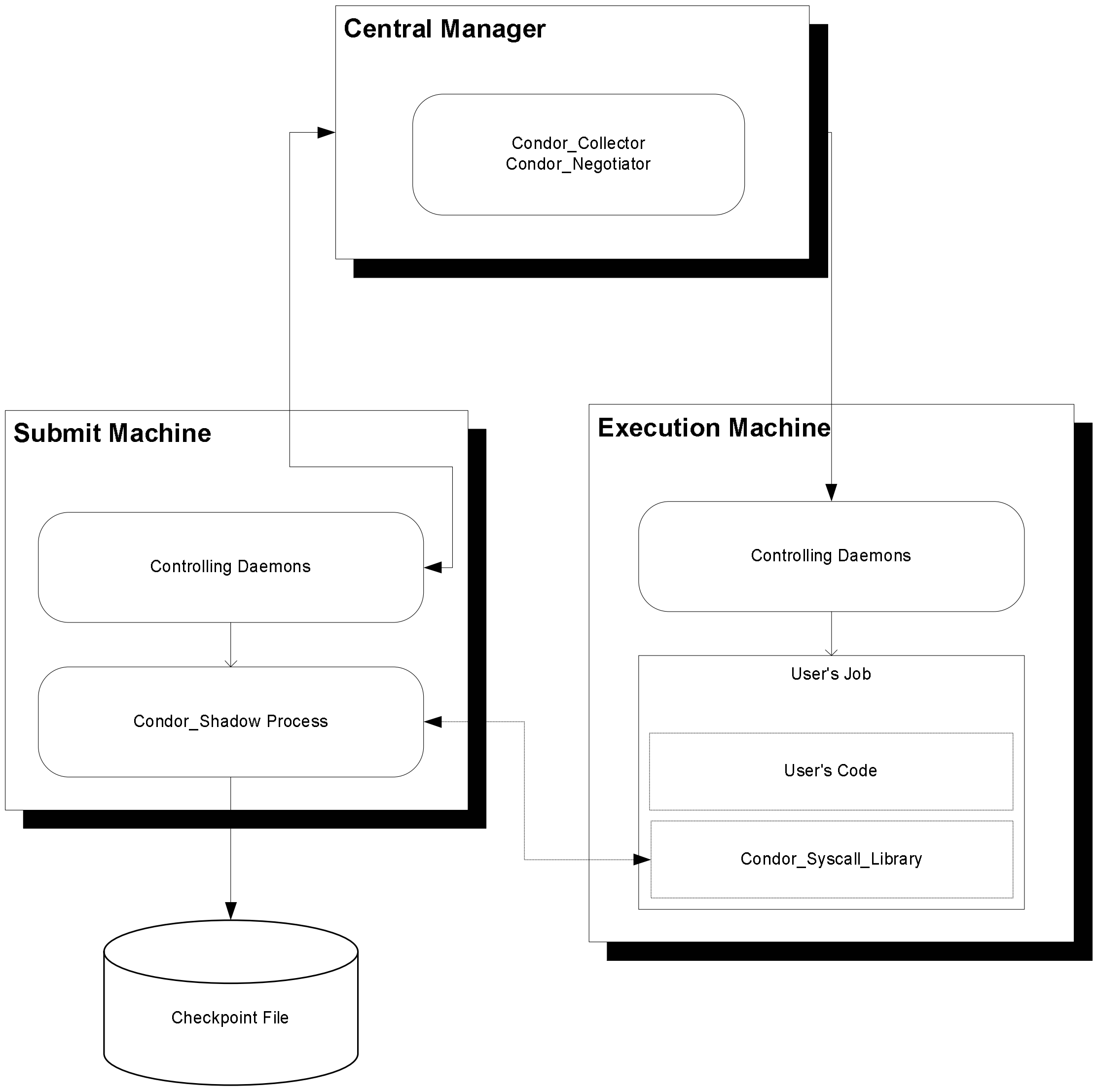


FIGURE 6.2: Pool (Cluster) components of the Condor HTC system.

central manager domain as a Pool. Condor's component model employs daemon operating system processes, such as *condor_collector*, to service the operation of the Condor system. Table 6.1 shows the role of these daemons in the Condor system.

Communication amongst daemon processes happens exclusively as remote procedure calls. However, except for the the recent provision for Globus resources, Condor's architecture and lack of an open API makes direct integration with other resources at the network and middleware level practically impossible. Interaction with the Condor system takes place at the application level through a set of command line programs, shown in table 6.2.

Compute resources within Condor's pool advertise their capabilities via a simple to use yet powerful, expressive proprietary data format called Classads [131]. They are used exclusively throughout the system for all data collection, resource and requirement

System Role	Condor Daemons	Description
System Management	Condor_master	Launching, monitor and control of all other daemon processes.
Resource Representation and Access	Condor_startd	Represents the resource, such as, execution computer, advertises its capabilities and controls access to resource.
Job Creation and Monitoring	Condor_starter	Creates a jobs execution environment, launches it and monitoring the job's running progress.
Resource Scheduling	Condor_schedd	Represents resource request; maintains a queue of jobs waiting and claims resources for job execution.
Job Management	Condor_shadow	Shadows the execution of a job, handling file transfers and providing remote control of allocated resource.
System Information Collection	Condor_collector	Collects information about all resources in Condor's pool. All daemons periodically send ClassAds to the collector.
Resource Matchmaking	Condor_negotiator	Matches jobs' requirements with resource capabilities. Periodically polls schedulers for jobs awaiting resource allocation.
Job Checkpointing	Condor_ckpt_server	Stores and retrieves checkpointed job information for their reliable operation.

Table 6.1: Categorisation and description of Condor's daemon services.

Type	Command Name	Description
Resource Identification, Monitoring and Discovery	condor_status	Returns status of machines as ClassAds
	condor_stats	View historical info about Condor pool
Resource Management	condor_findhost	Find a machine in Condor's pool that's removal has minimal impact on running jobs
	condor_glidein	Temporarily add Globus resource to Condor's pool
Job Management	condor_rm	Remove job from the queue
	condor_hold	Place job in suspended execution status
	condor_release	Release job suspended status
	condor_checkpoint	Force a job to checkpoint to a file (standard universe only)
Job Identification, Monitoring and Discovery	condor_history	View log of the state of completed or removed jobs
	condor_q	View job's status in queue
Job Scheduling	condor_submit	Submit jobs to queue
	condor_run	Submit shell jobs to queue
	condor_prio	Change execution priority of job
	condor_qedit	Change submitted job requirements
User Monitoring	condor_userlog	View job statistics from users log
User Management	condor_userprio	Manage a user's priority

Table 6.2: Categorisation and description of commands employed to operate Condor.

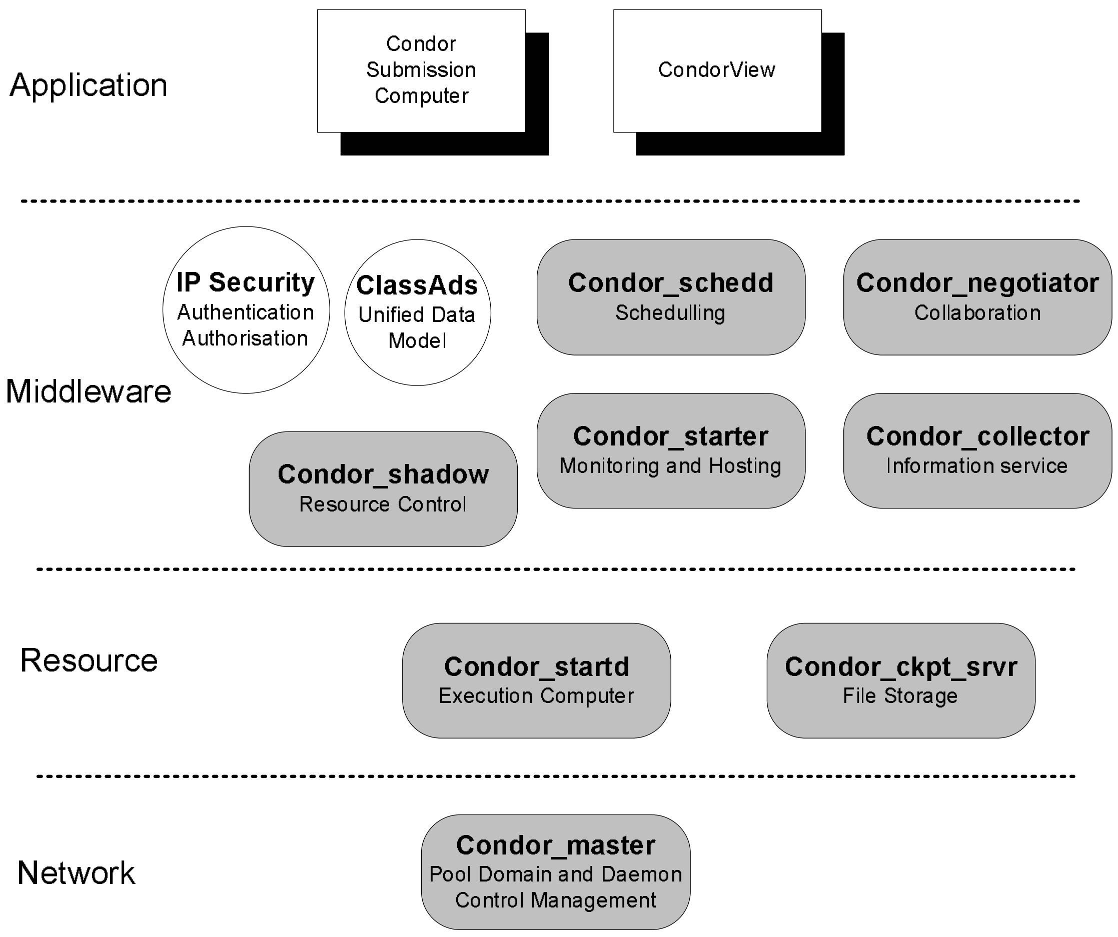


FIGURE 6.3: Architecture of a typical HTC clustering system.

description and monitoring purposes. Whilst their consistent employment throughout the system gives the appearance of a single system image and simplifies interaction, Classads remain a proprietary format for Condor.

6.2.3 Identification and Modelling of Resource Layers

Condor may be viewed as purely a resource provider at the lowest level of the Grid. However, it has component that fit into each of the higher levels of the Grid, shown in figure 6.3.

If we view the Grid model from the bottom up, at the network level we have RPC communication over TCP with the *condor_master* performing network operations that manage and monitor the other daemons of the system. At the resource level we have the execution computers in Condor's pool thats' access is controlled by the *condor_startd* daemon. Control, management and monitoring of the execution machines is provided by the daemons in the middleware level. Within the middleware level itself daemons collaborate with each other to provide sophisticated system functionality that is presented to the user at the application layer through contrib modules, such as the CondorView

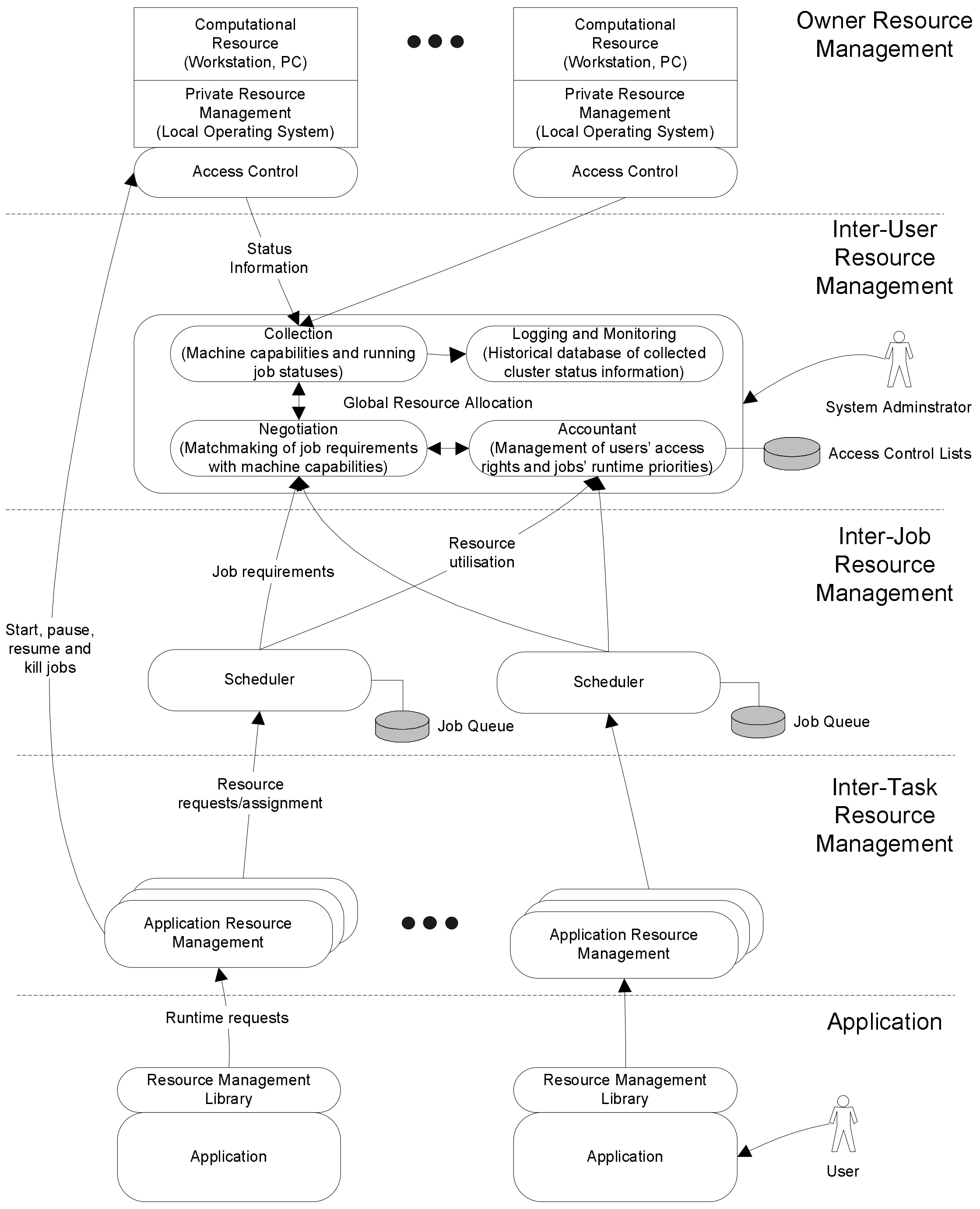

Computational Resource (Workstation, PC)
Private Resource Management (Local Operating System)
Access Control
Computational Resource (Workstation, PC)
Private Resource Management (Local Operating System)
Access Control
Owner Resource Management
Status Information
Inter-User Resource Management
Collection (Machine capabilities and running job statuses)
Logging and Monitoring (Historical database of collected cluster status information)
Global Resource Allocation
Negotiation (Matchmaking of job requirements with machine capabilities)
Accountant (Management of users' access rights and jobs' runtime priorities)
System Adminstrator
Access Control Lists
Inter-Job Resource Management
Resource utilisation
Job requirements
Start, pause, resume and kill jobs
Scheduler
Job Queue
Scheduler
Job Queue
Inter-Task Resource Management
Resource requests/assignment
Application Resource Management
Application Resource Management
Application
Runtime requests
Resource Management Library
Application
Resource Management Library
Application
User

FIGURE 6.4: Architecture of a typical HTC clustering system.

module for Web viewing of the state of the Condor Pool, and Condor commands, shown in table 6.2.

Identification and separation of components into layers in a user-oriented manner is an important step in providing a virtualised interface to the Grid. However, it does not take into account the collective and resource sharing operations apparent in multi-user, multi-job clustering software systems. A clearer picture can be derived using a resource-centric view.

Resource	Sharing Operation	Condor Commands
Inter-job	Scheduling	Condor_q, Condor_qedit and Condor_history
Inter-user	Discovery and information	Condor_status, and Condor_stats
Inter-task	Request and management	Condor_submit, Condor_hold, Condor_resume, Condor_rm

Table 6.3: User command program interaction within resource layers of Condor.

Figure 6.4 shows a generic clustering software stack for HTC cluster management system. It delineates the software into inter-user, inter-job and inter-task resource layers, whilst showing the interactions amongst features of each layer. Each layer represents a particular resource of the compute software system that that may form sharing operations with other resources in a larger distributed system. The layer provides:

inter-user a cluster information discovery resource for users of the cluster; providing, for instance, information about the clusters capabilities and historical usage records

inter-job a cluster scheduling resource for jobs spanning multiple users that provides, for instance, analysis, monitoring and historical information of the status of jobs in the execution queue.

inter-task a computer management resource for task with multiple jobs that provides, for instance, acquisition and control of computers

In, for example, the scenario of a engineering and design optimisation search PSE, engineers may have tasks, such as a wing design simulations, where software can only be run a particular computer platform. The PSE provides search tools to find suitable computers to run the simulation software and performs load-balancing between clusters to ensure that users' job requests are serviced quickly. The PSE could use the inter-user resource to discovery suitable computers and the inter-task resource to manage submission of jobs across a selection of clusters to ensure no cluster has too many jobs waiting for execution. The PSE could employ the inter-job layer to monitor the progression of jobs to check that they are being serviced in a timely fashion and if not reassign them to cluster that has a higher throughput of jobs.

This example shows a sophisticated sharing operation with interaction between resources in clusters and PSE, including resource discovery, load-balancing and monitor across multiple clusters to ensure desired qualities of service. We studied the Condor HTC system from a resource-centric perspective to find resources that fit into the generic model shown in figure 6.4. Table 6.3 identifies resource sharing operations that Condor offers and the commands that Condor provides to access the resources.

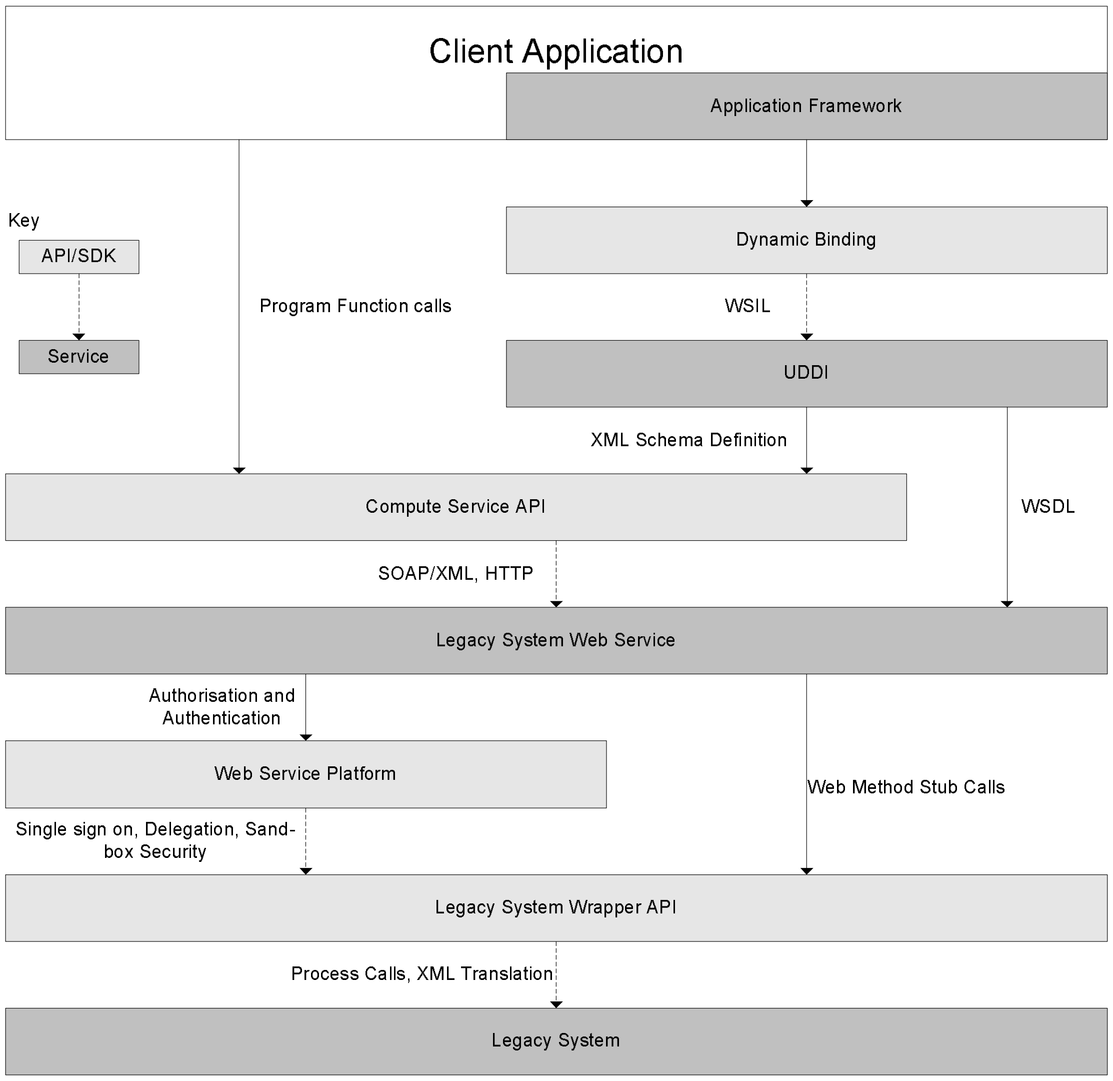


Figure 6.5: Open standard protocols stack of the compute resources and Computation Web Service. This shows the interfaces exposed by each layer of the Web Service implementation

6.3 Resource Sharing Operations through Web Services

We now describe methods to expose the resources in cluster management system for sophisticated sharing operations using service-oriented technologies and programmatically means. Using Microsoft's .NET framework [132] and ASP.NET, we have successfully wrapped a cluster management system, Condor, as a Web Service providing a generic interface that enables its seamless integration into the Grid. In addition, we show how the employment of open-standard technologies enable the interoperability, security and standard file transfer mechanisms needed in sophisticated Grid-enabled PSE, shown in figure 6.5.

Figure 6.6 shows an overview of a cluster management system exposed as a service. Access and sharing of compute and cluster management system resources is controlled

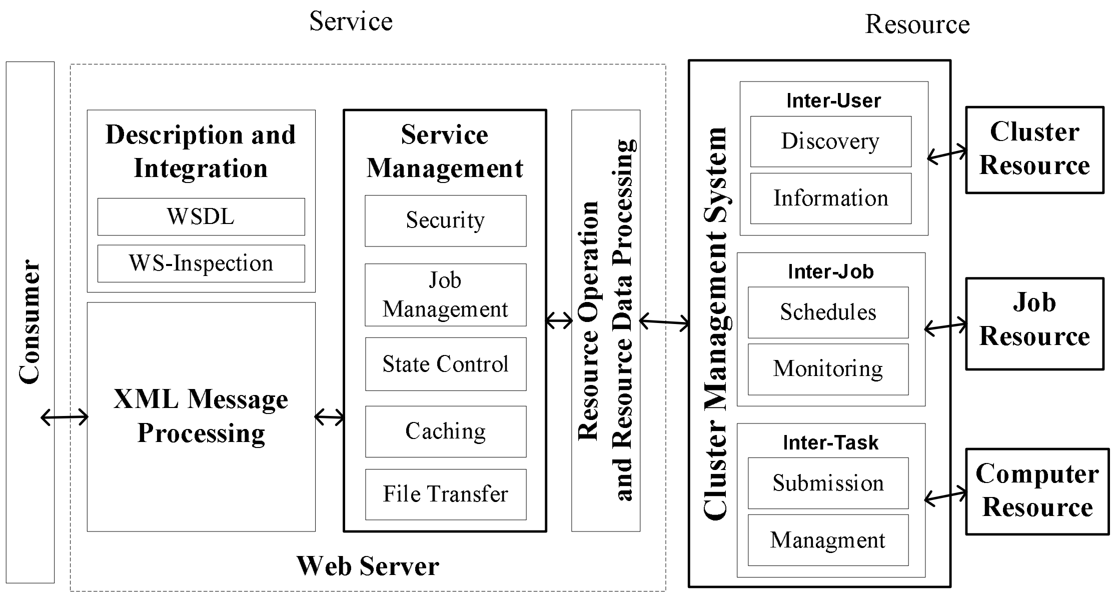


FIGURE 6.6: Architecture of Computation Web Service

through the service layer. The service adds file caching and management, security, and user and task management on top of the sharing operations of the cluster management system. The sharing operations of the particular underlying system are often combined and separated within the service layer in order to provide generic operation and behaviour to clients of the Web Service.

