A mechanism for creating web service interface to scientific applications.

Chen, Yu

Swansea University

School of Engineering

A Mechanism for Creating Web Service Interface to Scientific Applications

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in the School of Engineering of the

Swansea University

December 2007

Yu Chen

B.Sc., M.Sc.

Swansea University
Prifysgol Abertawe

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Declaration

I declare that this work has not already been accepted in substance for any degree, and is not being concurrently submitted in candidature for any degree

.....(Candidate)

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

.....(Candidate)

Statement 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

.....(Candidate)

09/11/2008.....Date

Acknowledgement

With a debt of gratitude, which cannot be adequately expressed in words, I thank my supervisor Prof. Oubay Hassan for his advice, guidance, and endless support during my research. His practical and sharp vision in research has not only been invaluable for my work on this thesis but also for my development as a researcher. In the past four years, engaging in any discussion with my supervisor has been an enjoyable practical lesson, where professionalism, devotion in duty, gentleness, assured care, and a feel of protection are abound. Thanks for his friendship and for being such an excellent listener and all for the financial support he has provided. I have been very fortunate to work with him.

I would like to extend my sincere gratitude to my second supervisor Prof. Nigel Weatherill for all the unforgettable generous financial support throughout my PhD program. Sincere thanks to Prof. Ken Morgan for his support on my thesis corrections, and to Dr. Jason Jones for many encouraging and exciting discussions.

I am deeply indebted to Prof. Derek Hill from UCL, Prof. Jo Hajnal from Imperial College and Prof. Daniel Rueckert from Imperial College for their guidance, insightful suggestions, and encouragements. I am indebted to them for the countless stimulating and fruitful discussions, which have trained me in research.

This study has greatly benefited from discussions with a number of colleagues at Swansea University, University College London, Imperial College and University of Oxford. In particular, I am grateful to the discussions and guidance of Dr. Ali Asghar Khanban and Dr. Gianlorenzo Fagiolo.

Thanks to Dr. I. J. Keshtiban, Dr. M. J. Banaie and Mr. Q. Hassan for their proof reading.

Last and certainly not the least, I wish to thank my family without whose moral support and encouragement this PhD would have been far from completion. I am and will ever remain indebted to them.

To my parents and other family members with loyalty and love.....

Abstract

Science is becoming increasingly multi-disciplinary and complicated. To solve complex scientific problems, we often need to integrate software and customize workflows to suit a particular problem. To make progress on key scientific issues, extended scientific collaborations are growingly dependent on complex workflows for data analysis and simulation.

Service Oriented Architecture has gained popularity in recent years within scientific research community. It has been broadly accepted as a means of structuring interactions among distributed software services. Service Oriented Architecture is a new paradigm for accessing, integrating and coordinating loosely coupled software systems in a standardized way. It aims to reduce the cost of building and maintaining complex software systems while increasing their re-usability.

However, most of the large industrial and scientific applications available today were written well before the introduction of Grid computing and Service Oriented Architectures. Hence, they are neither service oriented nor Grid-aware. There is a growing need to integrate them into Grid-aware applications based on Service Oriented Architecture. How to integrate these legacy applications into the Grid with the least possible effort and the best performance has become a crucial point.

The majority of the applications developed and used by scientific communities are command-line applications. They are written in FORTRAN, C, and a host of scripting languages. In addition to being fast and efficient, these applications represent state-of-the-art science; however, they are bound by many limitations which make it difficult to compose complex workflows from them and run them on a distributed set of resources. By converting these command-line legacy applications into Application Services, it

becomes easy to compose complex workflows from them and run them on the distributed resources.

There are some research programs aiming at integrating the legacy codes into Grid infrastructure. Some frameworks have been developed to compose and run scientific workflows on a Grid. A number of systems are available to allow scientists to Grid-enabling their existing applications without having to write extra code or modify their applications. But most of them do not provide a toolkit for wrapping an application as a Grid-aware Web service. Few of the systems have addressed the issue of security. This thesis presents an approach to reducing the required effort needed in developing Application Services for end users.

Also during the execution of complex scientific workflows, Application Services often become unavailable primarily due to the unreliable nature of the resources that host them. When an Application Service becomes unavailable, all workflows that are accessing it have to stop, and this means wasting a great deal of time and resources. This thesis offers a new solution to this problem, via providing a mechanism by which Application Service can be created on-demand from workflows in case it is unavailable.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Motivation

Science is becoming increasingly multi-disciplinary. To solve complex scientific problems, scientists often need to integrate software and customize workflows to suit a particular case. For example, Electromagnetic modelling employs Computer Aided Design to generate parametric design geometries, mesh generation software to discretise the flow domain, and Computational Electromagnetic solvers to obtain a solution [1]. In imaging studies, researchers tend to combine one or more image processing algorithms to form sequential image processing pipelines. For example, in tracking the progress of a brain tumour, the researchers may first segment the area of interest. Then, this segmentation would be used as an input to a series of rigid registrations that produce a set of transformations which could then be passed to transformation algorithm [2]. These large scientific collaborations not only highly demand data, applications, and compute resources sharing seamlessly, but also require several teams of specialists to work together closely, including the specialists in some specific fields, computer scientists, and compute resource providers.

To achieve the large scientific collaboration two significant barriers must be overcame. One barrier is introduced by heterogeneous nature of communities' resources including data, applications, and compute resources. The other barrier is introduced by legacy command-line applications. Most of the scientific applications are command-line applications, which have several limitations although they are fast, efficient and represent state-of-the-art science. First, it is not easy to solve computational challenges for the legacy applications. Traditionally, it is a long tedious task to perform a scientific computation on the resources provided by third party resources providers. Users interact with these computation resources at an absolutely fundamental level. Secondly, the

legacy applications are often used by internal research groups. They are usually run non-interactively without any user interaction and monitoring support, which makes them difficult to be steered by the users. Thirdly, the traditional scientific applications are not easy for collaborations. Most of the applications are often platform dependent and are difficult to integrate with the applications from other disciplines. Also they are command-line oriented. There is no generic method to describe their input parameters and output results. It is usually difficult to programmatically access these applications remotely.

Adaptable and flexible integration frameworks are highly required to meet challenges encountered in scientific problem solving environment - increasing complexity and large number of data, applications, resources, and researchers involved. Service Oriented Architecture on Grid infrastructure should be exploited to enhance the large scientific collaboration.

Conventional distributed computing could often assume to provide homogeneous computational power to solve computationally intensive problems. However, Grid is characterised by facilitating dynamic, flexible, secure and cross-organizational sharing of heterogeneous resources among dynamic collections of individuals in a transparent way. It is identified as the most promising infrastructure to provide resources for computationally extensive applications and storages for large-scale datasets. In particular, the Grid handles issues of authentication, authorisation, resource description and location, data transfer and resource accounting. This makes Grid technologies extremely useful to facilitate sharing of the resources across a distributed environment.

Service Oriented Architecture (SOA) is a computer systems architectural style for creating and using business processes, packaged as services, throughout their lifecycle. SOA also defines and provisions the IT infrastructure to allow different applications to exchange data and participate in business processes. These functions are loosely coupled with the operating systems and programming languages underlying the applications [3]. SOA is a component model that inter-relates different functional units of an application, called services, through well-defined interfaces and contracts between these services. The interface is defined in a neutral manner that should be independent of hardware platform, operating system, and programming language the service is

implemented in. This allows the services, built on a variety of such systems, to interact with each other in a uniform and universal manner.

This feature of having a neutral interface definition that is not strongly tied to a particular implementation is known as loose coupling between services. The benefit of a loosely-coupled system lies in its agility and ability to survive evolutionary changes in structure and implementation of internals of each service that make up the whole application. Tight-coupling on the other hand, means that the interfaces between the different components of an application are tightly interrelated in function and form, thus making them brittle when any form of change is required to parts or the whole application.

SOA have gained popularity in recent years within scientific Grid research community. It has been broadly accepted as a means of structuring interactions among distributed software services. Within Service Oriented Architecture all the resources (e.g., computational resources, data, and programs) are exposed as services. Access to these services is possible via generic interface definition mechanisms and thus allowing a transparent and uniform access to a range of distributed and heterogeneous resources (encapsulated behind the service definition).

However, most of the large industrial and scientific applications available today were composed well before Grid computing and Service Oriented Architecture appeared. Hence, they are neither service oriented nor Grid-aware. There is a growing need to integrate them into the Grid infrastructure based on Service Oriented Architecture. How to integrate these legacy applications into Grid with the least possible effort and the best performance becomes a crucial point. By converting these command-line legacy applications into Application Services, it becomes easy to compose complex workflows from them and run them on the distributed resources. In this present work, an Application Service is a Grid-aware application with a Web service interface that is described by Web Service Definition Language (WSDL).[4] The Application Service makes the application available through a Web service interface. The Application Service is Grid-aware, which means it can run the computation in the Grid environment. When a user invokes an Application Service with some input parameters, the service runs the application that it wraps on the distributed compute resource with those input parameters, monitors it and returns its output results. Providing the Web service

interface for the legacy applications alleviates many problems raised by the legacy applications mentioned above.

There are some research programs aiming at integrating legacy codes into the Grid infrastructure. Some frameworks have been developed to compose and run scientific workflows on a Grid. Some systems are available to allow scientists to Grid-enabling their existing applications without having to write extra code or modify their applications. Some tools are available to automate tasks of transforming the existing applications into Web services. But most of them do not provide a toolkit for wrapping an application as a Grid-aware Web service. Simply providing a Web service interface is not sufficient to make the application a usable component in a distributed computation. A toolkit is highly demanded to tackle existing problems, and automatically wrap the scientific applications as Grid-aware Web services without having to deal with the details of Web services technologies. Following are challenges the toolkit is facing:

Firstly, how can the toolkit make Application Services Grid-aware? Grid-aware means the Application Services can use distributed computational resources to run the applications. Our goal is to leverage the set of computational resources available across different sites on the Grid. So it is crucial for the Application Services to have ability to assess the remote computational resources. However, clusters at different sites run schedulers of their choice, such as Condor [5] and Sun Grid Engine [6]. Thus, it is mandatory that the Application Services provide the support to access any of existing schedulers and schedulers to appear in future. To end users these schedulers are accessible in a generic way, which means the users are able to access various schedulers via the Application Services in a transparent uniform fashion (not scheduler specific).

Another major concern is security. How does the Application Service allow users of a community run the scientific application on computing clusters provided by compute resource providers without having login accounts on the computing clusters? How can a service provider authorize a selected group of users with access to the Application Service without building a separate security infrastructure and without requiring the users to have login accounts on the machine hosting the Application Service?

Another challenge is scalability. For most of existing wrapping tools, service providers need to wrap the legacy applications offline and host them as persistent

services so they can be accessed from scientific workflows whenever needed. To this approach, scalability becomes one of the toughest challenges for large scientific collaborations. Sometimes a service provider needs to wrap and host hundreds of Application Services just for one scientific package, since it is not uncommon that many of scientific packages have hundreds of applications. Also whenever a new application is added into the framework, the service provider needs to develop and deploy a new Application Service.

Yet another challenge is reliability. In a Grid environment, Application Services running on a Grid, often become unavailable primarily due to unreliable nature of the Grid. Sometimes even though an application service may be available, it may not be usable because it may not meet some quality of service requirements. Under such circumstances, all workflows that are accessing that Application Service have to be stopped and can be resumed only after that Application Service becomes available. During the execution of complex workflows over a period of several hours, Application Service downtime could result in a considerable waste of time and resources. This is an important and widespread problem in large scientific communities that is intended to be addressed and solved in this thesis.

As explained in [7] (Taverna) , service failure is more complex and more likely than other failures. Fault tolerance mechanisms such as dynamic service substitution and retry are supported by Taverna. If a service failed because the machine it runs on is down, it is a candidate to be retried. If the service failed because the input data was invalid, it is inappropriate to try again. In addition to simply retrying the service invocation, it may be possible to locate an alternate service to invoke should the original service fail. In reality, only identical services running on an alternate service provider is deemed by users to be acceptably interchangeable [7]. Similar to Taverna, other current workflow systems, Triana [8] and Kepler [9], have tried to solve this problem by allowing the scientist to specify redundant Application Services for all the Application Services in the workflow.

However, these redundant services must be running at the time of workflow execution and suffer the same problem of unreliability as the primary copy. Moreover, in large scientific collaborations, owing to the large number of Application Services, providing redundancy consumes considerable resources. It is unrealistic to keep a large

number of Application Services persistent without a huge commitment in the form of resources and support infrastructure.

So, how is possible to provide a high availability of Application Services without actually requiring them to be persistent? How can the Application Service be created on demand in the event it is not kept persistent or is unavailable during the execution of a scientific workflow? How can the Application Service be created on demand in a way that is completely transparent to the user?

The several challenges that we have discussed so far are summarized below:

How to make Application Services Grid-aware? This means how to make the Application Services use distributed computational resources to run the applications. How to make the Application Services easy to be extended to support any of existing schedulers and schedulers to be appeared in future?

How to authorize a selected group of users with access to the Application Services without building a separate security infrastructure? How to allow the users to submit jobs to the resources provided by third-party without requiring the users to have login accounts on the computation resources and on the machine hosting the service?

How to provide a large number of applications as Application Services without problems of updating and maintaining source codes and deployments of all the Application Services?

How to create the Application Service on demand in the event it is not kept persistent or is unavailable during the execution of a scientific workflow?

1.2 Contributions

This thesis addresses all the above challenges and thus makes the following contributions:

i) An XML based language, Command-line Description Language (CoLDeL), to describe individual command-line application precisely.

ii) An Application Service Toolkit (ASToolkit) that is simple to wrap a large number of applications as Application Services, without problems of updating and maintaining source codes and deployments of all the Application Services.

iii) A Component plug-in mechanism that allows Application Services to be configured (at deployment time) with a Job Submission Component capable of interacting with the available resources.

iv) A WS-Security [10] based authorization mechanism by which service providers can control what users can invoke on their Application Services to run applications.

v) An Abstract Application Service (AAS) mechanism to create a specific Application Service on demand in the event it is not kept persistent or is unavailable during the execution of a scientific workflow. AAS is a generic Application Service. Scalability of this AAS mechanism is achieved by delivering applications through a dynamically reconfigurable AAS. This mechanism eliminates the need to keep all available applications wrapped as persistent Application Services.

vi) An overall framework for enabling the legacy applications and data on the Grid based Service Oriented Architecture.

vii) A Group Applications Service (GAS) mechanism to further optimize execution time of a workflow. GAS merges several Application Services into a single group service. It reduces the Grid overhead induced by the Web service invocation, scheduling, and data transfers.

viii) A mechanism to monitor and restart jobs.

CHAPTER 2

Legacy Applications Wrapping: Background and Related Work

2.1 Introduction

Nowadays, in every scientific domain, investigating complex phenomena requires great vast scientific collaborations. The reason for this is tied to the fact that science is becoming more multidisciplinary. Key progresses on scientific realms, are becoming increasingly dependent upon complex workflows of data analysis and simulation tasks. These workflows involve integration of many complex applications, each of which may be understood by only a limited number of specialists.

These legacy applications typically represent high quality and validated software. These applications that are developed by different teams of researchers are required to be integrated together to offer solution to large-scale scientific and engineering problems. Even each application requires aggregation and coordinated application of many widely distributed computing, libraries, and other resources.

Most legacy applications developed and used by scientific communities are command-line applications. They are written in FORTRAN, C, and some scripting languages. Traditional scientific applications have the following drawbacks which makes them difficult to be integrated together to solve large-scale problems.

2.1.1 Computational Challenges

High CPU/memory intensive scientific applications require access to high-end compute resources. Conventionally, to work out some complicated engineering or scientific problem, self-owned compute resources can not satisfy demand for compute power. Researchers either make some unnecessary hardware investments or outsource processing to external compute resources. This results in the current infrastructure being underutilized. A novel innovation is expected to meet constant demand for the compute power with the reality of underutilized resources.

Traditionally, it is a long tedious task to perform a scientific computation on resources provided by third-party resource providers. Users of high-performance computational resources have interacted with those resources at very rudimentary level – they obtain authorization (i.e. an account) and some amount of allocation, then they log in and interact with the resource through a low level interface (e.g. a command-line shell or ftp client) [11]. They launch an application by submitting a request to a scheduler or a queuing system. To perform one simulation job, the users have to open a new user account, recompile simulation codes, and learn different job schedulers on different resources. The users have to log in many times through the secure shell to run different jobs, and manually transfer input/output files. This hands-on procedure can be both time consuming and error prone.

This traditional approach to access the high-performance computational resources is not user-friendly. Hence, it is difficult to use. It suffers from a number of drawbacks. First, compute resources broker does not exist to help the users in choosing suitable resources. Second, low-level interfaces have a very steep learning curve, placing a cumbersome burden on the users to learn how to use the resources.Thirdly, inadequate job monitor and job management functionalities are provided to facilitate the users to track the jobs submitted. Fourthly, resource providers have to set up and maintain state (typically an account) for each user, which can be such a hassle for big communities.

2.1.2 Not Enough User Interaction

Traditionally, compute-intensive simulations are run non-interactively. Many of them lack a Graphical User Interface (GUI), which makes them difficult to be invoked by end users. Usually initial conditions and configuration parameters are recorded in a text based file format. The simulations read in input files and output results as files. The

whole large scaled simulations run without any user interaction during a job running process. The user only examines simulation results once the whole job is done.

However, if the result occurs in the early stages when simulation is not satisfied, the rest of the compute time will be spent on simulating something of absolutely insignificant interest. Even worse, if the initial parameters of the job fail to produce any meaningful useful results, then all of the CPU time spent on the simulation will be wasted. This waste can be avoided if the user interacts with the simulation job and checks the results on time. Also it is preferable if the user can steer the simulation by adjusting the parameters set [12].

2.1.3 Hard for Collaborations

Multi-disciplinary nature of engineering and scientific problems requires integrated applications developed by different research group and multi-source data from multiple data repositories. This highly demands collaborations among scientists and sharing of applications, data, and compute resources.

However, most of the applications are often platform dependent and are difficult to integrate with the applications from other disciplines. Also they are command-line oriented. There is no generic method to describe their input parameters and output results. It is usually a tough challenge to programmatically access these applications remotely.

Many applications have internal and external users worldwide. Though it does seem uncommon to travel around to have collaborations, it sounds like such a waste of both travel cost and time for scientists.

The conventional situation brings following challenging issues to communities:

i) How to share compute resources across organisations to satisfy demand of CPU/memory intensive scientific applications?

ii) How to provide interactive services to let users easily control and steer applications?

iii) How to build a collaborative problem solving environment to facilitate scientific collaborations across distributed organizations?

2.2 Grid

Grid, as explained by Ian Foster and Carl Kesselman, should enable "resource sharing and coordinated problem solving in dynamic, multi-institutional Virtual Organizations" [13]. By enabling the use of teraflop computers and petabyte storage systems interconnected by gigabit networks, Grid enables scientists to explore new avenues of research via conventional computing resources. Grid differs from other computational resources such as traditional supercomputers and clusters in the following key features: First, Grid coordinates resources not subjecting to centralized control. Second, Grid uses standard, open, general purpose protocols and interfaces. Third, Grid delivers non-trivial qualities of service.

Grid computing is akin to distributed computing, yet with a major focus on collaborations, data sharing, and interactions on a global scale, and delivering heterogeneous computational power to applications in a transparent manner. Grid ensures to get the most out of global compute resources. The Grid infrastructure is scalable and we can seamlessly add extra CPU power or other resources as required. In a Grid environment, the computation can be distributed across the global resources to achieve dramatic speed-up. Grid, offering flexible and secure sharing of resources, can couple applications and compute resources together under multiple ownership; hence, fully rise to the first challenge mentioned above - sharing compute resources across organizations.

[14] describes the most important capabilities of Grid: exploiting underutilized resource, parallel CPU capacity, Grid-enabled applications, virtual resource and virtual organizations for collaboration, access to additional resources, resource balancing, and increasing reliability relying on software instead of hardware.

2.3 Service Oriented Architecture

Service Oriented Architecture (SOA), can be regarded as a style of information systems architecture that enables the creation of applications that are built by combining loosely coupled and interoperable services [15]. SOA separates functions into distinct units (services), which can be distributed over a network and can be combined and reused to create business applications [16]. These services communicate with each other

by passing data from one service to another, or by coordinating an activity between two or more services.

These services inter-operate based on a formal definition (or contract, e.g., WSDL) that is independent of the underlying platform and programming language. The services have generic interface definition and can be accessed in a transparent and uniform way. The interface definition hides the implementation of the language-specific service. SOA-based systems can therefore be independent of development technologies and platforms. Application developers or system integrators can build applications by composing one or more services without knowing the services' underlying implementations.

Service Oriented Architecture is based on request/reply design paradigm for synchronous and asynchronous applications. Within a Service Oriented Architecture all resources including data and application's business logics or individual functions are modularized and presented as services for consumer/client applications. These distributed heterogeneous resources are encapsulated behind the service definition.

SOA has the following advantages:

i) *Uniform service semantics:* SOA services have self-describing interfaces in platform-independent XML documents.

ii) *Standard invocation mechanisms:* Usually SOA services communicate with messages formally defined via XML Schema.

iii) *Local/remote location transparency*

iv) *Interface level:* Service composition is based on compatibility at an interface level rather than an implementation level.

These advantages not only let users interact with the services easily, but also facilitate scientific applications collaborations across distributed organizations. This means that SOA offers a solution for the challenge two and the challenge three in the conventional situation.

SOA can be evolved based on existing system rather than requiring a full-scale system rewrite. As described in [17], we can realize following benefits if we focus our effort on the creation of services and applying existing techniques:

i) *Leverage existing applications:* An application can be constructed as an aggregation of existing components, via a suitable SOA framework which is made available to the community. To be able to use this new service, one requires only knowing the interface and the name. Internals of the service are hidden from the outside world, as well as the complexities of the data flow through the components that make up the service.

ii) *Consistent infrastructure:* Infrastructure development and deployment will become more consistent across all the different applications. Existing applications and newly-developed applications can be consolidated within a well-defined SOA framework.

iii) *Reduced cost:* As demands for applications evolve and new requirements are introduced, the cost of enhancing and creating new services by adapting the SOA framework and the services library, for both existing and new applications, is greatly reduced.

iv) *Continuous applications improvement:* SOA allows a clear representation of process flows. Service composition is based on compatibility at an interface level rather than an implementation level. This would allow for changing the application while keeping the same interface, and thus facilitates continuous applications improvement.

v) *Process-centric architecture:* In a process-centric architecture, the application is developed for the process. The process is decomposed into a series of steps, each representing a service. In effect, each service or component functions as a sub-application. These sub-applications are chained together to create a process flow capable of satisfying the users' need. This granularity lets processes leverage and reuse each sub-application.

Web service technology can be used as a basis for SOA. The World Wide Web Consortium (W3C) defines a Web service as below. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using

SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [18].

A further descriptive definition can be found in [19], where a Web service is defined as a platform and implementation independent software component that can be described using a service description language, published to a registry of services, discovered through generic mechanisms, invoked through a declared API, usually over a network, and composed with other services.

Due to widespread adoption of the Web service technologies, many standard protocols and tools have been defined and implemented and are available for use. Simple Object Access Protocol (SOAP) provides an XML-based messaging protocol between service providers and requestors to allow for applications exchange information over HTTP [20]. It is a lightweight protocol for information exchange in a decentralized, distributed environment. It consists of three parts: a) an envelope that defines a framework for describing what is in a message and how to process it, b) a set of encoding rules for expressing instances of application-defined data types, and c) a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols.

Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information [21]. It provides a way to describe and access Web services. The power of WSDL is that it expresses a program's interface in language-neutral XML syntax. WSDL does not directly enable remote function invocation, but does describe how to bind a particular interface to one or more remote invocation protocols.

Universal Description, Discovery, and Integration (UDDI) is the most widely recognized mechanism for publishing and discovery the Web services. It is an XML-based registry for businesses worldwide to list themselves on the Internet. Its ultimate goal is to streamline online transactions by enabling clients to find one another on the Web and make their systems interoperable [22].

Business Process Execution Language for Web Services (BPEL) provides a means to formally specify business processes and interaction protocols. BPEL provides a language for formal specifications of business processes and business interaction

protocols [23]. As such, it extends the Web services interaction model and enables it to support business transactions. BPEL defines an interoperable integration model that should facilitate expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

Web service makes interoperability easy. It is independent of programming languages, models, and system software. A client can remotely access a Web service by using standard well-defined mechanisms. This feature makes the Web services approach appealing to inter-organizational computing systems. The Web service technology has been adopted in industry as a standard for building enterprise applications. Adoption of Web service is useful for dynamic discovery and composition of services required for coordination of a decentralized set of resources.

2.4 Discussion about Some Alternative Technologies

2.4.1 .NET

The practical realization of Grid poses a number of challenges. Key issues that need to be dealt with are security, heterogeneity, reliability, application composition, scheduling, and resource management. The Microsoft .NET Framework [24] provides a powerful tool set that can be leveraged for all of these, in particular support for remote execution, multithreading, security, asynchronous programming, disconnected data access, managed execution and cross-language development, making it a great platform for Grid computing middleware. By providing developers with a comprehensive and consistent programming model and a common set of APIs, the .NET Framework helps developers to build applications in the programming language users prefer, across software, services, and devices.

The .NET Framework includes four pillars: Windows Presentation Foundation (WPF), Windows Workflow Foundation (WF), Windows Communication Foundation (WCF), and Windows CardSpace. WCF is Microsoft's unified programming model for building Service Oriented applications. It enables developers to build secure, reliable, transacted solutions that integrate across platforms and interoperate with existing investments. WCF simplifies development of connected systems and ensures interoperability. It unifies a broad array of distributed systems capabilities in a

composable and extensible architecture, spanning transports, security systems, messaging patterns, encodings, network topologies, and hosting models.

Microsoft's .NET Framework is a key framework for implementing commercial distributed systems for Windows-based platforms. Application developers rely on the .NET framework for its integrated tools, powerful functionality and ease of use. However, there are some challenges. .NET alone cannot deliver the scalability and reliability required for today's data intensive and computation intensive applications. .Net is limited on desktop computers those running variants of the Microsoft Windows operating system, thus severely limiting the ability to effectively utilize the none-Windows computing resources. Also it is difficult for .Net to integrate with current Grid software, which has been primarily written for Unix-based operating systems. The capabilities enabled by the .NET framework are important for Grid systems, but will not be considered by current work due to this limitation.

2.4.2 UDDI

Universal Description, Discovery and Integration (UDDI) is a platform-independent, XML-based registry for businesses worldwide to list themselves on the Internet [25] . UDDI is an open industry initiative, sponsored by OASIS, enabling businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet. A UDDI business registration consists of three components: White Pages — address, contact, and known identifiers; Yellow Pages — industrial categorizations based on standard taxonomies; Green Pages — technical information about services exposed by the business.

The UDDI specifications define a registry service for Web services and for other electronic and non-electronic services. A UDDI registry service is a Web service that manages information about service providers, service implementations, and service metadata. Service providers can use UDDI to advertise the services they offer. Service consumers can use UDDI to discover services that suit their requirements and to obtain the service metadata needed to consume those services.

The UDDI standard is the least understood and often the most maligned of the core Web Services standards. Unlike its now well-understood SOAP and WSDL, UDDI has

experienced limited and sporadic adoption by companies implementing Web Services-based SOAs [26].

The UDDI specifications supported a publicly accessible Universal Business Registry (UBR) in which a naming system was built around the UDDI-driven service broker. IBM, Microsoft and SAP announced they closed their public UDDI nodes in January 2006 [27].

"Basically, the UBR is a relic of an earlier vision for UDDI. The original vision for UDDI was as a standard that would help companies conduct business with each other in an automated fashion. The idea was that companies could publish how they wanted to interact, and other companies could find that information and use it to establish a relationship," said Jason Bloomberg, senior analyst at ZapThink. "Needless to say, this is not how companies do business -- there's always a human element to establishing a relationship. As a result, the UBR served as little more than an interoperability reference implementation. Now that UDDI has become more of a metadata management standard for SOA, there's little need for the UBR anymore."

2.4.3 CORBA

The Common Object Requesting Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together. While CORBA promised to deliver much in the way code was written and software constructed, it was much criticized during its history. Some of its failures were due to the implementations and the process by which CORBA was created as a standard; others reflect problems in the politics and business of implementing a software standard. These problems led to a significant decline in CORBA use and adoption in new projects and areas. The technology is slowly being replaced by Java-centric technologies [28].

Obviously, a number of external factors contributed to the fall of CORBA, such as the bursting of the Internet bubble and competition with other technologies, such as DCOM, EJB, and Web services. These factors cannot fully account for CORBA's loss of popularity, however. After all, if the technology had been as compelling as was originally envisaged, it is unlikely that customers would have dropped it in favour of

alternatives. Technical excellence is not a sufficient prerequisite for success but, in the long term, it is a necessary prerequisite. No matter how much industry hype might be pushing it, if a technology has serious technical shortcomings, it will eventually be abandoned.

The most obvious technical problem is CORBA's complexity—specifically, the complexity of its APIs. Many of CORBA's APIs are far larger than necessary. Developers who had gained experience with CORBA found that writing any nontrivial CORBA application was surprisingly difficult. The platform had a steep learning curve and was complex and hard to use correctly, leading to long development times and high defect rates.

Also CORBA provides quite rich functionality, but fails to provide two core features: security and versioning. For an e-commerce infrastructure, lack of security and versioning are quite simply showstoppers—many potential e-commerce customers rejected CORBA for these reasons alone. Please refer to [29], a comprehensive analysis of CORBA failures by Michi Henning in ACM queue about "The Rise and Fall of CORBA". The author is one of CORBA's former architects.