WSDL provides the standard mechanism to define the sharing operations of a virtualised clustering management system, listed in Appendix D. Clients perform interaction with the Web Service through operations represented by Web Methods defined in the WSDL document that are transferred via HTTP in the form of SOAP addressed XML messages. All interaction types including, instructions, queries and data transfers to the underlying resources happens through the service layer. From the outside, the service appears as a generic cluster management system rather than just an interface. This is important as it isolates clients from the details of the underlying implementation.

The service supports the following fundamental operations: submission of jobs, removal of jobs, querying of jobs status and querying of machine status, in addition to file transfer methods. Listings 6.2 shows the API of the services which is exposed via a WSDL interface listed in Appendix D. The following sections look at each of these operations.

Figure 6.7 shows the orchestration of a complete job submission process based on the service operations described above.

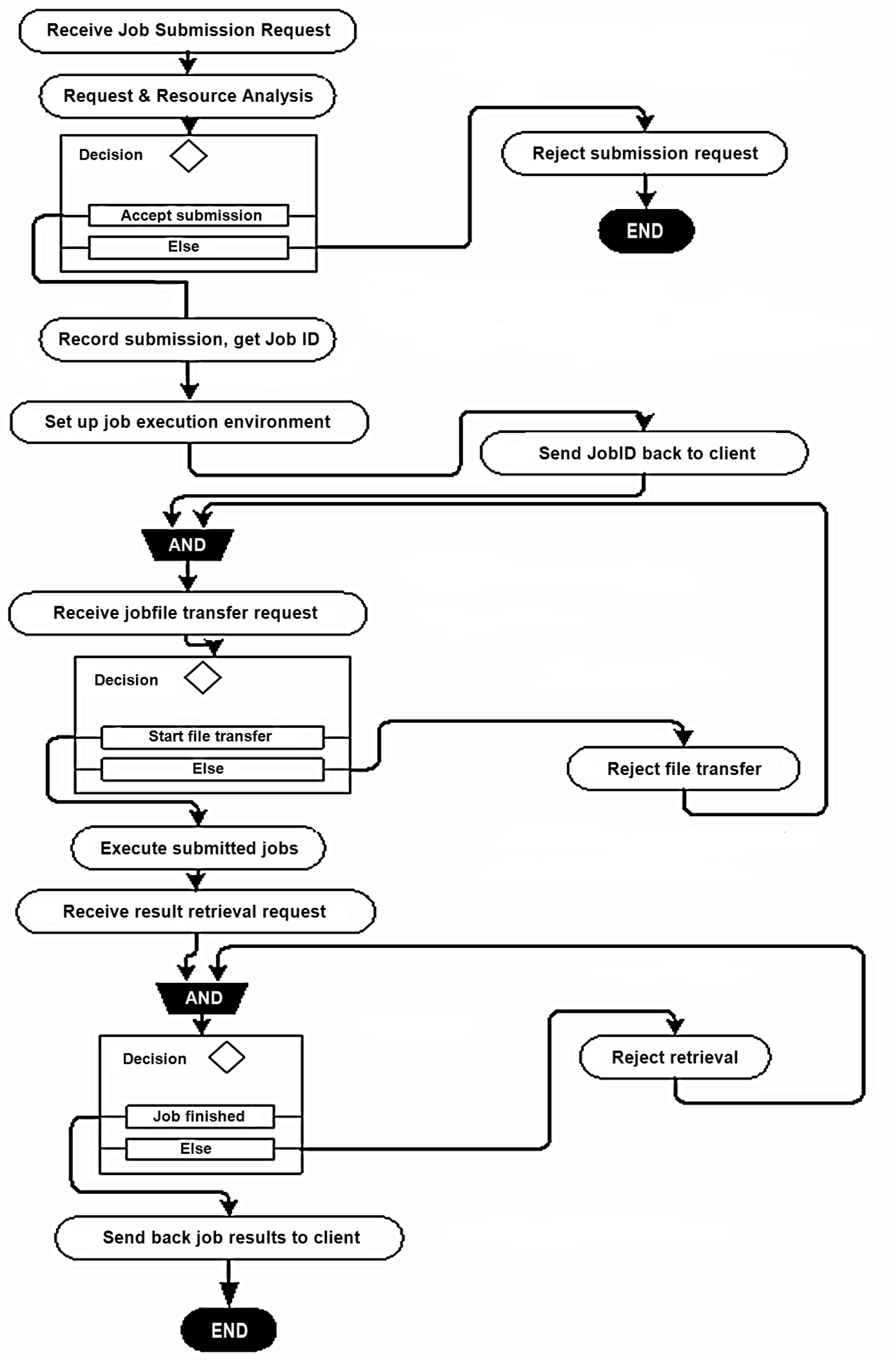

Receive Job Submission Request
Request & Resource Analysis
Decision
Accept submission
Else
Reject submission request
END
Record submission, get Job ID
Set up job execution environment
Send JobID back to client
AND
Receive jobfile transfer request
Decision
Start file transfer
Else
Reject file transfer
Execute submitted jobs
Receive result retrieval request
AND
Decision
Job finished
Else
Reject retrieval
Send back job results to client
END

Figure 6.7: The Job Submission Process

```
//SUBMIT
[WebMethod(Description="Submits a job to the Condor job queue and returns job ID.")]
public JobID Submit(SubmissionClassAd desc);

//Machine STATUS

[WebMethod(Description="Returns MachineClassAds for machines.")]
public MachineClassAd[] GetMachineStatuses(string[] machineNames, MachineStatusQuery query);

[WebMethod(Description="Returns MachineClassAds for machines.")]
public MachineClassAd[] GetAllMachineStatuses(MachineStatusQuery query);

//Job STATUS

[WebMethod(Description="Returns JobClassAd for user's jobs.")]
public JobClassAd GetJob(JobStatusQuery query, JobID id);

[WebMethod(Description="Returns JobClassAds for all user's jobs.")]
public JobClassAd[] GetAllJobs(JobStatusQuery query);

[WebMethod(Description="Returns JobClassAds for user's jobs.")]
public JobClassAd[] GetJobs(JobStatusQuery query, JobID[] ids);

//REMOVE

[WebMethod(Description="Removes user's job and returns a bool to determine if operation was
      succesful")]
public JobRemoveStatus RemoveJob(JobID id);

[WebMethod(Description="Removes user's jobs and returns a bool array to determine if operation
      was succesful")]
public JobRemoveStatus[] RemoveJobs(JobID[] ids);
```

Listing 6.2: Application Programmer Interface (API) of Computation Web Service

```
...
Requirements = (
     (Arch==INTEL && OpSys==WINNT50&&
      Memory>=128 && Disk>=100) ||
     (Arch==INTEL && OpSys==WINNT51&&
      Memory>=256 && Disk>=100)) &&
     (HasMatlab==true) &&
     (MatlabVersion== 6.1 )
...
```

Listing 6.3: Example of a ClassAd job submission request to the Condor

6.3.1 Computer Discovery

As with Web Services, resources should provide description, discovery and integration operations to enable their dynamic discovery and allocation. In heterogenous HTC compute clusters they share a common need to describe the capabilities of their computers to ensure that jobs with particular runtime requirements are matched with an appropriate computer for their correct execution. As well as runtime specific requirements, such as the operating system, machine architecture and disk and memory space, jobs need to specify environmental, file and execution specific details, such as job arguments and standard input, output and error streams. In addition, sophisticated compute clusters may also support instructions that tell the cluster system how it should transfer files, the number of executions of the job and the arguments to pass to each run of the job. The Condor system, for instance, employs Classads to describe resources (shown in listings 6.3), whilst GRAM employs RSL. The service exposes this operation through the GetMachineStatuses method.

```
...
<JobRequirements>
    <PlatformSpecification>
        <OperatingSystem>WINNT50</OperatingSystem>
        <MemoryRequirement>128</MemoryRequirement>
        <DiskRequirement>100</DiskRequirement>
        <Architecture>INTEL</Architecture>
    </PlatformSpecification>
    <PlatformSpecification>
        <OperatingSystem>WINNT51</OperatingSystem>
        <MemoryRequirement>256</MemoryRequirement>
        <DiskRequirement>100</DiskRequirement>
        <Architecture>INTEL</Architecture>
    </PlatformSpecification>
    <Runtime>
        <Version>6.1</Version>
        <Name>Matlab</Name>
    </Runtime>
</JobRequirements>
...
```

Listing 6.4: Example of a XML job submission request to the Compute Service.

6.3.2 Job Submission

Job submission is the operation to request and acquire a compute resource. The submission process happens as follows:

1. a job description is sent to the service, which provides job details and specify resource requirements
2. when accepted, all related job files are transferred to the service using file transfer service operations
3. the user requests to start the job

The format of the job descriptions is defined in the form of an XML Schema based on common resource specification languages, such as the Condor ClassAd and RSL. Every job description is validated against the schema to ensure data integrity and correctness. Listings 6.4 show an example of XML-based request to the compute service for job submission request. The Submit method of the service provide submission of jobs.

6.3.3 Resource Management and Job Monitoring

Job monitoring operations allow users to query the status of their jobs. The Optimisation Service WSDL, Shown in listings 6.5, is the response In Condor, jobs are often stopped and started by the system as and when the state of the machine is detected to be idle. A job can be monitored to see if it has completed, held, or idle waiting for a resource. Users may control the resource that job is executing on by suspending, resuming, removing, or restarting their job execution. Listings 6.5 show an example of XML-based response from the compute service to the client who requested the GetJob method to query the status of their jobs.

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <user>mjf</user>
    <messageID>UUID:67125375-3268763-36486-034872646</messageID>
    <relatesTo>UUID:7264876324-3284265-23874686-4673</relatesTo>
    <from>
        http://philip.sesnet.soton.ac.uk/ComputationService-NiceUser/Gateway.asmx
    </from>
  </soap:Header>
  <soap:Body>
    <ca:JobStatusResponse
        xmlns:ca="http://www.e-science.soton.ac.uk/webservices/computation">
        <Cluster Name="Southampton E-Science">
            <Job Name="OOMMF1">
                <MachineHost>e-science04.soton.ac.uk</MachineHost>
                <Status>Running</Status>
            </Job>
            <Job Name="OOMMF2">
                <MachineHost>e-science08.soton.ac.uk</MachineHost>
                <Status>Evicted</Status>
            </Job>
        </Cluster>
    </ca:JobStatusResponse>
  </soap:Body>
</soap:Envelope>
```

Listing 6.5: Example SOAP message from a Compute Service. The body contains the response message from a job status request sent to the server. The header contains information about the requestor (<user>), a unique identifier (<messageID>), its relationship to the request message (<relatesTo>) and the origin of the message (<from>).

6.4 Job Management

The Computation Web Service offers resource sharing to support multiple users with task containing multiple job. An essential requirement for a multi-user environment is security that in addition to providing access control to the service, it protects users files and data from unintended damage by other users or jobs and denies other users access to each others files. We have employed WS-Security [133] for message level security that provides encryption and signing of messages, and authentication and authorisation of users; enabling user access rights and roles to be configured in the service. WS-security has scalability advantages of traditional transport security, such as SSL, as messages can be verified on a trust basis negating the need to repeatable gain authentication from the original sender for any service that it may pass through.

In addition, the service provides systematic and robust job management to cater for jobs that may run from days to months offering recovery from computer failure. The compute jobs files are assigned a unique directory that stores its files locally before submission to a compute resources. Upon completion of a job's execution files will be transferred back in this directory. This directory is assured to be unique by its path and name that are generated respectively from the owner's unique username from their X509 Certificate and Universal Unique Identifier (UUID) job identifier generated by the service. Jobs from the same user will consequently sit in the same directory hierarchy. This provides a useful way to ensure security amongst users and provides a way to manage and insulate jobs from accidently writing over each others' files.

The service provides robustness and secure caching by using the same directory structure to hold information that links the jobs submission with its directory and holds a unique cache of files for the user that only he has access to. The stored submission information enables the service to reestablish which job is associate with which directory in the event of its failure.

6.4.1 State Management

As we discussed in section 4.3.5, a service must be able to contend with interactions and internal operations spanning multiple users and jobs. A cluster managements system contains many sharing operations, in particular Job Submission and execution, that demand state management.

Pure Web Services technologies do not address state management issues nor provide protocols defining the lifetime of the span of shared operation essential to preserving the integrity of a system. In order to provide state and lifetime management, a developer may either create their own new state management mechanisms from scratch or based on existing Web Server technologies, or employ extension technologies to the stateless Web Service model. We have implemented two versions of the Computation Web Service that explores both ideas.

The first version of our Web Service employed session and application state management functionality provided by the underlying Web Server hosting environment, IIS [134] and ASP.NET [135]. This method of implementation has distinct performance and reliability advantages with support for persistent long term state storage on a SQL database server. However, it also has many disadvantage stemming from its design as a mechanism for maintaining state across Web pages and HTTP/Web security model. Importantly support for long term session state requires HTTP cookies [136] on the client-side; a known security problem. In addition, a damaged or lost cookie would mean that the user would not be able to recover their session. Whilst the issue can be overcome by mapping session state onto application state it is then not possible to use cookies' lifetime management mechanism for creation, management and destruction of sessions.

A revision of the Web Service performed its own lifetime and persistent state management mechanism by recording state information using the directory structure mechanism discussed in the last section. Upon creation of a session, for instance on a successful request for job submission, a new unique directory would be created, with its name returned to the user as its unique session handle. All subsequent operations within the context of the resource management must pass this handle back to the Web Service enabling multiple session to be maintained concurrently. Each of these operations checks the certificate of the user to ensure that they are allowed to access that session. Subsequent operations will store or check for state information, such as the job ID returned by Condor or a record of the file requirements, to control and maintain the integrity of the job submission and resource management process. The lifetime of the session in this instance represents the life of the job. Only after the job has completed and result files have been uploaded can the session be allowed to timeout and be destroyed. Care must be taken to not destroy the session after the job has been assigned a compute resource, otherwise data may be lost. Consequently, simply timeout mechanism for cleaning up sessions are not possible because the length of the jobs' execution is an unknown factor.

A completely new Web Service was create using the transient Web Service model proposed in the OGSI [57]. OGSI supports dynamic creation and lifetime management of Web Service instances that's operation and data are tied to the specific context in which it was created. We employed this mechanism, to represent the submission and management of a resource for a job as a Web Service instance. There is a one-to-one relationship between the job, resource it was assigned and the Web Service instance. Therefore operations on it only act on the job and compute resource to which they were assigned. A stateless Web Service factory is responsible for the creation of the instances. It performs all the checks previously carried out by the request of submission operation before creating and return a unique Grid Service Handlers (GSH) [57] of the new Web Service instances. This allows the implementation of the Web instance to operate under the assumption that initial context is valid. This approach simplifies the way state information is stored and managed, removing the need for arbitrary measures, such as, storage of Condor Job IDs'.

6.4.2 File Transfer Management

Jobs in a HTC environment may need to transfer huge amounts of data to and from the compute resource. Whilst this overhead is insignificant compared to the length of time the job runs, a Web Service handling multiple jobs could be swamped and effectively suffer a form of denial of service attack if too many large file transfer operations occur concurrently. Therefore, it is important that file transfer operations are as fast and as efficient as possible.

Within the Condor system this problem does not exist because file transfers are distributed across the system with each submission machine transferring a job's files directly to the compute resource. However, our Web Service must receive and transfer all jobs' files causing a potential bottleneck. To overcome this architectural deficiency, we improved the data transfer mechanism of Web Services by employing DIME [137] implementation of the WS-Attachments standard [138]. In addition, file transfer control and caching schemes were employed to make our Web Service's file transfer operation more efficient.

Jobs generally need to transfer binary formatted files, such as the compiled executable. Unfortunately, due to text markup nature of the SOAP and XML standards they do not handle binary encode data and files efficiently. The only means to transfer binary files using SOAP is too encode them into arrays of bytes (Base64 encoding) and pass that as a fragment of an XML document. Consequently, these documents are often much larger then the original file. Transfer of huge files is unreliable as SOAP documents cannot be split and must be sent in one large chunk to the Web Service. DIME offers a more efficient means of transfer with files as binary attachments to the SOAP document. DIME has recently lost favour with the Grid Community because its functionality is not sufficient for supporting all the file sharing operations envisioned by the Grid. Alternatives such as SOAP MTOM.

6.5 Development with Legacy System

Cluster management system often have native data formats, modes of operation and protocols that differ in standard to not only Grid computing technologies but, the programming and development environment for the Web Service. Abstraction of the resource representation in addition to its sharing operations is needed aid developers build services on top of legacy system in a way common to the architecture of programming environment of their chosen Web Service hosting system. Within our Web Service we have employed a resource abstraction layer that bridges the native data, communication and operational model of the cluster management system into a common object-oriented model suitable for Web Service development.

The two most important Condor programs are *condor_submit* and *condor_status*. The first provides the user with the ability to monitor and query the machine pool. Submission of jobs performed by the *condor_submit* command requires a submit-description file. Other commands provide control, monitoring and querying of submitted jobs, shown in Table 6.2.

It is possible to launch any command line executable as a process from another program. Input to the command line executable is through the standard input, read files, pipes, and process start properties. Output retrieval requires reading from the standard output, standard error, written files, and pipes. The underlying API software thus communicates using standard IO and process function calls.

The development version of Condor 6.3.x provides an SDK however; this is for non-Microsoft Windows operating system. At a later stage, there are plans to replace the Condor wrapper API with the SDK upon the release of the release version of Condor 6.4. Code wrapped around Condor by modelling each of its programs as a process, facilitates the creation of an API. It then becomes possible to make use of these APIs in the Web Service code to expose a programmatic interface to Condor on the Internet. ClassAds

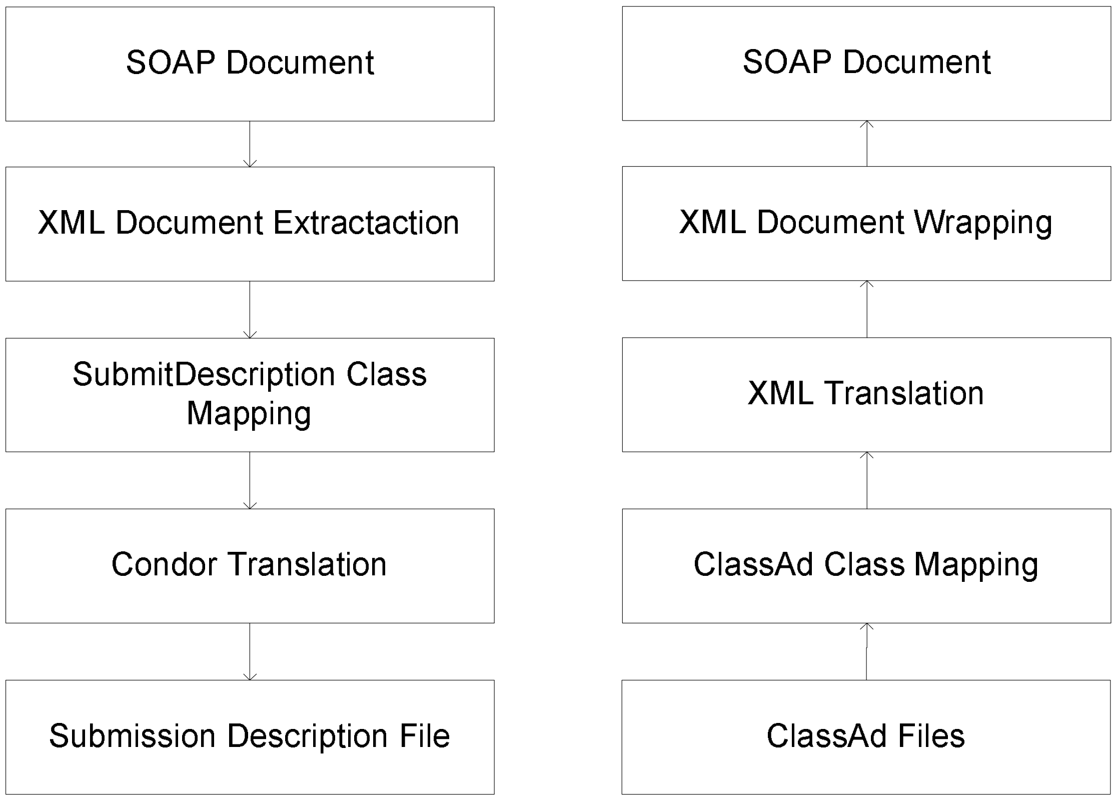


FIGURE 6.8: Condor XML Wrapping Process.

and Submission Description files (Job Requirement descriptors) are flat files, that are translatable to and from XML documents, see Figure 6.8, and have structures that are directly describable as XML Schema documents, as shown in Figure 6.5.

We have harnessed the Classads tools they have create and utilised them to create an abstract job submission request system for the Computation Web Service.

Web Services by nature transfer messages in XML format however Condor does not. So initially we constructed a set of C# class files that mapped the structures of Condor submission file. These classes where marked up with XML attributes such that they could be automatically serialised by the Web Service framework. We use these classes to contain the information about the job submission. The client constructs these classes and passes them to the Web Service via the submit method that takes these class instances and translates them to a format compatible with Condor using Classads.

6.5.1 Asynchronous Communication: Monitoring and Job Notification

After a job has terminated through either successful completion or failure, Condor notifies the owner of the job via email. While this is a useful feature and allows users to be quickly notified, Condor does not provide any other notification facilities other than polling the job queue until the job disappears.

Instead of polling, within the client's code for instance, call back routines are assigned to certain events generated by the Web Service. These events might be such things as job completion or failure, upon which the Web Service notifies the client of the event to which it performs the appropriate action, such as collecting the generated data or restarting the job with different parameters.

6.5.2 Discovery of Special Resources

Resource sharing operations extend to the Cluster Management System and the compute resources including memory, processor, disk capacities, database/archive facilities, and also specialist software environment or licensed applications. However, these specialist applications are often standalone desktop applications offering no common means for their description and discovery.

The Condor Classad mechanism system provides a convenient method for the discovery of resources with special capabilities in the Condor pool. Each machine within the Condor pool advertises its capabilities from attributes specified in its Condor system configuration file. These attributes will then be reflected in the information generated by the resource status query performed by Condor, which is passed on to the Web Service. The service will therefore be able to make the discovery by identifying the target attributes.

6.6 Review of Microsoft .NET based OGSI Implementations

The Open Grid Service Infrastructure (OGSI) defines a collection of standards for Grid service hosting environment. So far there have been a number of attempts to implement OGSI based on various software platforms. Here we presents a comprehensive review of two implementations of OGSI on the Microsoft .NET platform - OGSI.NET and MS.NETGrid. Through demonstrating the same Grid problem on both versions, we intend to reveal features presented in the implementations and highlight the differences, so as to provide first-hand experience and information for potential users, as well as suggestions for future improvement.

6.6.1 Motivation

The Open Grids Services Infrastructure (OGSI) [57] is an initiative by the Global Grid Forum. It builds upon both Grid and Web Services technologies to define standard mechanisms for creating, managing, and exchanging information among entities called Grid services.

Grid services are Web Services that follow a set of conventions which use interfaces and behaviours to describe how clients interact with these services. Combined with the OGSI mechanisms for creation and discovery of Grid Services, these conventions provide control, fault-resilience, and secure management of long-lived distributed state often required in advanced distributed applications. The OGSI specification proposes how Grid service instances are named and referenced; the base, common interfaces and behaviours that all Grid services implement; and the optional interfaces and behaviours associated with factories and service groups. However, the OGSI specification does not address how Grid services instances are created, managed, and destroyed within any particular hosting environment. While clients can invoke any Grid service instances on any platform that follows the conventions, service implementations are not necessarily portable among various hosting environments. The OGSI working group foresees that various Web Service platforms, such as J2EE [103], Microsoft .NET [79], and Windows or UNIX processes shall be used to implement the OGSI hosting environments.

Current OGSI hosting environments where created on the Java platforms, such as J2EE and Tomcat/Axis, and on operating systems, such as UNIX/Linux systems. Until now work on on implementing OGSI have not utilised the Microsoft's .NET framework. Whilst relatively new in comparison with the Java platform, it nevertheless offers comparable capabilities. In fact, the OGSA/OGSI standardisation process will benefit greatly from the creation of hosting environments on additional platforms. Therefore, there are now two attempts on OGSI implementation that harness the power of the .NET platform: the OGSI.NET project from the University of Virginia [139] and the MS.NETGrid project from EPCC [140].

Both OGSI.NET and MS.NETGrid provide platforms for the development and deployment of Grid services. Profoundly influenced by the Web Service model of Microsoft .NET, these two implementations have similar approaches to their Grid service infrastructures. However, many differences exist between them, such as container architecture, security model, and programming model. Here we presents a detailed review of the two .NET based OGSI hosting environments. In order to achieve a comprehensive understanding of how they work and what differences exist between them, we have constructed two functionally identical versions of part of an engineering design optimisation system on both platforms. The work also demonstrates that OGSI.NET and MS.NETGrid are both viable and powerful hosting environments for advanced distributed applications on the Grid.

6.6.2 Demonstration of .NET OGSI Implementations

To best leverage the capabilities of the .NET OGSI environments, a part of an existing Grid-enabled engineering design optimisation system was selected as the test scenario. Through the migration of the application to OGSI.NET and MS.NETGrid, we have

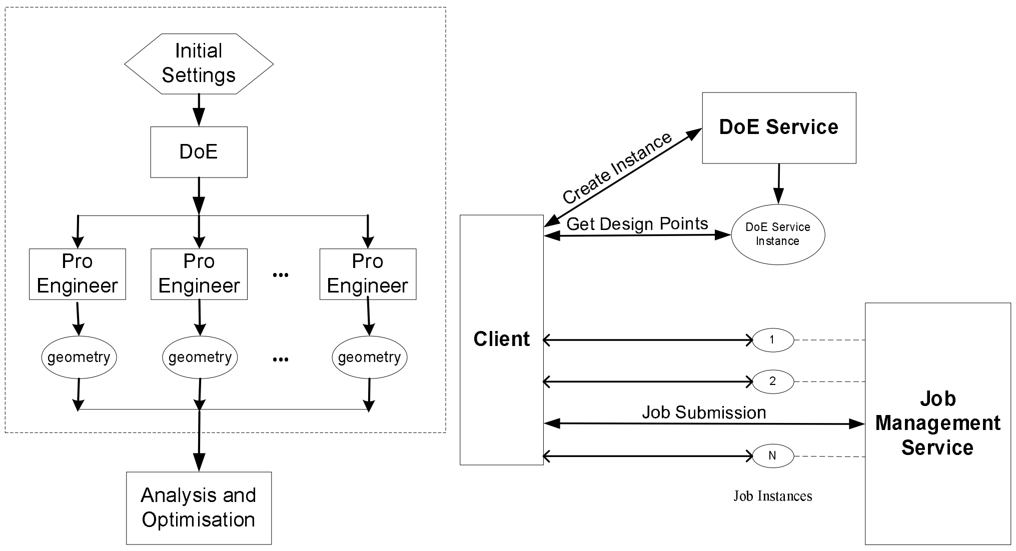
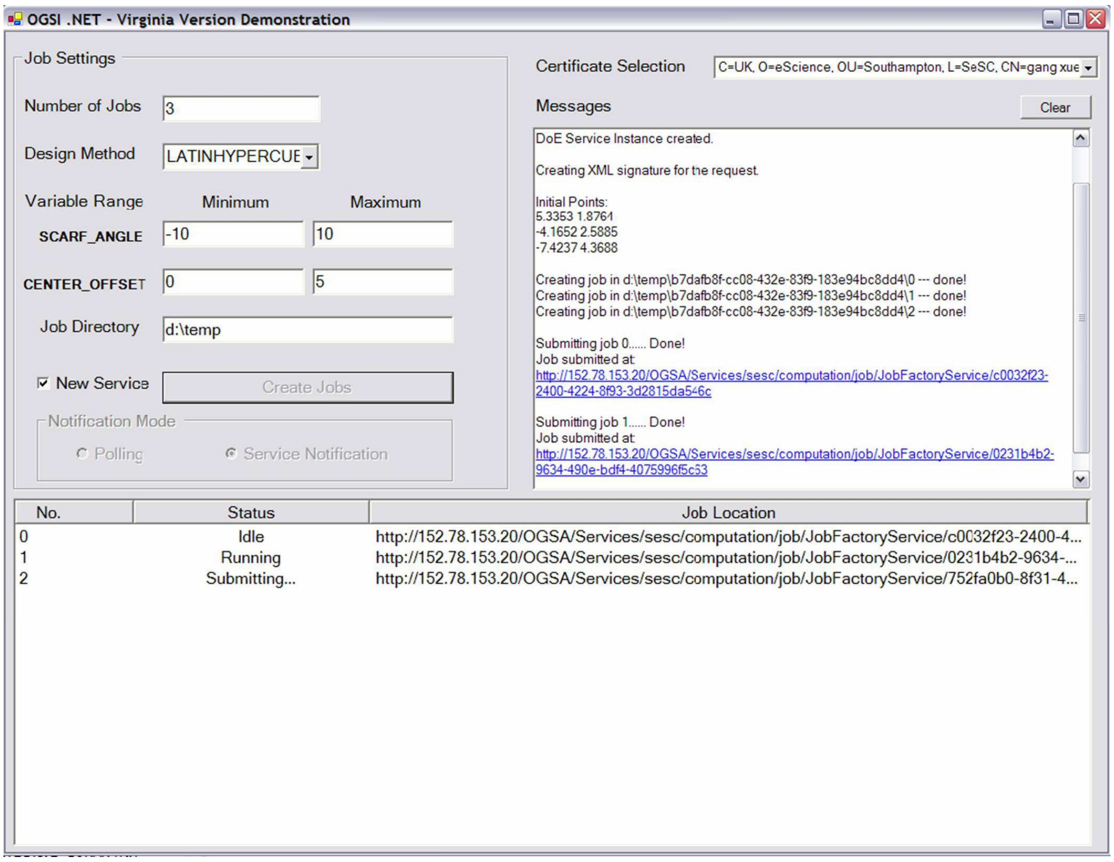


FIGURE 6.9: Engineering Design Optimisation in OGSI .NET Demonstration

been able to explore all aspects of the two platforms. The system demonstrated generates initial geometries for the optimisation of jet engine nacelle design. There are two Grid services involved in this application. The first one is a design of experiment (DoE) service, which provides starting points used for geometry generation based on the restrictions and design methods specified by the user. Being computationally intensive, geometry generation jobs are submitted to a Computation Web Service, which provides computational power and required engineering software. The process is illustrated in figure 6.9.

Both services in the demonstration need to be stateful, and are therefore suitable to exploit the OGSI transient Grid service model:

- The DoE service uses an engineering design system to create initial designs based on the requirements set by the user. As it is computationally expensive to run the design system, the service usually runs it only once, when the service instance is created. Therefore the DoE service needs to keep record of all results generated. In addition, the service also needs to keeps record of designs already retrieved by the user so that if more designs are required, no duplicate ones will be provided.
- The Computation Web Service enables execution of jobs on remote compute resources. The service provides functionalities for job submission, file transmission, and job management. As a Grid service, it utilises a factory service to create a unique transient service instance for every newly submitted job. The service instance handles the entire job submission procedure and maintains all its information, including job location, execution requirements, and job state. Grid service features such as Service Data and asynchronous notification model have also been exploited in the Computation Web Service to enable management functionalities, for example job status monitoring.



OGSI .NET - Virginia Version Demonstration

Job Settings

Number of Jobs: 3

Design Method: LATINHYPERCUE

Variable Range	Minimum	Maximum
SCARF_ANGLE	-10	10
CENTER_OFFSET	0	5

Job Directory: d:\temp

[✓ New Service]

Notification Mode: Polling [✓ Service Notification]

Certificate Selection: C=UK, O=eScience, OU=Southampton, L=SeSC, CN=gang xue

Messages

```
DoE Service Instance created.

Creating XML signature for the request.

Initial Points:
5.3353 1.8764
-4.1652 2.5885
-7.4237 4.3688

Creating job in d:\temp\b7dafb8f-cc08-432e-83f9-183e94bc8dd4\0 --- done!
Creating job in d:\temp\b7dafb8f-cc08-432e-83f9-183e94bc8dd4\1 --- done!
Creating job in d:\temp\b7dafb8f-cc08-432e-83f9-183e94bc8dd4\2 --- done!

Submitting job 0...... Done!
Job submitted at:
http://152.78.153.20/OGSA/Services/sesc/computation/job/JobFactoryService/c0032f23-
2400-4224-8f93-3d2815da546c

Submitting job 1...... Done!
Job submitted at:
http://152.78.153.20/OGSA/Services/sesc/computation/job/JobFactoryService/0231b4b2-
9634-490e-bdf4-4075996f5c63
```

No.	Status	Job Location
0	Idle	http://152.78.153.20/OGSA/Services/sesc/computation/job/JobFactoryService/c0032f23-2400-4...
1	Running	http://152.78.153.20/OGSA/Services/sesc/computation/job/JobFactoryService/0231b4b2-9634-...
2	Submitting...	http://152.78.153.20/OGSA/Services/sesc/computation/job/JobFactoryService/752fa0b0-8f31-4...

FIGURE 6.10: The OGSI .NET Demonstration Client Application

The operations demonstrated are controlled by the client application, which implements OGSI client protocols including transient services creation and management, security, and service notification. Figure 6.10 shows the client in operation.

6.6.3 Assessment of OGSI.NET and MS.NETGrid

In this section we evaluate the two .NET implementations of OGSI based on experiences from building the demonstration application. The assessment is done on the following aspects: specification compliance, platform integration, usability, and performance.

6.6.3.1 OGSI Compliance

The OGSI specification defines a set of Grid service port types that provides essential Grid service features. The following table compares implementations of these port types from OGSI.NET and MS.NETGrid in order to study their compliance with the OGSI specification.

As shown in table 6.4, OGSI.NET provides a complete implementation of the OGSI specification. While, MS.NETGrid only implements the *GridService* port type (although

OGSI Port Types	OGSI.NET	MS.NETGrid
GridService	Implemented	Implemented
Handle-Resolver	Implemented	Not implemented
Notification-Source	Implemented	Not implemented
Notification-Subscription	Implemented	Not implemented
Notification-Sink	Implemented	Not implemented
Factory	Implemented	Implemented
ServiceGroup/ ServiceGroup-Registration/ ServiceGroup-Entry	Implemented	Not implemented

Table 6.4: Comparison of OGSI Implementation - Port Types

OGSI Compliance	OGSI.NET	MS.NETGrid
Grid Service Description	Some pre-defined Grid service description elements included in the WSDL and XML files provided.	Pure WSDL 1.1 without service data and service data description.
Service Naming	Arbitrary service naming supported	Arbitrary service naming not supported
Unspecified Features	.NET remoting used in RPC styled service invocation.	None

Table 6.5: Comparison of OGSI Implementation - Other Issues

some *ServiceData* and operation message XML does not match the specification exactly) and *FactoryService* port types. Apart from the port type implementation, the two versions of .NET Grid service environment also differ in several issues listed in table 6.5.

6.6.3.2 .NET Integration

OGSI.NET and MS.NETGrid both build their system based on the MS .NET platform. However, due to the differences in design and implementation, they do not have the same level of .NET integration. Here the two versions are compared on a number of subjects concerning the .NET platform.

6.6.3.3 .NET Framework and Programming Language

The two versions are both implemented with MS Visual C#. Due to the multiple language support of .NET, it is possible to write Grid services using other languages supported by .NET, such as VB.NET, C++ and J#. Nevertheless, it should be noticed that OGSI.NET is built on .NET framework 1.1 and therefore not compatible with the older version of the framework. Although, the differences between version 1.0 and 1.1 of the .NET framework that may effect Grid service development are concerned with changes to the .NET security and ASP.NET security (affects only MS.NETGrid).

.NET Integration	OGSI.NET	MS.NETGrid
.NET Version Supported	MS .NET Framework 1.1	MS .NET Framework 1.0, 1.1
Implementing Language	C#	C#
ASP .NET Application Server	Container process does not run within the ASP.NET Web process.	Used as the environment for the Grid service container
Web Service Enhancements Features	Plug-ins supporting features such as DIME and WS-Routing provided by the package cannot be used in Grid services.	WSE features can be fully exploited.
System Security Restriction	Unlimited – GS container running as normal Windows service process.	Limited – GS container running in an application domain of the ASP.NET process
Container Security	Each GS instances run in a separate application domain, which provides isolation, unloading, and security boundaries for executing managed code.	No explicit security mechanism. All GS instances run within the same application domain.
Tools Support	Visual Studio .NET 2003 required; Automatic generation of proxy classes using “Add Grid Reference” plug in, Automatic generation of WSDL from Grid Service library via WSDLGenerator tool, GWSDL flattener for conversion of GWSDL to WSDL documents, XML and WSDL document handling programming libraries.	Visual Studio .NET 2002/2003
System Requirements	Windows 2000/XP/Server 2003; .NET framework 1.1; IIS 5.0.	Windows 2000/XP/Server 2003; .NET framework 1.0 or 1.1; IIS 5.0.

Table 6.6: Comparison of .NET Integration

The .NET framework supports backwards compatibility, such that the majority of code written for version 1.0 should work with 1.1. It is important to note that because the .NET framework also supports forwards compatibility, as newer versions become available developers will not be restricted to using the version of .NET utilised by the relative Grid containers, although care must be taken such that the application runs as expected on differing versions of the .NET Framework.

6.6.3.4 Application of ASP.NET and Web Service Enhancement

MS.NETGrid runs its Grid service container as an ASP.NET Web Application under the Microsoft Internet Information Server (IIS), and is therefore able to take advantage of ASP.NET features for Web service deployment and management. In addition, the Web Service Enhancement package that supports upcoming open standards for Web services such as WS-Security, WS-Routing and WS-Attachments can be used directly within its Grid service programming model without changes to the Grid service container. For OGSI.NET, on the other hand, the Grid service container runs as a stand alone Windows service process that provides similar service hosting functionalities similar to ASP.NET.

	OGSI.NET	**MS.NETGrid**
Factory	Various types of transient GS may utilise the same type of factory service.	For each Grid service, there must be a unique persistent factory service deployed.
Grid Service Configuration	XML based configuration format.	Standard ASP.NET Web configuration
Grid Service Deployment	All service library files are stored in the same folder; services need to be specified in the configuration; separate WSDL documents are required for each Grid service which must be placed in schema directory of the IIS root folder for automatic proxy generation; restart of the container process is required.	All service library files are stored in the same folder; services need to be specified in the configuration; the WSDL files do not exist in the file system; restart of IIS is required.
Grid Service Programming	No separation between service interface and service implementation logic.	Service interface and implementation are separated into the service proxy (interface) and service implementation (application logics).
Pre-defined and custom OGSI Port Types	Ports types are reusable among services and custom port types may be created. OGSI port types are implemented in separate classes and are linked into the Grid service classes through .NET attributes.	OGSI port types are implemented in the base GS proxy classes, which are inherited by the proxy class of each Grid service.
SDE definition and handling	SDEs are defined and configured using attributes. Class level SDEs may have custom handlers defined by marking up static 'set' and 'get' methods using attributes. Instance level SDEs handlers are assigned programmatically, where each SDE may be assigned multiple 'set' call backs and a single 'get' call back handlers.	SDEs are defined and configured programmatically. Instance level SDEs handlers are assigned programmatically, where each SDE may be assigned multiple 'set' call backs and a single 'get' call back handlers.

TABLE 6.7: Comparison of Grid Service Deployment and Programming Model

	OGSI.NET	**MS.NETGrid**
Resource Management	The container process is always running. Persistent services are only loaded fully into an application domain when they are invoked and stay in memory until the container process is stopped. Transient service instances are removed when they are destroyed or become invalid.	The container process is not loaded until the first GS is invoked. All persistent services are loaded when the container initialises and stay in memory until the ASP.NET web process is stopped. Transient service instances are marked for removal when they are destroyed or become invalid. However, there is no clean-up mechanism implemented.
Resource Usage	High and less scalable.	Moderate and scalable.

TABLE 6.8: Comparison of Grid Service Resource Management

It is, however, not obvious how new features supplied by platform and development tool vendors can be deployed without updating the container itself.

6.6.4 Grid Service Deployment and Programming

The most important aspect of an OGSI implementation is its support for the construction and deployment of Grid services. The two versions of .NET Grid service environment have both provided an effective service deployment and programming model. While similar in many ways, there remain some differences, which are listed in table 6.7.

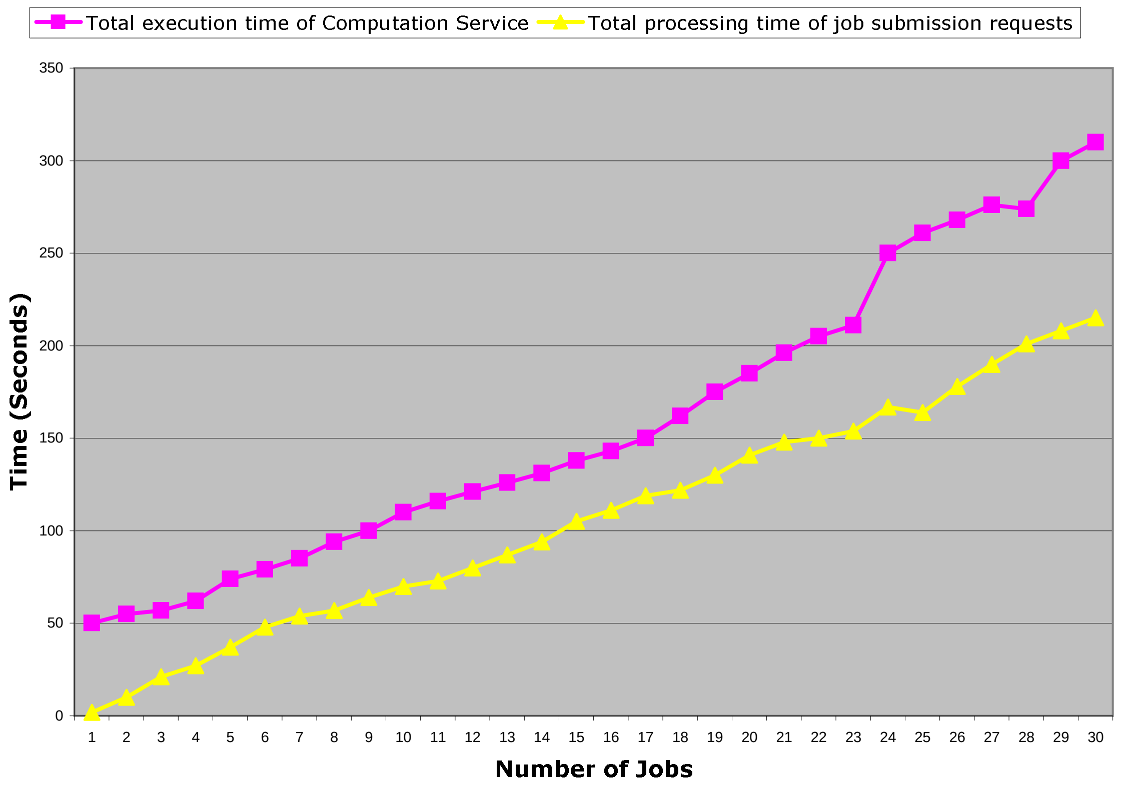


FIGURE 6.11: Scalability of Web Service submission node

6.7 Practical Application of Computation Web Service

Here we give a demonstration of the Computation Web Service for a parameter study of fluid dynamics in a 2-dimensional geometry of a nacelle for an aircraft engine, shown in 6.12. Four design variables that describe curves on the upper and lower edge of the nacelle are calculated across its surface. Parameter studies such as this enable engineers to more easily determine the impact of different wing geometries upon the quality of a design. Each simulation of the parameter study runs as job submitted to the Web Service from design environment in Matlab using a Java based client toolkit.

The job's files exceeds ten megabytes in size and include the executable for the simulation compiled from Matlab code, and supporting Matlab libraries for it to run. The cache mechanism of the Web Service reduces the network load and number of file transfers by only uploading new or changed files. After the initial job's files were uploaded subsequent jobs submission were significantly faster. A total number of six hundred and forty eight simulations jobs were successfully run over a cluster of twelve Windows NT nodes. The success of this shows that the Web Service is both robust and reliable, and provides seamless access to the Condor system from environments such as Matlab.

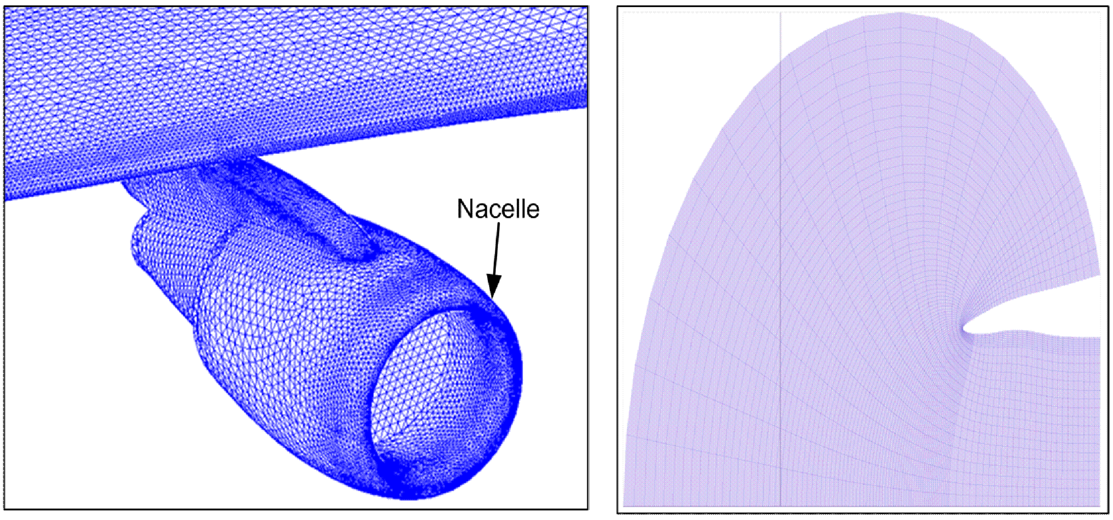


FIGURE 6.12: Visualised Result of a Four Dimensional CFD Parameter Studies

6.8 Summary

The efforts to implement OGSI on the Microsoft .Net Framework are examples that emphasise the importance of Web Service standards to aid interoperability. The ability to use a wider variety of platforms and hosting environments gives users and developers greater access to more software tools and resources. The Computation Web Service was successfully ported to the OGSI platform and was able to take advantage of the it's message events and lifetime management capabilities. Nonetheless, due to the success of its predecessor Globus, it is unlikely that OGSI platform will be widely supported because it will require significant software redevelopment to port existing Globus-based applications yet offering little added benefits. It provides a welcomed alternative to existing Web Service deployment platforms, however, it is unlikely that it will receive wide support in this scenario either, because platforms, such as Apache/Axis and ASP.Net, offers greater flexibility and do not constrain developers to conform to the OGSA architecture.

The Computation Web Service intends to provide a complete virtualisation of the computation resources that are essential to solving engineering design optimisation problems. The virtualisation allows the service to be generic to both the client environment and the target computation environment. This chapter introduces the design of the computation service that supports all operations throughout the entire job submission process. We have shown how to wrap a legacy service using Web Service technologies and demonstrated how it provides interoperability and integration with other components of a Grid-enabled PSE. In addition, within the Computation Web Service we have successfully implemented the strategies and approaches discussed in the previous chapter 4.

Chapter 7

Micromagnetics Problem Solving Environment

This chapter describes the work carried out developing a micromagnetic Web Service-based Problem Solving Environment (PSE), published in [141]. We explore the idea of orchestrating "Lightweight" message-based services intermediaries as transparent resources in the modelling, simulation and analysis micromagnetic problem solving process. In addition, we examine the viability of the TCP [142] transport protocol as a means to enable lightweight services that require no Web Server hosting environment.

Using a simple workflow protocol based on scripting principles, we show the power of Lightweight services to be quickly and effectively orchestrated into sophisticated distributed system; enabling transparent access to computational resources, seamless integration of legacy tools and applications, and knowledge capture and reuse. In addition, we show how the WS-Addressing [143] technology can be employed to implement workflows. As proof of concept, in section 7.6 we give the successful results [144] produced by our Grid-enable micromagnetic PSE.

The micromagnetic PSE demonstrates the power of the Computation Web Service, see chapter 6, using it as gateway through organisational network boundaries; specifically, firewalled subnets, enabling access to a large Condor cluster[1].

7.1 Motivation

Currently Web Service technologies are generally perceived and applied as RPC-style distributed objects, with interactions between services achieved through WSDL defined application programmer interfaces (APIs) and Web methods calls. Distributed object

[1]Which was the University of Southampton's E-Science test Condor cluster that contains nearly 1000 shared undergraduate PCs and dedicated computational machines situated across its campuses.

technology is an extension to the object-orientated design principles first developed for application programming. Whilst software engineers may have familiarity with this form of programming, they must realise that large-scale distributed systems cannot realistically be developed in the same manner as desktop applications. Distributed object technologies whilst, suitable for closed environments and controllable small-scale distributed systems, RPC-style architectures cannot feasible be applied to the Grid that inevitable will have to contend with many unknown factors and anarchical relationships amongst participants and resources. Therefore, it is vital that Grid system developers move away from the so called first generation RPC-style Web Services and start harnessing their message-passing and intermediary message exchange pattern capabilities.

Difficulties exist with the employment of HTTP as a transport protocol for Web Method and message style interactions between services. Its request-response mechanism and original target application, the transfer of small hypertext documents, affects HTTP's ability to effectively provide desired communication characteristics for services, in particular extreme high speeds, low latencies, asynchronous calls or long term connections. Another issue with its effective employment relates to its requirement for heavyweight HTTP capable Web Service hosting environments, such as ASP.NET and Tomcat. These provide necessary XML message processing, security controls and manageability necessary for handling of sophisticated and dynamic Web Service operation. However, whilst a suitable environment for services, it prevents lightweight clients from receiving asynchronous operations, such as notification. This is exemplified by the impossibility of using the OGSI notification system with desktop client applications that employ lightweight communication means, such as WSDL proxies.

7.1.1 SOAP Intermediaries and Message Exchange Patterns

Web Services employ the Simple Object Application Protocol (SOAP) for the addressing, description and encapsulation of XML documents, see section 2.5.4. SOAP can be thought of as a postal envelope used to carry a text-based payloads between Web Services. Its protocol defines a header, representing the outside of the envelope, that may have on it, amongst other things, an address and a body, representing the inside of the envelope, for carry any valid XML formatted data. SOAP's simple structure is extremely flexible and extensible, with well defined semantics for extension protocols, such as WS-Security [7] that enables the sending of signed and encrypted messages using SOAP.

Unfortunately, SOAP capabilities are currently under utilised. The majority of Grid system that employ it with Web Service use SOAP's HTTP transport support to receive and send Web Methods calls. Something that HTTP's post command would be able to carry out on it own. This superficial usage of SOAP negates its purpose and gives the impression that SOAP is an unnecessary overhead.

The true power of SOAP stems from its support for intermediaries [47] and their intended usage for complex message exchange patterns (MEPs). An intermediary is defined as "a SOAP receiver and a SOAP sender and is targetable from within a SOAP message. It processes the SOAP header blocks targeted at it and acts to forward a SOAP message towards an ultimate SOAP receiver" [47]. The ultimate receiver being the intended destination for the carried message. Intermediaries are derived from the concept of distributed message processing, where a message passes from sender to destination through a number distributed message processors or "intermediaries".

The message's defined path through the intermediaries is called the message exchange pattern (MEP) [47], shown in figure 7.1. How or what defines or implements the MEP is not specified in the SOAP specification, giving great flexibility but, opening up the possible emergence of different incompatible mechanisms. The SOAP distributed processing model can support many MEPs including but not limited to one-way messages, request/response interactions, and peer-to-peer conversations.

SOAP intermediaries perform processing on messages that they receive. Each type of intermediary has a specific role that must be defined according to the SOAP specification. Contained within the header of the SOAP envelope will be instructions that may or may not be targeted at a specific role. Intermediaries must process the headers and act upon the instructions that are meant for its role, removing it before passing the SOAP message on to the next intermediary.

SOAP intermediaries are either active or passive; with the former performing actions on the XML message contained in the SOAP body. Passive intermediaries only process the header information.

7.1.2 Lightweight Web Services

We define Lightweight Services as message-based distributed services or application components with common coarse-grained interfaces that only use the TCP protocol for transmission of SOAP messages and perform all the XML-message processing task themselves. By dropping HTTP we gain two things: first there is no perquisite for heavyweight Web Server hosting environment, such as ASP.NET or Tomcat, and second we gain the freedom from the performance, timeout and request/response issues of HTTP; enabling one-way transmission of SOAP messages.

By employing messages as the basis of communication and enforcing a simple common interface we ease the integration of lightweight services and make it far easier to formulate a workflow mechanism. This is similar to concept of a REST-style of architecture except that we allow intermediaries so the representation (e.g. the data values and structure of the XML message) may not be the sender's but the sum of the possible changes made by the intermediaries from the sender to ultimate receiver.

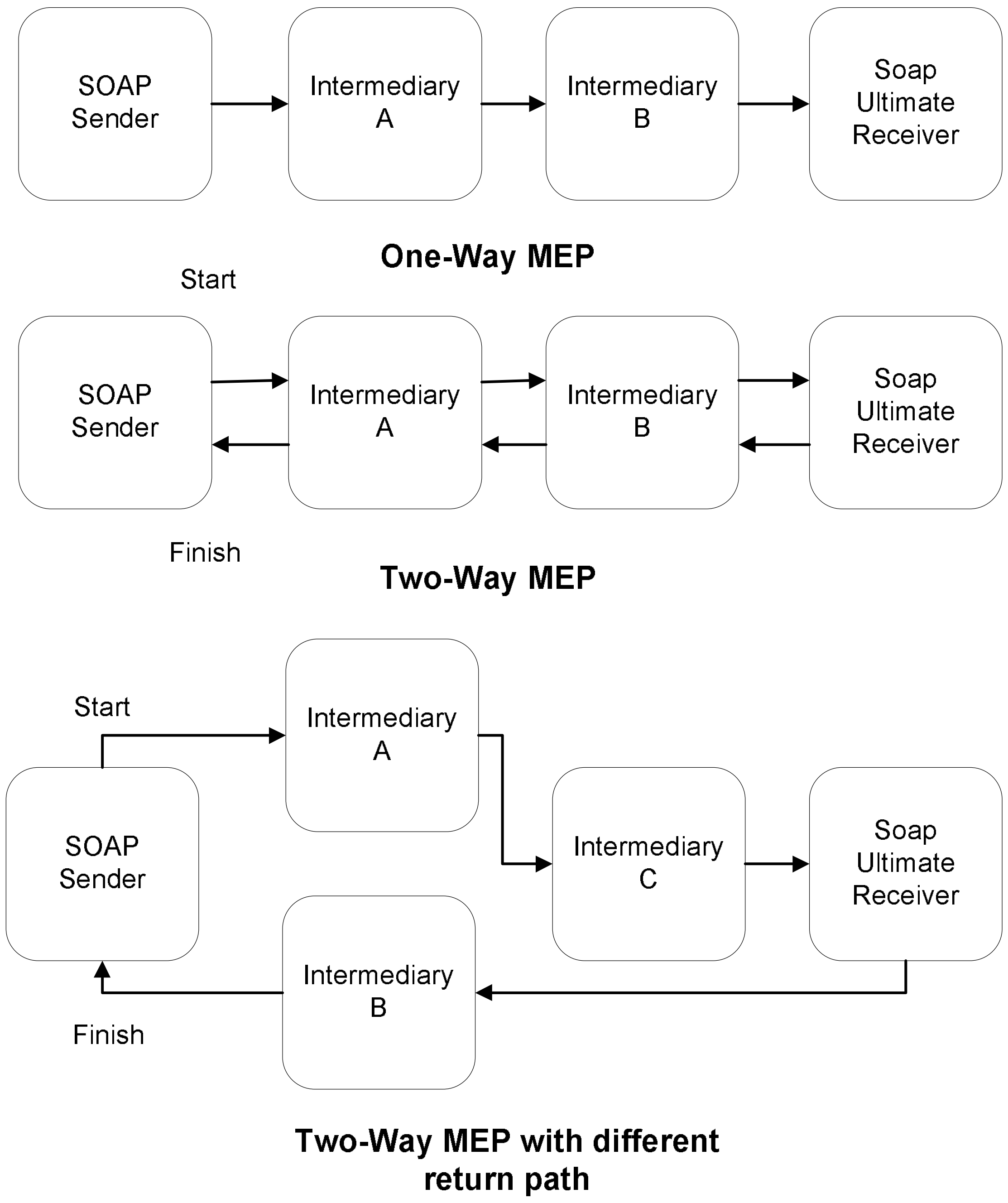


FIGURE 7.1: Example showing SOAP messages travelling in one-way and two-way message exchange patterns for SOAP sender and receiver, such as a Web Services, through intermediaries.

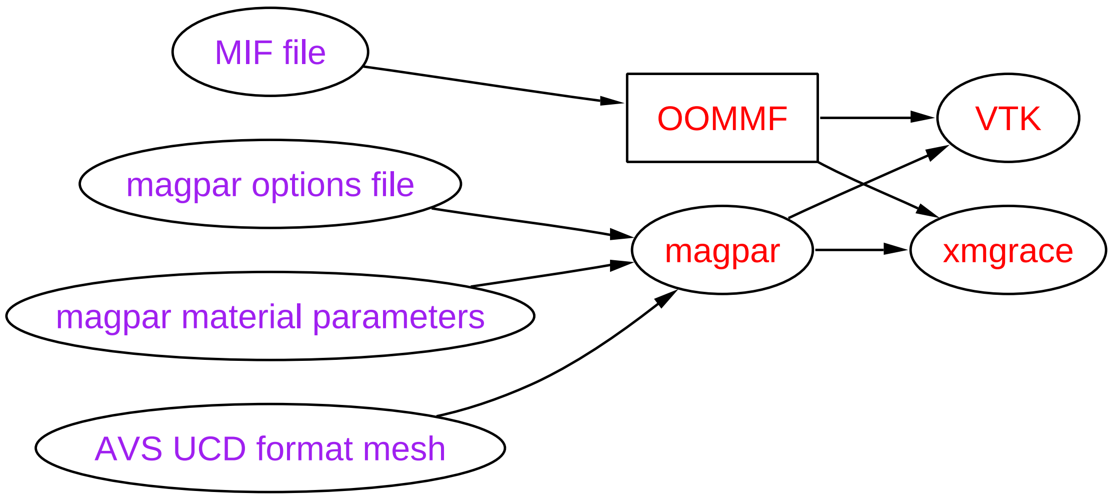


FIGURE 7.2: Shows an overview of the finite difference micromagnetic simulation process.

7.2 Analysis of the Micromagnetic Problem Solving Process

Effective research and resolution of micromagnetics problems requires the employment of applications such as, OOMMF and Magpar, for its simulation, modelling and visualisation respectively. In section 3.2 we gave an outline of a micromagnetic problem solving process, highlighting the application collaboration challenges faced by scientists. We will now look at a specific problem solving process in micromagnetics, Finite Difference method simulation, examining the typical application operational procedures and their interactions.

7.2.1 Finite Difference based Simulation

Scientists often employ the OOMMF numerical modelling package to perform Finite Difference (FD) based simulations. Effective usage of the OOMMF package requires pre- and post-processing of data flowing in and out of the application, see figure 7.2.

Preprocessing is primarily concerned with initialisation of the simulation. OOMMF requires parameters about the target systems such as, its geometry, material properties and simulation constraints. These may come from sources such as, the scientist themself, a database or a file, that must be passed to OOMMF in the form of a micromagnetic Input File (MIF), shown in listing 7.1.

The MIF format provides a powerful yet time consuming means to configure and initialise OOMMF that allows scientists fine-grained control over the behaviour and operation of sophisticated and long running simulations. The MIF format may only contain information for a single micromagnetic system requiring a new MIF file for each unique system.