2.5 The Application Service that Combines SOA and Grid

2.5.1 Wrapping Approach

The idea of Service Oriented Architecture is to achieve loose coupling among the interacting software. This advanced flexible architecture provides a foundation to allow Grid resources to be shared seamlessly. SOA is a widely accepted model for building grids, holding a lot of promise for grid-enabling scientific applications. In recent years, Web services have gained wide-spread acceptance in the Grid community as a standard way of exposing applications functionality to end users.

Most of the scientific applications are conventional FORTRAN or C applications that are configured by a parameter file or command-line arguments. They are not Web services. Consequently, to integrate them into the Service Oriented Architecture,theye must be embedded in a wrapper service. A wrapping approach is used to make a piece of code such as the simulation solver available as a self-contained reusable object to some glue layer. The glue layer is written in a high level language. It could be a Grid

fabric layer such as Web services, allowing interoperation of components running on different machines across a network.

The wrapping approach allows us to write the core of an application in a language like C or FORTRAN but controls its behaviour through a higher level language. This allows clients to easily interact the application via the high level language interface. Additionally, this avoids necessities of dealing with tedious low level details of interfacing to many third-party libraries. The wrapping approach also avoids incurring performance penalty resulted from writing the entire application in a higher level language. Moreover, the wrapping approach is very helpful in integrating the legacy applications into new technologies without rewriting the applications.

2.5.2 The Application Service

A Web service can wrap an application, enclose it and invoke it without the application having to be modified using the wrapping approach. In this fashion, a legacy application is wrapped as an Application Service. In this research, an Application Service is a Grid-aware application with a Web service interface that is described by the Web Service Definition Language. The Application Service makes the application available through a Web service interface. The Application Service is Grid-aware, which means it can run computations in the Grid environment. When a user invokes an Application Service with some input parameters, the service runs the application that it wraps on the distributed compute resource with those input parameters, monitors it and returns its output results. Providing a Web service interface for the legacy applications alleviates many problems raised by the legacy applications. By converting the command-line applications into the Application Services, it becomes easy to compose complex workflows from them and run them on the distributed resources.

Benefits of the Application Services are clear because of adoption of Grid and SOA. Application Services are Grid-aware. In a Grid environment, computations can be distributed across the available processors in a larger pool to achieve dramatic speed-up. Hence, users benefits from sharing the distributed resources. Our experience demonstrates that the geographically distributed compute resources on a Grid could readily be used as one in a computation of a complex job. The results reflect that dramatic speed-up is achieved only if the scenario being simulated is large enough, otherwise extra administration and the communication time associated with using the

resources on a Grid outweigh any possible gains in computation time. Speed differences in the processors of different resources also have an adverse effect on overall computation time of a job.

The advantages of Application Services include not only performance gain from the resources sharing, but also close collaborations established among engineers, engineering scientists, and computing scientists. Legacy applications are based on tightly coupled monolithic structure. They read input files and output final results. This makes them difficult to expand and collaborate according to users' different requirements. Legacy applications can be used as foundations and be decomposed into modules. These modules are further exposed to Application Services. Application Services are loosely coupled and flexibly distributed, with the following substantial benefits:

i) Flexible granular functionalities are accessible to the users or other applications via well defined interfaces. Code reusability is improved by the availability of invocations. In particular, Application Services can be naturally well integrated into a complex workflow, chaining different processing whose outputs are piped to the inputs of each other. Application Services allow different applications to exchange data regardless of operation systems or programming languages underlying those applications.

ii) Legacy applications are usually run locally. To run the legacy applications remotely, input/output data has to be explicitly specified in a task description. Users usually need manually transfer data between sites. Invoking a new execution of one same code on different data requires rewriting of a new task description. Alternatively, the users have to rely on some script language to manage batch jobs submission. Whilst, Application Services allow the users to use global distributed data, it decouples computation and data transferring. This decoupling is particularly important when considering restart of a job in case of the job failure. Also, Application Services can provide batch jobs submission support easily because it is written in high-level language.

iii) Application Services are an intermediate layer between the users and the Grid middleware. It adds an extra layer between applications invocation and the Grid infrastructure to which the jobs are submitted. Users do not need to know anything about the underlying middleware that will be directly invoked internally by the service. The different services might even communicate with different middleware and different Grid infrastructures.

iv) Application Services advertise themselves. So the users don't need to worry about how to find them.

v) Application Services make it possible to offer the service over internet and make software updates and maintenances procedures easy. in addition, they provide the users with the possibility to use the service based on pay per use rather than purchasing the software outright.

vi) Application Services change the conventional hands-on process and automate this process considerably. Application Services are easy to be integrated with a Grid portal. Geographically distributed participants can collaborate with each other, specify the service configuration, refer to global data repositories or archives, remote process computational simulations on demand, real time monitor the jobs and collaboratively analyze the results.

2.6 Two Wrapping Strategies

There are some research programs aiming at automating the transformation of a legacy application into an Application Service. Two main strategies are used in different grid middleware for describing and controlling application processing. The batch oriented approach is the most widely available and has been exploited for the longest time. The service oriented strategy has more recently emerged. It consists of using a standard invocation protocol for calling application code embedded in the service. Both strategies are valid in different circumstances, depending on factors such as granularity of codes, users and application area.

2.6.1 Service Oriented Wrapping

The service oriented strategy uses tightly coupled code wrapping technology that exposes low level functionalities. It uses a high level language to interact with a legacy application written in a low level language. The flexible granular functionalities in a low level language are accessible to users via a high level language interface. The service oriented strategy has more control and interaction with the legacy applications, although it requires accessing to the source codes.

In the service oriented strategy, it is typically assumed that service providers, would like to build Grid enabled Application Services using specific legacy software libraries. They have access to source codes. Some modifications are required to these legacy codes. Prior to wrapping, the legacy codes may need to be reconstructed and modified while main computation parts and existed functionalities are kept untouched. A wrapper layer written in a high level language is added to expose desired low level functionalities.

However, in the service oriented strategy, all application codes need to be changed and recompiled with the high level wrapper to become available as an Application Service. The service providers are often reluctant to invest efforts in writing specific wrapper for every application due to the following tentative reasons:

i) The complexity of standards often makes service conformity a matter of specialists. Some tools are available for helping in generating service interfaces but they cannot be fully automated and they all require a developer intervention.

ii) Standards tend to evolve quickly, especially in Grid area, obsoleting earlier efforts in a very short time scale.

iii) Multiple standards exist and one same application code may need to be executed through different service interfaces.

iv) In the case of legacy code, recompilation for instrumenting the code may prove very difficult or even impossible (in case of non-availability of source codes, compilers, dependencies, etc).

2.6.2 Batch Oriented Wrapping

The batch oriented strategy consists of a command-line description and a remote execution of an application code. The legacy code is provided as a black box with specified input and output parameters and environmental requirements. There is no requirement to access the source codes of the legacy application. Only the executable is available and required, alongside a user-level understanding of the application. But this strategy is relatively coarse-grained, in which the Application Service does not allow visibility of low level functionalities. The exposure of internal functionalities is limited to the command-line level.

The batch oriented wrapping is very common in both scientific and business applications when: the source codes are not available, or the programs are poorly documented and/or the necessary expertises to do any modifications have long left the organisation, or the application has to be ported onto the Grid within the shortest possible time and smallest effort and cost, or the functionalities are offered to partner organisations but the source is not.

In the batch oriented strategy, Application Services have service interfaces and features such as security, jobs and data management. The computation resources of the Grid are accessed through jobs submission. Each processing is related to an executable code and described through an individual computation job. A job description encompasses at least the executable code name and a command-line to be used for an application invocation. It may also includes additional parameters such as input and output files to be transferred before or after the execution, and additional job scheduling information such as minimum system requirements. Jobs can be described either directly via the command-line of a job submission tool, or indirectly through a job description file. Unless considering very simple code invocation use cases, description files are often required to specify the task in depth.

In the batch oriented strategy, application invocations are straightforward, through the legacy code command-line. It does not require any adaptation of application codes. Many Grid middleware are also batch oriented, such as Globus Toolkit [30] and Condor. This strategy is useful for Grid-enabling legacy applications without much effort.

In the batch oriented strategy, the service needs to precisely know the command-line format of an executable code, taking into account all of its parameters. Most of Application Services using this strategy rely on users to set up all these executable related information. It is not always the case when the users are not applications developers.

In the present work, both strategies have been adopted to wrap legacy applications. Application Service Toolkit is an automatic toolkit that wraps scientific applications as Application services and deploys them on the Grid. It is based on batch oriented wrapping strategy and is discussed in detail in Chapter 3. In GECEM problem solving environment, several legacy applications are wrapped as Application Services using service oriented wrapping strategy to achieve collaborative numerical simulation. Additionally, in GECEM Migrate Legacy Service Model, the batch oriented wrapping strategy is used to migrate the applications to the Grid environment. The details of GECEM are elaborated in Chapter five.

2.7 Related Work

2.7.1 Soaplab

.Soaplab is a framework for exposing command-line tools as Web services through the use of batch oriented wrapping. Soaplab service collection at European Bioinformatics Institute runs on top of more than hundred bioinformatic analysis tools from the EMBOSS package. It bears the capability to deploy a Web service in a container, starting from the description of a command-line tool. This command-line description, referred to as metadata of the analysis, is written for each application using ACD text format file and then converted into a corresponding XML format [31]. ACD format is a format used by EMBOSS [32]. EMBOSS 100+ programs are already distributed with their descriptions in this format.

Soaplab offers two types of services; namely, Analysis Service and Derived Analysis Service. Analysis Service is a service representing remote analysis applications using a generic interface. For Analysis Service, the individual applications have their input data and their results named. Once users know the input data names they can send their data to the analysis as the weakly-typed name-value pairs. Derived Analysis Service is a web

service representing some particular analysis application. European Bioinformatics Institute hosts more than hundred Derived Analysis Services for EMBOSS.

The Soaplab server is based on an internal CORBA-based AppLab [33] server and a Tomcat servlet engine [34].

AppLab application provides a CORBA [35] server for executing conventional command-line applications and sending results back to clients. AppLab is an automatically generated wrapper of command-line driven applications. It provides a uniform graphical user interface for the applications by using CORBA (for communication) and Java (for GUI). Applab develops a distributed object system, which provides an easy-to-use and well-defined access to a large set of existing command-line applications of different types.

After the Soaplab services are created and deployed in an Apache Tomcat container, users can access them using custom client programs, which provides graphical user interfaces to Soaplab services. Users use the unified API to find an analysis tool, discover what data is required and what data is produced, to start it and to obtain results.

Although Soaplab serves to wrap almost any command-line tool as a Web service, it has a number of limitations.

i) Soaplab uses CORBA on sever side for starting, controlling and monitoring applications but not Web services standards that are more widely adopted today [31]. To provide improved support for service providers, Soaplab has been rewritten in year 2007 with the removal of CORBA (AppLab) layer. The new version, called Soaplab-2, can be deployed in two configurations: document/literal-wrapped web services (using the JAX-WS webservices stack), or RPC/encoded web services. But Soaplab-2 only supports Analysis Service, and not Derived Analysis Service. Also, Soaplab does not have a Web services based notification model. This makes it difficult for Soaplab services to interoperate with other Web services based notifications systems, which are widely adopted today. However, it does have a CORBA-based event notification model.

ii) SoapLab does not support Grid standards for service level authentication and authorization [31]. Also, it does not have any other fine grained

authorization mechanism. This means that service providers cannot have fine-grained control on what operations users can invoke on their application services.

iii) The concept of Analysis Service is similar to our AAS. But the Analysis Service is weakly-typed whilst our AAS is strongly-typed.

iv) ACD is not the language particularly designed for command-line description.

v) SoapLab cannot run its applications on remote clusters of resources. It offers very little support for job monitoring and steering. Neither does it support the asynchronous mode of invocation. This makes it difficult for users to invoke long running applications.

vi) SoapLab cannot create an Application Service on-demand from workflows.

2.7.2 Generic Application Service

Generic Application Service (GAP) uses batch oriented wrapping. GAP is another system that provides a Web service interface to scientific command-line applications, which is part of In-VIGO system [36]. In-VIGO is a Grid system that makes extensive use of virtualization technologies to decouple user environments from physical resources, and subsequently creates such components to integrate application codes with core Grid components [37]. In GAP approach, a description of a legacy application's input and output information, and the required execution environment (e.g., machine architecture, operating system, and dependent libraries) are provided by developers. All information is provided in a single file, thus not requiring Java code to be written. A generic application service (a Grid service) interprets this information at runtime and configures itself into an Application Service that is specific to that legacy application. Validated requests to the generic service are submitted for execution on the Grid [36]. The GAP Service has some similarities with our Abstract Application Service (AAS). Both approaches attempt to wrap legacy applications as services. Both approaches develop a language to describe the command-line applications. However the differences lie in designs, implementations, and then capabilities of the services.

In GAP approach, a configuration language (CFG) has been developed to allow an application provider to specify information required to Grid-enable a legacy application.

The configuration language is specified in a grammatical framework that is designed to be powerful enough to represent the command-line applications. The grammar is designed to be a specification language based on feedbacks from application developers. The CFG specification is then mapped into XML, which is used in the back end of the enabling framework [36].

The concept of CFG is pretty similar to our Command-line Description Language (CoLDeL) used in AAS. Both contain information regarding general information for the application, execution environment information, and list of arguments accepted by the application. But CoLDel is more user-friendly and more powerful in terms of the expression of arguments. CoDel is an XML based language, which is human readable. It is more user-friendly compared with CFG, the unreadable specification language. CoLDel can specify each application with a rich semantic description and provide more useful information than CFG. This makes command generation easy and dynamic. CFG only provides support to express group arguments, but not to express conflict arguments; whilst CoLDel can specify both the dependencies and conflict of arguments. CoLDel also contains argument ID, argument order, the format appears on command-line, and so on, which are not specified in CFG. CFG uses SPS section to half hard record synopsis of the command. Only non-I/O arguments can be expressed and plugged into the final command. In CoLDel, I/O arguments contain even more information than non-I/O arguments, like file format, I/O direction, and name convention of output.

Another disadvantage of CFG is that, it is only used by services, but not users. GAP uses cfgParser and adaptors to map the CFG into XML [36], which is used in the back end of the service. But for the users, they have no data type provided to set up the job parameters. This means that GAP is a very weakly typed Web service, making it pretty much unusable by end users and usable only from a web portal using Application Service specific clients. Besides, GAP cannot be used in workflows where a strongly typed Web service is required. While in CoLDel, an XML schema has been defined, in which an initial number of types and a rich set of elements are declared. This ensures strongly typed data exchanging among services. The strong data typing not only makes exchange between clients and services easy, but also facilitates generic Web service workflow tools to compose these services easily.

GAP is built on top of Globus technology, whilst AAS is based on standard Web service technologies.

2.7.3 GEMLCA

Grid Execution Management for Legacy Code Architecture (GEMLCA) was developed by the University of Westminster to enable legacy code programs written in any source language to be easily deployed as a Grid Service without significant user effort. GEMLCA uses batch oriented wrapping. GEMLCA addresses issues of exposing the legacy codes as Grid services and provides a method for exposing and executing legacy applications through OGSI Grid Services [38]. GEMLCA creates a general solution to deploy existing legacy applications as Grid services without modifying source codes. GEMLCA services offer a front-end Grid service layer that communicates with clients in order to pass input and output parameters, and contacts a local job manager through Globus MMJFS (Master Managed Job Factory Service) to submit legacy computational jobs.

GEMLCA has been designed as a three-layer architecture: a front-end layer offers a set of Grid Service interfaces that any authorized Grid client can use in order to contact, run, and get the status and any result back from legacy codes [39]. This layer hides the second core layer section of the architecture that deals with each legacy code environment and their instances as Grid legacy code processes and jobs. The final layer, backend is related to Grid middleware where the architecture is being deployed. The GEMLCA implementation is based on Globus.

GEMLCA offers a comprehensive solution, since it includes portal and workflow access, and security solutions incorporating authentication, authorisation and security delegation mechanisms. It also offers end users with no programming knowledge the ability to port their applications to the Grid with relatively little effort. But it has some major limitations.

i) 1 GEMLCA is not lightweight. It represents a general architecture which includes GEMLCA client, GEMLCA resource (a set of Grid services), Grid host environment (Globus), and Compute Server. It allows an application provider to add a legacy application into the GEMLCA legacy codes list. But it does not generate a persistent Grid service for the specified

application. Instead GEMLCA resource creates an instance of the legacy code process returning a Grid service Handle (GSH) [39]. This means GEMLCA architecture has to be used as a whole. It is not easy to separate the GEMLCA client, the set of Grid services, and the backend Globus. Compared with GEMLCA, ASToolkit is lightweight. Service providers can get a standard Web service for each application wrapped. This Web service can be accessed in a standard manner by any client or workflow.

ii) GEMLCA is based on Globus infrastructure and is tightly bound with Globus. It requires Globus installation on GELMCA resource layer and Compute server.

iii) It is unable to deploy Grid services on remote hosts. Actually, this is a limitation of the Grid service containers that are available today. Unlike Web service containers like Tomcat that allow remote deployment of Web services, Grid service containers do not allow remote deployment of Grid services.

iv) GEMLCA Grid services do not transfer the files. The GEMLCA replies on portal or other client to upload or download the files via GridFTP [40].

v) GEMLCA does not support Message Level Security [41]. Hence, service providers do not have fine-grained control over which users have access to the operations in their application services.

2.7.4 Summary

There are a number of research tasks aiming at automating transformation of legacy codes into Application Services with their own set of advantages and disadvantages. However, there are some fundamental limitations that are not addressed by currently available results. These limitations which have been addressed by the work in this thesis are listed below:

i) Lack of a lightweight toolkit to create standard Web services for legacy applications.

ii) Absence of a fine grained authorization mechanism by which service providers can control whatever operations users can invoke on their application services.

iii) Need for a mechanism to create a Web service interface to a scientific application on-demand from workflows.

iv) Requirement for a scalable mechanism which enables wrapping a large number of applications as Application Services and, updating and maintaining source codes and deployments of all those Application Services.

CHAPTER 3

Command-line Description Language and Application Service Toolkit

3.1 Overview and Contribution

Web service architectures have gained popularity in recent years within scientific grid research communities. One reason for this is that web services allow software and services from various organizations to be combined easily to provide integrated and distributed applications. However, most applications developed and employed by scientific communities are not Web service oriented, and they are written in various programming languages (e.g., FORTRAN, C, scripting languages and others). These codes not only typically represent large scale investments in terms of time and effort that cannot be discarded, but also are high quality and extensively validated programs. There is a growing need to integrate them into grid applications based on service oriented architectures. The adaptation of these existing applications to Web services is becoming important as a way of harnessing validated tools in a new powerful operational environment provided by the Grid and Service Oriented Architecture.

A Web service can wrap an application, enclose it and invoke it without the application having to be modified. In principle, the task of wrapping an application as a Web service is not a huge task for a specialist trained in Web and Web service programming, but for most scientific application specialists, this is an extremely high barrier to surmount. There are a number of tools to help accomplish this task. These tools automate the task of transforming existing applications into Web services without having to deal with the details of Web services technologies. However, simply providing a Web service interface is not sufficient to make the application a usable

component in a distributed computation. One major concern is security. In particular, the question is how the Application Service allows the users of a community run the scientific applications on computing clusters provided by third party resource providers without having login accounts on the computing clusters? How can a service provider provide and authorize a selected group of users with access to Application Services without building a separate security infrastructure and without requiring the users to have login accounts on the host running the service?

Another problem is making the service usable directly from a web portal, as well as making it a component in a workflow. Also, how can the Web service use the distributed compute resource to run the application?

Our goal is to leverage the set of computational resources available across different sites on the grid. However, clusters at different sites run schedulers of their choice. The most commonly used schedulers include Condor and Sun Grid Engine (SGE) [1]. Hence, it is mandatory that the Application Service supports to access any of existing schedulers and schedulers to be appeared in future. To end users these schedulers are accessible in a generic way, which means the users are able to use them in a transparent uniform fashion (not scheduler specific). Furthermore, the users typically interact with schedulers such as Condor and SGE via command-line interfaces. However, in order to expose the applications as services, we need to access these schedulers programmatically.

Due to wide usage and huge potential of the Grid and Service Oriented Architecture, Application Service Toolkit (ASToolkit) [2] is implemented and allows scientists to provide a Web service interface to their existing applications without having to write extra code or modify their applications in any way.

The primary research contribution of this toolkit is as follows:

3.1.1 Command-line Description Language (CoLDeL)

CoLDel, an XML based language, is designed and developed particularly for the ASToolkit in order to describe individual command-line applications precisely. CoLDeL acts as a protocol so that different service providers can follow it to generate an Application Definition File for each scientific algorithm for use by ASToolkit Services. An XML schema has been defined which ensures strongly typed data

exchanging among services. An initial number of types and a rich set of elements are declared.

3.1.2 Automatic Toolkit for Wrapping All Command-line Oriented Applications

Most large scale computational facilities have traditionally operated their machines in batch mode and the ASToolkit is focused on these batch mode applications. It can wrap almost any command-line application (i.e., non-graphical), such as UNIX commands, or more sophisticated scripts written in Python, Perl, and so on. It does not require any modification to the wrapped applications. And, no source code is needed for the legacy applications. ASToolkit is an automatic toolkit that wraps scientific applications as Web services and deploys them on the grid. This Web Service is named Application Service. The Application Service is described by CoLDeL, presents a Web Service Description Language interface to potential clients and interacts with Grid resources via a component plug-in model.

ASToolkit makes command oriented application wrapping an automatic, easy and fast task. In addition, application providers do not need to be experts in web service standards, such as Web Services Description Language, Web Services Addressing, Web Services Security, or secure authorization, because the toolkit automatically generates these details.

3.1.3 Security

ASToolkit also automatically provides a WS-Security [3] based authentication and authorization system that allows selected users to securely interact with these services through automatically generated web interfaces, to compose scientific workflows using these services, and to monitor the status of their jobs on the grid.

3.1.4 Application Remote Execution and Access to Grid Resources

The Web Service generated by the ASToolkit is named Application Service. Application Service presents a Web Service Description Language interface for the application to potential clients and interacts with Grid resources via a component plug-in model. Application Service can enable but does not require the use of distributed resources via the Grid. The Application Service can support operation seamlessly in a highly distributed environment. The distributed functionality is enabled and controlled by components employing Grid middleware. While the Application Service can fully

support the distributed environment, it also can be easily used in a local environment. The role of the Grid-enabled ASToolkit service is to provide a uniform submission layer on top of different execution environments.

The ASToolkit provides a level of abstraction to the client that is much higher than services like Gram because it takes low level job submission details like environment variables and temporary file management out of the hands of the client.

In this chapter the features and implementation of this Web service wrapper are described together with how it has been used and tested in the context of medical image analysis.

3.2 Design

The primary consideration in developing the toolkit is to use standard software and keep the software requirements to a minimum. Only mature software with known reliability and performance characteristics is used for the toolkit.

To make the applications grid-aware, the following requirements have to be fulfilled: remote execution and access to Grid resources support for multiple concurrent users, access via a set of disparate clients, and the use of security mechanisms. Web service is good in grid enabling the applications. Web services are capable of serving multiple client requests concurrently. Also, since Web services are language and platform independent, they are easily accessible by clients written in different languages.

A toolkit is needed to automatically wrap scientific applications as Web services and deploy them on the grid. It can wrap any command oriented application and does not require any modification to the wrapped applications. The toolkit does not attempt to deploy any application. So the assumption is that either the application that the toolkit wraps has already been deployed and ready to run on some resource, or the application is ready to migrate to a remote resource to run. The deployment of the application is usually done by the application provider or service provider.

In addition, the toolkit should be easy and straightforward to use. The users do not need to have the knowledge of applications or Web service.

Furthermore, one of our primary goals is to couple together applications across the community. So the service from the toolkit should be easily coupled and orchestrated by workflow tools.

3.3 Architecture

3.3.1 ASToolkit Architecture

The toolkit can wrap almost any command-line application. Command-line Description Language is a generic language to describe individual command oriented applications precisely. Based on CoLDel protocol, the service provider or application provider will provide an Application Description File for each application wrapped. Application Service Component can process the ADF based on well known CoLDel protocol. It retrieves the dynamic information from ADF, and configures itself to the specified Application Service.

The toolkit does not generate any code for implementing the service interface. All the business logic, like Data Management Component, Application Service Component and so on, is pre-developed and packed as a library, and shared by all the Application Service.

The toolkit, actually Ant [4] script underneath, will pack all the needed documents into a Web application archive (WAR) file and deploy it onto a remote server. The Sun App Server [5] and Apache Tomcat both allow new Web applications to be installed while the container is running. To deploy a Web service on a remote machine dynamically, the toolkit needs to package the Web service implementation code, dependencies, and the deployment descriptor into a WAR file. The WAR file compresses all of this directory-structured content into one Java archive file (JAR). Usually, a Web service WAR file also includes schema (WSDL) files and the Web Service Deployment Description (WSDD).

Each application WAR file contains the same WSDL file, the same ASToolkit java business jars, same third party middleware, but different ADF file and service property file. A service property file describes some dynamic information that is required by an Application Service to be hosted on the specified host. It contains information closely

related to the Application Service server, which usually is different based on the different server's environment and requirement.

This architecture is simple and yet powerful. It makes the toolkit lightweight but highly configurable. No code generation is needed to create a service from its description.

The toolkit can be configured to complete part of the whole task. For example, it can only build the WAR file but not deploy it to the server. This Application Service could be deployed later on by just copying the corresponding WAR file into a specific directory designated by the hosting server. This directory is monitored by a daemon on the hosting server. Thus, any newly copied WAR file can be detected by the daemon and then be deployed into the container by this daemon.

Also, the toolkit can create client side stubs and server side skeletons. A stub is a client's local proxy for a remote object. The client uses the stub to communicate with the remote object-actually, the skeleton of the remote object. The skeleton is responsible for dispatching the client's communication to the actual remote object.

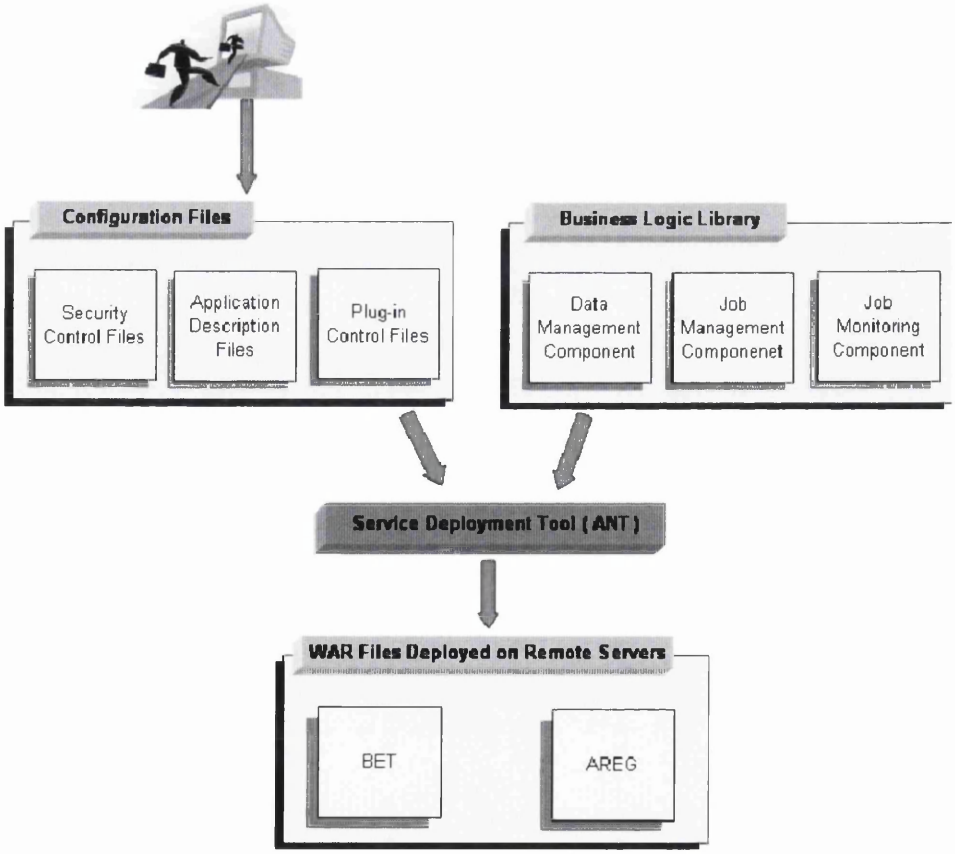

Figure 3.1: ASToolkit Architecture

Figure 3.1 describes the architecture of ASToolkit. The Service provider needs to manually edit the security control files to set up the security level. A set of security control file samples are provided. The service provider also needs to provide an Application Description File for the application wrapped. A plug-in control file needs to be set up based on which plug-ins the service provider is interested in to use. A set of business logic library is provided with the ASToolkit.

An Ant script based deployment tool is for the service provider to generate and deploy the customized Application Service. The deployment tool retrieves the required ADF from the ADF pool and generates an appropriately customized Application Service, which encapsulates the application, deploys the service onto the local server or remote server, and finally publishes the services in a registry service.

A Web interface is provided to let the service provider enter the information required in an application description, and generate the ADF on the fly. The ADF is pushed to the server side and registers with the ADF pool. Application providers can also register the ADF file which they had already via the Web interface.