```
4*atan(1.0)] set mu0 [expr 4*$pi*1e-7]

set TIMEDRIVER 0

Specify Oxs_BoxAtlas:atlas {
  xrange { 0 5E-08 }
  yrange { 0 5E-08 }
  zrange { 0 8E-08 }
}

Specify Oxs_RectangularMesh:mesh {
  cellsize { 5e-009 5e-009 5e-009 }
  atlas :atlas
}

Specify Oxs_UniformExchange {
  A    1.3e-011
}

Specify Oxs_UZeeman "
  multiplier [expr 0.001/$mu0]
  Hrange {
    { 500.0 0.0 0 -500.0 0.0 0 500 }
  }
"

Specify Oxs_Demag {}

Specify Oxs_EulerEvolve {
  alpha 0.5
  start_dm 0.01
}

Specify Oxs_TimeDriver {
  basename 5E-08-8E-08.mif_0008
  evolver Oxs_EulerEvolve
  stopping_dm_dt 0.01
  mesh :mesh
  stage_count 0
  stage_iteration_limit 0
  total_iteration_limit 0
  Ms { Oxs_ScriptScalarField {
    atlas :atlas
    script { Cone 795774.715459 }
   }
  }
  m0 { Oxs_UniformVectorField {
    norm 1
    vector { 1.0 0.0 0}
  }
 }
}

proc Cone { Ms x_in y_in z_in} {
    set x [expr 2.*$x_in - 1.]
    set y [expr 2.*$y_in - 1.]
    set z [expr 2.*$z_in - 1.]
    set left_side [expr ($x*$x) + ($y*$y)]
    set right_side [expr (($z+1)/2) * (($z+1)/2)]
    if { $left_side <= $right_side } {
         return $Ms
    }
    return 0
}

Destination archive mmArchive

Schedule    DataTable archive Stage 1 Schedule
Oxs_TimeDriver::Magnetization archive Stage 1
```

LISTING 7.1: MIF configuration file for a Cone OOMMF simulation

Effective handling of large numbers of micromagnetic simulations requires tools such as, Mifmaker, that automate the production of MIF files.

Mifmaker is a sophisticated Python script written by Richard Boardman whilst studying micromagnetics at the University of Southampton [94]. It is a proprietary tool that aids the user piece together their collected knowledge of micromagnetics whilst easing the task of generating the complex MIF format OOMMF input files. It works by taking a set of command line arguments pertaining to the parameters of the micromagnetic system. From these it generates a MIF file by validating the arguments values against the target system and creates default entries and values for any unspecified system parameters and OOMMF configuration. To do this, Mifmaker contains a database containing the magnetic properties of a small set of materials. Some of the data is the intellectual property of third parties and was lent to the author for the purpose of simulation for these persons. For security reasons and the highly focused target purpose of the tool, sharing and integration of Mifmaker is difficult. For instance, Mifmaker offers functionality that would benefit Magpar however its has incompatible operation and output format. Mifmaker would require a complete rewrite to fit in with Magpar and its additional preprocessing stages.

Visualisation of output data is essential for the effective analysis of the simulations. OOMMF returns results of simulations in a bespoke text and binary format that must be converted to a suitable for input into visualisation packages such as VTK.

7.3 Micromagnetic Message-based Web Service PSE

We encapsulate the pre- and post-processing stages of the micromagnetic using Lightweight Web Services that provide common behaviour, operation and data representation. The Web Services perform the translation to and from the applications' native data representation.

7.3.1 Coarse Grained Message-Based Web Services

The micromagnetic solving process flows data along a linear path passing output from the micromagnetic applications to the input of the next (e.g. from Materials database to Mifmaker to OOMMF). Operation of all applications in the micromagnetic process are limited to querying, input of data, execution and results retrieval.

Each Web Service offers operations pertaining to the applications functionality. However, where these do not fit in with common operational or behavioural characteristics the Web Service provides this while reusing as much existing logic as possible. This is achieved by either spreading or combining application functionality across or within Web Method operations respectively.

7.3.2 Intermediaries

Intermediaries provide the glue for binding Web Services together. We employ them in the micromagnetic PSE to perform the pre- and postprocessing of data for input and output of the OOMMF Web Service. The intermediaries are lightweight Web Services receiving, processing and forwarding SOAP messages.

7.3.3 Message Processing Chain

The Portal is the point of entry to the system and represents the start point for sending SOAP messages, see figure 7.3. The user of the system is presented with a browser interface to the Portal that contains a step by step guide that helps the user through the parameter selection for the micromagnetic simulation. Once the user has selected all the parameters he wants the browser posts this information to the Portal Web Service that generates a SOAP message containing this information.

The materials intermediary is the first step of the micromagnetics process. It takes as input a SOAP message containing the material name and its desired attributes along with other information sent by the micromagnetic PSE portal. The materials intermediary processes the message, removes the part of the SOAP message aimed at it, looks up requested information in its database and adds to the SOAP message the attribute values. It then sends this to the Mifmaker intermediary which processes the SOAP message performing the same process again. It extracts the message from the materials intermediary and parameters for generating the MIF file. It then sends the SOAP message to the OOMMF intermediary along with generated MIF files as attachments. The OOMMF intermediary file extracts the MIF files and generates the submission request and sends it along with MIF files to the Computation Web Service. The OOMMF service then sends a new SOAP message to the portal telling it that the job has started.

The user of the portal may then through the browser interface monitor and control the resource to which the OOMMF job was assigned. When the OOMMF intermediary receives messages from the Computation Web Service about change of status of the job it relays these to the portal which updates the browser with the information.

Once the job has completed the OOMMF intermediary downloads the results files from the Computation Web Service, informs the portal of the completion of the simulation, and forwards the results files as attachments to the visualisation intermediary. The visualisation intermediary translates the results files into a format specified by the portal user, sending this off to the selected Graphing intermediary that generates a rough plot of the data and returns this to the portal. The user can then decide if the plotted data looks ok or not and sends back possible instructions to the Gnuplot with modified visualisation parameters. The cycle repeats until the user is happy with the graph upon

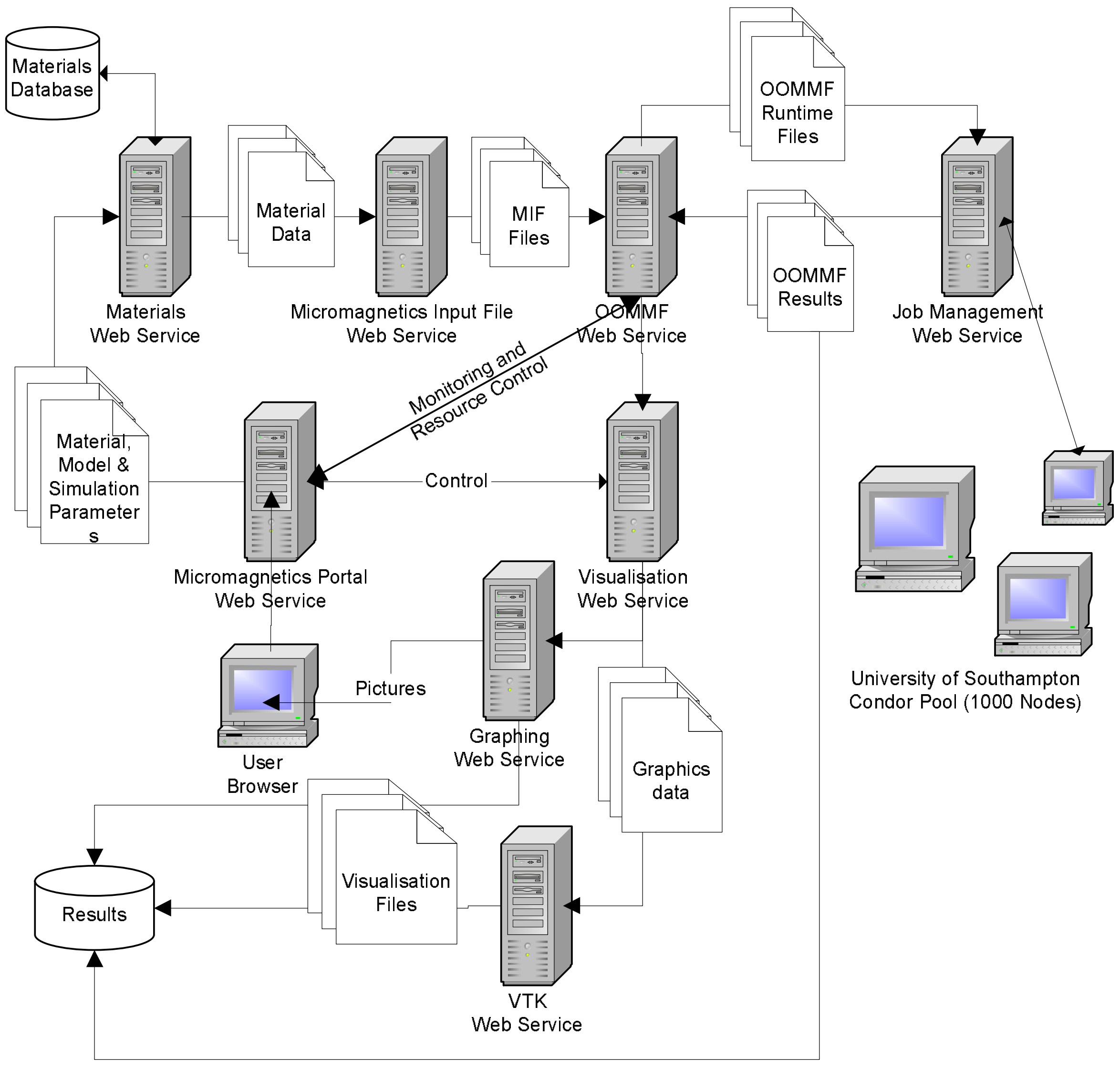

Materials Database
OOMMF Runtime Files
Material Data
MIF Files
OOMMF Results
Materials Web Service
Micromagnetics Input File Web Service
OOMMF Web Service
Job Management Web Service
Monitoring and Resource Control
Material, Model & Simulation Parameters
Control
Micromagnetics Portal Web Service
Visualisation Web Service
Graphing Web Service
Pictures
User Browser
Graphics data
University of Southampton Condor Pool (1000 Nodes)
Results
Visualisation Files
VTK Web Service

Figure 7.3: Message exchange pattern for the micromagnetics PSE

which the portal sends a command to the Graphing Tool to send the results off for full rendering on the VTK intermediary. Upon completion of rending all the results are sent to a location specified by the user in the initial stages, completing the process of modelling, simulation and analysis.

7.4 Knowledge Capture and Reuse

The micromagnetic PSE captures and reuses historical data to aid in the selection of computational resource with appropriate capabilities and provides estimates of calculation runtime for the detection of malfunctioning or endless jobs. In addition, this data helps scientists to gain a feeling for the reasonable size and complexity of tasks that may be run on available computational resources.

The system we have developed captures information from important stages of the micromagnetic solving process; collating and organising it by job into a database. This information is employed by micromagnetic PSE to suggest likely computational requirements to the user such as, minimum memory and disk size, for particular micromagnetic simulation, geometric and material parameter values. Upon user resolution of these values the PSE offers estimated runtimes for each job.

7.4.1 Collected Information and Historical Records

The micromagnetic PSE collects information from key stages of the solving process relating to the parameterisations of simulation, modelling and material, resource requirement specification and job runtime data. The historical Web Service contains unique records for each job, built on the fly as it receives input from the other Web Services.

A record is of the form of a simply XML data structure that may contain any number or type of information in XML format. The record has a unique identifier that relates it to the job; generated from the unique message identifier of the initial parametrisation message sent by the user.

Messages posted to the historical Web Service from a particular job pass the same initial message identifier along with their specific historical information. This may include the WS-Addressing headers of the intended messages that the historical information was derived from. The benefit of this means of knowledge capture and collation is that a Historical Web Service needs no knowledge of the micromagnetic process, Web Services or data types. All information relating to the origin, path and order of data produced by the Micromagnetic process is encapsulated in the message identifiers and the WS-Addressing headers within the records. Figure 7.4 shows a graph of the collected data about disk usage collected from the modelling of ferromagnetic cones, see section 7.6.

7.5 Computation Web Service in a Message-based Web Service PSE

Whilst the Computation Web Service was a good fit within the Geodise system, it posed several challenges within the micromagnetic system which contains services with message passing interfaces. Two important points of contention were established: how do we combine services with fundamentally different architectural styles (RPC and Message Passing) and how does the system handle application state fit within a primarily stateless REST-style architecture? Our solution was to create a bridge, the OOMMF intermediary, between the two architectural styles of services that performed the translation and state management between the messaging passing services and the Computation

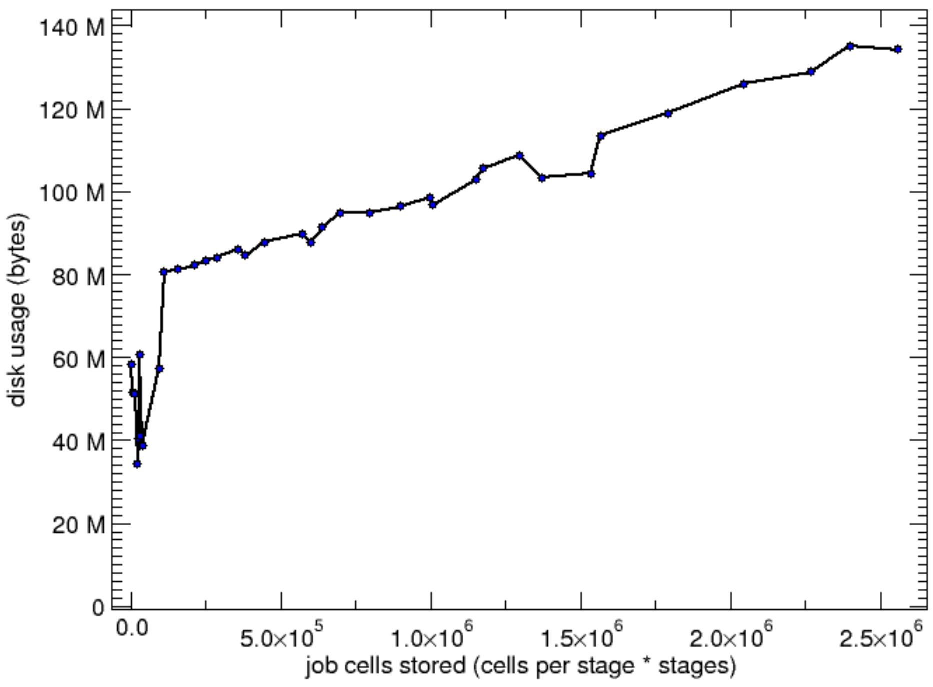


FIGURE 7.4: Graph showing a linear increase in compute node disk usage as the number of cells stored by micromagnetic simulation jobs rises. This is an example of the useful collected information by the PSE that could be used to precalculate the approximate amount of disk space required by a job.

Web Service. However, we believe this solution whilst effective does not fully address the issues of loosely coupling of stateful services nor does it offer a realistic solution for their seamless integration. Proposed further work would be to adapt a stateful RPC-style Web Service, such as the Computation Web Service, to a message based scheme to uncover the solution methodologies that would enable its seamless integration into the workflow of a Grid system.

7.6 Practical Application

The micromagnetic PSE has been employed in a parameter study of ferromagnetic cones that varied the height and diameter geometry of soft conical particles, shown visualised in Figures 7.5, 7.7 and 7.6.

Three studies were carried out with four, fifty and one hundred simulations jobs respectively. Each study represented a unique session of the portal. The user was able to successfully specify the parameters for each study using a few mouse clicks in a browser. The portal then generated the SOAP messages for all the jobs, linking them to the

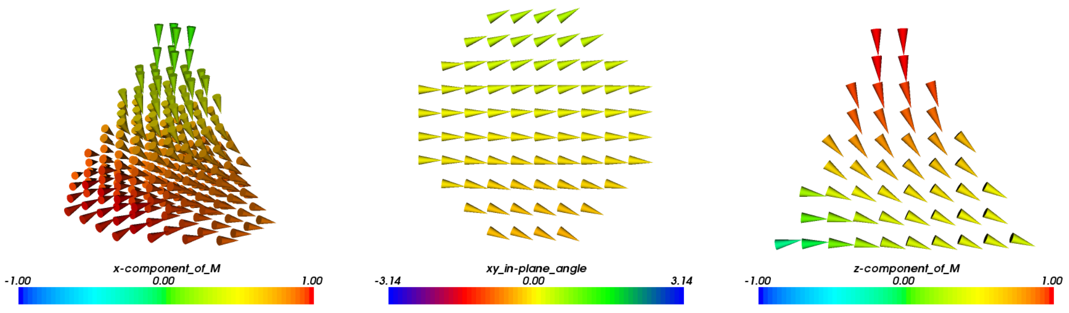


FIGURE 7.5: Cutplane ray traced visualisation of the systems magnetisation at zero applied field. Left most diagram shows that the system has a overall direction of magnetisation. Middle diagram is a cut plane in the y (up) and x (right) dimensions and the right diagram is a cutplane in the z (up) and x (right) dimensions.

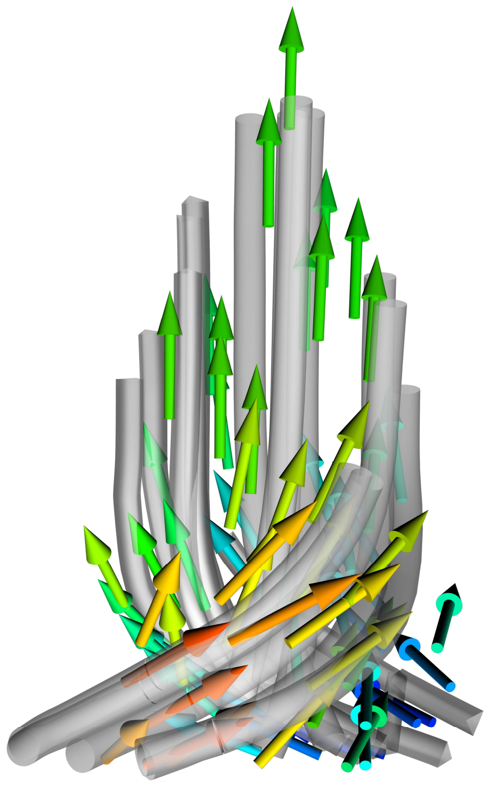


FIGURE 7.6: Large cone side ray traced visualisation of cones vortex core magnetisation at zero applied field. Tubes in visualisation shows the paths around the direction of magnetisation.

session and task via their unique MessageIDs. All jobs were successfully created on the Condor cluster and the user was able to monitor and control the progress of the tasks concurrently. All results files were successfully return to user which were used to generate the figures in this section.

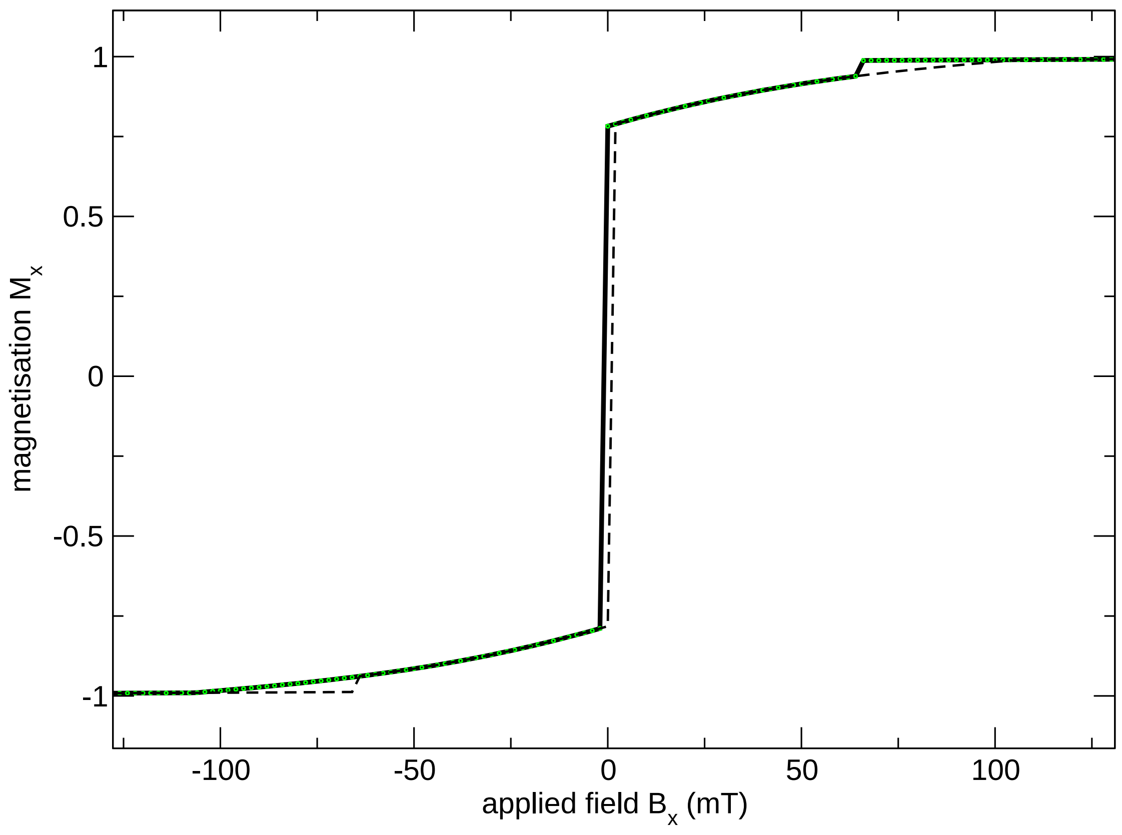


FIGURE 7.7: Graph demonstrates response of magnetic system to external magnetic field. The dashed black line shows the effect on the magnetisation of the particles as the applied field changes from negative to positive. The solid green and black line shows the effect on the magnetisation of the particles as the applied field is changed from positive to negative.

7.7 Summary

The Micromagnetics Grid-enabled PSE adopts an extremely loosely-coupled style to enable a level of interaction amongst resources in a way not previously seen in Grid computing. It demonstrates sophisticated workflow execution possibilities and seamlessly integrates computational and application services using a document-orientated message exchange paradigm through exploitation of the capabilities of WS-Addressing and SOAP message exchange patterns. That in addition allow the implementation of complex workflows and out-of-the-box knowledge capture and reuse. The design and approach of the Grid-enabled system is exemplified by the successful integration of micromagnetics tools with the Condor system, and the production of real-world results that otherwise would have been too difficult to achieve.

For the user, it offers a PSE that automates and removes much of the error-prone and repetitive computer science tasks scientists should not have to worry about. Access to resources is give in a uniform way using open Web Service standards that overcomes the difficulties users normally face when trying to integrate a plethora of heterogenous software and hardware resources. The user is instead presented with a set of distributed resources that appear as a unified collection of tools designed to work with each other. In addition, the added interoperability eases the ability for the user to construct sophisticated workflows that require minimal work to implement, maintain and resuse.

We have also demonstrated how message-driven services enable rapid development of sophisticated interoperable Grids. We conclude that message-driven services offer a more appropriate architecture for Grid computing than current first generation Web Method-based Web Services. In addition, we extol the concept of SOAP intermediaries as transparent resources that will enable the Grid to scale to the size of the Internet.

In the distributed micromagnetic system we demonstrate that TCP in conjunction with message-style Web Services provides a rapid, flexible and lightweight solution for Grid computing. In addition to enabling asynchronous interaction with clients, it opens the option of using Web Services as internal lightweight components of desktop applications and other Web Services. We offer as proof the successful results produced by the micromagnetic distributed system that provides a sophisticated virtual organisation of powerful lightweight Service components.

Chapter 8

Conclusion

The adoption of Service-Oriented Architectures (SOAs) in key aspects of distributed computing is the most important step towards the Grids emergence as a powerful and multifaceted global-scale system. Since the turn of the millennium, realisation of such a system has been furthered by the emergence of key enabling technologies, specifically the Web Service technologies of XML Schema, SOAP and WSDL, that have served to verify SOA as an extremely capable and suitable component model for Grid computing. These technologies provide the key ingredients of generality, simplicity and flexibility, essential for handling of and adaption to complicated, heterogeneous distributed computing environments, such as massive-scale high throughput computing (HTC) clusters.

Our early adoption of XML-based Web Service technologies, in experimental Grid systems and Problem Solving Environments (PSE), has been ratified by the technologies acceptance and application by the Grid communities for important Grid computing problems. This thesis helps to explain why Web Services technologies provide the appropriate means for implementation of Grid systems, and explores their application as a suitable SOA technology for Grid computing problems.

The motivations for Grid systems and specific challenges of Grid-enabled PSE were studied, in chapters 2 and 3, to establish the essential requirements for Grid technologies. We conclude from this that Web Service technologies offer Grid computing capacities not matched by existing distributed object and remote procedure call (RPC) based architectures, such as CORBA and DCOM.

This thesis has described our work carried out in not only engineering design optimisation systems (Geodise) but in the modelling, simulation and analysis of micromagnetic systems. These provide typical Grid computing scenarios requiring access, and dynamic and flexible collaboration of copious quantities of wide-ranging computational resources. We have demonstrated the successful results produced through concrete application of SOA technologies in both distributed environments. In our work on the Computation

Web Service we have selected an important resource in science and engineering problem solving, the compute cluster, and provided generic access and seamless integration of it into both PSEs. This illustrates how service-orientation should be applied to provide a complete virtualisation of resources on the Grid.

Virtualisation of resource types on the Grid provides an important step towards enabling resource consumers choice and easy dynamic migration between service implementations. Experiences with the design, development, management and application of the Computation Web Service has demonstrated how service-orientation should be applied to Grid computing. In addition, we have shown that bespoke systems, specifically the Condor HTC system, can be virtualised and offered as a generic compute services. This illustrates the ability of SOA to both enable reuse of legacy systems, preserving years of effort, and provide the high-level glue for seamless integration with other legacy systems and new resources.

The mode of operation and behaviour of heterogenous distributed resources can represent a real hurdle to creation of loosely-coupled Grids and the consequent creation of dynamic virtual organisations. In our work on numerical optimisation we identified operation and behaviour that would restrict a distributed systems to a tightly-coupled style undesirable in Grid computing. It was argued that optimisations operating in a forward-communication style did not allow separation of compute operations preventing choice of computing resource and generic control, monitoring and checkpointing of optimisations. We proposed an alternative operational mode for optimisation, the reverse-communication style, that enables loose-coupling of systems and addresses the deficiencies of the other style. In addition, we have implemented this style in a sample distributed Optimisation Service that demonstrates how operations can be adapted to enable seamless integration and creation of virtual organisations. As proof of this concept, this work has since been carried on by Gang Xue whom has successfully applied the principle to real engineering design optimisation problems within the GEODISE project.

Whilst Web Services have received wide acceptance by the Grid community, there remains questions over which SOA-style should be applied and the role of application state within Web Services. This has led to the proposal of different Grid service infrastructures, such as WS-GAF and WS-Resource Framework, along with a series of proposed Web Service specifications.

We explored the SOAP protocols currently overlooked transport neutrality and under used messaging abilities in a real world scenario of micromagnetic modelling, simulation and analysis. We demonstrated that the message-style of SOA has potential advantages over Web method-style architectures for intermediaries and sophisticated message exchange patterns. In particular, we found that as well as offering quality-of-service performance and reliability benefits for caching of data and routing of messages, intermediaries have a much broader role providing transparent access to services through gateways, translators and bridges. Their importance cannot be understated as clear parallels must be drawn with the Internet that's ability to effectively operate and perform on a global scale could not have been achieved without its vast network of intermediaries.

However, the role of intermediaries and message-style architectures in Grid computing extends further than this. In particular, we employed intermediaries in the distributed Micromagnetic PSE to enable transparent information capture, a task not easily achieved with Web Method-style of SOA.

In the distributed Micromagnetic PSE we demonstrate that TCP in conjunction with message-style Web Services provides a rapid, flexible and lightweight solution for Grid computing. In addition to enabling asynchronous interaction with clients, it opens the option of using Web Services as internal lightweight components of desktop applications and other Web Services. We offer as proof the successful results produced by the Micromagnetic PSE that provides a sophisticated virtual organisation of powerful lightweight Web Service components.

Users experience of the Micromagnetic PSE and Computation Web Service were positive. Both systems aided the users to concentrate on the problem that they where studying by removing the repetitive error-prone tasks of gaining access and integration of resources. Whilst perhaps not as flexible as scripting approaches they have both have aided produce work published in [141, 100].

One failing identified in our work was highlighted by the inelegant method required for enabling seamless integrating of the Computation Web Service into the distributed Micromagnetic PSE. The two SOA-styles simply do not mesh well and the role of application state in message-style architectures remains unclear. Further work on this topic will aid resolve the appropriate means for handling application state and distributed system state in general.

Agreement on the SOA-style for Grid computing must be made to avoid marked integration of services appearing across and within distributed systems. This will only be resolvable through investigation and experiences in adopting SOA technologies in both academic and commercial areas. Grid computing is a vast topic with many important areas left to research, including application state, workflow and intermediaries. Whilst we have implemented a system to define workflow in terms of redirection of message streams that has been successful in distributed Micromagnetic PSE, its suitability for general usage has yet to be answered. This is a key point as effective orchestration of Web Services must be done in a generic manner if we are to achieve sophisticated interaction patterns between disparate resources and Web Services. Whilst our adaption of the principles of scripting languages offer flexibility and robustness for workflows of service-oriented operations, it may prove with later work to be restrictive and not easily transferable to other workflow mechanisms.

BPEL4WS is a technological framework, supported by major software vendors, such as IBM and Microsoft, that is set to enable generic service orchestration. Further research into its ability to effective define workflows for scientific, engineering and business distributed systems is essential for the establishment of the Grid.

Service oriented architectures enable seamless integration and interoperability between the tools, technologies and platforms required to make the process of scientific and engineering discovery easier and more efficient. It is this integration leveraging open standards which will enable scientists and engineers to tackle the next generation of complex problems they face.

Bibliography

[1] Alistair Mills and David Boyd. Building an e-Science Grid for the UK. Annual report, UK Grid Engineering Task Force (ETF), July 2003. For period September 2002 to April 2003. http://esc.dl.ac.uk/ETF/public/l2g_final_report.pdf.

[2] Global Grid Forum. GGF: Area/Group overview. World Wide Web, 2006. http://www.ggf.org/ggf_areasgrps_overview.htm.

[3] Hugo Haas. W3C: Web Services Activity Statement. World Wide Web, 2005. http://www.w3.org/2002/ws/Activity.

[4] Steve Burbeck. The Tao of e-business services: The evolution of Web applications into service-oriented components with Web services. Technical report, IBM, October 2000. http://www-128.ibm.com/developerworks/library/ws-tao/index.html.

[5] CERN GirdCafe. What is the Grid? Definition, CERN, 2006. http://gridcafe.web.cern.ch/gridcafe/whatisgrid/whatis.html.

[6] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation, W3C, February 2004. http://www.w3.org/TR/REC-xml/.

[7] B. Atkinson. Global XML Web Services Specifications. web Services Security (WS-Security). MSDN Library Article, July 2003.

[8] Stratis Gallopoulos, Elias Houstis, and John Rice. Computer as Thinker/-Doer: Problem-Solving Environments for Computational Science. *IEEE Computational Science and Engineering*, Summer 1994. http://www-cgi.cs.purdue.edu/cgi-bin/acc/pses.cgi.

[9] ISO International Standard. Iso 10303-1:1994 industrial automation systems and integration product data representation and exchange - overview and fundamental principles. *TC184/SC4*, 1994.

[10] Yoshinobu Nakatani, Yasutaro Uesaka, and Nobuo Hayashi. Direct solution of the landau-lifshitz-gilbert equation for micromagnetics. *Japanese Journal of Applied Physics*, 28(12):2485–2507, December 1989.

[11] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms.* Prentice Hall, 2002.

[12] G.M. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS*, volume 30, pages 483–485, Atlantic City, N.I., April 1967. AFIPS Press, Reston, Va.

[13] John L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, 31(5):532–533, 1988.

[14] R.E. Benner, J.L. Gustafson, and R.E. Montry. Development and analysis of scientific application programs on a 1024 processor hypercube. In *SAND 88-0317*, Sandia National Laboratories, Albuquerque, N.M., Febuary 1988.

[15] M. Horstmann and M. Kirtland. Dcom architecture. In *MSDN Library.* Microsoft, July 1997.

[16] Sun Microsystems. Java remote method invocation - distributed computing for java. In *Sun Developer Network.* Sun Microsystems, 2004.

[17] L. Heaton. CORBA/IIOP Specification. In *Object Management Group Specification Catalog.* Object Management Group, Object Management Group, Inc., 250 First Ave. Suite 100, Needham, MA 02494, U.S.A., March 2004.

[18] F. Buschmann, R. Meunier, Hans. Rohnert, P. Sommerlad, M. Stal, P. Sommerlad, and M Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*, volume 1. John Wiley & Sons, 1 edition, August 1996.

[19] Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting bpel web services. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 621–630, New York, NY, USA, 2004. ACM Press.

[20] Petia Wohed, Wil Aalst, Marlon Dumas, and Arthur Hofstede. *Analysis of Web Services Composition Languages: The Case of BPEL4WS*, volume 2813. October 2003.

[21] Copeland-G. Freund T. Klein J. Langworthy D. Orchard D. Shewchuk J. Cabrera, F. and T. Web Services Coordination Storey. Ws-coordination. Technical report, W3C Web Services Description Working Group, www.w3.org/TR/wsdl12/, 2002.

[22] Ali Arsanjani. Service-oriented modeling (sic) and architecture. Technical report, SOA and Web Services Center of Excellence in IBM Global Services, November 2004.

[23] T. Berners-Lee. The World Wide Web: Past, Present and Future. Draft response to an invitation to publish in IEEE Computer special issue of October 1996. The special addition was later abandoned., August 1996.

[24] Web Services Roadmap. Web services protocols summary. Web Site, July 2006. http://roadmap.cbdiforum.com/reports/protocols/summary.php.

[25] T. Berners-Lee and et al. Uniform Resource Identifiers (URI): Generic Syntax. IETF RFC 2396, August 1998.

[26] R. Fielding, U. C. Irvine, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and Berners-Lee T. Hypertext transfer protocol – http/1.1. Online W3C RFC, June 1999.

[27] A. Le Hors and I. Jacobs. (html) 4.01 specification. W3C Recommendation, December 1999.

[28] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[29] R. Fielding. Principle Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002.

[30] S. Bethke, M. Calvetti, H.F. Hoffmann, D. Jacobs, M. Kasemann, and D. Linglin. Report of the steering group of the lhc computing review. Technical report, CERN European Organisation for Nuclear Research, February 2001.

[31] LHC@Home. What is lhc@home? Web Page, July 2006.

[32] S. Tetsuya. Annual Report of the Earth Simulator Center. Technical report, The Earth Simulator Center, Japan Marine Science and Technology Center, The Earth Simulator Center, Japan Marine Science and Technology Center, 3173-25 Showa-machi, Kanazawa-ku, Yokohama, 236-0001 JAPAN, March 2003.

[33] C. Qian, L. Lagacé, M. Massariol, C. Chabot, C. Yoakim, R. Déziel, and L. Tong. A rational approach towards successful crystallization and crystal treatment of human cytomegalovirus protease and its inhibitor complex. *Acta Crystallographica Section D*, 56(2):175–180, Feb 2000.

[34] S. Torquato. Modeling of physical properties of composite materials. *International Journal of Solids and Structures*, 37:411–422, 2000.

[35] O. J. Santollani, R. K. Agarwal, Y. Li, and M. A. Satyro. CAPE-OPEN Specialty Property Packages - New Thermodynamic Models and Software Integration Paradigm. In *Hyprotech User's Conference*, 2000.

[36] Jack Dongarra and et. al. MPI: A Message-Passing Interface Standard. Specification published in Message Passing Interface Forum, June 1995.

[37] L. Smarr. The coming of the grid. Talk given in a workshop on Building a Computational Grid Workshop, Argonne National Laboratory., September 1997.

[38] L. Smarr. The emerging national technology grid. Talk given at the 1998 Annual University of Kansas Center for Advanced Scientific Computing Distinguished Lecture, September 1998.

[39] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, chapter Computational Grids. Morgan-Kaufman, 1999.

[40] . I. Foste, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organisations. *Internation Journal of Supercomputer Applications*, 2001.

[41] I. Foster. What is the Grid? a Three Point Checklist. *Grid Today*, July 2002.

[42] David Snelling, Ian Robinson, and Tim Banks. Web services resource framework (wsrf) tc: Defining an open framework for modeling and accessing stateful resources using web services. Web Site, March 2004. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[43] Marc Snir, Steve W. Otto, David W. Walker, Jack Dongarr, and Steven Huss-Lederman. *MPI: The Complete Reference*. MIT Press, 1995.

[44] J. Basney and M. Livny. Deploying a High Throughput Computing Cluster, in High Performance Cluster Computing. *Prentice Hall Professional Technical Reference*, 1(5), May 1999.

[45] J. Bosak, T. Bray, Connolly D., E. Maler, G. Nicol, C. M. Sperberg-McQueen, L. Wood, and J. Clark. Guide to the W3C XML Specification ("XMLspec") DTD version 2.1. Guide, W3C, 1998.

[46] D. C Fallside. Xml schema part 0: Primer. Online W3C Recommendation, October 2001.

[47] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. F. Nielsen. SOAP Version 1.2 Part 1: Messaging Framework. Online W3C Recommendation, June 2003.

[48] J. B. Postel. Simple Mail Transfer Protocol. Online W3C RFC, August 1982.

[49] Vidur Apparao, Alex Ceponkus, Paul Cotton, Alex Ceponkus, Paul Cotton, David Ezell, David Fallside, Martin Gudgin, Oisin Hurley, John Ibbotson, R. Alexander Milowski, Kevin Mitchell, Jean-Jacques Moreau, Eric Newcomer, Henrik Frystyk Nielsen, Bob Lojek, Mark Nottingham, Waqar Sadiq, Stuart Williams, and Amr Yassin. Xml protocol (xmlp) requirements. Web Site, June 2002. http://www.w3.org/TR/2002/WD-xmlp-reqs-20020626.

[50] R. J. Bayardo, D. Gruhl, V. Josifovski, and J. Myllymaki. An evaluation of binary xml encoding optimizations for fast stream based xml processing. In *WWW '04:*

Proceedings of the 13th international conference on World Wide Web, pages 345–354, New York, NY, USA, 2004. ACM Press.

[51] D. Geer. Will binary xml speed network traffic? *Computer*, 38(4):16–18, 2005.

[52] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web: An introduction to soap, wsdl, and uddi. *IEEE Internet Computing*, 6(2):86–93, 2002.

[53] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture. W3C Working Group Note, W3C, February 2004. http://www.w3.org/TR/ws-arch/.

[54] MSDN. Visual Basic and Visual C# Concepts: Creating Web Applications and Services. Msdn library article, Microsoft, 2004.

[55] MSDN. *.NET Framework Tools: Web Services Description Language Tool (Wsdl.exe).* Micorosoft, 2004.

[56] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. Online W3C Note, March 2001.

[57] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling. Open grid services infrastructure (ogsi) version 1.0. Global Grid Forum Draft Recommendation, June 2003.

[58] T. Bellwood, L. Clment, and C. von Riegen. UDDI Version 3.0.1. Online Oasis Technical Committee Specification, October 2003.

[59] K. Ballinger, P. Brittenham, A. Malhotra, W. A. Nagy, and Pharies S. Web Services Inspection Language (WS-Inspection) 1.0. Online IBM DeveloperWorks Specification, November 2001.

[60] Kelvin Lawrence, Chris Kaler, and Don Flinn. Oasis web services security (wss) tc. Web Site, Feburary 2006. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

[61] IBM, Microsoft, RSA Security, and VeriSign. Web services security policy language. Web Site, December 2002. http://www-128.ibm.com/developerworks/library/specification/ws-secpol/.

[62] Martin Gudgin, Noah Mendelsohn, Mark Nottingham, and Herv Ruellan. Soap message transmission optimization mechanism. Web Site, January 2005. http://www.w3.org/TR/soap12-mtom/.

[63] Steve Ross-Talbot and Tony Fletcher. Web services choreography description language: Primer. W3C Working Draft, June 2006. http://www.w3.org/TR/2006/WD-ws-cdl-10-primer-20060619/.

[64] Tom Rutt, Jacques Durand, and Kazunori Iwasa. Oasis web services reliable messaging (wsrm) tc. OASIS, November 2004. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrm.

[65] Cox S.J., R.P Boardman, L. Chen, M. Duta, M.H. Eres, M.J. Fairman, Z. Jiao, M. Giles, C.A. Goble, G.E. Pound, A.J. Keane, C.M. Scott, N.R. Shadbolt, F. Tao, J.L. Wason, and G. Xue. Grid Services in Action: Grid Enabled Optimisation and Design Search. In *11th IEEE International Symposium on High Performance Distributed Computing HPDC-11*, page 413, 2002.

[66] Cox S.J., M.J. Fairman, G. Xue, J.L. Wason, and A.J. Keane. The Grid: Computational and Data Resource Sharing in Engineering Optimisation and Design Search. In *2001 ICPP Workshops*, pages 207–212, 2001.

[67] R. Stevens, K. Glover, C. Greenhalgh, C. Jennings, S. Pearce, P. Li, and et al. Performing in silico experiments on the Grid: a users perspective. In Cox S. J., editor, *UK e-Science All Hands Meeting*, pages 43–50. EPSRC, 2003.

[68] Lloyd S. L. GridPP - From Web to Grid. *Frontiers*, 16, Summer 2003.

[69] N. A. Walton. Astrogrid: Powering the virtual universe. *Astronomy & Geophysics*, 43(1):30, 2002.

[70] I. Foster and C. Kesselman. The globus project: A status report. In *Heterogeneous Computing Workshop*, pages 4–18, 1998.

[71] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*. IEEE Press, August 2001.

[72] A. S. Grimshaw, Wm. A Wulf, and Legion Team. Legion - The next logical step toward the world-wide virtual computer. *Communications of the ACM*, 40(1), January 1997.

[73] F. Baker. Requirements for IP Version 4 Routers. Internet Engineering Task Force (IETF), RFC 1812, 1995.

[74] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM symposium on Communications architectures & protocols*, volume 20 of *4*. ACM SIGCOMM Computer Communication Review, August 1990.

[75] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. Design and deployment of a national-scale authentication infrastructure. *IEEE Computer*, 30(12):60–66, 2000.

[76] T. Zhou. .NET Passport Simplifies E-Commerce User Management. *Windows IT Pro*, April 2002.

[77] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *ACM Conference on Computers and Security*, pages 83–91, 1998.

[78] B. C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33–38, September 1994.

[79] Borland Software Corporation. Make the microsoft .net framework real enterprise solutions for today! Webcast, December 2003.

[80] Inc. Sun Microsystems. Java 2 Platform, Standard Edition (J2SE platform), Version 1.4.2. Performance white paper, Sun Microsystems, Inc., 2004.

[81] T. A. Howes. An X.500 and LDAP database: Design and Implementation. Technical report, University of Michigan, 1995.

[82] R. Menday and P. Wieder. GRIP: The evolution of UNICORE towards a Service-Oriented Grid. In *Cracow Grid Workshop*, 2003.

[83] The Globus Project. The Globus Resource Specification Language RSL v1.0. Globus specification, May 2000.

[84] A.C. Catlin, M.G. Gaitatzes, E.N. Houstis, Z. Ma S. Markus, J.R. Rice, Nien-Hwa Wang, and S. Weerawarana. Softlab: A virtual laboratory framework for computational science.

[85] EleSoft Research. Elements engineering-scientific workspace, 2006.

[86] E. N. Houstis, J. R. Rice, S. Weerawarana, A. C. Catlin, P. Papachiou, K.-Y. Wang, and M. Gaitatzes. PELLPACK: A problem solving environment for PDE based applications on multicomputer platforms. *ACM Transactions on Mathematical Software*, 24(1):30–73, 1998.

[87] M. J. Donahue and D. G. Porter. OOMMF User's Guide, Version 1.0. Interagency Report NISTIR 6376, National Institute of Standards and Technology, Gaithersburg, MD, September 1999.

[88] Werner ScholzCorresponding, Josef Fidler, Thomas Schrefl, Dieter Suess, Rok Dittrich, Hermann Forster, and Vassilios Tsiantos. Scalable parallel micromagnetic solvers for magnetic nanostructures. In *Symposium on Software Development for Process and Materials Design*, volume 28, pages 366–383, Institute of Solid State Physics, Vienna University of Technology, Wiedner Hauptstrasse 8-10/138, A-1040, Vienna, Austria, October 2003.

[89] Gordon D Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods.* Oxford University Press, January 1986.

[90] K. Ho-Le. Finite element mesh generation methods: a review and classification. *Comput. Aided Des.*, 20(1):27–38, 1988.

[91] D. A. Thompson and J. S. Best. The future of magnetic data storage technology. *Research and Development (IBM)*, 44(3):311–322, May 2000.

[92] CA Ross. Patterned magnetic recording media. *Annual Review of Materials Research*, 31(1):203–235, 2001.

[93] Robert Edward; Hsiao Richard; Marinero Ernesto Esteban; Santini Hugo Alberto Emilio; Terris Bruce David Fontana, Jr. Patterned magnetic media and method of making the same using selective oxidation, January 2001.

[94] Richard P. Boardman. *Computer simulation studies of magnetic nanostructures.* PhD thesis, Engineering Sciences, 2005.

[95] W. Schroeder, K. Martin, and B. Lorenson. *The Visualization Toolkit, An Object-Oriented Approach To 3D Graphics.* Kitware, 2 edition, 1999.

[96] T. Williams and C. Kelley. GNUPLOT : An Interactive Plotting Program. Manual, Gnuplot, 1998.

[97] M. K. H. Fan, L.-S. Wang, J. Koninckx, and A. L. Tits. Software package for optimization-based design with user-supplied simulators. *IEEE Control Systems*, 1(9):66–71, 1989.

[98] A. J. Keane. *The OPTIONS Design Exploration System . Reference Manual and User Guide. Version B3.1.* University of Southampton, Department of Mechanical Engineering, University of Southampton, Highfield, Southampton, SO17 1BJ, U.K., b3.1 edition, June 2003.

[99] Cox S.J, M.J. Fairman, G. Xue, J.L. Wason, and A.J. Keane. The grid: Computational and data resource sharing in engineering optimisation and design search. In *ICPP Workshops*, pages 207–212, 2001.

[100] G. Xue, M. Fairman, G.E Pound, and S.J Cox. Implementation of a grid computation toolkit for design optimisation with matlab and condor. In *Euro-Par 2003 Parallel Processing, Lecture Notes in Computer Science*, number 2790, pages 357–365, 2003.

[101] S.J. Cox, Z. Jiao, and J.L Wason. Data management services for engineering. In *UK e-Science All Hands*, Sheffield, September 2002.

[102] A. J. Keane. OPTIONS design exploration system. Technical report, School of Engineering Sciences, University of Southampton, 2004.

[103] Sun Microsystems. *Enterprise JavaBeans Technology Downloads & Specifications.* Sun Microsystems, 2003.

[104] M.H. Eres, G.E. Pound, Z. Jiao, J. Wason, F. Xu, A.J. Keane, and S.J. Cox. Implementation of a grid-enabled problem solving environment in matlab. In *International Conference on Computer Science (ICCS), Lecture Notes in Computer Science*, volume Part IV, pages 420–429, 2003.

[105] I. Foster. What is the grid: A three point checklist. *Grid Today*, July 2002.

[106] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[107] Ian Hickson. Web forms 2.0. Online Opera Working Draft, March 2004.

[108] P. Bartholo. Applets Power the Client. Article, Sun Microsystem, Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, CA 95054, 1998.

[109] MSDN Library. *Web Services Description Language Tool (Wsdl.exe).* Microsoft Corporation, Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 U.S.A., 2004.

[110] superclusters.org. *Maui Users Manual.* CLUSTER RESOURCES INC. - Center for HPC Cluster Resource Management and Scheduling, CLUSTER RESOURCES INC. 11116 Conestoga Drive Covered Bridge Canyon, UT 84660, 2004.

[111] Jeffrey O. Kephart, Rajarshi Das, and Jeffrey K. MacKie-Mason. Two-sided learning in an agent economy for information bundles. In *Agent-mediated Electronic Commerce workshop at IJCAI '99*, 1999.

[112] Nelder and Mead. Downhill simplex method. *Computer Journal*, 7:308–313, 1965.

[113] *Numerical Recipes in C: The Art of Scientific Computing*, chapter 10.4 Downhill Simplex Method in Multidimensions, pages 408–412. Cambridge University Press, 2nd edition, 1992.

[114] E.H. Baalbergen and H. van der Ven. Spineware - a framework for user-oriented and tailorable metacomputers. *National Aerospace Laboratory*, page 16, September 1998.

[115] S. L. Padula, J. J. Korte, H. J. Dunn, and A. O. Salas. *Multidisciplinary Optimization Branch Experience Using iSIGHT Software.* Langley Research Center, 1999.

[116] Phoenix Intergration. Modelcenter 7.0. Web Site, 2006. http://www.phoenix-int.com/modelcenter.htm.

[117] J. Czyzyk, M.P. Mesnier, and J.J. More. The neos server. *Computational Science and Engineering, IEEE*, 5:68–75, 1998.

[118] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* 2nd edition, 2003.

[119] Rajkumar Buyya, David Abramson, and Jonathan Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, volume 1, pages 283–283, 2000.

[120] Henri Casanova and Jack Dongarra. NetSolve: A network-enabled server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, Fall 1997.

[121] Keith Seymour, Hidemoto Nakada, Satoshi Matsuoka, Jack Dongarra, Craig Lee, and Henri Casanova. *Overview of GridRPC: A Remote Procedure Call API for Grid Computing*, volume 2536. January 2002.

[122] R. Khare. Can xform transform the web? transcending the web as gui (graphical user interface). Technical Report Part II, 4K Associates, 2000.

[123] Sun Microsystems. *Servlets API and Documentation.* Sun Microsystems, May 2002.

[124] GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with Matlab. Example functions (single and multi-objective functions) 2 parametric optimization: De jong's function 1. Web Site, 2005. http://www.geatbx.com/ver_3_7/fcnindex-01.html#P85_2637.

[125] G. Xue, M. J. Fairman, and S. J Cox. Exploiting Web Technologies for Grid Architecture. In *CCGrid*, pages 272–273, 2002.

[126] G. Xue, M. J. Fairman, G. Pound, and S. J. Cox. Implementation of a Grid Computation Toolkit for Design Optimisation with Matlab and Condor. In *Euro-Par Conference*, 2003.

[127] G. Fox, T. Haupt, E. Akarsu, A. Kalinichenko, K. Kim, P. Sheethalmath, and C. Youn. The Gateway System: Uniform Web Based Access to Remote Resources. In *ACM Java Grande Conference*, 1999.

[128] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and Tuecke. S. A resource management architecture for metacomputing systems. In *IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[129] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[130] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, July 1998.

[131] R. Raman, L. Miron, and M. Solomon. Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching. In *Twelfth IEEE International Symposium on High-Performance Distributed Computing*, June 2003.

[132] D. Watkins. Handling language interoperability with the microsoft .net framework. Msdn library article, Monash University, October 2000.

[133] B. Atkinson and et. al. Web Services Security (WS-Security). IBM and Microsoft Corporation Specification Proposal, April 2002.

[134] Microsoft Corporation. Internet Information Services 6.0 Product Documentation. Technical report, Microsoft Corporation, 2004.

[135] A. Eide and et. al. *Professional ASP.NET Web Services.* Wrox Press, 1 edition, November 2001.

[136] Anonymous. Http cookie. Wikipedia, 2004.

[137] J. H. Gailey. Sending files, attachments, and SOAP messages via direct internet message encapsulation. *MSDN Magazie*, 2003.

[138] M. Powell. Understanding dime and ws-attachments. Msdn library article, Microsoft Corporation, July 2003.

[139] University of Virginia Grid Computing Group. Ogsi.net. Web Site, 2003. http://www.cs.virginia.edu/ gsw2c/ogsi.net.html.

[140] EPCC. The ms.netgrid project. Web Site, 2003. http://www.epcc.ed.ac.uk/ ogsanet/.

[141] M. J. Fairman, P. B. Boardman, J. Zimmerman, and H. Fanghor. Application of Lightweight Web Services for Micromagnetic Modelling, Simulation and Analysis. Currently under review for the Journal of Grid Computing.

[142] Information Sciences Institute. TRANSMISSION CONTROL PROTOCOL. Internet protocol, DARPA Internet Program, Defense Advanced Research Projects Agency, Information Processing Techniques Office, 1400 Wilson Boulevard, Arlington, Virginia 22209, September 1981.

[143] D. Box and et al. Web Services Addressing (WS-Addressing). Technical report, Global XML Web Service, 2004.

[144] R. P. Boardman, M. J. Fairman, J. Zimmerman, and H. Fanghor. Micromagnetic modelling of ferromagnetic cones. Under review for the Journal of Material Sciences.

Conference Publications

1. Cox, S. J., Fairman, M. J., Xue, G., Wason, J. L., and Keane, A. J. 2001. The Grid: Computational and Data Resource Sharing in Engineering Optimisation and Design Search. Proceedings of the 2001 ICPP Workshops p.207-212.
2. Xue, G., Fairman, M. J., and Cox, S. J. 2002. Exploiting Web Technologies for Grid Architecture. Proceedings of the 2002 CCGrid p. 272-273.
3. Cox S.J, Boardman, R.P, Chen, L, Duta, M, Eres, M.H, Fairman, M.J, Jiao, Z, Giles, M, Goble, C.A, Pound, G.E, Keane, A.J, Scott, C.M, Shadbolt, N.R, Tao, F, Wason, J.L, Xue, G. (2002) Grid Services in Action: Grid Enabled Optimisation and Design Search. Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002, p 413
4. Xue, G., Fairman, M. J., Pound, G., and Cox, S. J. 2003. Implementation of a Grid Computation Toolkit for Design Optimisation with Matlab and Condor. Proceedings of the 2003 Euro-Par Conference.
5. Pound, G.E, Eres, M. H, Fairman, M.J, Xue, G, Keane, A.J, and Cox, S.J. Grid middleware for engineering design search and optimisation. Proceedings of UK e-Science All Hands Meeting 2003, pp. 736-743

Technical Reports

1. Cox, S. J., Takeda, K., Xue, G. and Fairman, M.J., 2001. Microsoft .NET and the Grid. Report for the Dept of Trade and Industry. 3rd August 2001.

Original C Language Source Code for Amoeba Optimisation