3.3.2 Application Service

3.3.2.1 Application Service Architecture

Figure 3.2 represents architecture of Application Service. Details regarding structure of messages and operations supported by the service are presented to clients in a consistent WSDL interface. All the Application Services have the same interface. The behaviour of the Application Service is decided by the components (plug-ins) used. The choice of components essentially depends on the plug-in controller in the service property file, which specifies the specific plug-ins to use for data management, job submission and application description. More specific details on the application wrapped by service are described in detail in the associated Application Description File. This would typically include algorithm classification information, a full definition of input/output parameters and all other command-line parameters. At back end a concrete service implementation is built with appropriate Web Service Components (plug-ins). The plug-ins in use interact(s) with underneath compute resources. Data Management Component interacts with the distributed data repositories to download/upload the input/output files. Job Submission Component communicates with

the Application Description File and interprets the job as wrapped application job. It also interacts with the distributed computational resources and submits jobs to them.

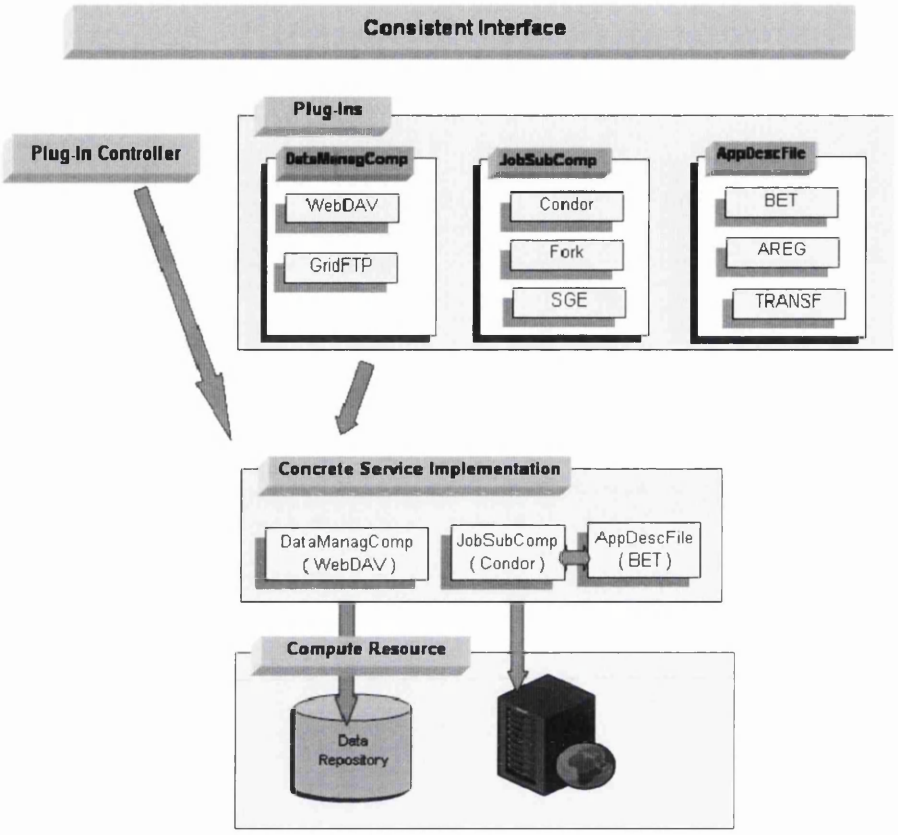

Figure 3.2: Application Service Architecture

3.3.2.2 Web Service Interface

An Application Service makes an application available through a Web Service interface. It is implemented as a Java JAX-RPC-compliant [6] Web Service deployable in any Java Servlet compliant container. This opens up the choice of deployment platforms depending on the scalability requirements.

The Web Service interface makes use of WS-Security to protect message exchange as well as authenticating and authorizing users. The Web Service interface demonstrates the use of the application as a networked multi-user service.

The Application Service is multi-threaded, i.e., if multiple clients invoke operations of a service, a separate thread is generated for each client. State handling is managed internally and transparently based on unique jobHandlers.

The behaviour of the submitJob operation is customized for a specified application by means of a specified Application Description File.

3.3.2.3 Web Service Descriptions Modularisation

For the Application Service, WSDL file is separated into distinct modular components in order to improve re-usability and manageability. These modular components include an XML Schema file, an abstract WSDL file, and a concrete WSDL file, which respectively contain data type definitions, abstract definitions, and specific service bindings. The specific or concrete service definitions depend on the abstract service definitions, which in turn depend on the data type definitions.

This modularisation improves the readability and manageability. It helps in writing clearer service definitions by separating the definitions according to their level of abstraction. The technique also improves opportunities for certain types of extension and reuse. The same data type definitions can be used across many abstract services, and the same abstract services can be offered through many different bindings, at many addresses. As services grow, however, this may evolve into a tree of documents with the data type definitions at its root, branching into several abstract services, documents, and further fanning out to concrete services.

3.3.2.4 How it Works?

After a user specifies all the parameteric values for a job following CoLDel (job description), the client sends a Simple Object Access Protocol message to the Application Service to invoke the submitJob operation. The Application Service interacts with the Data Management Component to download the input files from the data server to the Application Service server side. Invocations on the Application Service are passed through to the appropriate WS component; the component then takes the relevant steps required to fully realize the invocation. For the FORK component a script would be formed and submitted to the local server. The script would invoke the application with the precise command. For the Condor component a further series of Web service invocations would be performed. The component would interact with the Condor submission service and submit the job to the Condor pool. After the job is completed, Data Management Component uploads the output results back to the data server.

To monitor the job progress, the client passes the jobHandler to the service to invoke the monitorJob operation. And the service will return the jobStatus back to the client. This pull model is used which is based on a request/response paradigm. The client sends

a request to the server, then the server answers. In this model, the data transfer is always initiated by the client, hence avoids the possible firewall problem. Most firewalls are set up to allow outgoing HTTP requests and incoming responses but not incoming requests.

3.4 Features of ASToolkit

The ASToolkit provides the following features:

i) Easily and rapidly wrap any command oriented application as a Web service without modification to the selected application.

ii) Enables strong data typing. ASToolkit services have inputs and outputs defined in detail using XML schemas. Initially the number of types and a rich set of elements are declared. This ensures strongly typed data exchange among services. The strong data typing not only makes validation of inputs easy, but also facilitates generic Web service workflow tools to compose these services easily.

iii) Adopts a consistent interface and plug-in model. All the ASToolkit services have the same interface and can easily be extended to support any Grid scheduling. ASToolkit services are extensive and are not tied to any specific scheduling middleware. It can use various schedulers to submit jobs on resources.

iv) Hide the computational resource from the user. The user only interacts with algorithm services, not computational resources.

v) Implement with WS-Security, which addresses the three security requirements: message authentication, message integrity and message confidentiality.

vi) Use a Service Provider Account Model to make user account management simpler. There is no need to open a user account for each user on computational resource.

vii) Support data transfer for jobs. ASToolkit services can stage in input data files from a user specified location before running the application, and can

stage out the output data files after the job is completed. This is done using WebDav [7].

viii) Support concurrent and asynchronous job submissions.

ix) Support job monitoring. The application services can monitor the status of jobs and return output results to the user. The steer service can monitor any jobs on different hosts.

x) Have capability to return job specification provided by user for each job. This is useful to resubmit the job if the job fails.

3.5 Implementation

3.5.1 Consistent Interface

As Figure 3.2 shows, all the Application Services use the same interface no matter what application is wrapped. Regardless of the number of wrapped and deployed applications what input data and options are expected, what output is produce, or what syntax the command-line applications have, they all use the consistent interface and are controlled by the same methods.

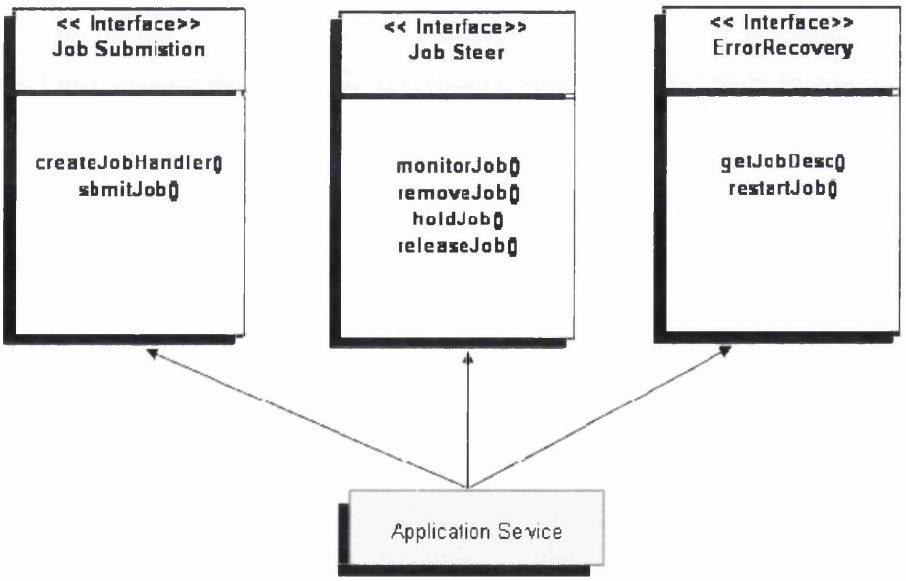

Figure 3.3: Application Service Interfaces

From Figure 3.2 we can also see that all Application Services use the same interface no matter what grid scheduler they interact with--FORK, CONDOR, GRAM [8], or others. Application Service is Grid-enabled and it can submit jobs to distributed

compute resources. Since various schedulers run on different sites, it is mandatory that these schedulers can be accessed in a consistent way for maximum code reuse. Furthermore, Web service implementations interact with these schedulers programmatically rather than their regular command-line interfaces.

This consistent approach can offer a lot of benefits to users who are only interested in the applications themselves, but not in the underlying complicated compute resource, or the diverse scheduler middleware. By providing one consistent interface to all applications, we effectively hide the complicated grid environment from the user.

There are two ways to produce WSDL, top-down and bottom-up. In top-down, WSDL specification should be developed from scratch, optionally starting from some XML domain vocabulary. While in bottom-up, WSDL specification is derived from some existing server side component interface. Bottom-up WSDL creation is straightforward, as powerful tools are available. But developers need keep an eye on the generated type and port names. They might not match with naming conventions. Also it is sometimes difficult to map all language constructs of a programming language to WSDL. This is because the WSDL is independent of the implementation language. Another frequently occurring problem is that an existing component interface does not meet the requirements for a well-defined Web service interface. For example, unsupported data types appear in method signatures or inheritance is used. Whenever the tools cannot handle the WSDL generation, it is safer to hand code the WSDL. By creating the WSDL file first developers will ultimately have more control over the Web service, and can eliminate interoperability issues that may arise when creating a Web service using the bottom-up method.

We use top-down approach and carefully design the contract. Especially our Application Service uses industry standard XML schemas to describe the data structures. A static WSDL is used for every Application Service due to the consistent interface. Two Application Services can be distinguished by their unique URLs, and their associated Application Definition File.

The port interfaces of Application Service are depicted in Figure 3.3. The Application Service exposes many operations via several WSDL port interfaces, namely jobSubmit, JobSteer, and errRecovery. These port interfaces include many operations

like createJobHandler, submitJob, monitorJob, RemoveJob, holdJob, releaseJob, getJobDesc, and restartJob.

The operation createJobHandler allows the client to create a job. This operation requires one argument which is a job description object. This object contains all the application command parameters set up from the client. It is set up following the CoLDeL and will be validated by the server against associated Application Description File. Once appropriate checks have been made (validation of inputs, data type checks, values supplied for options, mandatory arguments setup, and so on), this operation returns to the client a unique job identifier.

The execution of a remote application can be initiated by calling submitJob. The submitJob operation provides a generic entry point for using the application. The invoke operation requires one jobHandler. Once the server retrieves the jobDesc object back based on the jobHandler, the actual application launches. How or where this processing takes place is an implementation detail and is effectively hidden from the user. Thus the client only deals with one generic WSDL interface.

Since the application processing may take considerable time we must release the client from the call to invoke immediately. Otherwise, client timeout errors are inevitable.

The operation monitorJob allows the client to get a job specific status object. getJobDesc allows user to get the job description object back. This object can be used directly to resubmit the job in case of job failure. RemoveJob, holdJob, and releaseJob enable the client to control the job.

3.5.2 Component Plug-in Model

The Application Service is simply a wrapper that exists to provide the client with full details regarding the specific application the service can perform and a means of invoking a distributed computation. It has the capability to transfer the data files, and create running instances on the distributed compute resource when the user provides any additional needed parameters. When so invoked the Application Services responds by doing three things. First, it combines the information from the user and Application Description File to create a concrete job description. Second, it transfers the input files and output files between the Application Service server and data server. Third, it

interacts with backend compute resources that can be used to do the application computation.

The Data Management Component, Job Submission Component and ADF are used to achieve the above functionality. The Data Management Component is in charge of file transfer between the two sites. Job Submission Component is responsible for managing the application execution. ADF provides all the specific application related information. These three components work together to complete the submitJob operation, the main operation provided by Application Service.

Using the information in the JobDescription from the user, Application Service first interacts with the Data Management Component to stage in the input files for running the application. Combined with the information from the user and ADF, the Job Submission Component prepares the job and submits the application as a job to the compute resource. Application Service then keeps checking the status of the application job. After the application execution is complete, the Application Service interacts with the Data Management Component, and stages the output files of the application to the specified output data repository.

In a Grid context the compute resources may exhibit considerable heterogeneity. In our test bed of four distributed sites this can be clearly seen, with various combinations of hardware, operating systems and middleware present at each site. Given the varying administrative policies at each site and the fact that each organization has already spent considerable time developing their respective infrastructures it is not feasible to require all participants to upgrade or move to a common solution. Instead, we require that the Application Service adapt to the local resources: this is achieved through the use of a Job Submission Component plug-in model that allows the Application Service to be configured (at deployment time) with a Job Submission Component capable of interacting with the available resources.

The Application Description File is also added as a plug-in at run time to allow the Application Service to configure itself to become the specified Application Service based on the associated ADF.

Figure 3.2 shows the component plug-in model we use for the Application Service. "JobSubmissionComponent" like FORK, CONDOR, GRAM, etc., is the component capable of interacting with distributed compute resources and submitting application

jobs to it. It is encapsulated behind the consistent interface of the service. BET, FAST, and FLLIRT are Application Definition Files that describe the application in detail following CoLDeL protocol. Both "JobSubmissionComponent" and Application Definition File are prepared and plugged in at deployment time or run time in order that the Application Service can dynamically construct itself to be the individual application service running on a specific compute resource. The use of the component plug-in model ensures that our services are not tied to any specific middleware package or any specific application. It is straightforward to implement other plug-ins to submit to other schedulers. This makes the Application Service easy to fit in the increasingly diverse and complex Grid middleware.

We have currently prototyped components to interact with the local server and the Condor SOAP, Fork and Condor. The fork implementation translates an incoming request combined with the Application Definition File, into a simple command-line and forks this on the host server of the Application Service, whilst the Condor implementation takes the incoming request and programmatically interacts with the SOAP interface of the Condor scheduler at a remote site.

For example, at Oxford we have a Linux machine dedicated as an Application Service compute resource. We need to be able to set up an application running environment and construct the precise command, submit to the local server, and check status; and the FORK Component provides this functionality.

At another site, UCL, we have a Condor pool accessible through the Condor SOAP API. Application Services wishing to make use of this resource are configured with the Condor component. Requests to the Application Service are delegated to this component which maps them to Condor specific API calls.

Abstraction of the ADF and the "JobSubmissionComponent" provides us with two levels of flexibility:

i) Clients deal only with the Application Service interface. Changes can be made behind this interface without affecting clients in any way.

ii) The Application Service deals only with plug-in components – the service itself has no intrinsic knowledge of the underlying compute resources and

applications. We avoid coupling the Application Service with any one specific middleware or any application.

3.5.3 Strong Data Typing

Traditionally, scientific applications are invoked via command-line. And the user needs set up a set of arguments to run the command. Since the advent of the Web service technologies into the Grid world, several projects have attempted to expose their applications as Web services. However, it is very often the case these Web services use simple strings or other generic type to represent the arguments set up. Although this may provide remote execution and access to Grid resources via a Web service interface, this is not very flexible and robust. This is because the string based arguments setup are not strongly typed.

To enable strong data typing, a number of types and a rich set of elements are defined using XML schemas, like jobDescription, serviceDescription, algorithm, argument, and so on. The user needs to specify the job in detail using these data types. The incoming message of the createJobhandler operation is jobDescription, a strong data type. This jobDescription will be checked and validated by the Application service.

The strong data typing has the following advantages:

i) The strong data typing helps to properly define the web interface and minimize the network overhead.

ii) These strong data types are the abstract from all command oriented applications. All the applications can be described by these data types in detail. Hence, they do not require additional negotiation between clients and the service provider to understand these data passed or expressed in strong typed form.

iii) Without a strong data type, it is prone to message related exceptions due to inconsistencies between the format of sent data and the format of accepted data. Application Service code is required to be liberal in what it accepts, which adds extra coding complexity. With strong data typing, the Web service is robust, because only highly constrained data enters Application Service.

iv) The client does not have to have full knowledge about the application. The client can just set up the input files for the job, and leave the rest of the argument setup to the Application Service.

v) Application Service can validate and correct the jobDescription set by the user.

vi) The strong data typing not only makes exchange between client and service easy, but also facilitates generic Web service workflow tools to compose these services easily.

The strong data typing also has some disadvantages. It is difficult to develop and requires the developer to have a working knowledge of XML and WSDL. It makes the Web interface unstable due to the frequent change of the data types, particularly for immature Web services where the required data is subject to negotiation and revision. A modular WSDL is introduced to stabilise the WSDL, in which strong data type system is abstracted into a 100% XML Schema compliant data model.

3.5.4 Command-line Description Language

Command-line Description Language (CoLDeL), an XML based language, is designed in order to describe individual applications precisely. CoLDeL acts as a protocol so that different service providers could follow it to generate an Application Definition File for each scientific algorithm for use by Application Services. An XML schema has been defined which ensures strongly typed data exchanging among services. An initial number of types and a rich set of elements are declared.

The design of the CoLDel is simple but powerful. It provides a generic approach to the abstraction of command oriented application's configuration. It makes the application service lightweight but highly configurable. No business logic code generation is needed to create a service from its description. CoLDel is also helpful to enable generic Web service workflow tools to compose Application Services.

In our current implementation, CoLDel has the following advantages:

i) Conforming to the XML schema, CoLDeL can specify each application with a rich semantic description and provide as much useful information to the service/workflow/client as possible.

ii) CoLDeL provides a set of default argument values from Application Definition File, so that the user needs not set all argument values for each job execution – this is useful as there can be hundreds of arguments to be set up, but the user is often only interested in a few of them.

iii) CoLDel supports data types and constraints on arguments values to ensure that all the arguments values are acceptable. It can facilitate validation of the user's job configuration. The argument set up from the user can be validated against its data type and constraints before execution. This increases the probability of successful completion of the execution. The benefit of this validation is particularly obvious for long-running applications or for applications that form part of a workflow.

iv) CoLDel specifies the dependencies and conflict of arguments, which can help to validate the input dependencies.

Below is the schema definition for argument (named port in the schema).

```
- <void index="1">
  - <object class="uk.ac.neurogrid.appws.service.common.Port">
    - <void property="id">
        <int>2</int>
      </void>
    - <void property="portName">
        <string>-t</string>
      </void>
    - <void property="portDisplayName">
        <string>Input Image Type. 1-T1, 2-T2, 3-PD(default T1)</string>
      </void>
    - <void property="portDescription">
        <string>Input image type. 1-T1, 2-T2, 3-PD(default T1)</string>
      </void>
    - <void property="mandatory">
        <boolean>false</boolean>
      </void>
    - <void property="argumentType">
        <object class="uk.ac.neurogrid.appws.service.common.ArgumentType" field="pair" />
      </void>
    - <void property="cmdLineRepresentation">
      - <object class="uk.ac.neurogrid.appws.service.common.CommandLineRepresentation">
        - <void property="flag">
            <string>-t</string>
          </void>
        - <void property="avalue">
            <string>1</string>
          </void>
        - <void property="defaultValue">
            <string>1</string>
          </void>
        - <void property="recommendValue">
            <string>NA</string>
          </void>
        - <void property="rangeValue">
            <string>NA</string>
          </void>
        - <void property="enumValue">
          - <array class="java.lang.String" length="3">
            - <void index="0">
              - <object class="java.lang.String">
                  <string>1</string>
                </object>
              </void>
            - <void index="1">
              - <object class="java.lang.String">
                  <string>2</string>
                </object>
              </void>
            - <void index="2">
              - <object class="java.lang.String">
                  <string>3</string>
                </object>
              </void>
            </array>
          </void>
        </object>
      </void>
    </object>
  </void>
```

Figure 3.4: Argument Data Type with Associated Metadata

As shown in Fig 3.4, the argument data type contains a lot of associated metadata for each argument of the algorithm command-line, like defaultValue, recommendValue, rangeValue, and so on. This can be used by third party portal or workflow to help guide the user in the setup of a valid parameter. In the case of argument set up, the portal can use the metadata from ADF to present the user with default value, recommended value and value range. The user then refers to this to set up the argument with a valid parameter. This helps the user to set up the arguments in a user friendly way. In particular, it is useful when the user does not have a lot of experience on applications. This assist in argument setup greatly reduces the chance of job failure and rapidly increases the opportunity to get better results from the application.

```
- <void index="9">
  - <object class="uk.ac.neurogrid.appws.service.common.Port">
    - <void property="id">
        <int>10</int>
      </void>
    - <void property="portName">
        <string>-or</string>
      </void>
    - <void property="portDisplayName">
        <string>Output restored image</string>
      </void>
    - <void property="portDescription">
        <string>Output restored image.</string>
      </void>
    - <void property="mandatory">
        <boolean>false</boolean>
      </void>
    - <void property="argumentType">
        <object class="uk.ac.neurogrid.appws.service.common.ArgumentType" field="flag" />
      </void>
    - <void property="outPortType">
      - <object class="uk.ac.neurogrid.appws.service.common.FileType">
        - <void property="ifdefault">
            <boolean>false</boolean>
          </void>
          <!--  when filename is number 15, means get file base from port 15  -->
        - <void property="fileName">
            <string>15</string>
          </void>
        - <void property="fileSuffix">
            <string>_restore.nii</string>
          </void>
        - <void property="fileOrPath">
            <object class="uk.ac.neurogrid.appws.service.common.FileOrPath" field="filePartialNeedSuffix" />
          </void>
        - <void property="fileFormat">
            <object class="uk.ac.neurogrid.appws.service.common.FileFormat" field="NII" />
          </void>
        - <void property="portDirection">
            <object class="uk.ac.neurogrid.appws.service.common.PortDirection" field="out" />
```

Figure 3.5: Input/output Argument Data Type with Associated Metadata

Fig 3.5 shows that for all the input files and output files, "FileType" is defined to provide full information regarding the file, like “WebDavDir” "filename," "fileFormat," etc. All the well known file formats are defined in the type "FileFormat”. This

information allows the workflow composer to compose services by connecting outputs of a service to the inputs of other services only if they semantically correct match.

CoLDel, the abstraction of applications description, provides a protocol to describe the command oriented applications. It renders the Application Service the capability to describe and process any command oriented application in a generic way. CoLDel is not only used by Application Service, but is also shared by client and workflow. Different parties follow the same protocol to specify and process the application description, which makes the application arguments set up an easy and safe task. Therefore, it avoids the failure or poor performance that is caused by improper arguments setup. CoLDel is also very helpful for workflow to connect the services and orchestrate the input/output files in a controlled manner.

On the client side, the client builds strongly typed data defined by CoLDel to supply specific argument values for the application through an incoming SOAP message. The input data arguments have to be set by the user. The user needs to set up the Uniform Resource Identifier of the input files. On the service side, the service provider writes an ADF for each application following CoLDel before service deployment. At run time, Application Service validates the client setup and sets up other mandatory arguments based on the information provided in ADF. JAVA XMLDecoder and XMLEncoder are applied for converting an "algorithm" object to and from its equivalent XML document representation. The Application Service generates an "algorithm" object from associated ADF, combines this with the "algorithm" object provided by the client, determines the specifics on how to properly build and construct the command with the appropriate parameters, and subsequently submits the job.

3.5.5 Application Description File

The CoLDeL schema defines all the data type needed to describe the application, including "FileFormat", "ArgumentType", "CommandLineRepresentation", "Port", and so on. An Application Descripion File is an XML document created from CoLDel schema to fully describe a specific application. It provides meta-data about an existing application and is usually supplied by the service provider or application provider. The information provided in an Application Description File defines the semantics of an application and enables the Application Service to expose an application automatically as a Web service.

Before wrapping the application, the application provider must write an Application Definition File for this individual application conforming to CoLDeL schema. The ADF has to be complete enough so that the service can dynamically compose the command-line at run time and retrieve the input data files. As a consequence, the ADF contains the following information that can be categorized into three categories:

i) General information. This includes the algorithm name, contributing institution, versioning information, and brief description.

ii) The execution environment information. This describes the requirement on the execution environment such as platform, libraries required and environmental variables. This information is used to construct the execution environment at run time.

iii) The argument's description. This provides all the information for each argument of the algorithm command-line, including mandatory information, argument type, default values, value range, dependency information, naming conventions of outputs, and so on.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <java version="1.5.0_10" class="java.beans.XMLDecoder">
  - <object class="uk.ac.neurogrid.appws.service.common.Algorithm">
    - <void property="algorithmName">
        <string>bet</string>
      </void>
    - <void property="algorithmVersion">
        <string>1.0</string>
      </void>
    - <void property="algorithmDescription">
        <string>Brain segmentation tool.</string>
      </void>
    - <void property="location">
        <string>/home/ng/medicalApp/fsl-3.3.7/bin</string>
      </void>
    - <void property="exec">
        <string>bet</string>
      </void>
    - <void property="category">
        <string>segmentation</string>
      </void>
    - <void property="subcategory">
        <string>brain extraction</string>
      </void>
    - <void property="environmentVars">
      - <array class="uk.ac.neurogrid.appws.service.common.EnvironmentVar" length="3">
        - <void index="0">
          - <object class="uk.ac.neurogrid.appws.service.common.EnvironmentVar">
            - <void property="varName">
                <string>FSLDIR</string>
              </void>
            - <void property="varValue">
                <string>/home/ng/medicalApp/fsl-3.3.7</string>
              </void>
            </object>
          </void>
```

Figure 3.6: ADF_BET Part1

```
- <void index="1">
  - <object class="uk.ac.neurogrid.appws.service.common.EnvironmentVar">
    - <void property="varName">
        <string>PATH</string>
      </void>
    - <void property="varValue">
        <string>${FSLDIR}/bin/:${PATH}</string>
      </void>
    </object>
  </void>
- <void index="2">
  - <object class="uk.ac.neurogrid.appws.service.common.EnvironmentVar">
    - <void property="varName">
        <string>LD_LIBRARY_PATH</string>
      </void>
    - <void property="varValue">
        <string>${FSLDIR}/lib:${LD_LIBRARY_PATH}</string>
      </void>
    </object>
  </void>
 </array>
</void>
- <void property="others">
  - <array class="java.lang.String" length="2">
    - <void index="0">
      - <object class="java.lang.String">
          <string>. ${FSLDIR}/etc/fslconf/fsl.sh</string>
        </object>
      </void>
    - <void index="1">
      - <object class="java.lang.String">
          <string>export FSLOUTPUTTYPE=NIFTI</string>
        </object>
      </void>
    </array>
  </void>
```

Figure 3.7: ADF_BET Part2

Figures 3.6 and 3.7 indicate a simplified example of an ADF used for a medical image brain extraction service. BET ADF that conforms to the CoLDeL schema has three main elements: serviceInfo, environmentVars, and ports.

First, some generic information related to the application is included, like algorithmName, algorithmVersion, algorithmDescription and executable name and location. As the elements' names imply, they specify the name and version of the algorithm, a short description of the algorithm, and the name of the executable and its location.

Then environment variables are specified inside of the <environmentVars> element, if any, that need to be passed on to the application before executing it on this host.

The environmental variables FSLDIR, PATH and LD_LIBRARY_PATH are specified for the BET application.

The generic information and environmental variables information are used by Application Service only. This is vital information for Application Service to create the right environment and retrieve the right executable.

```
- <void property="ports">
  - <array class="uk.ac.neurogrid.appws.service.common.Port" length="13">
    - <void index="0">
      - <object class="uk.ac.neurogrid.appws.service.common.Port">
        - <void property="id">
            <int>1</int>
          </void>
        - <void property="portName">
            <string>bet</string>
          </void>
        - <void property="portDisplayName">
            <string>bet</string>
          </void>
        - <void property="portDescription">
            <string>bet.</string>
          </void>
        - <void property="mandatory">
            <boolean>true</boolean>
          </void>
        - <void property="argumentType">
            <object class="uk.ac.neurogrid.appws.service.common.ArgumentType" field="avalue" />
          </void>
        - <void property="cmdLineRepresentation">
          - <object class="uk.ac.neurogrid.appws.service.common.CommandLineRepresentation">
            - <void property="avalue">
                <string>/home/ng/medicalApp/fsl-3.3.7/bin/bet</string>
              </void>
            </object>
          </void>
        </object>
      </void>
```

Figure 3.8: ADF_BET Part3

Moreover, Figs 3.8 and 3.9 show that the command arguments information is defined inside of element <ports>, which is an array that contains many <port> elements. Each port stands for an argument of the algorithm command-line. It contains all the meta-data related the argument, e.g., id, portName, portDisplayName, portDescription, mandatory, argumentType, cmdLineRepresentation, conflictPorts, dependPorts, inPortType, outPortType and so on.

"mandatory" specifies if this argument is mandatory to be set. "conflictPorts" explains which arguments cannot bet set together with this argument.

"argumentType" tells the format of argument appears on the command-line. The possible "argumentType" are "FLAG" (e.g., -debug), "VALUE" (e.g., inputfile.suffix), "PAIR" (e.g., -Sx1 100) and "NOT". "cmdLineRepresentation" specifies how the argument exactly appears on the command-line. For each argument, "cmdLineRepresentation" is made up of two parts, "flag" and "avalue". "flag" is always

static. “avalue” is dynamic, and can be set by user or workflow. ADF provides a lot of associated meta-data for the “avalue” setup, like "defaultValue", "recommendValue", "enumValue" and "rangeValue". These meta-data can be used by the portal to help guide the user in the selection of a valid parameter. It is very helpful in terms of validation and assistance of arguments setup.