```c
#include "stdafx.h"
#include <math.h>
#include "nrutil.h"
#define TINY 1.0e-10 /*A small number.*/
#define NMAX 5000 /*Maximum allowed number of function evalua-*/
#define GET_PSUM \ /*tions.*/
    for (j=1;j<=ndim;j++) {\
        for (sum=0.0,i=1;i<=mpts;i++)\
            sum += p[i][j];\
        psum[j]=sum;\
    }
#define SWAP(a,b) {swap=(a);(a)=(b);(b)=swap;}

void amoeba(float **p, float y[], int ndim, float ftol,
            float (*funk)(float []), int *nfunk)
    /*Multidimensional minimization of the function funk(x) where x[1..ndim] is a vector in
     ndim
    dimensions, by the downhill simplex method of Nelder and Mead. The matrix p[1..ndim+1]
    [1..ndim] is input. Its ndim+1 rows are ndim-dimensional vectors which are the vertices of
    the starting simplex. Also input is the vector y[1..ndim+1], whose components must be
     preinitialized
    to the values of funk evaluated at the ndim+1 vertices (rows) of p; and ftol the
    fractional convergence tolerance to be achieved in the function value (n.b.!). On output,
     p and
    y will have been reset to ndim+1 new points all within ftol of a minimum function value,
     and
    nfunk gives the number of function evaluations taken.*/
{
    float amotry(float **p, float y[], float psum[], int ndim,
        float (*funk)(float []), int ihi, float fac);
    int i,ihi,ilo,inhi,j,mpts=ndim+1;
    float rtol,sum,swap,ysave,ytry,*psum;
    psum=vector(1,ndim);
    *nfunk=0;
    GET_PSUM
    for (;;) {
        ilo=1;
        /*First we must determine which point is the highest (worst), next-highest, and lowest
        (best), by looping over the points in the simplex.*/
        ihi = y[1]>y[2] ? (inhi=2,1) : (inhi=1,2);
        for (i=1;i<=mpts;i++) {
            if (y[i] <= y[ilo]) ilo=i;
            if (y[i] > y[ihi]) {
                inhi=ihi;
                ihi=i;
            } else if (y[i] > y[inhi] && i != ihi) inhi=i;
        }
        rtol=2.0*fabs(y[ihi]-y[ilo])/(fabs(y[ihi])+fabs(y[ilo])+TINY);
        /*Compute the fractional range from highest to lowest and return if satisfactory.*/
        if (rtol < ftol) { /*If returning, put best point and value in slot 1.*/
            SWAP(y[1],y[ilo])
                for (i=1;i<=ndim;i++) SWAP(p[1][i],p[ilo][i])
                    break;
        }
        if (*nfunk >= NMAX) nrerror("NMAX exceeded");
        *nfunk += 2;
        /*Begin a new iteration. First extrapolate by a factor -1 through the face of the
    simplex
```