```
- <void index="7">
  - <object class="uk.ac.neurogrid.appws.service.common.Port">
    - <void property="id">
        <int>8</int>
      </void>
    - <void property="portName">
        <string>-f</string>
      </void>
    - <void property="portDisplayName">
        <string>Fractional intensity threshold</string>
      </void>
    - <void property="portDescription">
        <string>Fractional intensity threshold (0--1); default=0.5; smaller values give larger brain outline estimates</string>
      </void>
    - <void property="mandatory">
        <boolean>false</boolean>
      </void>
    - <void property="argumentType">
        <object class="uk.ac.neurogrid.appws.service.common.ArgumentType" field="pair" />
      </void>
    - <void property="cmdLineRepresentation">
      - <object class="uk.ac.neurogrid.appws.service.common.CommandLineRepresentation">
        - <void property="flag">
            <string>-f</string>
          </void>
        - <void property="avalue">
            <string>0.5</string>
          </void>
        - <void property="defaultValue">
            <string>0.5</string>
          </void>
        - <void property="rangeValue">
            <string>0;1</string>
          </void>
        </object>
      </void>
    </object>
  </void>
```

Figure 3.9: ADF_BET Part4

“inPortType” and “outPortType” provides information for the input file and output file, respectively including “ifdefault”, “WebDavDir”, “filename”, “fileFormat”, “filePrefix”, “fileSuffix”, etc. In some applications, the output file name is based on the input file name. “dependPorts” tells where to get this input file name.

The information contained in <ports> is shared by the user, workflow, and Application Service.

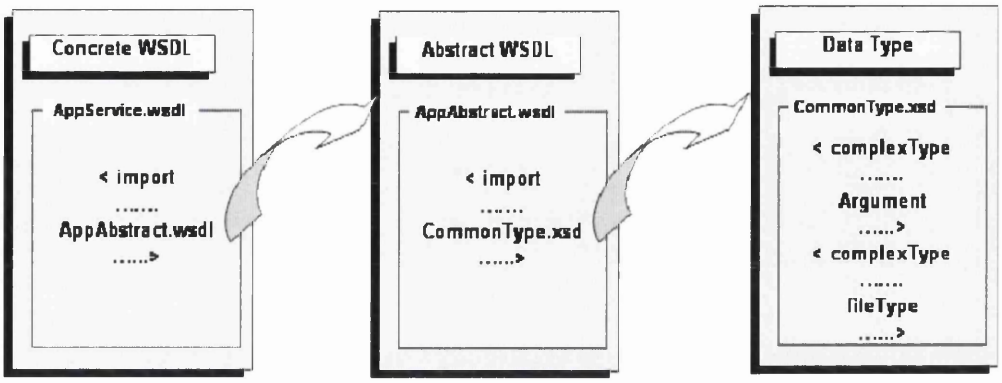

Figure 3.10: Web Service Descriptions Modularisation

3.5.6 Modular WSDL

Figure3.10 shows how concrete WSDL imports abstract WSDL, which imports common data type schema.

As mentioned above, an initial number of types and a rich set of elements are declared in the WSDL which ensure strongly typed data exchange between client and service. It makes the WSDL interface unstable due to frequent change of the data types. Particularly in our case, most data types are designed to describe the applications. With further applications being brought in, the data types are subject to frequent negotiation and revision. A modular WSDL is introduced to keep the WSDL stable, in which the type declarations of a Web Service are moved into a separate document.

All the application related types are separated from the WSDL definitions and put in an XML schema document named commontype.xsd.

The NgappServiceAbstract.wsdl file defines what the Application Service does by defining the data types and business operations of the Web Service. The file imports XML schema commontype.xsd as immediate children of the <wsdl:types> element, and defines different <wsdl:message> and <wsdl:portType> elements.

The NgappService.wsdl is the concrete WSDL file that defines how and where to invoke a service by defining network protocol and service endpoint location with the <wsdl:binding> and <wsdl:service> elements. The NgappService.wsdl file incorporates

the NgappServiceAbstract.wsdl file using <wsdl:import> or <wsdl:include>. These elements should be the first immediate children of the <wsdl:definitions> element. <wsdl:include> is used when two wsdl files have the same namespace and <wsdl:import> is used to combine wsdl files from different namespaces. This approach greatly improves component re-usability as the same NgappServiceAbstract.wsdl file can have multiple service bindings.

Abstraction of the Web Service type declarations into a 100% XML Schema compliant data model has many important advantages. This encourages collaboration between the different partners involved in the data model design process and assures interoperability. This also leverages the advanced capabilities of XML schema for precisely constraining complex scientific data.

Abstraction of the Web Service type system into a 100% XML Schema compliant data model produces several important advantages:

i) Separation of Roles. The type declarations are fully abstracted and developed in isolation from the network protocol and communication specific details of the WSDL file. In doing this, the focus becomes centred upon the business/scientific requirements of the data model. This greatly encourages collaboration between the scientists who are involved with the description of scientific data and data model design.

ii) Data Model Re-usability. Existing XML Schema can be re-used instead of re-designing a new type system for each new Web Service. This helps reduce development efforts, cost and time.

iii) Isolation of Changing Components. The data model is the component that is most subject to change, often in response to changing scientific requirements. Its isolation therefore limits the impact on other Web Service components such as the concrete WSDL file implementation.

iv) Full XML Schema Functionality. The XML Schema type system leverages the more powerful features of the XML Schema language for description, constraint and validation of complex data. This has proven invaluable for the description and constraint of complex scientific data.

3.5.7 Data Management

In many scientific enterprises, the input files are large and they may be stored in some remote location. Consequently, we do not assume that the user will upload the file directly to the portal or the service, though this is possible. Rather, Uniform Resource Identifier (URI), a compact string of characters for identifying an abstract or physical resource, is adopted to provide the direct link to data files. It is often better to pass the Uniform Resource Identifier (URI) for the input files to the service.

The Data Management Component handles data transfer between two sites using WebDav. The Application Service interacts with the Data Management Component to transfer data between service provider and data repositories. It retrieves the input data from the data repositories and puts the results back to the data repositories. The information exchanged between the Application Service client and service via SOAP only contains references of input/output, the URIs. No large data is included in SOAP messages. Since we are using WebDav to manage data, the URI contains information of the WebDav server, WebDav folder name, and file name.

We believe using a common data transfer protocol would eliminate the current duplication of effort in developing unique data transfer capabilities for different storage systems. For the current implementation, Application Service only takes input data located in the WebDav folder on the remote data server. WebDav is used to transfer data files between the Application Service and data server, which is a secure, efficient data transport mechanism. If needed, other data management plug-ins are possible and can easily be fitted into the Application Service due to the component plug-in model used.

When a client invokes an application, the Application Service creates a new working directory for each job at the server site. The input data files described via URIs are archived from data repositories to the corresponding working directory through WebDav. The application is run inside the working directory in the FORK case, or submitted to a computational resource. The output will be brought back to this working directory from the remote computational site. Finally, Application Service transfers all the output files back to the WebDav folder on the data server through WebDav. The URIs of output files are provided which allows them to be located and retrieved later.

The data upon which each job acts may come from two different sources: 1. Users may directly upload data files from their local machine to a remote data server. In this case, users can use the WebDav browser to drag the files to the data server directly in an easy and secure way. 2. Users may issue a query via a web interface. This query is resolved against one or more distributed databases. The matching data set can be used in later computations.

The "applicationName" along with the attribute "targetNamespace" uniquely identify the application. The "hostName" specifies the name of the host on which the application has been deployed whereas the "executable" specifies the location of the application on that host. The "tmpDir" specifies a temporary directory that the application service can use to stage input files to the application. It is also used to temporarily store log files, standard out, standard error, intermediate files and output files of the application.

3.5.8 Security

Security is a critical requirement and must be accounted for by any geographically distributed Grid community. There is high demand to protect data confidentiality and integrity in Grid enabled applications. The traditional security mechanisms for homogeneous systems do not scale to heterogeneous environments operated by different organizations. A reliable yet easy to use security infrastructure is therefore important.

There are three requirements of the security: authentication, integrity and confidentiality. Authentication is to ensure that parties within a business transaction are really who they claim to be. Integrity is to validate the integrity of business information exchanged in a transaction ensuring that a message's content has not been altered or corrupted during its transmission over the Internet. Confidentiality is to make the information exchanged in Web services requests and responses unreadable. The purpose is to ensure that anyone accessing the data would need the appropriate algorithms and security keys to decrypt the data before being able to access the actual information.

Transport-level mechanisms like HTTP over Secure Sockets Layer (HTTPS) provide these capabilities, but transport level security isn't flexible enough for some applications. The difference of HTTPS and WS-Security are exhibited below:

i) Transport based security (HTTPS) is bound to HTTP. It only secures the transport channel between two points. This means HTTPS provides point-to-

point security, securing the connection between the sender and receiver of the message. This solution is incomplete if intermediaries between the endpoints forward or process the message. While with WS-Security, message itself is secure rather than underlying transport. Signature and encryption persists with messages.

ii) HTTPS is bound to HTTP. While web services are decoupled from an underlying transport, and can use transports like SMTP and JMS in addition to HTTP.

iii) HTTPS encrypts the entire message, while WS-Security can encrypt only a portion of the message.

It is important to justify the decision to use WS-Security because WS-Security has an impact on the overall response time and the number of simultaneous requests that the service can support. HTTPS provides a significantly better performance solution than what is possible when using XML Digital Signature and XML encryption. For NeuroGrid Application Service, business logic is more complex and distributed on remote systems. The overall processing times of business logic executed by the Application Service implementations are long. The impact of WS-Security on response time does not result in much a difference. Also due to stringent requirements for patient confidentiality and authenticity of results, we choose WS-Security instead of transport level security to ensure secure data management in a distributed environment. For NeuroGrid Application Service, digital signature and message level encryption capability provided by WS-Security is the good choice. We have a full implementation based on WS-security. The Application Service addresses the three security requirements outlined below:

i) Authentication is used to ensure the identity of the message senders.

ii) Digital signatures are used to ensure a message's integrity, that its content has not been altered or corrupted during its transmission over the network.

iii) Encryption is used to ensure message confidentiality.

Every user uses an X.509 certificate issued from NeuroGrid CA, a trusted Certificate Authority. NeuroGrid CA issues two types of certificates, NeuroGrid user certificate

and Guest Certificate. NeuroGrid user certificate has the access to the data nodes belonging to the user's group, and all the Application Services. The Guest Certificate has access to all the Application Services, and one data node dedicated for guest users.

The certificate contains identity credentials and has a pair of private and public keys associated with it. The proof of identity presented by a party includes the certificate itself and a separate piece of information that is digitally signed using the certificate's private key. By validating the signed information using the public key associated with the party's certificate, the receiver can authenticate the sender as being the owner of the certificate, thereby validating their identity.

In NeuroGrid Application Service, the SOAP message is digitally signed. We use the private key of the sender's X.509 certificate to digitally sign the SOAP body of a Web service request. Likewise, a Web services response is digitally signed to ensure data integrity.

An X509Token is used to allow Application Service to have fine grained control on what operations users can invoke. Token is an XML document which is a detailed policy document that authorizes a user to invoke a particular operation on an Application Service. It is signed by the Application Service's credentials (X509 proxy certificate) and is issued to the user. The user needs to present the token to the Application Service before it can invoke an operation. Thus X509Token allows Application Service providers to have fine grained control on what operations users can invoke on their Application Services. X509Token is automatically generated by Application Services and is completely transparent to Application Service providers.

The security set up can be conveniently controlled by Security Configuration Files. In a sign-encrypt set up, the client first constructs a SOAP message to invoke the service. If the Application Service supports tokens, the client attaches the user's capability token to the SOAP message, signs the message with the user's X509 proxy certificate and encrypts it. Then the user sends the message to the service.

On the server side, the service first decrypts the message. Then it verifies the signature on the SOAP message. It then verifies the capability token to ensure that it has not expired and ensure that the token authorizes the user to invoke the operation on the service. The SOAP message is then sent to the SOAP Message Processor for further processing. The server signs and then encrypts and sends the response.

In a typical usage scenario, the user invokes the service provided by the service provider, and then the service provider accesses compute resources on behalf of the user. This means compute resources have to trust all the users trusted by service provider, and hence brings the account management burden to the resource provider. We introduce a Service Provider Account Model to avoid this problem. In this model, the user does not have direct access to compute resources and they are completely decoupled from compute resources where jobs are effectively run. The user only has access to the Application Service. A special user account is set up for the service provider, which is trusted by compute resources. The service provider acts as an active agent between the user and compute resources. It authenticates and authorizes the users, and serves the user requests. It also accesses related compute resources on behalf of itself. The compute resources respond to these requests due to their trust of the service provider.

3.5.9 Service Provision and Deployment

We assume that the application has been already deployed on some host or has been ready to migrate to run on some compute resource. This is because Application Services do not attempt to deploy any application. Hence, in order to generate a customized Application Service that can be deployed in the Application Service hosting environment (i.e., Apache Tomcat) as a Web Service, the application installation or preparation is first. Also an Application Description File, which specifies the installation and configuration details, has to be provided. A security control file has to be in place to control the security setup of the Application Service, like signing, encrypting, and so on.

```
#webdav client certificate directory, used by ngapp service
server.aswebdavclient.certs=/home/ng/certs/webdavclient

#alias of webdav service at ngapp server side
#ngnodem03.cs.ucl.ac.uk.webdav.alias=ngnodem03.cs.ucl.ac.uk
#ngnodem14.ediamond.ox.ac.uk.webdav.alias=ngnodem14.ediamond.ox.ac.uk

#webdav alias on ngnodem14
ngnodem01.ediamond.ox.ac.uk.webdav.alias=ngnodem01
ngnodem02.ediamond.ox.ac.uk.webdav.alias=ngnodem02
ngnodem03.cs.ucl.ac.uk.webdav.alias=ngnodem03
ngnodem04.cs.ucl.ac.uk.webdav.alias=ngnodem04
ngnodem05.cs.ucl.ac.uk.webdav.alias=ngnodem05
ngnodem06.ediamond.ox.ac.uk.webdav.alias=ngnodem06
ngnodem07.ediamond.ox.ac.uk.webdav.alias=ngnodem07
ngnodem08.ediamond.ox.ac.uk.webdav.alias=ngnodem08
ngnodem11.ediamond.ox.ac.uk.webdav.alias=ngnodem11
ngnodem12.ediamond.ox.ac.uk.webdav.alias=ngnodem12
ngnodem14.ediamond.ox.ac.uk.webdav.alias=ngnodem14
ngnodem15.ediamond.ox.ac.uk.webdav.alias=ngnodem15

#Intermediate output directory
intermediate.output.directory=/home/ng/tomcat50-jwsdp/jobs

#ngapp implementation class
ngapp.impl.class=uk.ac.neurogrid.appws.core.App_ForkServiceImplementor
```

Figure 3.11: A Service Property File

Moreover, a service property file is set to describe some dynamic information that is required by an Application Service to be hosted on the specified host. It contains information closely related to the Application Service server, which is usually different based on the different server's environment and requirement. A sample service property is shown in Figure 3.11

server.asWebDavclient.certs specifies where the security certificates files are located, which are used to invoke the Data Management Service. WebDavserver.alias tells the alias for a specified data management server. Because the Data Management Service is hosted on 8 different nodes, we provide a list of aliases for all data servers. The Application Service can dynamically get the right alias based on which data server is in use. intermediate.output.directory tells the Application Service where to put all the intermediate output. ngapp.impl.class specifies which Job Submission Component plug-in (FORK, CONDOR, etc.) is to be used for Application Service.

An intuitive deployment tool has been developed for the generation and deployment of the customized Application Service. The deployment tool enables the application provider to upload an Application Description File via a portal interface. The ADF file

will be registered with the ADF Pool. At the service provider side, the deployment tool generates an appropriately customized Web Service, which encapsulates the application, and finally deploys the service within the remote Application Service hosting environment.

3.5.10 ASToolkit Client Environment

ASToolkit client applications usually run on PCs or workstations connected to the Internet. ASToolkit client environment offers the ASToolkit client API, which has a high level Java API interface to hide the complexity of accessing remote Application Services from the user. ASToolkit client API is mainly used for communicating with the Application Services. It also handles the preparation of service input data and post-process the service output data. The API may be used not only by the end user, but also by client side application developers, like portals, to construct advanced Grid applications that interact with remote Application Services.

Figure 3.12 shows that ASToolkit client contains several layers, high level Java API interface, client business logic (job submission, job steering), a security handler, and service proxies. The top layer of the ASToolkit client is the high level Java API interface. The Java API provides a set of classes for dealing with job submission and job steering at a high level of abstraction hiding the details of the underlying interaction with Application Services. The API hides the complexity of dealing with remote Application Services from the user.

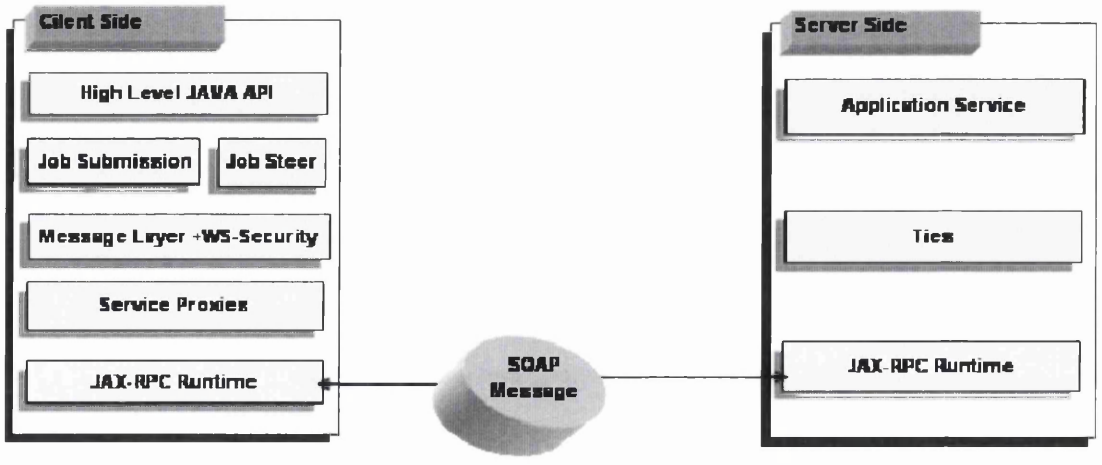

Figure 3.12: ASToolkit Client API Abstraction Layers

Security is handled at the lower layer of the ASToolkit client and is completely hidden from the user. This prevents the user from dealing with low level issues such as message generation, signing, and encryption.

In the lowest layer, ASToolkit client provides service proxies for the toper layer to interact with. Service proxies are stubs that are placed on the client and connect a web service client to the JAX-RPC runtime. ASToolkit automatically generates these and packs them in ASToolkit client. The user does not have to deal with the complexities of stub generation of client side. Service proxies are in charge of interacting with remote Application Services and handle job execution and job monitoring, as well as job steering.

Jobs can be executed synchronously or asynchronously. If they are executed synchronously, the client is blocked until the remote execution is complete. The job outputs are returned as a response to the original request. However, this style of invocation is not always appropriate. Jobs may possibly spend a lot of time being queued if the resources happen to be heavily loaded. Furthermore, if the jobs are long running, the client will stay blocked until the job finishes, or possibly times out.

To overcome this shortcoming, in ASToolkit client, jobs are launched asynchronously. A response is immediately sent back to the client with a jobID for the job being executed. The clients can use this jobID to query the service for job status and job metadata at a later time. This makes the service stateful. Apart from the job status and metadata about job inputs, outputs and other command arguments setup, the service status also includes file transfer information and job history. Files are used to keep all this information on the server side, and are accessed via both Application Service and a generic steer service.

3.6 A Sample to Wrap an Application

To understand how the ASToolkit can be used to wrap the applications we describe a typical use sample. (Let us)Suppose there is an application BET needed to be wrapped as a Web Service. BET [9] is Brain Extraction Tool that runs from the command-line with a set of arguments options. BET expects to find an input file and produces an output file. The service provider would like to create an Application Service that will allow all the qualified users (the people who hold NeuroGrid certificates) to run this

program. We assume that BET has been installed on some host. For example, (let us) assume that it is installed at Linux box hairyviolet.cs.ucl.ac.uk and located in the directory path /home/ng/medicalapps/bet. ASToolkit does not do anything related to application deployment or installation. We install the application in the same directory path on different servers. ASToolkit allows the application to be installed anywhere as long as it is accessible because the location of application is specified in the Application Description File and will be dynamically loaded by Application Service at run time.

There are the following actions that our application provider and service provider must take to create a BET Application Service:

i) Write an Application Description File (ADF) for BET, which is a simple Extensible Markup Language (XML) description of the application conforming to CoLDel schema. ADF is described in detail in section 3.5.5. An ADF_BET is provided in Figure 3.5. The document contains general application information, the execution environment information, and application arguments information. If the application is to be run on Condor or another Grid environment, ADF_BET needs to list all the executables to be staged into the remote execution node.

ii) Edit Security Policy File, SPF_BET, contains the policy information concerning which individuals and groups are authorized to invoke the service, and how to secure the message between users and service. The application provider can simply use the default Security Policy File without even knowing this.

iii) The application provider can upload the ADF_BET and SPF_BET from the portal through a Web interface. The executables are also needed to be uploaded for the non-fork Application Service. Then the service developer will take over the task and wrap BET as an Application Service using ASToolkit. Or the application provider can wrap it using ASToolkit with minimum software installation, ANT and Java.

iv) The service provider manually edits the service property file which contains some dynamic information that is required by an Application Service to be hosted on the specified host.

v) The service provider gets ready the ADF_BET, SPF_BET and/or executables. Then the provider manually edits the ANT property file to let the ANT know

where to find all these ADF, SPF and/or executables. Executables are only prepared and put in the right location known by ANT script for the none-fork Application Service.

vi) The service provider runs the ANT script through command-line, or eclipse. ANT script includes service building and remote service deployment.

3.7 Summary

In this chapter, we have presented CoLDel, an XML based language for describing individual command-line applications precisely. CoLDeL provides an standard to service providers, so that the service providers can generate an Application Definition File for each scientific algorithm for use by the Application Service. CoLDeL is also a protocol between client and service, which makes the exchange between client and service easy. It also facilitates generic Web service workflow tools to compose Application services easily.

We also have presented ASToolkit, a toolkit for wrapping scientific applications as Web services. We described the technical details of the ASToolkit architecture and implementation. We describe a typical use sample to demonstrate how the ASToolkit can be used to wrap the applications.

CHAPTER 4

NeuroGrid Framework

4.1 Introduction

Current neuroimaging research is characterized by small-scaled studies carried out in single centers. Many groups make their algorithms available for download over the Web, which is not a convenient or efficient way for neuroimaging research groups to share algorithms for regular image analysis. Also it is very unusual to share data. When data is shared, subtle differences between the image acquisitions normally inhibit reliable quantitative analysis of aggregated data. Furthermore, data curation in neuroimaging research tends to be poor, with images normally archived on removable storage media that rapidly become obsolete, making aggregation of data between or within sites difficult, if not impossible, on those occasions when it is desired [42-44].

NeuroGrid framework helps to overcome these problems by the integration of image storage and image analysis algorithms and by the collaboration of work within sites. The NeuroGrid framework connects sites for rapid and secure flow of data, enables distributed data analysis with image analysis tools and interoperable databases, and enhances collaboration between researchers in different clinical and methodological areas. Therefore, this framework can aid data sharing, data analysis sharing, and compute resource sharing. It allows current algorithms and existing data management procedures to be more accessible and interoperable in the Grid environment, so there are low barriers to sharing and time is not wasted on re-engineering well established algorithms.

Currently, one of the main obstacles for neuroimaging research and other academia field take-up of Grid technology is the existence of a large amount of legacy code that is inaccessible as Grid/Web services. The ASToolkit concept and its integration with the

Grid portal technology eliminates this problem and can lead to a breakthrough in the establishment of scientific Grids. NeuroGrid framework stretches the usability of scientific Grids where most of the codes are written in FORTRAN or other languages, and now all these applications are accessible as Application services. The framework also enables users to discover these Application services, interact with them, and interact with composed scientific workflows from a user friendly Web portal. The framework is built for neuroimaging community, but the approach is generic and can be used for any other community.

In this chapter, we present a detailed description of the NeuroGrid framework, which forms a basis for some of the work introduced in this dissertation. First, we present the design goals for NeuroGrid, and then discuss its architecture in detail. We also highlight the Abstract Application Service, Group Applications Service Optimization and Scientific Workflow mechanisms and describe how they are implemented and used within the framework. Finally we conclude with some real user cases in scientific communities.

4.2 Design Goals

NeuroGrid is a large-scale system addressing the key issues of data- and services-sharing in an open and changing community. The complexities of the NeuroGrid are two-fold: the heterogeneity of initiatives and stakeholders and the high level of specialization of the various sub-fields that can contribute to the Grid system. Both of these complexities pose several challenges to requirements engineers and architecture designers. In such a context, a clear understanding of the key goals and requirements is crucial to develop a system that fits the users' expectations.

4.2.1 Functional Requirement

NeuroGrid framework has four primary functional goals:

4.2.1.1 Legacy algorithms easily to be accessed

We need to provide an ability to allow legacy algorithms easily to be accessed and run in the Grid environment.

We need to provide mechanisms to migrate the legacy applications to dynamic Grid environments. Also, we wish to provide mechanisms to support job monitoring and job restart for fault tolerance.

Most large scale computational facilities have traditionally operated their machines in batch mode; the toolkit is focused on these batch mode applications. The toolkit should be able to wrap almost any command-line application (i.e., non-graphical), such as UNIX commands or more sophisticated scripts written in Python, Perl, and so on. It should not require any modification to the wrapped applications. A toolkit will provide a set of tools that wraps scientific applications as Web services. The toolkit is also able to orchestrate generation, deployment and installation of the Application service.

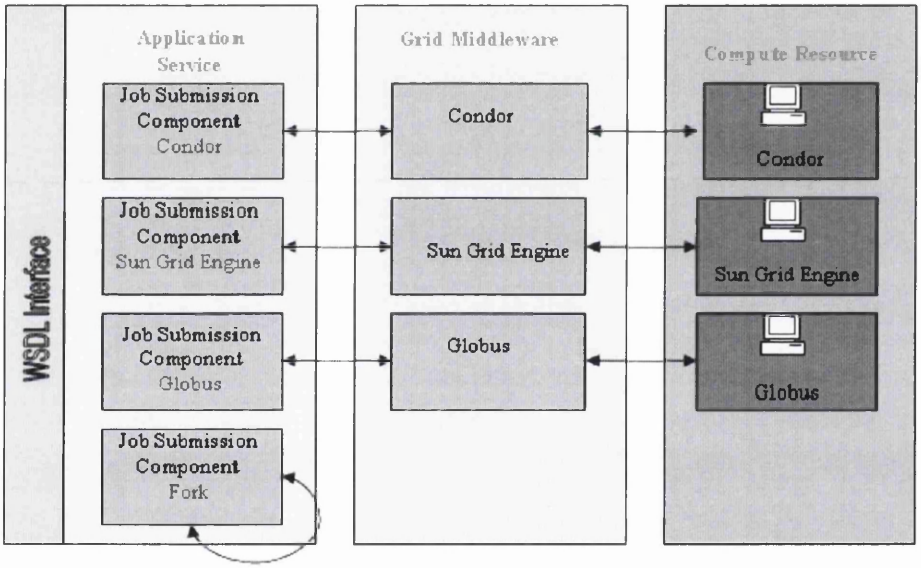

Figure 4.1: Grid Enabled Application Service

As depicted in Figure 4.1, algorithms will be wrapped and presented as Web services in NeuroGrid framework, which is termed as Application services. The exact implementation details of the algorithm and platform will be abstracted away using a common Web service interface.

Figure 4.1 also shows that the role of the Application service is to provide a uniform submission layer on top of different execution environments. Various Job Submission Components can run the legacy algorithms in the various Grid environments, such as Globus, Condor, and Sun Grid Engine.

The Application service can enable but does not require the use of distributed resources via the Grid. The Application service with job submission component FORK can run the application locally without complicated Grid infrastructure.

4.2.1.2 Data to be more accessible and interoperable

We need to allow existing data management procedures to be more accessible and interoperable. Data Access provides uniform access to heterogeneous data repositories. Metadata uses metadata to discover data, and explicitly query metadata registries for metadata about data within the system. Data publishing provides basic means through which distributed data can be published within the system and accessed and retrieved later.

A federated database will be built to facilitate the data sharing. A data management service will allow user to publish and query to and from the federated database. This service also provides functionalities for files editing, management, and transfer on the remote server using WebDav protocol. A fine-grained access control system is developed to provide arbitrarily complex access controls on the federated database. This service was developed by NeuroGrid Oxford team, and has been integrated with our NeuroGrid framework seamlessly.

4.2.1.3 Graphical user interfaces to access applications and data

We need to provide graphical user interfaces to access a large number of Application services and federated database from a scientific portal, and yet keep the portal lightweight and manageable. We will provide a Web portal which is a gateway through which users may access services, invoke workflows, and manage data. There are two types of users: service providers who use the portal to create the application services for legacy applications and end users who interact with the services through the Web interface.

4.2.1.4 A lightweight workflow composer

We need to provide a lightweight workflow composer to compose sequential workflows from Application services. It will be possible to define and store sequential workflows within NeuroGrid that will join together algorithms wrapped as Application services.

4.2.2 Architectural Requirement

Below are the important high level requirements for the system.

4.2.2.1 Languages

Java TM and J2EE are proven technologies with multiple vendors offering compatible products. It is a good platform for deploying Web services, and offers the good solutions to the really hard part of developing Web services, such as security, messaging capabilities, distributed transaction management, connection pool management, and handling huge numbers of simultaneous users [45].

4.2.2.2 Platform

Linux is the platform of server that will host the Application toolkit, Application services, data management services, workflow services, and Web portal. Client-side software should support any platform.

4.2.2.3 Modularity of Components

The software should be constructed in a modular way based on components, where a software component provides specific functions via a well defined public interface [46]. Components interact with other components through their interfaces. It should be possible to replace a component with a different implementation respecting the same interfaces without perturbing the rest of the system.

This also ensures no one component in the system is responsible for providing all of its capabilities. This principle allows components to support "plug and play" architecture.

4.2.2.4 SOA

It is intended to build the NeuroGrid infrastructure using WS-I [47] compliant Web services with a minimal set of extensions which currently include WS-Security.

The Web service interactions will be abstracted away by the Java APIs. However, it will be perfectly possible to achieve full functionality by talking directly to the underlying Web services.

4.2.2.5 Distributed Environment

The system should enable, but not require, the use of distributed resources via the Grid. The system should support operation seamlessly in a highly distributed

environment. The Grid-enabled functionalities are enabled and controlled by components employing Grid middleware. The system should concern distributed operation in its design and should use the agreed standard services for distributed operation. While the system should fully support the distributed environment, it also should be used easily in local environments.

4.2.2.6 Fault Tolerance and Robustness

Security services should not have any possible single point of failure. Data management services and Application services should show some degree of fault tolerance.

4.2.2.7 Extensibility and Modifiability

It must be possible to add new services and resources to the system once deployed.

4.2.2.8 Integrate-ability

The system must integrate heterogeneous components whether project-specific or legacy.

4.2.2.9 Technology Independence

The underlying implementation does not dictate the architecture of the system, and vice versa.

4.2.2.10 Security

Security is a key in the Grid environment. The core of the security infrastructure will be an X.509 [48] certificate authority used to issue NeuroGrid certificates. These certificates will be required for any direct connection to the NeuroGrid. It will not be possible to download any file or invoke any Application service without a certificate. Each user certificate will contain the name of the user as well as their research group. The research group will be used to provide a simple degree of access control.

The standard security mechanisms, like WS-Security, will be used to enforce integrity and confidentiality on Application services messaging.

Baseline security architecture is as follows:

i) User is authenticated at the Web portal by using his/her credential, an X.509 certificate issued by a trusted certification authority.

ii) When interacting with support components, the user is identified by his X.509 certificate.

iii) Authorization and access control are performed on each data node employing local access rules.

4.2.2.11 Access Control

The most basic level of access control will be that data access will only be granted to people with valid NeuroGrid certificates. A simple level of access control will be that files on each node will be divided into those which can be accessed by any other member of the NeuroGrid consortium and those that can only be accessed by the local research group; this will be achieved using the research group identifier that forms part of the user's certificate. More fine grained access controls will be enabled by use of XACML (eXtensible Access Control Markup Language) [49] policies at each of the local nodes. These can be used to provide arbitrarily complex access controls.

4.2.2.12 Interface

There are three ways of dealing with the NeuroGrid:

i) Java APIs -- These will provide the greatest functionality of all the interfaces; they can be used to create applications that deal directly with the NeuroGrid using a local user certificate.

ii) Portals -- The end-user (human) interface to Web applications will be through a portal, specifically a Web portal that will require the client to authenticate themselves using a NeuroGrid certificate. The Web portal is required to mask the complexity of the distributed environment from users while providing fully distributed functionality. Most services will be integrated and accessible through the portal. JSP is adopted to build the Web portal. Applets can be embedded in the portal for more interactivity.

iii) Web Servers -- Sometimes there is a need to support users who are not part of the NeuroGrid consortium. These people will not be given certificates and will not be able to use the portals or to download any files. All data presented to/received from the users will be handled by

the Web server. The Web server will have a NeuroGrid certificate which will enable it to interact directly with the NeuroGrid.

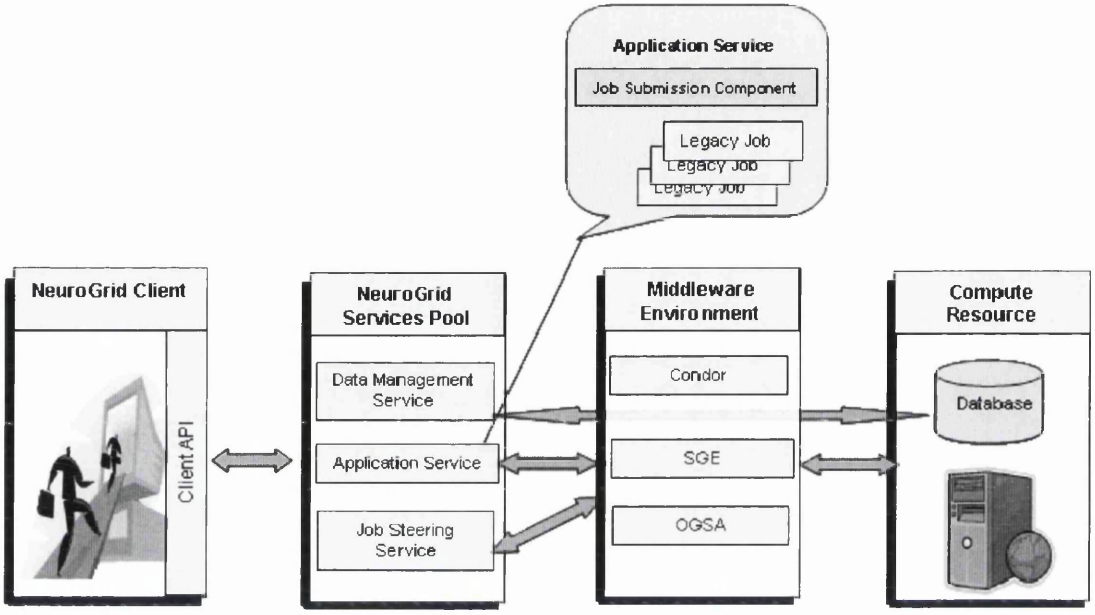

Figure 4.2: NeuroGrid Framework Architecture

4.3 NeuroGrid Framework Architecture

4.3.1 Architecture

NeuroGrid represents a general architecture for data and legacy applications sharing on the Grid environment. The high level NeuroGrid conceptual architecture is represented in figure 4.2. It is designed to be modular and adaptable based on a service-oriented approach. It is organized in multiple components to promote interoperability and allow reuse of core components. There are four basic components in the architecture as displayed in Figure 4.2, which are NeuroGrid Client, NeuroGrid Services Pool, Middleware Environment and Compute Resource. These components could be installed on physically different nodes by different roles. Each of these components simulates an encapsulated black box committed to deliver a well-defined functionality to the layer above that, independent of the underlying Grid middleware solution.

The end user communicates with the NeuroGrid Client by sending and receiving SOAP over SSL, and utilizes the NeuroGrid Client for core functionalities. The

NeuroGrid Client in turn forwards some incoming requests to NeuroGrid Services Pool and relies upon underlying Web services to provide functionalities to the client.

The NeuroGrid Client is built into a Web portal to provide a user-friendly Web-based interface through which the end user can remotely launch and monitor the algorithms. It also provides the interface to publish and query the data to and from the global image databases. It provides seamless integration of a collection of legacy applications and data across geographically distributed virtual organizations. The NeuroGrid Client presents the data and applications to the end user through a browser and an Internet connection, and hides the end user from the complexities of the underlying Grid infrastructure. There is no any installation for end user site. The end user site only consists of a compatible Web browser so it is lightweight and can operate across firewalls.

The NeuroGrid Services Pool consists of a collection of Web services including Application Services, Data Management Service, and Job Steering Service. Application Services are the applications with a standard Web service interface independent of the implementation languages and platforms. The application provider has the flexibility to move services to different machines or to move a service to an external Compute Resource provider. The same application services can support different client types. These applications—a pool of Web services—are the fundamental components of this architecture. Since Web services are platform- and language-independent, a Web service technology is also used to implement support services among applications, portals and Compute Resource. Data Management Service and Job Steering Service have been developed to provide the following functionalities: data transfer, data publishing and query, workflow management, job monitoring, and others.

The Middleware Environment is middleware provided by third party, such as GT3, GT4, SGE, Condor, database related software and others. This layer is a bridge to connect the NeuroGrid Services Pool to the Compute Resource. The system administrator of the Compute Resource is in charge of the installation of this layer. The Compute Resource includes the underlying computational resource and data storage resource.

4.3.2 Hardware

The computers used in NeuroGrid framework can be broken up into the following categories:

4.3.2.1 Data Server Nodes

They store the image files and associated databases. They also host the data management service to provide the functionalities to access the files and database.

4.3.2.2 Application Server Nodes

These are Application Service servers for the NeuroGrid framework, each of which hosts a number of Application Services. In none-Fork cases, the servers interact with Grid middleware and submit application jobs to Computation Nodes. In Fork cases, the Application Server Nodes are also the Computation Nodes, which means computation is running on Application Server Nodes.

4.3.2.3 Computation Nodes

These are the nodes to run the applications. The related middleware from a third party are installed to manage the jobs submitted to Computation Nodes. It is not compulsory that these Computation Nodes be dedicated to NeuroGrid framework. They can be any standard job management environment, such as Condor, SGE, and FORK, with which the Application Services can interact.

4.3.2.4 Portal Server Nodes

These nodes act as the portal servers. All direct access to the Grid will be via one of these servers. They also host the workflow services and handle all interaction with the Application Services. They may also optionally host Application Services. This means the portal and Application Services can be hosted either on physically different nodes or on same nodes.

4.3.2.5 Web Server Nodes

For some of the research teams, it is necessary to support connections from users who do not hold NeuroGrid certificates. These users will interact with separate Web servers that will act as client machines to the NeuroGrid. There should be no restrictions on the platform used to host these Web servers, aside from the ability to call through to the Java APIs.

4.3.2.6 Portal Clients

These are the workstations used to interact with the NeuroGrid framework by end users. It is important that these clients only make outward connections to avoid issues with firewalls. They will interact directly with the server nodes.

4.3.2.7 Web Clients

These are the users' machines outside of the NeuroGrid consortium. No special requirements are to be placed on these machines.

4.3.2.8 Software

Mature software with known reliability and performance characteristics needs to be used for the NeuroGrid framework. The primary consideration in developing the NeuroGrid framework is to use standard software and keep the software requirements to a minimum.

We chose Java since there are a comprehensive set of freely available tools to build Web service applications. Apache Tomcat is used as the Servlet container that is the official Reference Implementation for the Java Servlet and JavaServer Pages [50] technologies. Web services are built using the JavaTM Web Services Developer Pack (Java WSDP). The Java WSDP is an all-in-one package that contains key technologies to simplify building Web services using the Java 2 Platform [51]. The package includes a set of technologies that can be used to create and deploy secure, reliable, transactional, interoperable Web services and clients [52].

4.3.3 Roles of Framework

NeuroGrid framework brings together five distinct roles: service consumer, service provider, application provider, service broker, and computing resource provider. Each participant can play multiple roles. Figure 4.3 illustrates the relationship among these five roles.

Service providers host the Application Services. They perform the Grid-enabling process and are in charge of the deployment, hosting, maintaining of the Application Services. They also need to install the application if needed. They are the service developer with deep knowledge about Web services. They are not developer of legacy applications, so they usually do not have knowledge of legacy applications Grid enabled.

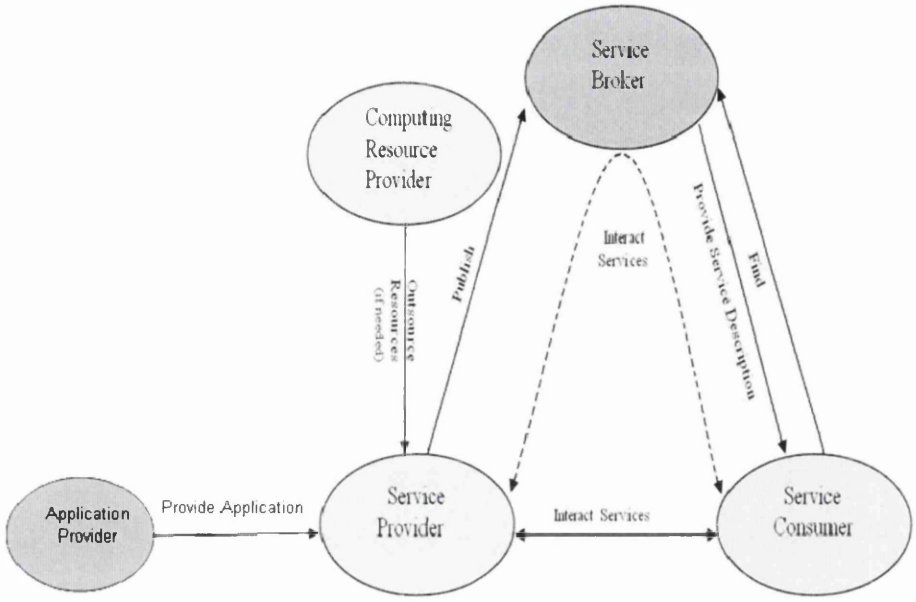

Figure 4.3: Roles Involved in NeuroGrid Framework

Application providers are application specialists who provide the legacy applications or the information on where to get the legacy applications. They perform the help role in Grid-enabling process via providing an Application Description File for each application. They are expected to be very knowledgeable about the application to be Grid-enabled. However, they are not expected to have the knowledge about Web service, JAVA, XML and other Application Service related technologies.

Service brokers are responsible for providing the portals, orchestrating the Application Services, and managing the Grid resources. They have an in-depth knowledge of the Grid computing middleware and portal system. They are not anticipated to have in-depth knowledge of the applications of the Web portal provides.

Computing Resource providers are system administrators responsible for providing computing resource to scientific applications. It is not compulsory for them to be dedicated to NeuroGrid framework. They manage the standard job management environment, such as Condor, and SGE, with which the Application Services can interact.

Service consumers are clients (humans or other services) who use the Application Services. Service consumers are expected to have a working knowledge of how to use

the application invoked. They need to prepare for the input data and set up the related parameters for the application. They do not need to install the applications. Also, it is expected that most end users are not IT experts. They are not expected to know the underlying Grid/Web related middleware.

4.4 Implementation

4.4.1 Abstract Application Service (AAS)

4.4.1.1 Introduction and Contribution

This section describes a scalable Abstract Application Service (AAS) mechanism to Grid-enable legacy scientific applications on Grids. In the context of this section, 'Grid-enabling' means turning an existing application, installed on a Grid resource, into a service and generating the application-independent user interface to use that application through a Web portal. The focus of this mechanism is to create the specific application service on demand in the event it is not kept persistent or is unavailable during the execution of a scientific workflow. The unique contribution of this work is the design and implementation of this mechanism, which we term 'Abstract Application Service (AAS)'. AAS can create specific Application Service instance on demand in a way that is completely transparent to the user and provides a high availability of Application Services without actually requiring them to be persistent. The novel aspect of the mechanism is that AAS creates Application Service by configuring itself on the fly to become a particular Application Service in need, not by instantiating the Application Service. This is achieved by the dynamic combination of the common abstraction for legacy applications and application description using specially designed Command-line Description Language (CoLDeL). This combination allows AAS to dynamically configure itself to a particular Application Service just in time. An AAS may have several concrete instances running at the same time on the grid, and each concrete service instance may have a different legacy application associated with it.

AAS can create the specific application service on demand. For example, Registration workflow invokes its constituent application services in the order specified in the workflow control file, with the data provided by the scientist. Let us assume that during the execution of the workflow, the workflow finds the AREG service is not available. Instead of stopping the workflow execution, the workflow sends a message to

AAS to check if AREG is included in the AAS. If so, it will invoke the AAS with the application name and version specified. AAS dynamically configures itself to become an AREG service, which is a Web service interface to the AREG application. Thus, in this example, even though the AREG service is not available during the execution of the workflow, we are able to create it just in time using AAS, invoke it and continue executing the workflow.

4.4.1.2 Scalability

ASToolkit and most of the reviewed existing wrappers are static compared with Abstract Application Service. In the context of ASToolkit, we need to wrap the legacy applications offline and host them as persistent services so they can be accessed from scientific workflows whenever needed. To this approach, scalability becomes one of the biggest challenges for large scientific collaborations. Sometimes a service provider needs to wrap and host hundreds of Application Services just for one scientific package, because it is not uncommon that many of scientific packages have hundreds of applications. Also when a new application is added into the framework, service providers need to develop and deploy a new Application Service. To solve this scalability problem, we adopt AAS as a generic mechanism to optimize the whole application execution at run time.

AAS is a generic application service. Scalability of this AAS approach is achieved by delivering the applications through a dynamically reconfigurable AAS. A legacy application can be launched by invoking the AAS, which is driven by a pool of Application Description Files. This applications pool contains many applications. Disregarding how many are installed and deployed, and what input data they expect, they all are described using the same language (CoLDeL), having the same service interface, and being controlled by the same methods.

This mechanism obviates the need to keep all the available applications wrapped as persistent Application Services. This generic service highly simplifies Application Service development because it can wrap any command-line oriented executable with a minimal effort. The application provider only needs to write the Application Description File for the application and register it with the Abstract Application Service.

4.4.1.3 Registry Service

For each application, we have a configuration document that contains the static information related to the application. The Application Description File is based on Command-line Description Language (CoLDeL). It is a XML-based language for describing individual applications precisely. Conforming to the XML schema, CoLDeL can specify each application with a rich semantic description and provide as much useful information to the service, workflow and user as possible. CoLDeL is described in detail in section 3.5.4. The Application Description File is written by the Application Service provider and registered with the registry service. To register with the registry service, the application provider first uploads the Application Description File to the portal. The portal then pushes the Application Description File to Application Description Pool. The registry service then updates central registered XML file containing all the information of registered applications. After being registered with a well-known registry service, the application can be discovered by the portal or a workflow.

The registry service holds an Application Descriptions Pool which contains all Application Description Files for all available registered applications. Once given the dynamic application name and version information from AAS, the registry service will get the right Application Description File from pool and return the AppDescription object to AAS.

The available applications information is central registered in an XML file. The file is also available at the workflow side. So, in case the registry service is down, or workflow ensures the information it holds is up to date, workflow also can get available applications information locally instead of obtaining from the registry service remotely.

4.4.1.4 Some Details of AAS

This common interface is same as a particular Application Service. Further detail is introduced in section 3.5.1.

The client, for example workflow, provides AAS with the application information (name, version, etc...) at the time of launching the service. This information is contained in the SOPA message. Essentially this incoming SOPA message sent to the service can be viewed as an abstract invocation of a particular application containing specific parameter values supplied by the end user, and application information supplied

by workflow. Upon receiving this message, AAS can be configured with specific components that will take the message along with the Application Description File and translate this abstract invocation into a concrete invocation. This means, AAS configures itself to a particular Application Service the client needs and specifies.

AAS uses both the static and dynamic information about an application to launch the application required. The dynamic information is provided by workflow or other clients when it contacts AAS, including the application name, version, and job related parameters. The static information is provided by the Application Description File which AAS retrieves from Registry Service. Upon retrieving the Application Description File, AAS can configure itself to become that Application Service instance. The Application Service instance combines the dynamic and static information to launch the right application with the right configuration.

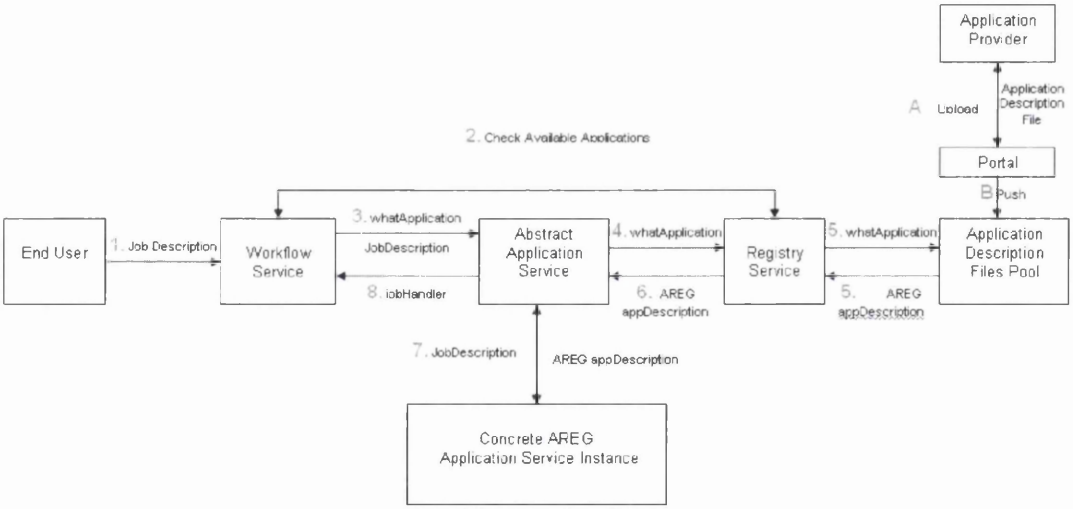

Figure 4.4: Dynamic creation of Application Service Using AAS

Figure 4.4 describes how AAS configures itself to an AREG service just in time. The numbers in Figure 4.4 illustrate the detailed flow of service interactions among workflow service, AAS and the registry service. Letters A and B in Figure 4.4 illustrate how the application provider registers the Application Description File (ADF) with the registry service. In step A, the application provider uploads the AREG ADF to portal via the portal interface. In step B, the portal pushes the AREG ADF to the registry service, which then registers the AREG ADF with ADF Pool.

Step 1. The end user intends to run AREG service and passes the jobDescription object to workflow service, which contains the required service name, service version and other job related parameters.

Step 2. Workflow service finds out that there is no persistent ARG service available. Then, it aims to use AAS as an AREG service. Workflow service generates whatApplication object based on the need, and the jobDescription object based on the user's requirement. Workflow queries a well-known registry service to obtain the available applications.

Step 3. If the application required is available, the workflow service then sends a SOAP message to AAS with whatApplication and jobDescription have provided.

Step 4. After receiving the message, AAS verifies its authenticity and ensures the user is authorized. AAS passes the whatApplication to Registry Service, and asks for needed AppDescription object.

Step 5. The registry service gets the AppDescriptoin object from the Application Description Pool.

Step 6. The registry service returns the AppDescriptoin object to AAS.

Step 7. Based on the information of whatApplication, AAS configures itself to become the particular AREG Application Service. With combined information from AREG AppDescription and jobDescription, AAS launches a concrete AREG job.

Step 8. AAS returns a jobHandler to workflow service.

The implementation of AAS essentially depends on the resources available within the Grid. In NeuroGrid framework, we have implemented two plug-in components. One is FORK, which will translate an incoming request into a simple command-line and fork this on the host server. The other is Condor, which takes the incoming request and contacts with Condor submission node to submit job to the Condor pool. Because of the heterogeneity of grid resources, it may be beneficial to envisage numerous other components, for example, mapping the incoming request to a Resource Specification Language (RSL) [53] fragment and submitting to a Globus GRAM gatekeeper, or

interacting with a high performance parallel cluster via Sun Grid Engine or other schedulers. The implementation of new plug-in components is straightforward because the system is constructed in a modular way based on components design.

4.4.2 Group Applications Service Optimization

In the following section, we explore a Group Applications Service (GAS) strategy to further optimize the execution time of a workflow. GAS merges several sequential Application Services into a single service. It reduces the grid overhead induced by the Web service invocation, scheduling, and data transfers.

4.4.2.1 What is Group Applications Service?

'Services grouping', in the service oriented workflow is usually not as straightforward as in the task oriented workflow, for two reasons. First, the services included in the workflow are totally independent from each other. Every service can use a different data transfer and job submission approach. Second, the Application Services and the Grid infrastructure executing the jobs do not have any knowledge regarding the workflow and the job dependencies. To tackle the problem, we propose a GAS solution to group sequential Application Services included in the workflow, thus allowing more elaborated optimization strategies. In this approach, we merge several sequential Application Services into a single service, which we term Group Applications Service.

GAS not only fulfils the tasks of a set of Application Services, but also takes over the role of workflow service to assemble the applications and orchestrate the data transfer and input/output. The contribution of this approach is to move the jobs assembling and orchestrating tasks from workflow level to Application Service level. Actually GAS acts in dual roles as both Applications Service and workflow. This provides an opportunity for GAS to optimize and offer the most efficient performance based on its knowledge of not only applications, but also the workflow and job dependencies.

On the Grid, large-volume data transfer across sites is common, resulting in large execution time penalties to many Grid applications. Such data transfers can drastically affect data-intensive application performance. In NeuroGrid framework, for each atomic Application Service, the input files of the application need to be staged in from a WebDav data sever to the application server side. Additionally, output files need to be sent back to the WebDav data server.

Usually these Application Services are completely independent. Consequently, in a sequential workflow, for chaining two independent Application Services A0 and A1, output data of A0 first needs to be returned to the data server before being sent back as an input to A1. If a workflow contains several atomic Application Services, each service will go through this data transfer procedure. To maximize efficiency of workflow, the time spent in files transfer should be minimized. GAS can reduce the execution time of a workflow by avoiding unnecessary files transfer. A set of applications are put in one group and are released as a GAS. This GAS is used instead of invoking many atomic Application Services involved in the workflow. It reduces the Grid overhead induced by the Web service invocation and lessens data transfers times where it may also reduce the parallelism. GAS is suitable for use when related sequential Application Services involved in the workflow and are all provided at the same site.

4.4.2.2 Registration Workflow vs. Registration GAS

Here we use registration as a sample to compare the difference between registration workflow and registration group service. Consider the registration workflow made of three services and represented on top of Figure 4.5. There are four components involved in this workflow—workflow service and three Application Services, BET, AREG and TRANSF. These Application Services are separate services and are invoked independently and sequentially in the order of BET, AREG, and TRANSF. Data transfers are handled by each Application Service, which takes the input from remote WebDav server and uploads the output to the WebDav server. The output of one service will serve as the input of next service. Here the output of BET is used as the input of AREG; the output of AREG is the input of TRANSF. Service invocation and the input/output connection between the services are handled at the workflow service level.

On the bottom of figure 4.5, BET, AREG, and TRANSF are grouped in a single Registration Group Applications Service. By interacting with this GAS instead of three independent Application Services, the overhead associated with each Application Service invocation is dramatically decreased. Also this Registration GAS has the knowledge regarding the job dependencies and input/output connection from workflow level, so it is capable of invoking the three applications sequentially and orchestrates the input/output among these applications. Instead of taking the input from WebDav server only as Application Service does, GAS can directly take the output of the previous

service located at the Application server site. Thus, GAS efficiently reduces the data transfer across the grid.

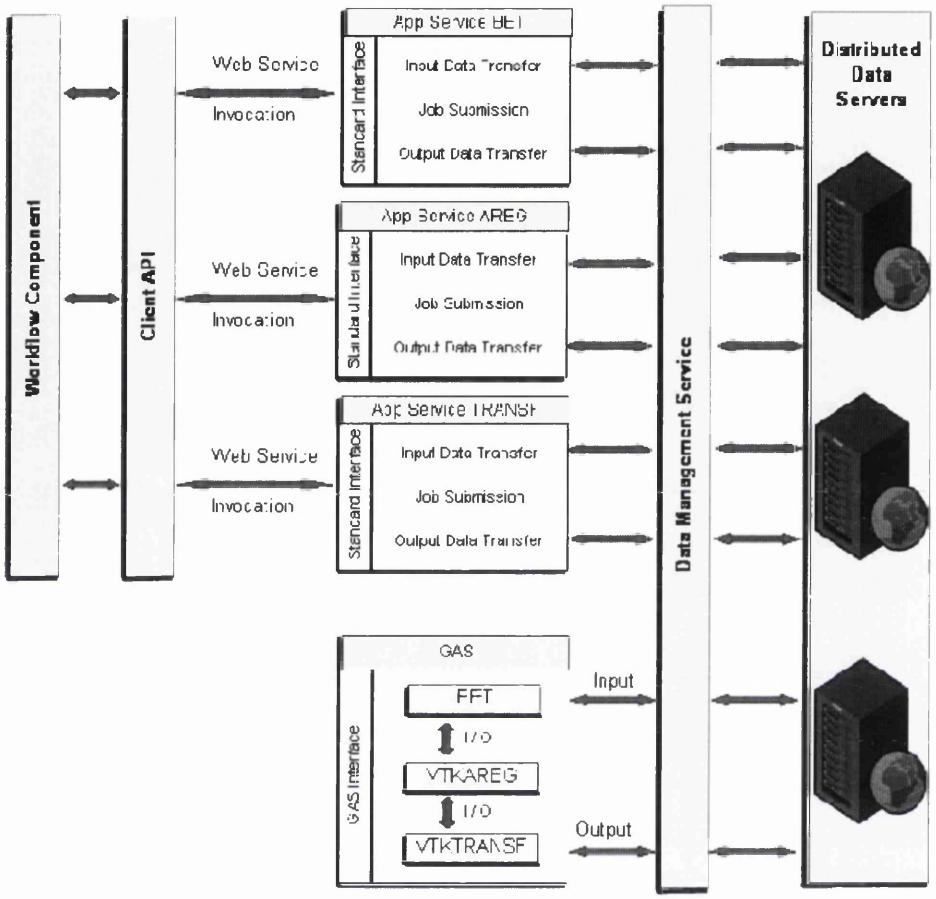

Figure 4.5: Registration Workflow, vs. Group Service.

4.4.2.3 Implementation: Group Description File

In Figure 4.6, we represent the overall architecture of GAS and some application scenarios. GAS is a generic approach which can group any applications. For any newly-added group, all a service provider needs to do is to provide the related Application Description Files and a Workflow Description File.

Now, let us suppose we are going to optimize registration workflow using the grouping strategy we explained in the last sections. Registration workflow consists of three legacy applications, BET, AREG and TRANSF. It is a sequential workflow as exhibited in the Figure 4.15.

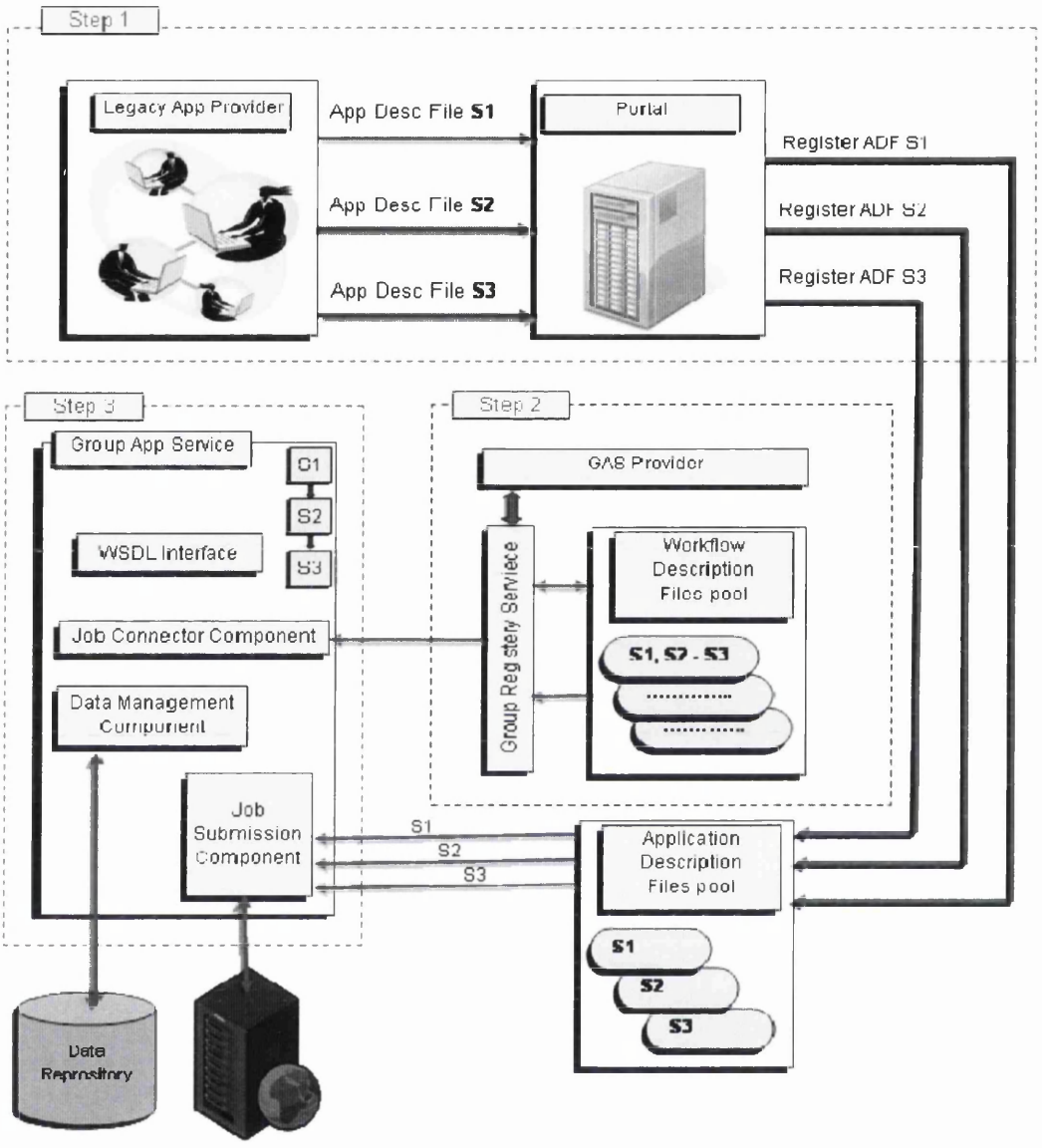

Figure 4.6: Group Application Service Architecture

Based on our grouping rule, we can group these three applications as a Registration Group Service.

In step 1, the legacy application provider uploads an Application Description File (ADF) for each legacy code to a portal. The portal then pushes the ADF to the ADF pool. In this case, they are bet_0.1.xml, areg_2.0.xml and transf_2.0.xml. These are Application Description Files that are used in the individual Application Service. Further details about ADF can be found in section 3.5.5.

In step 2, the GAS provider provides a Workflow Description File (WDF) which is the same as the one used in the workflow engine. The greater detail about WDF is introduced in section 4.4.3. This WDF is registered with a well known Group Registry Service so it can be discovered by the portal or a workflow.

In step 3, GAS will use Jobs Connector Component to mange the connection between applications (intermediate files) based on the WDF, and also delegate the atomic jobs to Job Submission Component sequentially.

Jobs Connector Component takes a role of sequential workflow engine and orchestrates the data between the services. It contacts with Data Management Component to get the input data from the WebDav repository for the first application, Registration, which is termed head application. Then GAS delegates the atomic job submission to Job Submission Component. When the head service finishes, JCC will take the output of head application and use it as the input of next application, AREG, which we term chain application. Then JCC delegates the atomic job submission to Job Submission Component again for the second atomic job. When all the chain applications jobs are finished, JCC invokes the last application, TRANSF, termed tail application, via JSC. Finally, JCC interacts with the Data Management Component to upload the output to the data server.

Essentially GAS combines the functionality of Application Service and Workflow Service together. It reuses most components used by Application Service and Workflow Service. It utilises Job Submission Component when managing each atomic job submission. It also makes use of Jobs Connector Component (JCC), a layer to orchestrate the applications like a sequential workflow. GAS is accessible through the same interface as sequential workflow. This is designed deliberately to allow clients to treat GAS as workflow without knowing the backend optimization.

The group job submitted by GAS contains several application sub-jobs. At the return of the GAS job submission, the client gets several sub-job handlers back. These sub-jobs are the same as atomic jobs submitted via Application Service. Clients can monitor and steer the sub-jobs individually in the same way they do on the atomic jobs from Application Service.

4.4.2.4 Performance

GAS can lead to different level of speed up depending on the size of the data. In data-intensive application cases, a huge amount of data transfer is needed across sites that introduce high overheads. To precisely quantify how group strategy influences the application performances, we model the workflow execution time and GAS execution time for different configurations.

4.4.3 Application Services in Scientific Workflow

We have described how to wrap a single application as a Web service. In this section we describe how to create workflows from the services using our Sequential WorkFlow Component (SWFC). SWFC provides an easy-to-use tool that allows a service provider to connect Application Services together to form workflows. End users can execute the workflows on the Grid instead of executing the Application Services one-by-one. Users can submit batch jobs and monitor the progress of a workflow.

4.4.3.1 Need of Assembling Applications

Usually we need to assemble several applications together to solve big scientific problems. It is very common to build applications by assembling legacy codes for processing and analyzing data. This assembling allows code reusability without introducing a redesign and redevelopment task to the application developers. In imaging studies, researchers tend to combine one or more of the image processing algorithms to form sequential image processing pipelines [54]. An applications pipeline is essentially the composition of one or more algorithms that exhibit a linear flow of data between stages. For example, in tracking the progress of a brain tumour, researchers may first segment the area of interest. Then, this segmentation would be used as input to a series of rigid registrations that produce a set of transformations which could then be passed to transformation algorithm.

At present, the following problems exist when forming this pipeline: first, algorithms may be provided by a variety of providers, requiring users to first locate and then install the software they are interested in; second, composition is tightly coupled to specific processing applications. For example, it can be cumbersome to use another registration algorithm replacing the one used in the pipeline. Also, there is no agreement among the various algorithms developers. It is possible an incompatibility occurs, for example, due

to differing image formats, at some stage in the pipeline. Frequently users have to mediate among the algorithm-specific vocabulary and perform a kind of data conversion.

Scientific workflow is needed to facilitate this applications assembling procedure. Workflow describes the behaviour of complex applications and their composition. The logic of such composed applications is described through a set of computation tasks to perform and data dependencies imposing constraints on the order of processing. In NeuroGrid framework, the ability to compose Application Services constitutes an essential requirement.

4.4.3.2 Task Based and Service Based Scientific Workflows

Many workflows have been proposed with very different approaches to compose a set of computation tasks. As discussed in section 2.5, two main strategies have been proposed and implemented in Grid middleware, batch oriented strategy and service oriented strategy. Current available workflow managers can be categorized into two groups—task based and service based.

In the task based workflow, workflow composes the batch oriented applications. Each computation job includes the information of not only the processing (executable and command-line parameters) but also the data (static declaration). Workflow processors directly represent the computing tasks. The user needs to provide the application executables and the precise command-line parameters. All the related data to be processed are statically described in the workflow. Condor Directed Acyclic Graph Manager (DAGMan) is a task based workflow manager [55]. DAGMan is a meta-scheduler for Condor. It manages dependencies between jobs at a higher level than the Condor Scheduler.

In the service based workflow, workflow composes the service-oriented applications. Services are naturally well-suited for representing and chaining workflow components. The service oriented approach has been implemented in different workflow managers, including the Kepler, Taverna and Triana. Kepler is an extensible open source scientific workflow system that provides scientists with a graphical user interface to register and discover resources and to interactively design and execute scientific workflows using emerging Web and Grid-based technologies to distributed computations. Kepler can orchestrate standard Web services linked with both data and control dependencies [9].

Taverna is an open source workflow tool which provides a workflow language and graphical interface to facilitate the easy building, running, and editing of workflows over distributed computer resources. Tavena targets bioinformatics applications and is able to enact Web service and other components like SoapLab services and Biomoby [56] services. It implements high-level tools for the workflow description, Feta semantic discovery engine [7]. Triana is decentralized and distributes several control units over different computing resources. Triana implements both parallel and peer-to-peer distribution policies [57, 58].

4.4.3.3 Sequential Workflow Component

Although existing workflow systems are able to support complex computations and data repositories in a distributed environment, and some support Web service, they do not meet security requirements from the NeuroGrid Application Service. Also in the NeuroGrid community, only sequential image processing pipelines are needed to combine one or more of the image processing algorithms.

To meet NeuroGrid framework requirements, we develop a practical but powerful Sequential WorkFlow Component (SWFC) to orchestrate the applications. This is a lightweight workflow engine that allows users to run and monitor the sequential applications pipelines.

In the Sequential WorkFlow Component, the appropriate structure of a processing pipeline is defined. The pipeline exhibits the following characteristics:

i) Each pipeline is composed of one or more distinct sequential processing stages.

ii) In each stage only one application is executed over a set of data sets. A stage therefore comprises of one job.

iii) An ordering representing the dependencies between each stage is imposed on the pipeline.

iv) Input of each stage can be from users or refer to outputs produced at a previous stage of the pipeline, thus providing a means of data flow between stages and achieving transparent data flow.

Typically a workflow comprises a sequence of distinct processing stages, each corresponding to and realized by an Application Service. Each stage can reference data

produced at the preceding stage, thus allowing transparent (potentially distributed) data flow between workflow stages. The logic of such a composed application is described through a set of computation tasks to perform and data dependencies imposing constraints on the order of processing. These image processing pipelines are defined once by the researcher and provided to SWFC.

The SWFC is light and is not really in charge of any data management and application execution. It does not deal with direct application execution and resources, but invokes the Application Services instead. SWFC is actually an I/O connector among applications. It acts as a bridge between the user and Application Services. From WDF, SWFC gets whatever services to invoke and related I/O information. It generates the job description for each job on the fly, the right form that Application Services required, and then it invokes the specified Application Services sequentially. All the data movement is done in Application Service via interacting with Data Management Component. This workflow allows users to submit batch sequential pipeline jobs in one go, further monitor the jobs, and restore the jobs when the jobs are failed.

4.4.3.4 Benefits of Workflow

Below are the benefits of employing a workflow:

i) Enables executions of multiple applications in one go.

ii) Simplifies re-execution by reusing existing pre-defined workflows.

iii) Abstracts away the detail behind each application.

In NeuroGrid framework, the chief benefit to the end user is that the common workflows are pre-defined and offered by portal, so the users can quite easily instantiate complicated image processing pipelines by simply picking a workflow off the peg and passing it the appropriate images, datasets and parameters.

4.4.3.5 Data Type Match and Conversion

Typically a workflow will be composed of a number of distinct processing stages. Each stage is related to and realized by an Application Service. Each stage can reference data produced at the preceding stage, thus allowing transparent (potentially distributed) data flow between workflow stages. It is essential this data flow occurring between workflow stages is compatible in terms of the type of data produced and consumed at each stage. By checking the related Application Description File, Sequential WorkFlow

Component can determine the compatibility between stages. Where a type mismatch occurs the Workflow Component has two options: search for a conversion service capable of casting between the two types and transparently insert this service into the workflow, or failing this, reject the workflow and inform the user of the nature of the incompatibility.

4.4.3.6 Workflow Description File

The workflow-needed information is expressed in an XML document that allows the structure of the required processing pipeline to be defined. This XML document, which is termed Workflow Description File (WDF), is a collection of information that, taken together, provide precise definitions of scientific processors used to process raw and derived datasets. The following basic information is included, which is consistent with the definition of the pipeline.

i) Processes definition that offers a brief description of all processes included in the processing pipeline. WDF works with Application Description Files together, which contain a complex and detailed description of each process.

ii) Dataflow description of each scientific process.

iii) Data derivation description that provides a precise description of how a dataset was derived.

Based on the information above provided by WDF, SWFC orchestrates the Application Services and the required datasets. SWFC executes the workflow by invoking its constituent Application Services in the order specified in the WDF with the data specified in WDF.

For every workflow, there is one WDF, which contains information necessary to construct the process pipeline and orchestrate the applications. A WDF contains links and references to specific services and all related input and output files. There are three types of services involved in a workflow -- head service, chain service, and tail service. For the head service, all the input files information is from the user. The input and output files for the chain service and tail service are concept file names. To enable remote retrieval of these files by the Application Service, these concept file names in WDF will be substituted by a real physical file identifier (URI or local file with path) on the fly after the workflow is initialized.

The WDF has four main elements: wfDescription, wfName, wfid, and services. The services element describes the services invoked. Each service element contains the name of the service, a short description of the service, serviceId, order, input files and output files. In our example of a Registration workflow, the WDF has the top-level structure shown below.