```
        across from the high point, i.e., reflect the simplex from the high point.*/
        ytry=amotry(p,y,psum,ndim,funk,ihi,-1.0);
        if (ytry <= y[ilo])
            /*Gives a result better than the best point, so try an additional extrapolation by
 a
            factor 2.*/
            ytry=amotry(p,y,psum,ndim,funk,ihi,2.0);
        else if (ytry >= y[inhi]) {
            /*The reflected point is worse than the second-highest, so look for an
 intermediate
            lower point, i.e., do a one-dimensional contraction.*/
            ysave=y[ihi];
            ytry=amotry(p,y,psum,ndim,funk,ihi,0.5);
            if (ytry >= ysave) { /*Cant seem to get rid of that high point. Better*/
                for (i=1;i<=mpts;i++) { /*contract around the lowest (best) point.*/
                    if (i != ilo) {
                        for (j=1;j<=ndim;j++)
                            p[i][j]=psum[j]=0.5*(p[i][j]+p[ilo][j]);
                        y[i]=(*funk)(psum);
                    }
                }
                *nfunk += ndim; /*Keep track of function evaluations.*/
                GET_PSUM /*Recompute psum.*/
            }
        } else --(*nfunk); /*Correct the evaluation count.*/
    } /*Go back for the test of doneness and the next*/
    free_vector(psum,1,ndim); /*iteration.*/
}

#include "nrutil.h"
float amotry(float **p, float y[], float psum[], int ndim,
             float (*funk)(float []), int ihi, float fac)
 /*Extrapolates by a factor fac through the face of the simplex across from the high point
 ,
 tries it, and replaces the high point if the new point is better.*/
{
    int j;
    float fac1,fac2,ytry,*ptry;
    ptry=vector(1,ndim);
    fac1=(1.0-fac)/ndim;
    fac2=fac1-fac;
    for (j=1;j<=ndim;j++) ptry[j]=psum[j]*fac1-p[ihi][j]*fac2;
    ytry=(*funk)(ptry); /*Evaluate the function at the trial point.*/
    if (ytry < y[ihi]) { /*If it s better than the highest, then replace the highest.*/
        y[ihi]=ytry;
        for (j=1;j<=ndim;j++) {
            psum[j] += ptry[j]-p[ihi][j];
            p[ihi][j]=ptry[j];
        }
    }
    free_vector(ptry,1,ndim);
    return ytry;
}
```

Listing 1: Original C Language Source Code for Amoeba Optimisation

Java source code for Amoeba Optimisation Algorithm Session EJB

```java
package optimisation;

import java.rmi.RemoteException;

import javax.ejb.EJBException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

import javax.ejb.CreateException;

import optimisation.AmoebaState;

/**
 * @ejb.bean name="Ameoba"
 *           display-name="Name for Ameoba"
 *           description="Description for Ameoba"
 *           jndi-name="ejb/Ameoba"
 *           type="Stateless"
 *           view-type="remote"
 */
public class Ameoba implements SessionBean {

    //ALGORITHM MAXIMUMS

    private static final int NMAX = 5000; //Maximum function evaluations

    //AMOEBA STATES

    public static final int START_LOOP     = 0;
    public static final int INITIALISATION = -1;
    public static final int FINISHED       = -2;
    public static final int NMAX_ERROR     = -100;
    public static final int AMOTRY_ERROR   = -101;

    public Ameoba() {
        super();
        // TODO Auto-generated constructor stub
    }

    public void setSessionContext(SessionContext ctx)
        throws EJBException,
        RemoteException {
        // TODO Auto-generated method stub

    }

    public void ejbRemove() throws EJBException, RemoteException {
        // TODO Auto-generated method stub

    }

    public void ejbActivate() throws EJBException, RemoteException {
        // TODO Auto-generated method stub

    }

    public void ejbPassivate() throws EJBException, RemoteException {
        // TODO Auto-generated method stub

    }

    /**
     * Default create method
     *
     * @throws CreateException
     * @ejb.create-method
     */
    public void ejbCreate() throws CreateException {
        // TODO Auto-generated method stub
    }

    /**
     * Business method
     * @ejb.interface-method  view-type = "remote"
     */
    public AmoebaState bootstrap(int ndim, float ftol)
    {
        //setup the state data for the optimisation
        //first element of array start from 1

        AmoebaState result = new AmoebaState();

        int i=0, ihi=0, ilo=0, inhi=0, j=0, mpts=0, nfunk=0, state=INITIALISATION;
        float rtol=0.0f, summ=0.0f, swap=0.0f, ysave=0.0f, ytry=0.0f, psum[], x[];
        Amotry amotry;
        float p[][], y[];

        //x[] is arguments for func f(x)
        x = new float[ndim+1];

        //p's ndim+1 rows are ndim-dimensional vectors which are the vertices of
        //the starting simplex. Also input is the vector y[1..ndim+1],
        //whose components must be preinitialized to the values of f(x) evaluated
        //at the ndim+1 vertices (rows) of p
        p = new float[ndim+2][ndim+1];
        y = new float[ndim+2];

        //first iteration of reverse commuincation loop to preinitialise p
        //see initialisation case of next() for continuance of preinitilisation
        //i starts at 1 and is incremented by 1 till it reaches ndim+1
        i=1;
        if( i <= (ndim+1) ) {
            for (j=1;j<=ndim;j++)
                x[j]=p[i][j]=(i == (j+1) ? 1.0f : 0.0f);

            //call to f(x) would have gone here
            //instead we return x[] to client which must calculate result of f(x) and pass too
      next
        }

        mpts = ndim+1;
        psum = new float[ndim+1];
        nfunk = 0;

        amotry = new Amotry(p,y,psum,ndim,ihi,-1.0f);

        //save state
        result.i = i;
        result.ihi = ihi;
        result.ilo = ilo;
        result.inhi = inhi;
        result.j = j;
        result.mpts = mpts;
        result.nfunk = nfunk;
        result.ndim = ndim;
        result.state = state;

        result.rtol = rtol;
        result.summ = summ;
        result.swap = swap;
        result.ysave = ysave;
        result.ytry = ytry;
        result.psum = psum;
        result.p = p;
```

```java
        result.y = y;
        result.ftol = ftol;
        result.x = x;

        result.amotry = amotry;

      //return optimistaion id
        return result;
    }
    /**
     * Business method
     * @ejb.interface-method view-type = "remote"
     */
    public AmoebaState next(AmoebaState state)
        throws optimisation.FunctionEvaluationException
    {
        AmoebaState ns = null;
        try
        {
            //make a copy of the state which will then be updated and
            //returned to the client each time f(x) need calculation on new x values
            ns = (AmoebaState)state.clone();
        }
        catch(Exception ex)
        {
            //this shouldnt happen as clone is implemented but allows program to compile
            throw new optimisation.FunctionEvaluationException(ex);
        }

        while(true)
        {
            switch(ns.state)
            {
                case INITIALISATION:

                    //continue preinitialise of p (see bootstrap)
                    //keep returning x to client for calc of result of f(x) until i reaches ndim+1
                    ns.y[ns.i++] = ns.res; //store results of f(x_i) for ordering
                    if( ns.i <= (ns.ndim+1) ) {
                        for (ns.j=1;ns.j<=ns.ndim;ns.j++)
                            ns.x[ns.j] = ns.p[ns.i][ns.j] = (ns.i == (ns.j+1) ? 1.0f : 0.0f);
                        ns.state = INITIALISATION;
                        return ns;
                    }

                    //do final preinitiliastion by getting psum and then start the optimisation
                    //psum give us the summation for calculating of the centroid in ordering
                    //this only need happen once hence is in the initialiastion after p[][] has finished looping
                    for(ns.j=1; ns.j<=ns.ndim; ns.j++) {
                        for(ns.summ=0.0f, ns.i=1; ns.i<=ns.mpts; ns.i++)
                            ns.summ += ns.p[ns.i][ns.j];
                        ns.psum[ns.j] = ns.summ;
                    }

                    ns.state = START_LOOP;
                    break;

                case START_LOOP:

                    ns.ilo=1;
                    //****
                    //Order
                    //****
                    //First we must determine which point is the highest (worst), next-highest, and lowest
                    //(best), by looping over the points in the simplex.
                    if( ns.y[1] > ns.y[2] ) {
                        ns.ihi = 1; ns.inhi = 2;
                    } else {
                        ns.ihi = 2; ns.inhi = 1;
                    }

                    for (ns.i=1;ns.i<=ns.mpts;ns.i++) {
                        if (ns.y[ns.i] <= ns.y[ns.ilo])
                            ns.ilo=ns.i;
                        if (ns.y[ns.i] > ns.y[ns.ihi]) {
                            ns.inhi=ns.ihi;
                            ns.ihi=ns.i;
                        } else if (ns.y[ns.i] > ns.y[ns.inhi] && ns.i != ns.ihi)
                            ns.inhi=ns.i;
                    }

                    ns.rtol=(float)(2.0f*Math.abs(ns.y[ns.ihi]-ns.y[ns.ilo])/
                            (Math.abs(ns.y[ns.ihi])+Math.abs(ns.y[ns.ilo])));


                    //Compute the fractional range from highest to lowest and return if satisfactory.
                    if (ns.rtol < ns.ftol) { //If returning, put best point and value in slot 1.
                        ns.swap=ns.y[1];
                        ns.y[1]=ns.y[ns.ilo];
                        ns.y[ns.ilo]=ns.swap;
                        for (ns.i=1;ns.i<=ns.ndim;ns.i++) {
                            ns.swap=ns.p[1][ns.i];
                            ns.p[1][ns.i]=ns.p[ns.ilo][ns.i];
                            ns.p[ns.ilo][ns.i]=ns.swap;
                        }
                        ns.state = FINISHED; //completed optimisation
                        return ns;
                    }

                    //check we have not exceed our number of maximum number of iterations
                    if (ns.nfunk >= NMAX) {
                        ns.state = NMAX_ERROR;
                        throw new FunctionEvaluationException("next: maximum function evaluations exceed");
                    }
                    ns.nfunk += 2;

                    //****
                    //Start Reflect
                    //****

                    //Begin a new iteration. First extrapolate by a factor ?1 through the face of the simplex
                    //across from the high point, i.e., reflect the simplex from the high point.
                    //set up amotry and wait for next res
                    ns.amotry.reset(ns.p,ns.y,ns.psum,ns.ndim,ns.ihi,-1.0f);
                    ns.x = ns.amotry.getEvaluationPoint();
                    ns.state = 1;
                    return ns;

                case 1:


                    //we have got back the result from amotry so continue with revesre communication loop
                    ns.amotry.state = ns.amotry.next(ns.res, ns.p, ns.y, ns.psum);
                    if( ns.amotry.state == StatefullOptimiser.FINISHED ) { //amotry has completed
                        //get result of amotry and goto next state
                        ns.ytry = ns.amotry.getResult(); //save reflection point f_r
                        if (ns.ytry <= ns.y[ns.ilo])
                        {

                            //setup amotry and wait for next res
                            ns.amotry.reset(ns.p,ns.y,ns.psum,ns.ndim,ns.ihi,2.0f);
                            ns.point = ns.amotry.getEvaluationPoint();
                            ns.state = 2; //goto expand
                            return ns;
                        }
                        else if (ns.ytry >= ns.y[ns.inhi])
                        {
                            ns.ysave=ns.y[ns.ihi];

                            //setup amotry and wait for next res
                            ns.amotry.reset(ns.p,ns.y,ns.psum,ns.ndim,ns.ihi,0.5f);
                            ns.point = ns.amotry.getEvaluationPoint();
                            ns.state = 3;
                            return ns;
                        }
                        else
                        {
                            --ns.nfunk;

```

```
                            //goto start of loop
                            ns.state = START_LOOP;
                            break;
                        }
                    } else if( ns.amotry.state != StatefullOptimiser.ERROR ) { //keep looping
    until amotry finishes
                        //wait for next res
                        ns.point = ns.amotry.getEvaluationPoint();
                        ns.state = 1;
                        return ns;
                    } else { //something has gone terrible wrong
                        ns.state = AMOTRY_ERROR;
                        throw new EJBException("next: "+ns.amotry.getErrorMessage());
                    }

                case 2:
                    ns.amotry.state = ns.amotry.next(ns.res, ns.p, ns.y, ns.psum);
                    if( ns.amotry.state == Amotry.FINISHED ) {
                        //get result of amotry and goto next state
                        ns.ytry = ns.amotry.getResult();
                        ns.state = START_LOOP;
                        break;
                    } else if( ns.amotry.state != StatefullOptimiser.ERROR ) {
                        //wait for next res
                        ns.point = ns.amotry.getEvaluationPoint();
                        ns.state = 2;
                        return ns;
                    } else {
                        ns.state = AMOTRY_ERROR;
                        throw new EJBException("next: "+ns.amotry.getErrorMessage());
                    }

                case 3:
                    ns.amotry.state = ns.amotry.next(ns.res, ns.p, ns.y, ns.psum);
                    if( ns.amotry.state == StatefullOptimiser.FINISHED ) {
                        //get result of amotry and goto next state
                        ns.ytry = ns.amotry.getResult();
                        if (ns.ytry >= ns.ysave) {
                            ns.i=1;
                            while( ns.i <= ns.mpts ) {
                                if (ns.i != ns.ilo) {
                                    for (ns.j=1;ns.j<=ns.ndim;ns.j++)
                                        ns.p[ns.i][ns.j]=ns.psum[ns.j]=0.5f*(ns.p[ns.i][ns.j]+
    ns.p[ns.ilo][ns.j]);
                                    //wait for res;
                                    ns.point = ns.psum;
                                    ns.state = 4;
                                    return ns;
                                }
                                ns.i++;
                            }
                            ns.nfunk += ns.ndim;
                            for(ns.j=ns.i; ns.j<=ns.ndim; ns.j++) {
                                for(ns.summ=0.0f, ns.i=1; ns.i<=ns.mpts; ns.i++)
                                    ns.summ += ns.p[ns.i][ns.j];
                                ns.psum[ns.j] = ns.summ;
                            }
                        }
                        ns.state = START_LOOP;
                        break;
                    } else if( ns.amotry.state != StatefullOptimiser.ERROR ) {
                        //wait for next res
                        ns.state = 3;
                        return ns;
                    } else {
                        ns.state = AMOTRY_ERROR;
                        throw new EJBException("next: "+ns.amotry.getErrorMessage());
                    }

                case 4:
                    ns.y[ns.i++] = ns.res;
                    while( ns.i <= ns.mpts ) {
                        if (ns.i != ns.ilo) {
                            for (ns.j=1;ns.j<=ns.ndim;ns.j++)
                                ns.p[ns.i][ns.j]=ns.psum[ns.j]=0.5f*(ns.p[ns.i][ns.j]+ns.p[ns.
    ilo][ns.j]);
                            //wait for res;
                            ns.point = ns.psum;
                            ns.state = 4;
                            return ns;
                        }
                        ns.i++;
                    }

                    ns.nfunk += ns.ndim;
                    for(ns.j=ns.i; ns.j<=ns.ndim; ns.j++) {
                        for(ns.summ=0.0f, ns.i=1; ns.i<=ns.mpts; ns.i++) ns.summ += ns.p[ns.i]
    [ns.j];
                        ns.psum[ns.j] = ns.summ;
                    }

                    //goto start
                    ns.state = START_LOOP;
                    break;

            default:
                if( ns.state == FINISHED ) {
                    return ns;
                } else if( ns.state == NMAX_ERROR ) {
                    throw new FunctionEvaluationException("next: maximum function evaluations
    exceeded");
                } else if( ns.state == AMOTRY_ERROR ) {
                    throw new EJBException("next: "+ns.amotry.getErrorMessage());
                } else {
                    throw new EJBException("next: an invalid state has been entered.");
                }

            } // switch
        } // loop
    }
}
```

LISTING 2: Java source code for Amoeba Optimisation Algorithm Session EJB

WSDL of the Compute Web Service

```xml
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://www.geodise.org/CompWS/" xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespace="http://www.geodise.org/CompWS/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.geodise.org/CompWS/">
      <s:element name="GetFiles">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="id" type="tns:JobID" />
            <s:element minOccurs="1" maxOccurs="1" name="timeout" type="s:int" />
            <s:element minOccurs="1" maxOccurs="1" name="retrieve" type="tns:RetrieveFiles" />

          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="JobID">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="Cluster" type="s:unsignedInt" />
          <s:element minOccurs="1" maxOccurs="1" name="Proc" type="s:unsignedInt" />
        </s:sequence>
      </s:complexType>

      <s:simpleType name="RetrieveFiles">
        <s:restriction base="s:string">
          <s:enumeration value="All" />
          <s:enumeration value="New" />
        </s:restriction>
      </s:simpleType>
      <s:element name="GetFilesResponse">
        <s:complexType>
          <s:sequence>

            <s:element minOccurs="1" maxOccurs="1" name="GetFilesResult" type="s:boolean" />
            <s:element minOccurs="0" maxOccurs="1" name="files" type="tns:ArrayOfJobFile" />
            <s:element minOccurs="0" maxOccurs="1" name="info" type="tns:TerminationClassAd" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfJobFile">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="JobFile" nillable="true" type="tns:JobFile" />

        </s:sequence>
      </s:complexType>
      <s:complexType name="JobFile">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Data" type="s:base64Binary" />
          <s:element minOccurs="0" maxOccurs="1" name="Filename" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="TerminationClassAd">
```

```

        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Machine" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="JobID" type="tns:JobID" />
          <s:element minOccurs="0" maxOccurs="1" name="Executable" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="Terminated" type="tns:Terminated" />
          <s:element minOccurs="1" maxOccurs="1" name="StatusCode" type="s:int" />
          <s:element minOccurs="1" maxOccurs="1" name="SubmittedTime" type="s:dateTime" />
          <s:element minOccurs="1" maxOccurs="1" name="CompletedTime" type="s:dateTime" />
          <s:element minOccurs="1" maxOccurs="1" name="RealTime" type="s:long" />

          <s:element minOccurs="1" maxOccurs="1" name="ImageSize" type="s:unsignedLong" />
          <s:element minOccurs="1" maxOccurs="1" name="UserCPUTime" type="s:long" />
          <s:element minOccurs="1" maxOccurs="1" name="SystemCPUTime" type="s:long" />
          <s:element minOccurs="1" maxOccurs="1" name="RemoteCPUTime" type="s:long" />
          <s:element minOccurs="1" maxOccurs="1" name="AllRunTimes" type="s:long" />
          <s:element minOccurs="0" maxOccurs="1" name="Administrator" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="WSUser" type="s:string" />
        </s:sequence>
      </s:complexType>

      <s:simpleType name="Terminated">
        <s:restriction base="s:string">
          <s:enumeration value="normally" />
          <s:enumeration value="abnormally" />
          <s:enumeration value="checkpointed" />
        </s:restriction>
      </s:simpleType>
      <s:element name="Wait">
        <s:complexType>

          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="id" type="tns:JobID" />
            <s:element minOccurs="1" maxOccurs="1" name="timeout" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="WaitResponse">
        <s:complexType>
          <s:sequence>

            <s:element minOccurs="1" maxOccurs="1" name="WaitResult" type="s:boolean" />
            <s:element minOccurs="0" maxOccurs="1" name="info" type="tns:TerminationClassAd" /
    >
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="Submit">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="desc" type="tns:SubmissionClassAd" />

          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="SubmissionClassAd">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="Executable" type="tns:JobFile" />
          <s:element minOccurs="1" maxOccurs="1" name="Requirements" nillable="true" type="tns
    :LogicalExpression" />
          <s:element minOccurs="1" maxOccurs="1" name="Rank" nillable="true" type="tns:
    ComparisonExpression" />
          <s:element minOccurs="1" maxOccurs="1" name="EmailNotification" type="tns:
    EmailNotification" />

          <s:element minOccurs="1" maxOccurs="1" name="EmailAddr" nillable="true" type="s:
    string" />
          <s:element minOccurs="1" maxOccurs="1" name="Environment" nillable="true" type="tns:
    Environment" />
          <s:element minOccurs="1" maxOccurs="1" name="Log" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="Universe" type="tns:Universe" />
          <s:element minOccurs="0" maxOccurs="unbounded" name="SubmissionJobGroupClassAd" type
    ="tns:SubmissionJobGroupClassAd" />
          <s:element minOccurs="0" maxOccurs="unbounded" name="SharedDataFiles" nillable="true
    " type="tns:JobFile" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="LogicalExpression">

        <s:sequence>
```

```xml
        <s:element minOccurs="0" maxOccurs="1" name="Expression">
          <s:complexType>
            <s:sequence>
              <s:any />
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:sequence>

    </s:complexType>
    <s:complexType name="ComparisonExpression">
      <s:complexContent mixed="false">
        <s:extension base="tns:LogicalExpression" />
      </s:complexContent>
    </s:complexType>
    <s:simpleType name="EmailNotification">
      <s:restriction base="s:string">
        <s:enumeration value="ERROR" />

        <s:enumeration value="UNDEFINED" />
        <s:enumeration value="Never" />
        <s:enumeration value="Always" />
        <s:enumeration value="Complete" />
        <s:enumeration value="Error" />
      </s:restriction>
    </s:simpleType>
    <s:complexType name="Environment">
      <s:sequence>

        <s:element minOccurs="0" maxOccurs="1" name="environment">
          <s:complexType>
            <s:sequence>
              <s:any />
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:sequence>
    </s:complexType>

    <s:simpleType name="Universe">
      <s:restriction base="s:string">
        <s:enumeration value="ERROR" />
        <s:enumeration value="UNDEFINED" />
        <s:enumeration value="vanilla" />
        <s:enumeration value="standard" />
        <s:enumeration value="pvm" />
        <s:enumeration value="scheduler" />
        <s:enumeration value="globus" />

        <s:enumeration value="mpi" />
      </s:restriction>
    </s:simpleType>
    <s:complexType name="SubmissionJobGroupClassAd">
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="Input" nillable="true" type="tns:JobFile" />
        <s:element minOccurs="1" maxOccurs="1" name="Output" nillable="true" type="s:string" />
        <s:element minOccurs="1" maxOccurs="1" name="Error" nillable="true" type="s:string" />
        <s:element minOccurs="1" maxOccurs="1" name="Arguments" nillable="true" type="s:string" />

        <s:element minOccurs="0" maxOccurs="1" name="Priority" type="tns:Priority" />
        <s:element minOccurs="1" maxOccurs="1" name="Queue" type="s:unsignedLong" />
        <s:element minOccurs="0" maxOccurs="unbounded" name="Data" nillable="true" type="tns:JobFile" />
      </s:sequence>
    </s:complexType>
    <s:complexType name="Priority">
      <s:attribute name="priority" type="s:short" use="required" />
    </s:complexType>
    <s:element name="SubmitResponse">

      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="SubmitResult" type="tns:JobID" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="GetMachineStatuses">
      <s:complexType>
        <s:sequence>

          <s:element minOccurs="0" maxOccurs="1" name="machineNames" type="tns:ArrayOfString" />
          <s:element minOccurs="1" maxOccurs="1" name="query" type="tns:MachineStatusQuery" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:complexType name="ArrayOfString">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="string" nillable="true" type="s:string" />
      </s:sequence>

    </s:complexType>
    <s:simpleType name="MachineStatusQuery">
      <s:restriction base="s:string">
        <s:enumeration value="Available" />
        <s:enumeration value="Claimed" />
        <s:enumeration value="All" />
      </s:restriction>
    </s:simpleType>
    <s:element name="GetMachineStatusesResponse">

      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="GetMachineStatusesResult" type="tns:ArrayOfMachineClassAd" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:complexType name="ArrayOfMachineClassAd">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="MachineClassAd" nillable="true" type="tns:MachineClassAd" />

      </s:sequence>
    </s:complexType>
    <s:complexType name="MachineClassAd">
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="MyType" type="tns:MyType" />
        <s:element minOccurs="1" maxOccurs="1" name="TargetType" type="tns:TargetType" />
        <s:element minOccurs="0" maxOccurs="1" name="Name" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Machine" type="s:string" />
        <s:element minOccurs="1" maxOccurs="1" name="Rank" type="s:float" />