```
- <java version="1.5.0_10" class="java.beans.XMLDecoder">
  - <object class="uk.ac.neurogrid.workflow.core.wf.Wf">
    - <void property="wfDescription">
        <string>Registration. BET-VTKAREG-VTKTRANSFORMATION</string>
      </void>
    - <void property="wfName">
        <string>Registration</string>
      </void>
    - <void property="wfid">
        <int>1</int>
      </void>
    - <void property="services">
      - <array class="uk.ac.neurogrid.workflow.core.wf.Service" length="3">
        - <void index="0">
          - <object class="uk.ac.neurogrid.workflow.core.wf.Service">
            - <void property="servicename">
                <string>bet_v_1_0</string>
              </void>
            - <void property="serviceid">
                <int>1</int>
              </void>
            - <void property="order">
                <int>1</int>
              </void>
            - <void property="inputfiles">
              - <array class="uk.ac.neurogrid.workflow.core.wf.IoAtomic" length="1">
                - <void index="0">
                  - <object class="uk.ac.neurogrid.workflow.core.wf.IoAtomic">
                    - <void property="filename">
                        <string>FROMUSER</string>
                      </void>
                    - <void property="portid">
                        <int>2</int>
                      </void>
                    - <void property="hardcontrol">
                        <int>0</int>
                      </void>
                    </object>
                  </void>
                </array>
```

Figure 4.7: Registration Workflow Description File-Part1

```
- <void property="outputfiles">
  - <array class="uk.ac.neurogrid.workflow.core.wf.IoAtomic" length="1">
    - <void index="0">
      - <object class="uk.ac.neurogrid.workflow.core.wf.IoAtomic">
        - <void property="filename">
            <string>regout_bet.nii</string>
          </void>
        - <void property="portid">
            <int>3</int>
          </void>
        - <void property="hardcontrol">
            <int>1</int>
          </void>
        </object>
      </void>
    </array>
  </void>
 </object>
</void>
- <void index="1">
 - <object class="uk.ac.neurogrid.workflow.core.wf.Service">
  - <void property="order">
      <int>2</int>
    </void>
  - <void property="serviceId">
      <int>2</int>
    </void>
  - <void property="servicename">
      <string>vtkareg_v_2_0_0</string>
    </void>
    <!-- Not only files from previous service. Also includes files from user. -->
  - <void property="inputfiles">
    - <array class="uk.ac.neurogrid.workflow.core.wf.IoAtomic" length="4">
        <!-- target. from previous service output. -->
      - <void index="0">
        - <object class="uk.ac.neurogrid.workflow.core.wf.IoAtomic">
          - <void property="filename">
              <string>regout_bet.nii</string>
            </void>
          - <void property="portid">
  - <void property="hardcontrol">
      <int>1</int>
    </void>
  </object>
</void>
<!--                                                    source. from user. -->
- <void index="1">
 - <object class="uk.ac.neurogrid.workflow.core.wf.IoAtomic">
  - <void property="filename">
      <string>FROMUSER</string>
    </void>
  - <void property="portid">
      <int>3</int>
    </void>
  - <void property="hardcontrol">
      <int>0</int>
    </void>
  </object>
 </void>
- <void index="2">
 - <object class="uk.ac.neurogrid.workflow.core.wf.IoAtomic">
  - <void property="filename">
      <string>FROMUSER</string>
    </void>
  - <void property="portid">
      <int>4</int>
    </void>
  - <void property="hardcontrol">
      <int>0</int>
    </void>
  </object>
 </void>
```

4.8: Registration Workflow Description File-Part2

Figures 4.7 and 4.8 illustrate the partial XML representation of registration workflow. Some generic information related to the workflow is included, such as wfName, wfId, and wfDescription. For registration workflow, it contains three stages, BET, AREG, and TRANSF, which is defined inside the <services> element. Each stage is defined as a service, including information like, servicename, serviceid, order, inputfiles, and outputfiles.

In the first stage, BET, the servicename is bet_v_1_0; serviceid is 1; order is 1. The BET URL is defined to reference the appropriate BET service. This reference is usually obtained dynamically through the combination of servicename and host information. Host information can be achieved through a query on the registration service. Next, each input/output file is defined for each service to be executed. Information related to input files is placed inside the <inputfiles> element which includes the portid, file name, and the origin of the file. The portid is the same identifcation number used in Application Description File. There are two types of resources for input files; one is from a user physically located at WebDav server, and the other is from the previous stage, the outcome of the previous service. BET is the first service of registration workflow, called head service, so all the input files are from the user on the WebDav server. In a similar fashion, output files-related information is placed inside the <outputfiles> element. There is one output file in BET stage, which is named regout_bet. When writing the WDF, it is not clear where the output files will be placed, so in WDF, only files name is specified, not the real file URL. The Workflow Component will dynamically resolve this variable reference to a file URL.

Similarly, the next two stages of the workflow, namely an AREG service and a TRANSF service, are defined. Again we reference the Application Services to be used for these two stages and specify the input and output files. In the last service of the workflow (tail service), the output files are not known by the workflow until the job runs, because for some applications, the number and type of output files are not certain, based on the parameters set up from users. Therefore, the WDF lists all the possible output files for tail service. The SWFC will dynamically generate the solid output files list combining the information from the users and WDF.

The Java language has an excellent API for converting objects to and from XML. Given a stored WDF document representing a workflow, it is a fairly simple matter to

generate Java objects from their equivalent XML representations. These objects can then be used and manipulated immediately for workflow instantiation.

4.4.3.7 Limitation of WDF

This WDF concept works well for most cases. It allows variable cardinality for inputs to applications, but only for certain (fixed) input files. It has problems with some algorithms which can take variable numbers of inputs. As an example, the atlas generation algorithm (atlas) from the Insight Toolkit [59] takes a variable length array of images as input to produce a single composite output image. This array could realistically comprise as few as two images or, just as easily, more than hundred.

This uncertain range from one to many cardinality distinguishes the stage from others. Although this sounds conceptually quite simple, representing this in a workflow at the abstract level is quite difficult to achieve. This is due to the fact that keeping track of each branch of an undetermined number of input files is impossible without knowing the exact number of inputs concerned.

4.4.3.8 Generic Application Service in Workflow Component

Let us assume that during the execution of the registration workflow, SWFC finds the BET service unavailable. Instead of stopping the workflow execution, the SWFC sends a message to Generic Application Service with the whatApplication object and jobDescription object. The Generic Application Service dynamically configures itself to a solid Web service interface to the BET application. Then WFC can run the BET application and continue to execute the rest of the workflow. Thus, in the above example, even though the BET service is unavailable during the execution of the workflow, we will be able to create it just in time using Generic Application Service, invoke it and continue executing the workflow.

4.4.3.9 Interface of Workflow

Workflow Service has several operations to support job submission and monitoring. CreateJob/CreateBatchJob creates a job ID/IDs for single job/batch jobs and returns the IDs to the user. The sendJob and sendBatch operations provide an entry point for using any of the applications pipeline. The send submission operation requires one argument that is essentially a description of command-line-related parameters using Command-line Description Language (CoLDeL). Once appropriate checks have been made

(validation of inputs, type checks, values supplied for mandatory parameters, etc.), the actual Application Services are launched. How or where these Application Services are located is an implementation detail and effectively hidden from the user. Thus the client only deals with one generic workflow interface.

The workflow also has getWSInfo and getBatchWSInfo operations. If the job fails, the user can pass the job ID to workflow, get the user's parameter setup back, and then resubmit the job easily. Job ID is returned to the user after job submission. GetJobStatus, getBatchJobStatus and getWorkFlowStatus provide monitoring functionality to allow the user monitor the jobs submitted. To determine the status of a particular service invocation, the user needs to use these operations via passing the job ID. The actual business logic is provided by Steer Service. Workflow interacts with Steer Service underneath while providing a simple interface to user.

4.4.3.10 Generic Workflow Service

While the concept of a predefined sequential workflow is clearly useful, there is a problem with regard to reusability. Since each service and its input/output links must be clearly specified, a workflow designed to registration workflow could not be used for another set of applications. A dynamic workflow model was therefore needed to represent the abstract of workflow and construct a solid workflow according to the demand of the user. We use the same approach as Generic Applications Service to develop a Generic Workflow Service. Scalability is achieved by providing a common abstraction for a category of workflows and providing a "generic" workflow service to orchestrate any of registered workflow-related applications.

In this approach, we pool a set of Workflow Description Files for the predefined workflows, such as Registration Workflow, Segmentation Workflow, and N3 Workflow. The Generic Workflow Service can dynamically retrieve the required WDF and configure itself to be the demanding particular workflow service. For every new workflow service, the application developer only needs to write the WDF for the new workflow and register it with the Generic Workflow Service.

4.4.4 Portal

There are various possibilities available for hosting the services to be made available to the neuroimaging scientists. Given that user friendliness is a key aspect, a Web-based

project portal was developed. This portal provides a personalizable environment that the neuroimaging scientist is offered to explore all of the (Grid-related) software, data resources and general information associated with the NeuroGrid. The traditional open-standard-based J2EE technologies have been used to develop the NeuroGrid portal. Figure 4.9 is a screenshot of the portal.

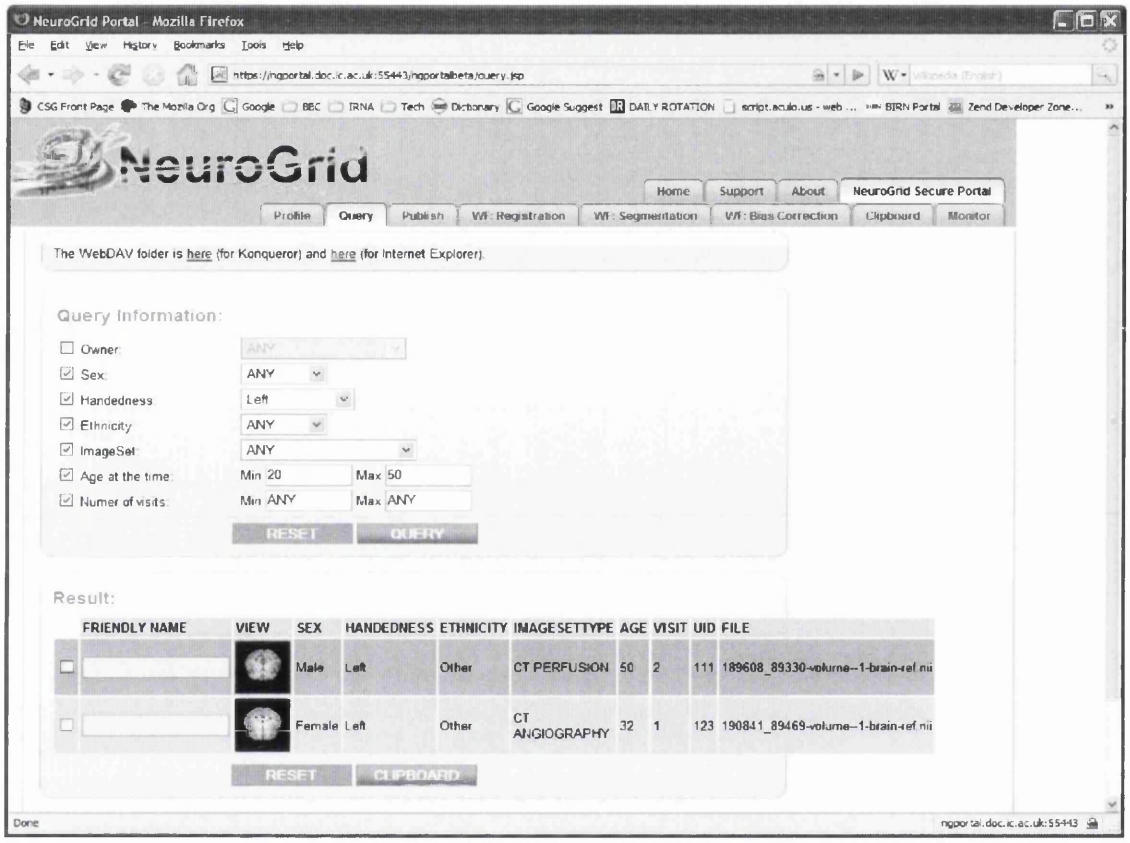

Figure 4.9: NeuroGrid Framework Portal

4.4.4.1 Security Issues

The integral part of the portal technology is security. The neuroimaging researchers have been issued (by the U.K. e-Science Certification Authority) X.509 certificates that need to be embedded into their browsers. Based on the role and research group of each NeuroGrid portal user (e.g. psychosis, dementia, etc.), the X.509 certificate is used to limit what services the NeuroGrid portal user sees and subsequently is allowed to invoke.

It is very important to make sure the database and all the data transactions over the net are secure. In order to achieve this, it was decided to use secure http and secure

WebDav for all transactions. The neuroimaging researchers have been issued (by the NeuroGrid Certification Authority) X.509 certificates that need to be embedded into their browsers. Based on the role and research group of each NeuroGrid portal user (e.g. psychosis, dementia, etc.), the X.509 certificate is used to limit what services the NeuroGrid portal user sees and subsequently is allowed to invoke. This certificate provides user identification for all other tractions between any nodes involved in the process. This approach also eliminates the need for a username and password, because users are recognised by their certificates.

There is a WebDav space for each user on a remote server (defined by the certificate) that works as data pool. All parties who need to share large amount of data, like images, have secure access to this pool. As an example, user can upload an image from a local computer into this pool and the image can be read by a web service to run an algorithm on, and the result can be put back onto the pool, and available for the user to read it back.

4.4.4.2 Functionalities

The portal consists of two parts: secure and non-secure. The non-secure part consists of static pages with only one goal: help the user to set up the secure connection. In the non-secure part, the users can see information about the portal and a step by step guide on how to obtain and install their certificate into their browser. This guide covers four browsers: Internet Explorer, Firefox, Konqueror and Safari. In this part the user learns how to get hold of a valid certificate and how to import it into the browser.

The secure part is the main section. The secure portal has following functionalities: users can query the database and publish into the database; users can run workflow jobs on their own data and the data they have queried; users can monitor the jobs; and users can see and modify their personal profile on the portal. These are the basic and fundamental functionalities that were expected to be included in the NeuroGrid portal.

4.5 User Cases

The framework has found real applications in scientific communities. Initially the workflow services are designed and used by users from many universities within the NeuroGrid project, such as Oxford, UCL, Imperial College, Edinburgh, Nottingham,

Cambridge, and Newcastle. The users of the framework have been divided into four categories based on their skill level and the flexibility they require when using complex neuroimaging applications.

Application scientists: They primarily conduct research to improve the numerical models for the neuroimaging community. Usually they just provide the applications to be wrapped by others via ASToolkit. Sometimes these users use the ASToolkit to wrap their applications as Application Services for the community. These users are the application providers. They may be also service providers if they wrap the services by themselves.

Neuroimaging scientists: They have a great deal of experience in using multiple applications. They use the framework to compose and run the workflows from Application Services and analyze their output. They can also work with service providers and contribute their workflows to community.

Neuroimaging users: They do not have vast profound experience on the Application Services, or they intend to use the distributed Compute Resource. They only run the pre-composed workflows with some parameters setup through the portal.

Neuroimaging users: They are only interested in the data and use the portal to access the distributed data server.

In the real world, there are many requirements from these different users, which are listed below:

i) Security is critical when working with confidential patient data, but users are reluctant to manage complicated security systems. It is required to add in the strict security with minor users' effort.

ii) The neuroimaging community has a large number of command-line scientific applications. A cost-effective mechanism is needed to wrap a large number of applications as Application Services without the hassles of maintaining and updating the source code.

iii) Since most application providers are application developers, they have little knowledge about Web services and related technologies. An easy-to-use toolkit is needed for application scientists to wrap their applications as

application services with minimal or no learning curve. This means the toolkit should be easily used by none Web service experts.

iv) Because the amount of scientific applications is huge, a mechanism is needed to dynamically generate the Application Service on the fly. This is to avoid hosting huge amount of Application Services.

v) The application scientists continually conduct research to improve the numerical models. They often add features and improvements to their applications from time to time. Application Service needs allow application scientists change the underlying application while keeping the same interface to the end users.

vi) Application Services will be used by many geographically-located end users. A user-friendly interface is needed for these end users to access the Application Services and data server. There should be no software installation requirement at the end users' side.

vii) The Application Services need to be scalable enough to support a few hundred concurrent end users.

viii) End users need to run the applications without having a login account on the compute resource.

The framework meets all of the above requirements via introduction of ASToolkit, Abstract Application Service, Group Applications Service Optimization and Scientific Workflow. ASToolkit offers a number of features, which are discussed in Chapter 3.

We have successfully used ASToolkit to wrap scientific applications as Web services for the neuroimaging community. Also, the toolkit can be used easily for command-oriented applications in other fields. As mentioned in Chapter 3, with the focus on simplicity and configurability, all the ASToolkit services employ a consistent interface and the same business logic. ASToolkit services can be distinguished by their unique URLs and their associated Application Definition File. This approach makes wrapping easy but stable. The service provider only needs to create one appropriate Application Description File and specify server behavior in the configuration file. The ASToolkit will build the Application Service and deploy it on the remote server.

In the real world, imaging studies tend to utilize sequential pipelines of image processing algorithms where the results of one algorithm are used as the input to a subsequent step. These pipelines are defined by the researcher and then applied to specific data sets. Scientific workflow has been used to create the sequential scientific pipelines. The workflow is expressed in a XML-based Workflow Description File. The workflow engine coordinates the execution of a series of sequential Application Services as specified in the Workflow Description File. This workflow engine allows the user to submit sequential pipeline jobs at one time, and further monitor and restore them if the job fails. It also has batch job submission functionalities. All the Application Services are pre-composed into workflows and accessible via the Web-based portal.

Below some of the scientific Application Services/workflows wrapped by the toolkit are described. Each of these workflows has been tested on sample MR images from the IXI data set (ww.ixi.org.uk) and found to operate effectively from a simple Web page interface that can be used by non-experts.

4.5.1 Brain Extraction and Segmentation Workflow

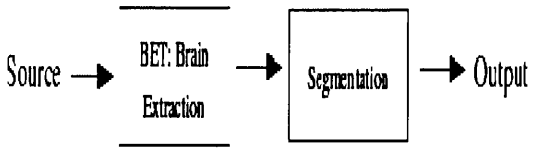

Figure 4.10: Brain Extraction - Segmentation Workflow

As shown in Figure 4.10, two algorithms wrapped as Application Services are used in this segmentation workflow. One is FMRIB BET, a Brain Extraction Tool, and the other is FMRIB FAST [60], an automated tissue classification tool. This workflow segments the brain from MR image sets, removes surrounding and peripheral tissues, then classifies voxels into different tissue groups and returns a segmented image. Figures 4.11, 4.12, 4.13, and 4.14 show the results of Segmentation workflow.

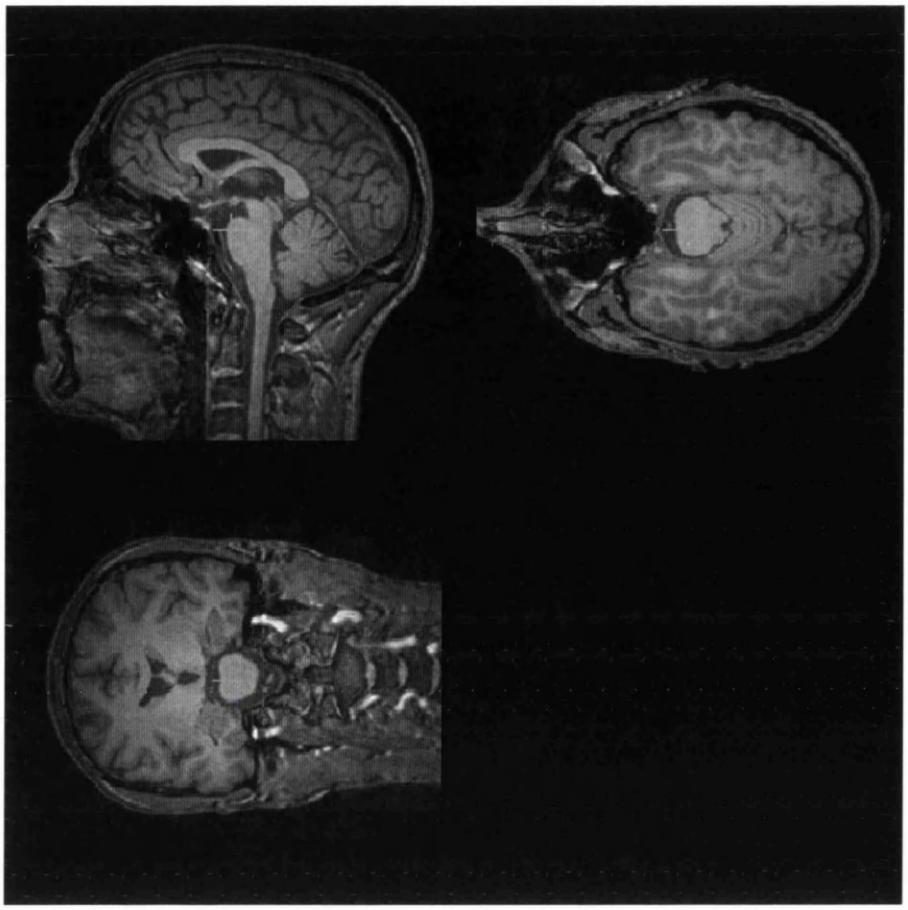

Figure4.11: Original Scanned Data

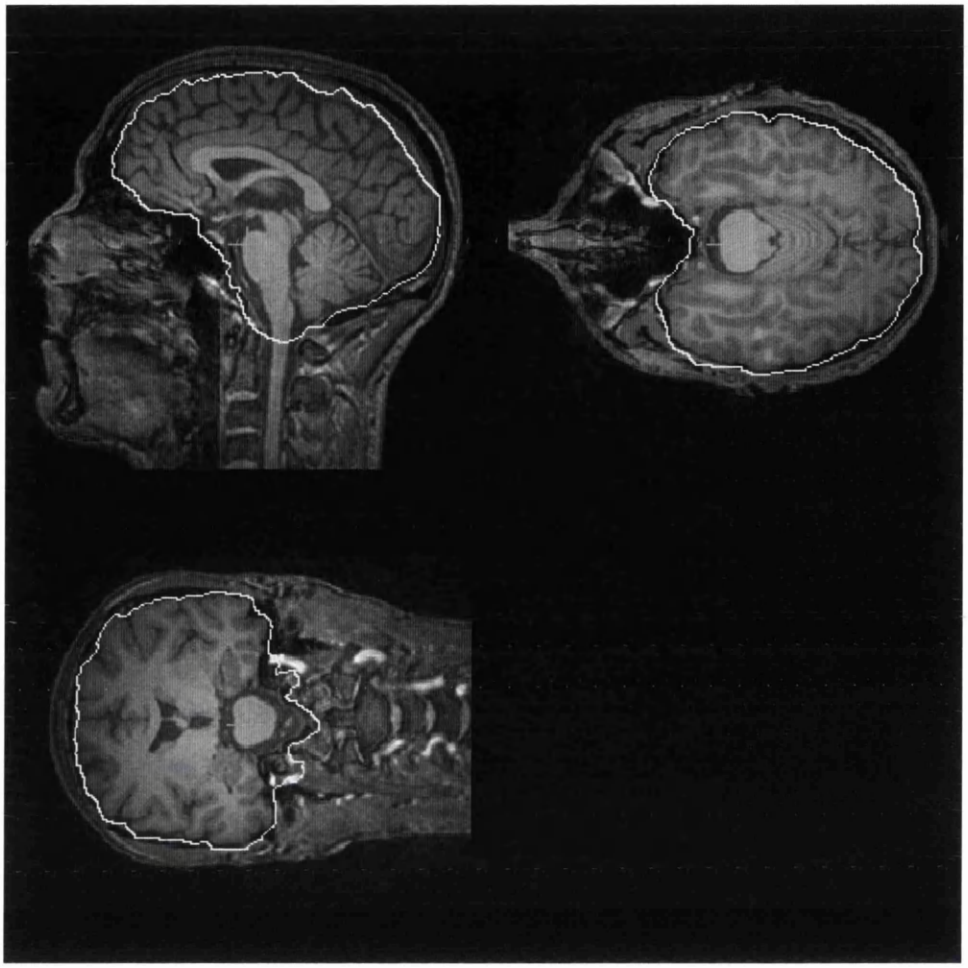

Figure 4.12: BET Results

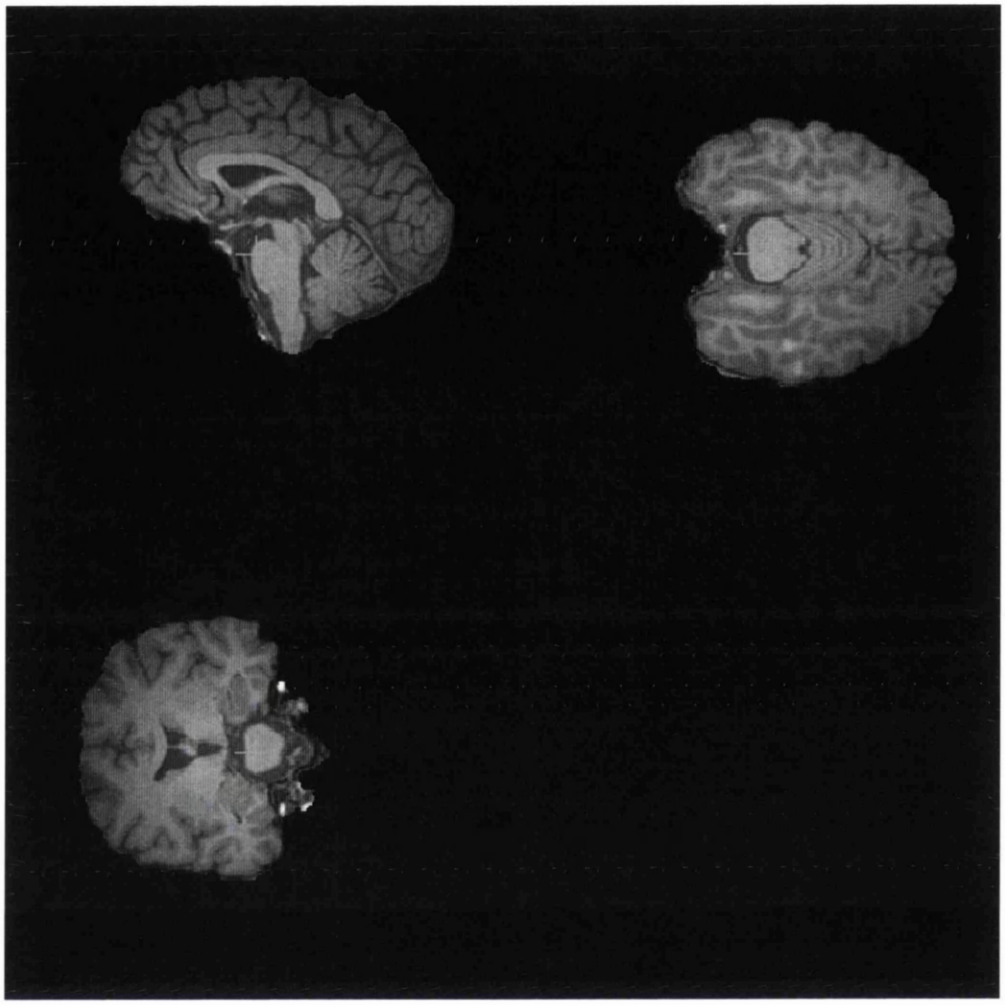

Figure 4.13: BET Results

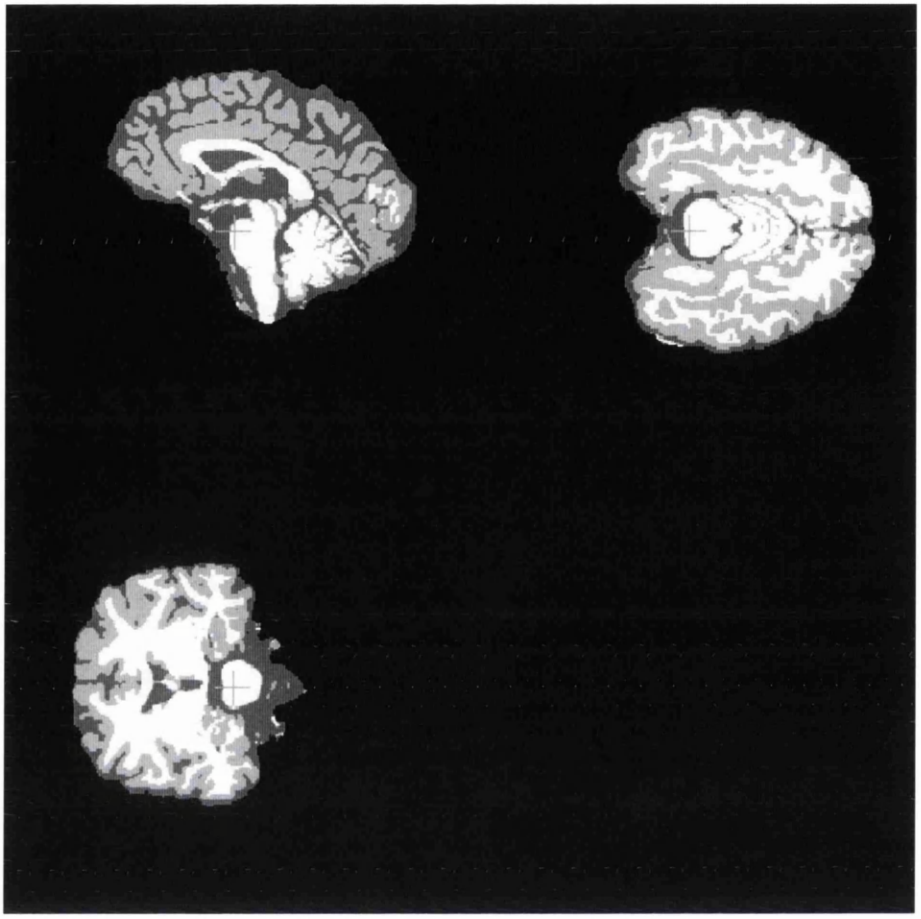

Figure 4.14: Fast Results

4.5.2 Brain Extraction, Affine Registration and Transformation Workflow

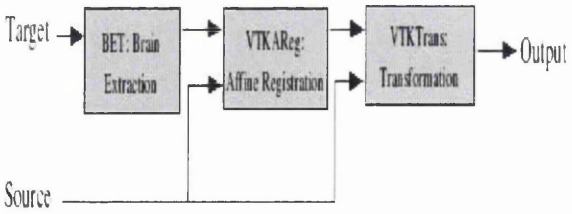

Figure 4.15: Brain Extraction - Affine Registration - Transformation Workflow

Figure 4.15 shows there are three Application Services in this registration workflow. They are brain extraction (BET); affine registration (IRTK) [61]; and transformation (IRTK) as determined by the registration process. This workflow automatically aligns a source image with a reference image. First, the train extraction is used to eliminate non-brain tissue from the reference image, so the registration is focused on matching the brain. Next, the quality of alignment is determined by a similarity measure between the

two images. The alignment uses twelve parameters affine transformation, three rotations, three translations, three scaling factors, and three skew factors. Finally, the input image is transformed according to the parameters computed by the registration step. The workflow involves three distinct Application Services associated with three different input and configuration files.

Following Figures 4.16 and 4.17 compare the brain images before and after Registration workflow.

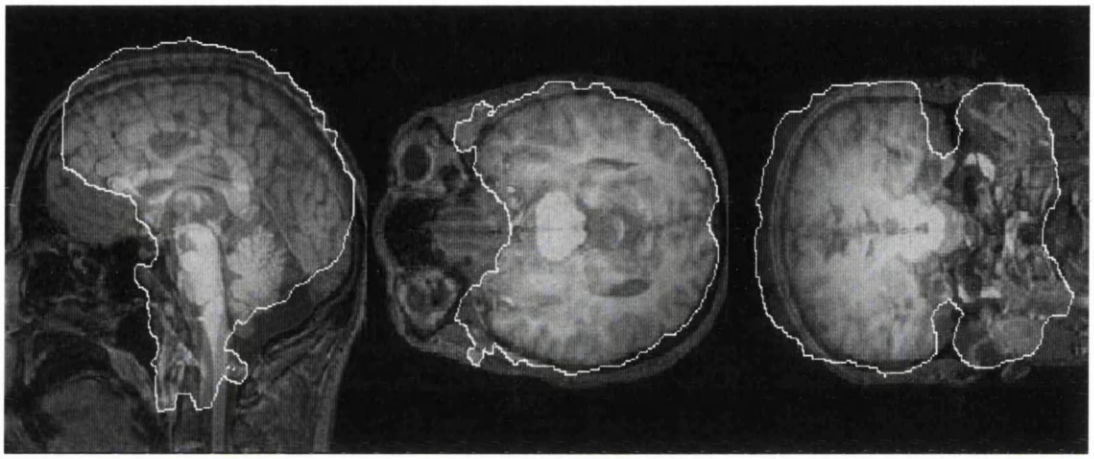

Figure 4.16: Before Registration

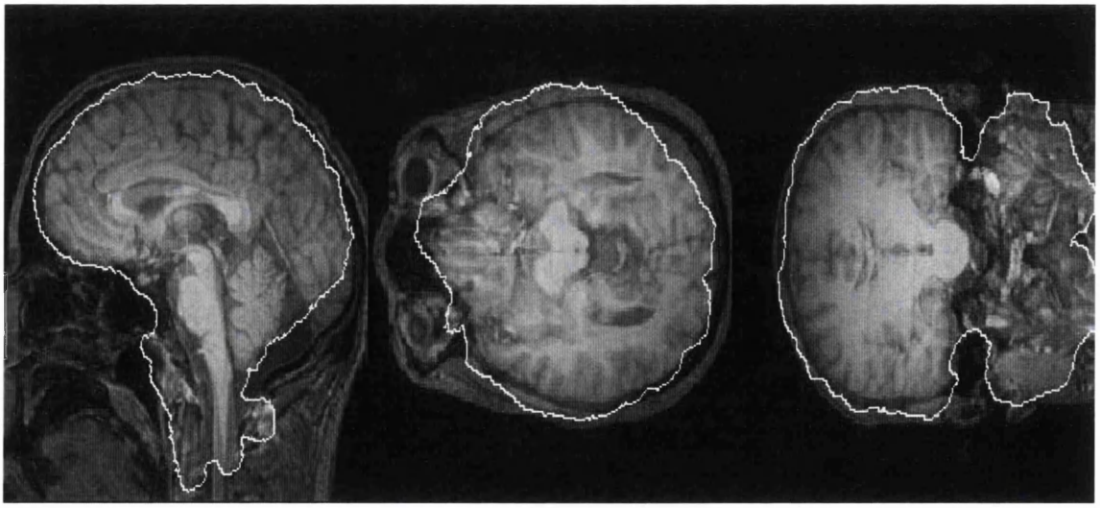

Figure 4.17: After Registration

4.5.3 Image Intensity Correction Workflow

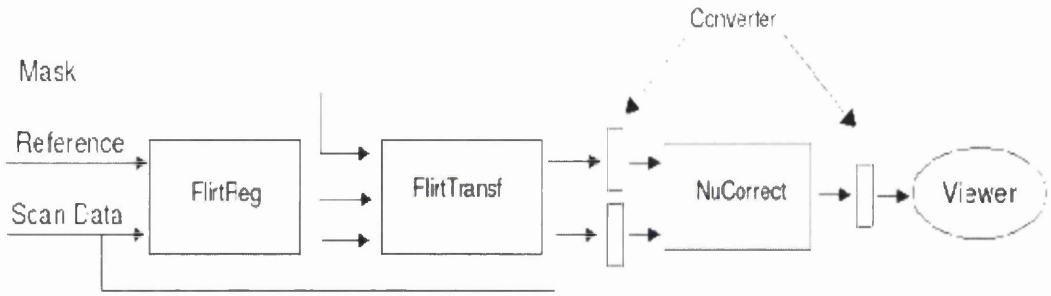

Figure 4.18: Flirt Registration - Flirt Transformation - Intensity Correction Workflow

As shown in Figure 4.18, the algorithms wrapped in this workflow are FMRIB FLIRT [62], image registration, MNI N3 [63], intensity correction, and FMRIB converter – which applies image transformation. The process is: aligns mask with target image, estimates bias-field correction using N3, applies correction, returns corrected image. This image intensity correction workflow includes six independent Application Services. The results of Intensity Correction Workflow can be shown via following Figures 4.19 and 4.20.

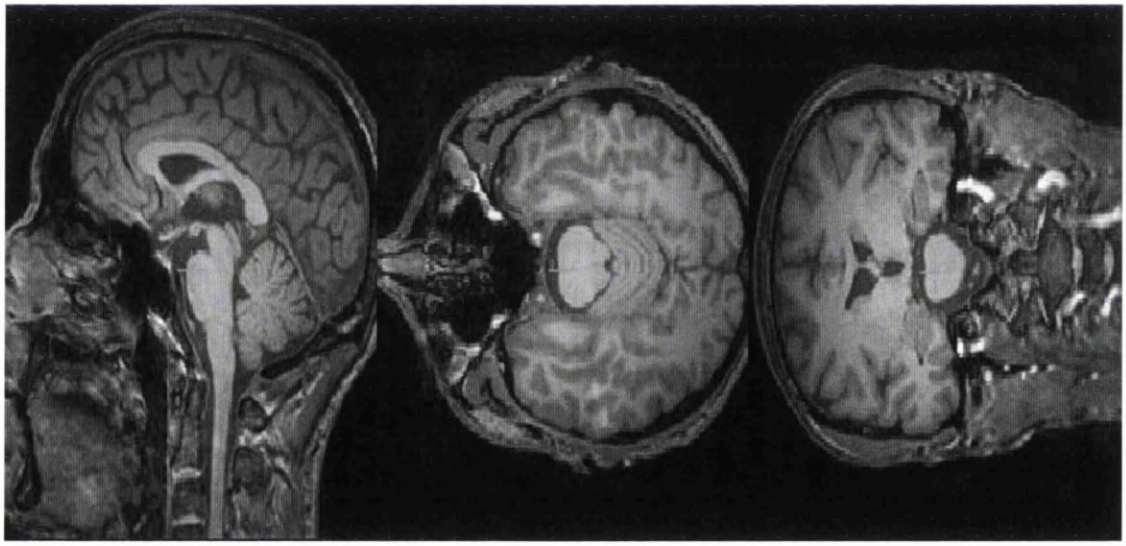

Figure 4.19: Before Intensity Correction

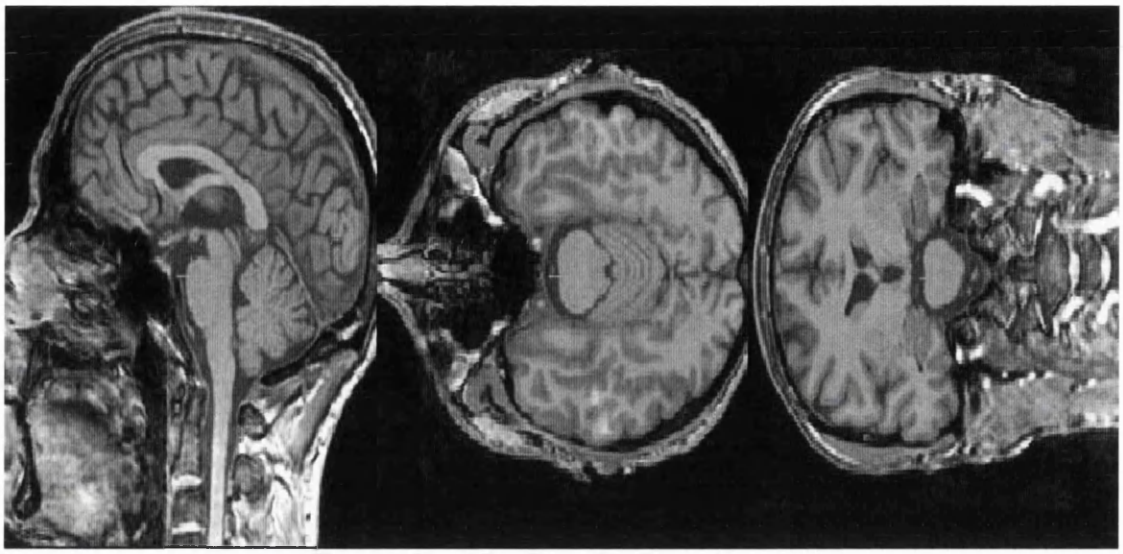

Figure 4.20: After Intensity Correction

4.6 Users Feedback

After initially used and tested within NeuroGrid project, the NeuroGrid portal is further publicised to more scientific communities, like FSL and SPM, and more institutes, like Clinical Sciences Centre, Imperial College London, Division of Clinical Neurosciences, University of Edinburgh, Department of Radiology & Biomedical Imaging, University of California, San Francisco, UCLA (University of California, Los Angeles) Health System, and so on. NeuroGrid project users have full access to the integrated database and workflow capabilities. Non-NeuroGrid users have access to the workflows, but not the stored data, via guest certificates. Users' feedback is positive. But it is hard to predict what impact such scientific portals will have on scientific communities in future.

The use of certificates is not popular amongst users. Some users have difficulties installing them. Even though help webpage is provided, users are generally more comfortable when directly shown how to proceed. Most researchers comment that they prefer a more familiar user name and password style identity verification process. Also the Web portal facilitates users in accessing the system from a variety of platforms, for example starting a job from the work location and then checking the job result later from home. This requires the users to install certificates on different platforms.

The access control security model is accepted by the users. Different exemplar datasets belong to different groups and are accessed by the users within the group only

in most cases. Although there is demand on sharing public data and application among groups, access control on data and application is an important capability.

The concept of WebDav folders is initially foreign to the users, but the portal is helpful in getting users in using this facility. Lack of universal support for the WebDav protocol in standard browsers is a minor irritation. However, associating the WebDav folder as a network place is a widely used strategy. This facilitated simple and integrated access completely consistent with local data storage solutions.

The workflows are proved popular, particularly the brain extraction tool (BET), tissue classification, data re-orientation and DICOM anonymisation tools. The system was found to be reliable however, when occasionally services fail to execute, lack of feedback is a problem. Once jobs have been submitted, the job monitor provided information about job status, but if the process fails, there was no way for a user to identify what has gone wrong. This strongly suggests that a more detailed help page and more explanatory information on errors could have been invaluable.

The available workflows have kept growing based on the requests from the users. The ASToolkit is useful to wrap algorithms as Web services rapidly. The concept of Grid enabled Application Service is very helpful to meet the computational resource demanding with the number of the users growing.

Criticisms of the portal are mostly related to latencies in response and to occasional failures in submitted tasks. These could result in frustration as with any interactive system. Error reporting is a key ingredient in maintaining confidence in the portal if a problem occurred elsewhere in the system, and this can be substantially improved with more specific error messages being provided.

CHAPTER 5

GECEM: a Problem Solving Environment Using Wrapping Approach

5.1 Introduction

Grid-Enabled Computational Electromagnetics (GECEM) [64-68] is a problem solving environment aimed at exploring Grid technology in engineering designs. It is a multi-disciplinary effort undertaken jointly with researchers from academic and commercial partners. The problem solving environment brings together engineers, Mesh experts, Computational Electromagnetic simulation experts, and computer scientists to achieve numerical simulation and visualization. The GECEM problem solving environment strongly focuses on geographically distributed resources sharing and collaborations, and expects a novel innovation in Engineering designs based on the Grid technology. A critical issue in success of this innovation is the ability to closely couple scientific applications developed by engineering scientists with middle-tier support provided by computer scientists.

GECEM, a service-oriented simulation problem solving environment based on the Grid technology, provides a platform for engineers and scientists to share their collective skills, applications, data and computational resources in a secure, reliable and scalable manner. It is a distributed computing infrastructure designed to facilitate the engineers and the scientists to remotely initiate, run and monitor engineering services aiming at achieving collaborative numerical simulation and visualization. It enables the seamless integration of heterogeneous compute resources that span multiple administrative domains and locations across the world, and provides flexible and secure access to these resources to all the participants through a user friendly interface.

Through the seamless collaboration among virtual organizations, distributed resources can be more effectively used to tackle complicated engineering problems and contribute to the exploration of problem in extremely complicated conditions. We believe such an adaptable and flexible integration framework is able to meet challenges encountered in engineering problem solving environments -- increasing complexity, highly compute resource demand, and large number of applications involved.

5.2 Some Details

GECEM Grid infrastructure is based on Globus middleware, which has proven to be a powerful and acceptable reliable basis for our work. For applications involved in GECEM, a collection of OGSI compliant Grid services are developed at the corresponding service provider sites. The service oriented approach provides a more flexible and interactive environment in terms of service discovery, invocation, steering, and notifications. Grid Resource Allocation and Management (GRAM) [69] is used as interface to start jobs on computational resources, which will contact and submit the jobs into a back-end scheduling system, such as Condor, LoadLeveler [70], OpenPBS [71], and so on. GridFTP [40] is used to transfer any data files between computational resources and storage resources . JavaCoG [72] is adopted as the main programming interface to Globus-based services such as GRAM, MDS [73], and GridFTP. It also provides a client-side API for MyProxy [74] and has extensive Grid Security Infrastructure (GSI) [75, 76] support. GSI is a portion of the Globus Toolkit that provides fundamental security services needed to support the Grids in terms of message-level security, transport-level security and authorization. MyProxy, a combination of an on-line credential repository and an on line certificate authority, has been chosen to manage X.509 Public Key Infrastructure (PKI) security credentials.

These services are implemented as a collection of Web and Grid services, each developed at the corresponding services site. Clients can access the GECEM services in different ways. A GECEM Portal presents these services to the users through a simple user-friendly interface, and also hides the users from complexities of underlying Grid infrastructure. The portal provides seamless integration of a collection of heterogeneous computational and data intensive applications across geographically distributed virtual organisations.

Portlets [77] is adopted to enable service brokers to create interactive services, which plug and play with portals via the Internet, and thereby open up many new integration ability. The GECEM portal is based on GridSphere [78] and runs as portlets in any standard portlet container, which provide an interface to the user to access the Grid environment.

5.3 GECEM Architecture

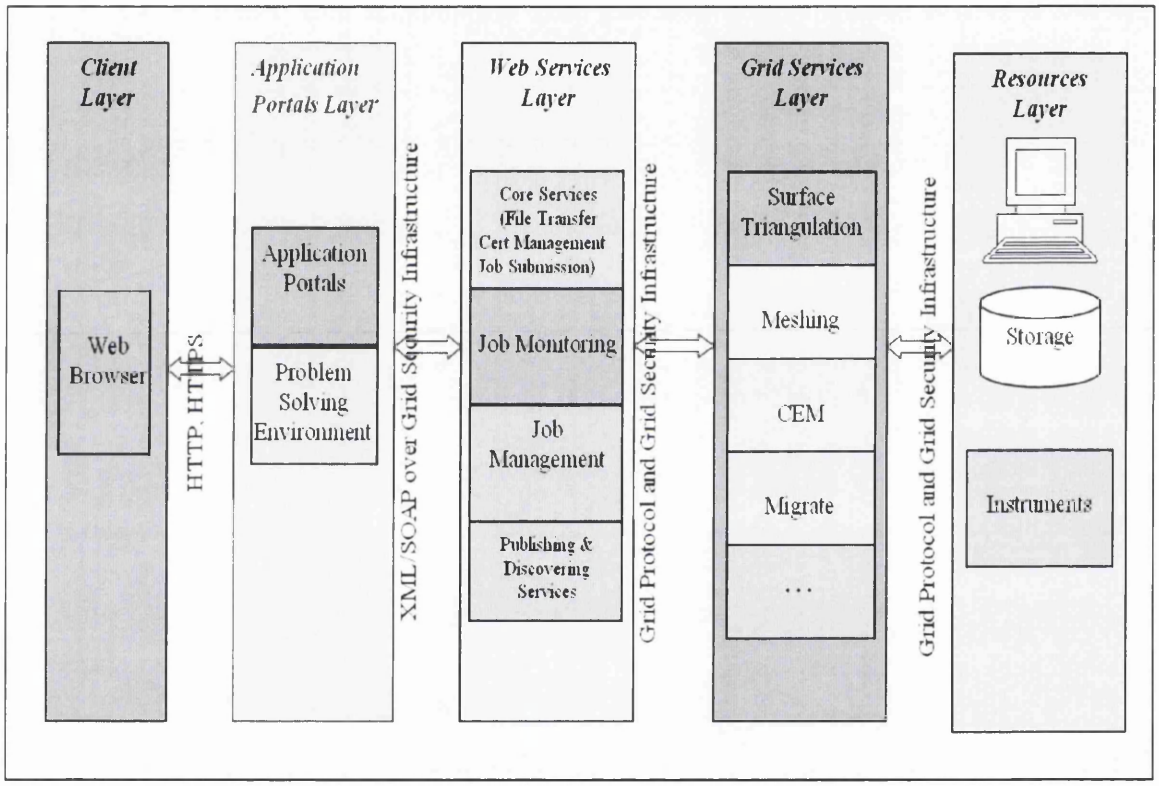

Figure 5.1: GECEM n-layer service-oriented Architecture

Service Oriented Architecture (SOA) is architecture with special properties, comprised of components and interconnections that stress interoperability and location transparency. The idea of SOA is to achieve loose coupling among interacting software. This advanced flexible style of architecture provides a foundation to allow Grid resources to be shared seamlessly. SOA is adopted on the GECEM Grid infrastructure. Web services, Grid services, and Grid portal have been seamlessly integrated into a GECEM service oriented framework. As shown in Figure 5.1, the layered architectural

framework is built and comprised of a client layer, a portal layer, a web/Grid services layer, and a compute resources layer. Clients communicate with the portal layer by sending and receiving SOAP over SSL, and utilize the portal for core functionalities. The portal in turn forwards some of incoming requests to a series of Grid and Web services and relies upon underlying Grid and Web services to provide functionalities to the clients.