        <s:element minOccurs="0" maxOccurs="1" name="CpuBusy" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="CondorVersion" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="CondorPlatform" type="s:string" />
        <s:element minOccurs="1" maxOccurs="1" name="VirtualMachineID" type="s:int" />
        <s:element minOccurs="1" maxOccurs="1" name="ImageSize" type="s:int" />
        <s:element minOccurs="1" maxOccurs="1" name="JobUniverse" type="tns:Universe" />
        <s:element minOccurs="1" maxOccurs="1" name="VirtualMemory" type="s:unsignedInt" />
        <s:element minOccurs="1" maxOccurs="1" name="Disk" type="s:unsignedInt" />
        <s:element minOccurs="1" maxOccurs="1" name="CondorLoadAvg" type="s:float" />

        <s:element minOccurs="1" maxOccurs="1" name="LoadAvg" type="s:float" />
        <s:element minOccurs="1" maxOccurs="1" name="KeyboardIdle" type="s:long" />
        <s:element minOccurs="1" maxOccurs="1" name="ConsoleIdle" type="s:long" />
        <s:element minOccurs="1" maxOccurs="1" name="Memory" type="s:unsignedInt" />
        <s:element minOccurs="1" maxOccurs="1" name="Cpus" type="s:unsignedInt" />
        <s:element minOccurs="0" maxOccurs="1" name="StartdIpAddr" type="s:string" />
        <s:element minOccurs="1" maxOccurs="1" name="Arch" type="tns:Arch" />
        <s:element minOccurs="1" maxOccurs="1" name="OpSys" type="tns:OpSys" />
        <s:element minOccurs="0" maxOccurs="1" name="UidDomain" type="s:string" />

        <s:element minOccurs="0" maxOccurs="1" name="FileSystemDomain" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Subnet" type="s:string" />
        <s:element minOccurs="1" maxOccurs="1" name="TotalVirtualMemory" type="s:unsignedInt" />
        <s:element minOccurs="1" maxOccurs="1" name="TotalDisk" type="s:unsignedInt" />
        <s:element minOccurs="1" maxOccurs="1" name="KFlops" type="s:unsignedInt" />
        <s:element minOccurs="1" maxOccurs="1" name="Mips" type="s:unsignedInt" />
        <s:element minOccurs="1" maxOccurs="1" name="LastBenchmark" type="s:dateTime" />
        <s:element minOccurs="1" maxOccurs="1" name="TotalLoadAvg" type="s:float" />
        <s:element minOccurs="1" maxOccurs="1" name="TotalCondorLoadAvg" type="s:float" />

```

```
          <s:element minOccurs="1" maxOccurs="1" name="ClockMin" type="s:long" />
          <s:element minOccurs="1" maxOccurs="1" name="ClockDay" type="tns:ClockDay" />
          <s:element minOccurs="1" maxOccurs="1" name="TotalVirtualMachines" type="s:
     unsignedInt" />
          <s:element minOccurs="1" maxOccurs="1" name="CpuBusyTime" type="s:long" />
          <s:element minOccurs="1" maxOccurs="1" name="CpuIsBusy" type="s:boolean" />
          <s:element minOccurs="1" maxOccurs="1" name="State" type="tns:State" />
          <s:element minOccurs="1" maxOccurs="1" name="EnteredCurrentState" type="s:dateTime"
     />
          <s:element minOccurs="1" maxOccurs="1" name="Activity" type="tns:Activity" />
          <s:element minOccurs="1" maxOccurs="1" name="EnteredCurrentActivity" type="s:
     dateTime" />

          <s:element minOccurs="0" maxOccurs="1" name="Start" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="Requirements" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="CurrentRank" type="s:float" />
          <s:element minOccurs="1" maxOccurs="1" name="RemoteUser" nillable="true" type="s:
     string" />
          <s:element minOccurs="1" maxOccurs="1" name="RemoteOwner" nillable="true" type="s:
     string" />
          <s:element minOccurs="1" maxOccurs="1" name="ClientMachine" nillable="true" type="s:
     string" />
          <s:element minOccurs="1" maxOccurs="1" name="JobID" nillable="true" type="tns:JobID"
      />
          <s:element minOccurs="1" maxOccurs="1" name="JobStart" type="s:dateTime" />
          <s:element minOccurs="1" maxOccurs="1" name="LastPeriodicCheckpoint" type="s:
     dateTime" />

          <s:element minOccurs="1" maxOccurs="1" name="LastHeardFrom" type="s:dateTime" />
        </s:sequence>
      </s:complexType>
      <s:simpleType name="MyType">
        <s:restriction base="s:string">
          <s:enumeration value="ERROR" />
          <s:enumeration value="UNDEFINED" />
          <s:enumeration value="Job" />
          <s:enumeration value="Machine" />

        </s:restriction>
      </s:simpleType>
      <s:simpleType name="TargetType">
        <s:restriction base="s:string">
          <s:enumeration value="ERROR" />
          <s:enumeration value="UNDEFINED" />
          <s:enumeration value="Job" />
          <s:enumeration value="Machine" />
        </s:restriction>

      </s:simpleType>
      <s:simpleType name="Arch">
        <s:restriction base="s:string">
          <s:enumeration value="ERROR" />
          <s:enumeration value="UNDEFINED" />
          <s:enumeration value="INTEL" />
          <s:enumeration value="ALPHA" />
          <s:enumeration value="SGI" />
          <s:enumeration value="SUN4u" />

          <s:enumeration value="SUN4x" />
          <s:enumeration value="HPPA1" />
          <s:enumeration value="HPPA2" />
        </s:restriction>
      </s:simpleType>
      <s:simpleType name="OpSys">
        <s:restriction base="s:string">
          <s:enumeration value="ERROR" />
          <s:enumeration value="UNDEFINED" />

          <s:enumeration value="HPUX10" />
          <s:enumeration value="IRIX6" />
          <s:enumeration value="LINUX" />
          <s:enumeration value="OSF1" />
          <s:enumeration value="SOLARIS251" />
          <s:enumeration value="SOLARIS26" />
          <s:enumeration value="SOLARIS27" />
          <s:enumeration value="SOLARIS28" />
          <s:enumeration value="WINNT40" />

          <s:enumeration value="WINNT50" />
          <s:enumeration value="WINNT51" />
        </s:restriction>
      </s:simpleType>
      <s:simpleType name="ClockDay">
        <s:restriction base="s:string">
          <s:enumeration value="ERROR" />
          <s:enumeration value="UNDEFINED" />
          <s:enumeration value="Sunday" />

          <s:enumeration value="Monday" />
          <s:enumeration value="Tuesday" />
          <s:enumeration value="Wednesday" />
          <s:enumeration value="Thursday" />
          <s:enumeration value="Friday" />
          <s:enumeration value="Saturday" />
        </s:restriction>
      </s:simpleType>
      <s:simpleType name="State">

        <s:restriction base="s:string">
          <s:enumeration value="ERROR" />
          <s:enumeration value="UNDEFINED" />
          <s:enumeration value="Owner" />
          <s:enumeration value="Unclaimed" />
          <s:enumeration value="Matched" />
          <s:enumeration value="Claimed" />
          <s:enumeration value="Preempting" />
        </s:restriction>

      </s:simpleType>
      <s:simpleType name="Activity">
        <s:restriction base="s:string">
          <s:enumeration value="ERROR" />
          <s:enumeration value="UNDEFINED" />
          <s:enumeration value="Idle" />
          <s:enumeration value="Busy" />
          <s:enumeration value="Suspended" />
          <s:enumeration value="Vacating" />

          <s:enumeration value="Killing" />
          <s:enumeration value="Benchmarking" />
        </s:restriction>
      </s:simpleType>
      <s:element name="GetAllMachineStatuses">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="query" type="tns:MachineStatusQuery"
    />
          </s:sequence>

        </s:complexType>
      </s:element>
      <s:element name="GetAllMachineStatusesResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetAllMachineStatusesResult" type="
    tns:ArrayOfMachineClassAd" />
          </s:sequence>
        </s:complexType>
      </s:element>

      <s:element name="GetJob">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="query" type="tns:JobStatusQuery" />
            <s:element minOccurs="0" maxOccurs="1" name="id" type="tns:JobID" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:simpleType name="JobStatusQuery">

        <s:restriction base="s:string">
          <s:enumeration value="Running" />
          <s:enumeration value="All" />
        </s:restriction>
      </s:simpleType>
      <s:element name="GetJobResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetJobResult" type="tns:JobClassAd" /
    >
```

```xml

          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="JobClassAd">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="MyType" type="tns:MyType" />
          <s:element minOccurs="1" maxOccurs="1" name="TargetType" type="tns:TargetType" />
          <s:element minOccurs="0" maxOccurs="1" name="ID" type="tns:JobID" />
          <s:element minOccurs="1" maxOccurs="1" name="QDate" type="s:dateTime" />

          <s:element minOccurs="1" maxOccurs="1" name="CompletionDate" type="s:dateTime" />
          <s:element minOccurs="1" maxOccurs="1" name="Owner" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="NTDomain" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="LocalUserCpu" type="s:float" />
          <s:element minOccurs="1" maxOccurs="1" name="LocalSysCpu" type="s:float" />
          <s:element minOccurs="1" maxOccurs="1" name="RemoteUserCpu" type="s:float" />
          <s:element minOccurs="1" maxOccurs="1" name="RemoteSysCpu" type="s:float" />
          <s:element minOccurs="1" maxOccurs="1" name="ExitStatus" type="s:int" />
          <s:element minOccurs="1" maxOccurs="1" name="NumCkpts" type="s:unsignedInt" />

          <s:element minOccurs="1" maxOccurs="1" name="NumRestarts" type="s:unsignedInt" />
          <s:element minOccurs="1" maxOccurs="1" name="CommittedTime" type="s:dateTime" />
          <s:element minOccurs="1" maxOccurs="1" name="CondorVersion" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="CondorPlatform" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="Iwd" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="Universe" type="tns:Universe" />
          <s:element minOccurs="1" maxOccurs="1" name="Cmd" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="MinHosts" type="s:unsignedInt" />
          <s:element minOccurs="1" maxOccurs="1" name="MaxHosts" type="s:unsignedInt" />

          <s:element minOccurs="1" maxOccurs="1" name="WantRemoteSyscalls" type="s:boolean" />
          <s:element minOccurs="1" maxOccurs="1" name="WantCheckpoint" type="s:boolean" />
          <s:element minOccurs="1" maxOccurs="1" name="JobPrio" nillable="true" type="tns:Priority" />
          <s:element minOccurs="1" maxOccurs="1" name="NiceUser" type="s:boolean" />
          <s:element minOccurs="1" maxOccurs="1" name="Env" nillable="true" type="tns:Environment" />
          <s:element minOccurs="1" maxOccurs="1" name="EmailNotification" type="tns:EmailNotification" />
          <s:element minOccurs="1" maxOccurs="1" name="UserLog" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="CoreSize" type="s:unsignedInt" />
          <s:element minOccurs="1" maxOccurs="1" name="Rank" nillable="true" type="s:string" />

          <s:element minOccurs="1" maxOccurs="1" name="In" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="Out" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="Err" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="BufferSize" type="s:unsignedInt" />
          <s:element minOccurs="1" maxOccurs="1" name="BufferBlockSize" type="s:unsignedInt" />
          <s:element minOccurs="1" maxOccurs="1" name="TransferFiles" type="tns:TransferFiles" />
          <s:element minOccurs="1" maxOccurs="1" name="ImageSize" type="s:unsignedInt" />
          <s:element minOccurs="1" maxOccurs="1" name="ExecutableSize" type="s:unsignedInt" />
          <s:element minOccurs="1" maxOccurs="1" name="DiskUsage" type="s:unsignedInt" />

          <s:element minOccurs="1" maxOccurs="1" name="Requirements" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="Args" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="ProcId" type="s:int" />
          <s:element minOccurs="1" maxOccurs="1" name="User" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="OrigMaxHosts" type="s:unsignedInt" />
          <s:element minOccurs="1" maxOccurs="1" name="Status" type="tns:Status" />
          <s:element minOccurs="1" maxOccurs="1" name="CurrentHosts" type="s:unsignedInt" />
          <s:element minOccurs="1" maxOccurs="1" name="RemoteHost" nillable="true" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="ShadowBday" type="s:dateTime" />

          <s:element minOccurs="1" maxOccurs="1" name="JobStartDate" type="s:dateTime" />
          <s:element minOccurs="1" maxOccurs="1" name="ServerTime" type="s:dateTime" />
        </s:sequence>
      </s:complexType>
```

```
  <s:simpleType name="TransferFiles">
    <s:restriction base="s:string">
      <s:enumeration value="ERROR" />
      <s:enumeration value="UNDEFINED" />
      <s:enumeration value="ONEXIT" />

      <s:enumeration value="ALWAYS" />
    </s:restriction>
  </s:simpleType>
  <s:simpleType name="Status">
    <s:restriction base="s:string">
      <s:enumeration value="ERROR" />
      <s:enumeration value="UNDEFINED" />
      <s:enumeration value="idle" />
      <s:enumeration value="running" />

      <s:enumeration value="removed" />
      <s:enumeration value="completed" />
      <s:enumeration value="held" />
    </s:restriction>
  </s:simpleType>
  <s:element name="GetAllJobs">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="query" type="tns:JobStatusQuery" />

      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="GetAllJobsResponse">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="GetAllJobsResult" type="tns:
ArrayOfJobClassAd" />
      </s:sequence>
    </s:complexType>

  </s:element>
  <s:complexType name="ArrayOfJobClassAd">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="JobClassAd" nillable="true"
type="tns:JobClassAd" />
    </s:sequence>
  </s:complexType>
  <s:element name="GetJobs">
    <s:complexType>
      <s:sequence>

        <s:element minOccurs="1" maxOccurs="1" name="query" type="tns:JobStatusQuery" />
        <s:element minOccurs="0" maxOccurs="1" name="ids" type="tns:ArrayOfJobID" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:complexType name="ArrayOfJobID">
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="JobID" nillable="true" type="
tns:JobID" />
    </s:sequence>

  </s:complexType>
  <s:element name="GetJobsResponse">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="GetJobsResult" type="tns:
ArrayOfJobClassAd" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="RemoveJob">

    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="id" type="tns:JobID" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="RemoveJobResponse">
    <s:complexType>
      <s:sequence>

```

```
            <s:element minOccurs="1" maxOccurs="1" name="RemoveJobResult" type="tns:
    JobRemoveStatus" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:simpleType name="JobRemoveStatus">
        <s:restriction base="s:string">
          <s:enumeration value="MarkedForRemoval" />
          <s:enumeration value="JobAlreadyTerminated" />
          <s:enumeration value="DoesNotExist" />

          <s:enumeration value="CondorFailure" />
        </s:restriction>
      </s:simpleType>
      <s:element name="RemoveJobs">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="ids" type="tns:ArrayOfJobID" />
          </s:sequence>
        </s:complexType>

      </s:element>
      <s:element name="RemoveJobsResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="RemoveJobsResult" type="tns:
    ArrayOfJobRemoveStatus" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfJobRemoveStatus">

        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="JobRemoveStatus" type="tns:
    JobRemoveStatus" />
        </s:sequence>
      </s:complexType>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="GetFilesSoapIn">
    <wsdl:part name="parameters" element="tns:GetFiles" />
  </wsdl:message>

  <wsdl:message name="GetFilesSoapOut">
    <wsdl:part name="parameters" element="tns:GetFilesResponse" />
  </wsdl:message>
  <wsdl:message name="WaitSoapIn">
    <wsdl:part name="parameters" element="tns:Wait" />
  </wsdl:message>
  <wsdl:message name="WaitSoapOut">
    <wsdl:part name="parameters" element="tns:WaitResponse" />
  </wsdl:message>

  <wsdl:message name="SubmitSoapIn">
    <wsdl:part name="parameters" element="tns:Submit" />
  </wsdl:message>
  <wsdl:message name="SubmitSoapOut">
    <wsdl:part name="parameters" element="tns:SubmitResponse" />
  </wsdl:message>
  <wsdl:message name="GetMachineStatusesSoapIn">
    <wsdl:part name="parameters" element="tns:GetMachineStatuses" />
  </wsdl:message>

  <wsdl:message name="GetMachineStatusesSoapOut">
    <wsdl:part name="parameters" element="tns:GetMachineStatusesResponse" />
  </wsdl:message>
  <wsdl:message name="GetAllMachineStatusesSoapIn">
    <wsdl:part name="parameters" element="tns:GetAllMachineStatuses" />
  </wsdl:message>
  <wsdl:message name="GetAllMachineStatusesSoapOut">
    <wsdl:part name="parameters" element="tns:GetAllMachineStatusesResponse" />
  </wsdl:message>

  <wsdl:message name="GetJobSoapIn">
    <wsdl:part name="parameters" element="tns:GetJob" />
  </wsdl:message>
  <wsdl:message name="GetJobSoapOut">
    <wsdl:part name="parameters" element="tns:GetJobResponse" />
  </wsdl:message>
  <wsdl:message name="GetAllJobsSoapIn">
```

```
    <wsdl:part name="parameters" element="tns:GetAllJobs" />
  </wsdl:message>

  <wsdl:message name="GetAllJobsSoapOut">
    <wsdl:part name="parameters" element="tns:GetAllJobsResponse" />
  </wsdl:message>
  <wsdl:message name="GetJobsSoapIn">
    <wsdl:part name="parameters" element="tns:GetJobs" />
  </wsdl:message>
  <wsdl:message name="GetJobsSoapOut">
    <wsdl:part name="parameters" element="tns:GetJobsResponse" />
  </wsdl:message>

  <wsdl:message name="RemoveJobSoapIn">
    <wsdl:part name="parameters" element="tns:RemoveJob" />
  </wsdl:message>
  <wsdl:message name="RemoveJobSoapOut">
    <wsdl:part name="parameters" element="tns:RemoveJobResponse" />
  </wsdl:message>
  <wsdl:message name="RemoveJobsSoapIn">
    <wsdl:part name="parameters" element="tns:RemoveJobs" />
  </wsdl:message>

  <wsdl:message name="RemoveJobsSoapOut">
    <wsdl:part name="parameters" element="tns:RemoveJobsResponse" />
  </wsdl:message>
  <wsdl:portType name="CompWSSoap">
    <wsdl:operation name="GetFiles">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Gets files from
     completed jobs. Blocks if job not finished.</wsdl:documentation>
      <wsdl:input message="tns:GetFilesSoapIn" />
      <wsdl:output message="tns:GetFilesSoapOut" />

    </wsdl:operation>
    <wsdl:operation name="Wait">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Waits until selected
     user job has completed and returns true if it completes.</wsdl:documentation>
      <wsdl:input message="tns:WaitSoapIn" />
      <wsdl:output message="tns:WaitSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="Submit">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Submits a job to the
     Condor job queue and returns job ID.</wsdl:documentation>

      <wsdl:input message="tns:SubmitSoapIn" />
      <wsdl:output message="tns:SubmitSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetMachineStatuses">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Returns
     MachineClassAds for machines.</wsdl:documentation>
      <wsdl:input message="tns:GetMachineStatusesSoapIn" />
      <wsdl:output message="tns:GetMachineStatusesSoapOut" />
    </wsdl:operation>

    <wsdl:operation name="GetAllMachineStatuses">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Returns
     MachineClassAds for machines.</wsdl:documentation>
      <wsdl:input message="tns:GetAllMachineStatusesSoapIn" />
      <wsdl:output message="tns:GetAllMachineStatusesSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetJob">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Returns JobClassAd for
      user's jobs.</wsdl:documentation>
      <wsdl:input message="tns:GetJobSoapIn" />

      <wsdl:output message="tns:GetJobSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetAllJobs">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Returns JobClassAds
     for all user's jobs.</wsdl:documentation>
      <wsdl:input message="tns:GetAllJobsSoapIn" />
      <wsdl:output message="tns:GetAllJobsSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetJobs">

      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Returns JobClassAds
     for user's jobs.</wsdl:documentation>
      <wsdl:input message="tns:GetJobsSoapIn" />
      <wsdl:output message="tns:GetJobsSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="RemoveJob">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Removes user's job and
      returns a bool to determine if operation was succesful</wsdl:documentation>
      <wsdl:input message="tns:RemoveJobSoapIn" />
      <wsdl:output message="tns:RemoveJobSoapOut" />

    </wsdl:operation>
    <wsdl:operation name="RemoveJobs">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Removes user's jobs
     and returns a bool array to determine if operation was succesful</wsdl:documentation>
      <wsdl:input message="tns:RemoveJobsSoapIn" />
      <wsdl:output message="tns:RemoveJobsSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CompWSSoap" type="tns:CompWSSoap">

    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetFiles">
      <soap:operation soapAction="http://www.geodise.org/CompWS/GetFiles" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>

    </wsdl:operation>
    <wsdl:operation name="Wait">
      <soap:operation soapAction="http://www.geodise.org/CompWS/Wait" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>

    </wsdl:operation>
    <wsdl:operation name="Submit">
      <soap:operation soapAction="http://www.geodise.org/CompWS/Submit" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>

    </wsdl:operation>
    <wsdl:operation name="GetMachineStatuses">
      <soap:operation soapAction="http://www.geodise.org/CompWS/GetMachineStatuses" style="
     document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>

    </wsdl:operation>
    <wsdl:operation name="GetAllMachineStatuses">
      <soap:operation soapAction="http://www.geodise.org/CompWS/GetAllMachineStatuses" style="
     document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>

    </wsdl:operation>
    <wsdl:operation name="GetJob">
      <soap:operation soapAction="http://www.geodise.org/CompWS/GetJob" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>

    </wsdl:operation>
    <wsdl:operation name="GetAllJobs">
```

```
      <soap:operation soapAction="http://www.geodise.org/CompWS/GetAllJobs" style="document"
    />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>

    </wsdl:operation>
    <wsdl:operation name="GetJobs">
      <soap:operation soapAction="http://www.geodise.org/CompWS/GetJobs" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>

    </wsdl:operation>
    <wsdl:operation name="RemoveJob">
      <soap:operation soapAction="http://www.geodise.org/CompWS/RemoveJob" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>

    </wsdl:operation>
    <wsdl:operation name="RemoveJobs">
      <soap:operation soapAction="http://www.geodise.org/CompWS/RemoveJobs" style="document"
    />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>

    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CompWSSoap12" type="tns:CompWSSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetFiles">
      <soap12:operation soapAction="http://www.geodise.org/CompWS/GetFiles" style="document"
    />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>

      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Wait">
      <soap12:operation soapAction="http://www.geodise.org/CompWS/Wait" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>

      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Submit">
      <soap12:operation soapAction="http://www.geodise.org/CompWS/Submit" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>

      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetMachineStatuses">
      <soap12:operation soapAction="http://www.geodise.org/CompWS/GetMachineStatuses" style="
    document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>

      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetAllMachineStatuses">
      <soap12:operation soapAction="http://www.geodise.org/CompWS/GetAllMachineStatuses" style
      ="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>

      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetJob">
      <soap12:operation soapAction="http://www.geodise.org/CompWS/GetJob" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>

      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetAllJobs">
      <soap12:operation soapAction="http://www.geodise.org/CompWS/GetAllJobs" style="document"
      />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>

      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetJobs">
      <soap12:operation soapAction="http://www.geodise.org/CompWS/GetJobs" style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>

      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="RemoveJob">
      <soap12:operation soapAction="http://www.geodise.org/CompWS/RemoveJob" style="document"
      />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>

      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="RemoveJobs">
      <soap12:operation soapAction="http://www.geodise.org/CompWS/RemoveJobs" style="document"
      />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>

      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CompWS">
    <wsdl:port name="CompWSSoap" binding="tns:CompWSSoap">
      <soap:address location="http://localhost/ComputationService/CompWS.asmx" />
    </wsdl:port>

    <wsdl:port name="CompWSSoap12" binding="tns:CompWSSoap12">
      <soap12:address location="http://localhost/ComputationService/CompWS.asmx" />
    </wsdl:port>
  </wsdl:service>
```

```
</wsdl:definitions>
```

Listing 3: WSDL Interface of the Compute Web Service