A pool of OGSI-compliant Grid services are fundamental components of this architecture. The implementation, deployment and communication of these Grid services are based on Globus Toolkit and Web services technologies. Since Web Services are platform and language independent, Web services are used to implement support services between the applications and the portal. Web services components are deployed to provide support functionality, such as certificates management, jobs submission, files transfer, jobs monitoring and management, and publishing and discovering services. End user interface is dominated in the use of portal servers. Portlets is adopted to enable service brokers to create interactive services, which plug and play with portals via the Internet, and thereby open up many new integration abilities. The Grid portal is based on GridSphere and runs as portlets in any portlet container, which provides an interface to the user to access the Grid environment.

The client layer only consists of a Web compatible browser for the purpose of lightweight. Any system that can run a Web browser is capable of serving as a client for GECEM. The portal layer provides an end user interface, through which the users can access the services from anywhere with a browser and an Internet connection.

The portal is a user friendly Web-based interface that remotely launches and monitors computational simulations on GECEM computational resources at remote sites. It provides seamless integration of a collection of heterogeneous computational and data intensive applications across geographically distributed virtual organizations. The portal presents the applications to the end users through a browser and an Internet connection, and hides the users from complexities of the underlying Grid infrastructure.

The portal is a very important aspect of GECEM and we have spent a significant amount of effort on the GECEM portal to address CEM applications specific issues. Two portals are developed in order to suit different research environment needs, a service-site portal and a general portal. The former provides an interface for service

providers to access the services; while the latter offers a service for the majority of users. GECEM portals provide additional features such as application-specific data transformation between big-endian and little-endian, and input files transformation between XML and text-based format. Our experience shows that portal is a lightweight and easy to use solution to link the resources with the users. But more sophisticated approach is needed to achieve more client side assistance in some complicated cases, for example, an interactive visualization of complex results set.

The business logic layer includes a collection of Web and Grid services provided by middleware and applications provider. The applications are decomposed into component-oriented services, which are exposed in a generic interface independent of implementation languages and platforms. A service provider has flexibility to move the services to different machines, or to move services to an external provider. One service can support different client types. These applications, as a pool of OGSI-compliant Grid services, are the fundamental components of this architecture. Since Web Service is platform and language independent, it is used to implement support services between applications and portal.

The resources layer includes underlying computational resources, data storage resources, and any instrument involved.

5.4 Service Oriented Wrapping

5.4.1 Introduction

The approach of making a piece of code such as simulation solver available as a self-contained reusable object to some higher-level glue layer is often termed wrapping. In this case, the glue layer is a high-level language; in a more general case, it could be a Grid fabric layer such as Web Services, allowing interoperation across a network of components running on different machines. As discussed in Chapter 2, two strategies are adopted for describing and controlling application processing, batch oriented wrapping strategy and service oriented wrapping strategy. Here we introduce the service oriented wrapping strategy used in the GECEM problem solving environment.

In GECEM, a few meshing and simulation applications used in the community are integrated with the SOA and Grid environment through interactive service oriented

wrapping approach without re-design and re-development. For example, Meshing Grid Service is a mesh generation service to mesh flow domain. CEM Grid service provides sophisticated simulation on an incident electromagnetic wave and a general scatterer. These services are easy to be invoked remotely without legacy codes migration. It encourages collaborations between organizations to deliver better outcomes by combination of various applications geographically located.

In GECEM framework, these OGSI-compliant Grid services are the fundamental components of GECEM architecture. The implementation, deployment and communication of these Grid services are based on Globus Toolkit and SOA.

To be better adapted for the Grid environment, different legacy libraries are precompiled and the proper one will be chosen to provide run time support according to the platform on which the service is hosted. To be fully merged with SOA principle, instead of monolithic structure with a Web service interface, a single application is split into several Grid service based components. Minimal modifications are required to the legacy application.

5.4.2 How to wrap

Before wrapping, the legacy application needs to be reconstructed and modified while main computation parts and existed functionalities are kept untouched. We list operations and functionalities provided by the application, and make it clear that how communication goes on between the functionalities. In each functionality context, we pay attention to how the job is invoked, when and how the response is sent back, and how the job information is indicated. With all this information, the legacy application is re-constructed and further divided into smaller, independent components. Instead of the legacy monolithic structure, we split the single application into several discrete components. Also we adjust the components to make sure they are capable of working independently and not rely on other resources.

After the reconstruction, we use service oriented wrapper strategy to add a Grid fabric layer for each component. The service oriented wrapper strategy can be used for generic cases where the legacy codes (written in FORTRAN, C or others) need to be accessed from other environments. Computational intensive parts of component, written in FORTRAN, still remain the same. These core calculation parts are presented as C

wrapped library for easily plugging into Grid environment. Then a Java wrapper is defined with native methods. Java Native Interface [79] is adopted to enable the integration of the Java wrapper with legacy code written in other languages, and allow Java code to operate with existing applications and libraries. This wrapper strategy keeps the core computation part in a language like C or FORTRAN, but controls its behaviour through a high level language. This strategy not only keeps the performance strengths of FORTRAN, but also allows the programmer to easily interface the applications through the high level language.

The Java wrapper and the library wrapped need to be further exposed as a Web/Grid service. We need to consider how to implement these individual services, OGSI-compliant or Web services. By the time we developed the services, Globus Alliance had not contributed the WS-Resource Framework (WSRF) implementation. We choose the OGSI-compliant Grid service to expose the Application Services. For the simulation services to work in a larger e-science context, some issues have to be addressed, such as security and interoperability with other services and consumers. In interoperability aspect, Grid service enhances the web service, which is stateless and non-transient. OGSI provides consistent mechanisms for statefulness (SDEs), stateful interactions, transient instances, service lifetime management, notification on state changes and security infrastructure, which are all key requirements for GECEM framework. GSI is the security architecture that provides the fundamental security services and guarantees reliable invocation, authentication, authorization, delegation and confidential communication [76]. It does provide a useful base to Grid infrastructure for easing some security concerns.

For every application (volume mesh service and electromagnetic simulation service), a persistent application factory service which creates transient service instances on demand is defined and hosted,. The factory implements the Factory portType, which provides an operation to create Grid service instances. NotificationSource portType and GridService portType are also implemented, which provide generic and notification functionalities to the Grid services. A WSDL is created manually for each application Grid service. These services are compatible with OGSA's well-defined interfaces and specific conventions addressing discovery, dynamic service creation, lifetime management, notification, and manageability. They also ensure high

security based on authentication, authorization, and incorporate credential delegation. At client site, the client uses the grid middleware (client) to invoke the Grid services and receives serviceLocators for the newly created service instances.

The whole wrapping procedure is summarised and described below:

i) Split the single application into several discrete components.

ii) For each component, define the Java wrapper with native methods. This Java wrapper loads and links to the native implementation.

iii) Run the Java to get the header file, which will be included in the C wrapper.

iv) Change the FORTRAN main to a subroutine, with command line arguments passed as parameters.

v) Create a C wrapper to invoke FORTRAN subroutines. Add control code in the C wrapper to control the FORTRAN computation loops and data transfer.

vi) The FORTRAN code and C wrapper are compiled together as a dynamic link library, which can be loaded and linked into the Java Virtual Machine. This shared library needs to be present where the grid service is provided.

vii) Finally the Java wrapper is exposed as the grid service implementation.

5.4.3 Stateful Grid Service for Data Sharing

One of the fundamental requirements for the Grid is to share application data managed internally with processes running outside the application environments. File system can be used to transfer data among various processes in a local environment. But it falls short in the Grid environment where processes can be dispatched to run on different resources. The OGSI supports the data sharing across compute resources through the concept of the Grid Service Instance.

Due to concerns for security and interoperability requirement, we implement the components as a collection of stateful Grid services. This stateful model avoids passing state information between service and consumer. All variables and data in an application process are handled by a unique Grid Service Instance, identified by Grid Service Handles (GSHs), which can maintain state between invocations. Some data in

the application process is exposed through some well-known Service Data Element, which allows multiple processors or consumers to share the same data by sharing the GSH. A Grid Service Handle can be seen as a pointer to a particular stateful interaction, which is useful to access data through process and represent interaction state.

According to our experience, OGSI provides abilities to dynamically create transient stateful service instances, which is helpful for data sharing in the Grid environment. But it is achieved through the introduction of the Grid Service Instance, which is object-oriented, and results in tightly-coupled applications. Specification of high level interfaces needs to be emphasized instead of creating transient service instance at the infrastructure level. Grid services need to move towards a service-oriented architecture adopting secure reliable messages to couple processes.

5.4.4 Others issues

Service providers can publish details of service descriptions to allow easy discovery through community registries. GECEM services can be easily invoked via a command-line interface, grid portal or other remote procedure call without going through repeated program initiation and termination. But such Grid services cannot be run simply across the grid environment like a batch job. They must be installed and deployed within a Grid service host environment (such as an Apache Tomcat servlet container) on the compute node that runs the service oriented application.

5.5 Migrate Legacy Service Model - Batch Oriented Wrapping

5.5.1 Model Introduction

With batch oriented strategy, existing applications can be run on available computers in a Grid environment. GECEM applications are conventional FORTRAN applications that read input files and generate a set of result files. Originally GECEM legacy applications were invoked from command-line. Input files and other configuration information were specified in command-line parameters. These applications usually remotely run on traditional HPC systems. Users are required to log in to system and submit jobs to a queuing system. Input files and output results have to be transferred manually. Perfectly fitted to this scenario, a Migrate Legacy Service Model (MLSM) is

developed to drive the applications automatically through a browser, hence change traditional manual driven via terminal.

MLSM brings together three distinct roles: service consumer, service provider, and computational resource provider. It allows input data from user site A, application executables from service provider B and work together at computational resource C provided by a resource provider. The user case we are addressing here is that the input data, the applications, the compute resources are owned by different entities. This model provides the applications with a gateway to the Grid environment. The main idea is to allow the consumers to remotely process the applications to fulfil extra computing resources requirement through a user-friendly interface. In this model, a user only focuses on configuration, steering, and monitoring services without being aware of the details of the underlying Grid infrastructure.

MLSM changes conventional job submission through user log-in in many aspects. The service supports different client format. It brings in security based on GSI. It also has delegation functionality, so the service can further invoke other service on behalf of the user. Processes requested via the service are launched as service handlers rather than jobs. Data that is generated from the service requests may or may not belong to an individual user.

MLSM is designed for batch-driven, command-line oriented applications. It can work with all the command-line oriented executables written in any language. It typically suits parallel applications, which are hard to be migrated into the Grid environment using service oriented wrapping approach described in Chapter 2. In this model, there is no (any) modification on legacy code, so source code is not necessary to be available. It provides an easy solution to migrate the legacy applications to the Grid environment with the smallest effort and cost. The Grid middleware is responsible for security, resource control, scheduling, and so on. But an executable application tends to have special requirements for different platform versions, such as collections of libraries, JAR files, and any other environmental conditions. The heterogeneous nature of computing resources still remains a significant barrier in this context. Pre-process work on the executables and input data is required to tailor to various environments. A job broker may be needed to allocate suitable resources for the executable according to its specific requirements.

5.5.2 Model Architecture

The Migrate Legacy Service Model is composed of a number of pieces of software interacting over several different sites, client site, service provider site, computation site, and data site. The Grid portal is provided for service invocation, through which a user needs to point out Universal Resource Identifiers (URIs) of input files and to specify a service configuration. The Migrate Legacy Service (MLS) passes all these info and requests a legacy job. Globus Toolkit, a middleware layer, handles secure communications and data transfer among different sites.

5.5.2.1 Client Site

At the client site, the user interacts with the Migrate Legacy Service through a standard browser. A Grid Portal is provided to invoke the services. Http, instead of GridFTP, is used to bring results back to user site after a job finished. This avoids any installation from the user site.

5.5.2.2 Service Provider Site

At the service provider site, a Migrate Legacy Service is hosted and it serves as a bridge between users and compute/data resources. MLS is a Grid service that provides the applications with a gateway to the Grid environment. Front end, MLS provides a Grid service interface to communicate with the user. Backend, it contacts a job manager through Globus GRAM to migrate a legacy job to allocated computational resource. Also it contacts GridFTP servers on different sites to transfer large amounts of data among geographically distributed storage systems. MLS is implemented both through programmatic job submission against computer Grid using the Java CoG Kit, and through executing scripts including a set of Globus commands in a defined environment (JAVA, Perl, etc.).

Figure 5.2 illustrates events sequence of this migration procedure. The consumer stores his/her proxy on a MyProxy server and interacts with the portal. The portal retrieves a proxy for the consumer from MyProxy server and then uses it to contact the MLS on behalf of the consumer. The MLS submits the job to the computational site via Globus job manager. Upon receiving the MLS request, Globus Resource Allocation Manager's gatekeeper of the allocated compute resource spawns the application migrated sequentially or across multiple computing nodes in parallel. In this process, input files and application executables, appointed by the consumer, are staged to the

chosen compute nodes by MLS. Upon completion, results are brought back to where the consumer appointed. The consumer can submit, query and retrieve the results of Grid jobs.

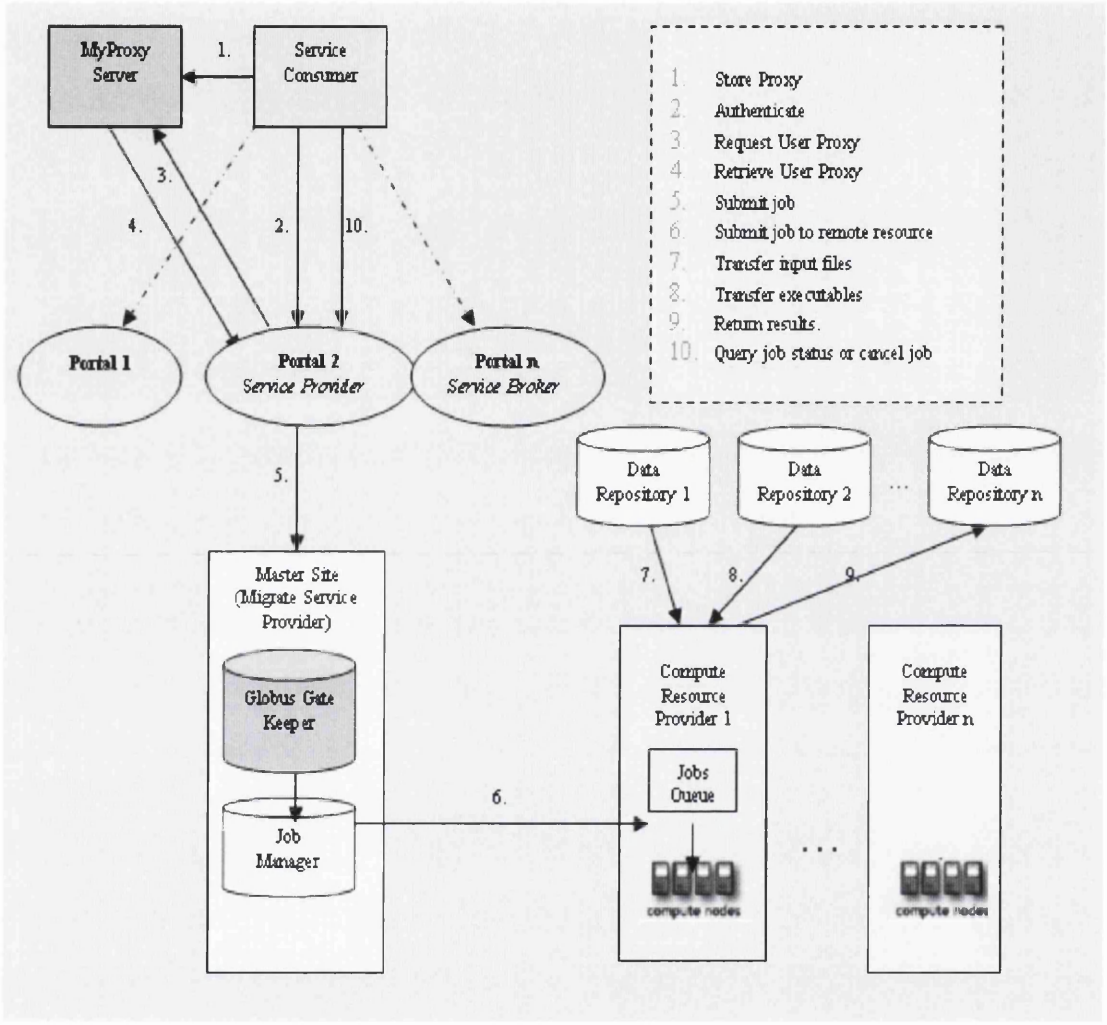

Figure 5.2: Migrate Events Sequence

The Migrate Legacy Service is an OGSI compliant service with delegation functionality. Delegation is a key factor to guarantee the MLS further invokes other services on behalf of the users.

5.5.2.3 Computational Site

At the computational site, a Globus container runs continuously where it offers a persistent GRAM service to response the Migrate Legacy Service's request and forwards a job to a related local scheduler. It also provides a GridFTP service to response file transfer requests from the Migrate Legacy Service.

5.5.2.4 Data Site

At the data site, a GridFTP service is provided to put a third-party file transfer into action, which is steered by the Migrate Legacy Service.

There is no installation work at the client site. At the service provider site, service providers need to install Globus. They also need to deploy and host a Migrate Legacy Service on the machine where the application executables are available. At the computational site and the data site, Globus needs to be installed to response job requests from the Migrate Legacy Service.

5.5.3 Implementation Issues

We adopt a well-documented and industry standard approach, Resource Specification Language (RSL), to express our requirements. Resource consumption information and other job information are provided, such as name of the executable file, files to stage in and stage out, files clean up, maximum memory required, CPU usage, etc. A job defined in this way can either be sent via Globus commands or be submitted programmatically to Globus GRAM.

Independent sub-jobs strategy is used in Migrate Legacy Service. Instead of submitting one big job with input staging and output back in one go, we divide the whole MLS job into several independent sub-jobs, input staging, legacy processing, and output back. This is partly because of flexible security control over different sub-jobs, and partly because of efficiency of the computational site. Further explanation is given below regarding these benefits.

Due to three distinct roles involved, service consumer, service provider, and computing resource provider, it brings in a big challenge in a security context. As Migrate Legacy Service is a Grid service, so operations are executed as 'container owner' by default, a service provider who hosts the service in this context. But some functionality, such as the user's input data retrieval, has to be executed as the user. We divide the job into several sub-jobs. Sub-jobs can be controlled to run as different identities, for example, retrieving the user's data as the user, retrieving application executables as the service provider, and submitting a job as the user or the service provider. This is achieved through fine-grained control over security properties of MLS, such as authentication mechanisms required to access the methods of MLS and

run-as identities of MLS. Account and credential management mechanism is adopted to organise and pass the proper credential needed to satisfy security requirements from each role, which is described in detail in section 5.6.

Independent sub-jobs strategy also brings more efficiency and flexibility to compute site compare with a single job. For example, the user can resubmit a "legacy job" without re-transfer input files if previous submission failed. Also, if the "legacy job" is terminated or failed, the user still can transfer partial result back. Also "input staging" and "results back" jobs only need GridFTP server and data storage, but not any other special resources demand, whereas "legacy processing" most possible has very strict and high computational resource requirements. If all these sub-jobs were bound together as a single job, computational resources would be engaged for the whole job. Infact, these resources were left idle while file transfer was carrying on.

With independent sub-jobs strategy, the user can control and steer many sub-jobs instead of one big job, and it is easy to get clear and correct information of each sub-job.

5.6 Security

5.6.1 Security Issues

Security is a critical requirement and must be accounted for by any geographically distributed Grid community. To achieve a specific goal, Grid extends the conventional homogeneous system to a more heterogeneous environment which brings a diverse set of users, data, applications, and compute resources together. Security control and management of these different resources are often the responsibility of entirely different organizations. Traditional security mechanisms for homogeneous system do not scale to heterogeneous environment belonged belonging) to different organizations. A GECEM security model is set up to address security management and collaboration issues among virtual organization. This security model plays a key role in supporting and enforcing the overall security requirement, and in providing authentication, authorization, role management, access control, delegation, and so on.

To provide a robust security infrastructure, GECEM security model is based on Grid Security Infrastructure (GSI) assisted with Grid Accounts and Credentials Management Model.

All web transactions are executed under a Secure Socket Layer via HTTPS. A secure HTTP connection means that data sent to and received from an HTTP server are encrypted before being sent out over the internet.

Based on Public Key Infrastructure, Grid Security Infrastructure prohibits a centrally managed security system. Key of this technology is the introduction of a Public Key Infrastructure credential, which consists of a proxy certificate and its corresponding private key. The credential is a certificate generated from the user's certificate and acts as a representative of the user. This credential is passed and verified across organizational boundaries; hence, it allows heterogeneous system to be secured on a distributed basis. The introduction of credential has many befits. It provides "single sign on" capability, which avoids the need of private key associated with the user certificate. The lifetime of credential can be set limited, which reduces the influence in case of exposure. It is also possible to flexibly control rights granted to the credential.

Dual authentication and authorization is performed between system services and service consumer based on X.509 certificates. GSI defines and implements useful security services for authentication and delegation, which is proved helpful and easy to work with in some uncomplicated context. GSI has been used throughout the GECEM system and hence presents the user a consistent security mechanism for compute, data, and application resource. However, our experience shows that tasks of creating and managing the user accounts and credentials used by GSI are complicated and difficult due to complicated security requirements the Grid faces. Grid Accounts and Credentials Management Model are adopted to solve this problem, as will be further discussed below. Some middleware is needed for automating user registration, credential creation, and credential management tasks.

GECEM application services also have delegation functionality and the client side is also set with fully delegation, which allows delegation of credentials for computations that involve multiple sites, and provides the user with a single secure access point. This "single sign-on' mechanism allows the user to easily access many diverse data, applications, compute resource owned by different owner without visiting many sites. The user can simply ignore the underlying complicated Grid infrastructure and no longer have to type in different accounts and passwords for different resources. This "single sign-on" makes it easy for the user to access resources across organizational

boundaries, but it lays a challenge to user management for Grid sites. Typically in large scale VOs, an end user may neither have access to his/her physical certificate and key, nor have an account login on the remote resource.

In a typical usage scenario, the user applies for a user certificate and key from trusted certificate authority. UK National CA and Singapore CA are trusted by GECEM system. We make some effort to recognize Singapore CA and create new opportunities for sharing Grid resources between UK and Singapore.

Also the user is required to have a user account on the Grid resources he/she is entitled to use. The user's distinguished name (DN), associated with each certificate, must be individually registered on a mapfile of the Grid resource. The user generates a limited lifetime proxy using X.509 personal certificate and key pair. This proxy is passed among the Grid resources for authentication. The subject of the proxy is verified against the DN entry recorded on the Grid mapfile. Authorization and access control is performed based on the verification. With delegation functionality, the service can use the user's proxy to invoke other services on other Grid resources. The further invoked service performs all authentication and authorization against the user proxy, without having to trust the intermediate sites which forwarded the job. No further password request is handed to the user for invocation of a new service. At each site, after passing authentication and authorization the user certificate is mapped to local account, which may be different at different sites.

MyProxy is used to store the user's credentials. The user can store his/her proxy with a defined amount of time on a MyProxy Server. When the user logs into GECEM portal, the portal contacts with the MyProxy server and retrieves the credential with limited lifetime for the user. This short-term credential is used by portal to access Grid resources on the user's behalf by signing request with the private key belonging to the proxy certificate. With MyProxy's credentials delegation, the proxy can be retrieved directly from the web browser. Also the private key of the user's digital certificate can be avoided to be sent over a network.

To further circumvent client firewall restrictions, all communications to GECEM services are client initiated, where the client invokes a service and polls for responses / results at a later date (no Web Service call backs are made to the client). This architectural model clearly separates service providers from the client.

Accounts and Credentials Management Model is composed of several models, Credentials Management Model (CMM), Service Provider Account model, User Appointed Compute Resource model and Accounts Pool Model.

5.6.2 Credentials Management Model (CMM)

Scenario a: Migrate Legacy Service needs to access different resources belonging to different roles in one single job.

Migrate Legacy Service needs to access various kinds of resources owned by different entities including data, application executable, and computational resource. We introduce a third-party CMM to assist the resources sharing in a controlled manner.

In Migrate Legacy Service, it allows input data from the user site A, application executables from the service provider B, work together at compute resource C provided by the resource provider. The user case we are addressing here is that input data, applications and compute resources are owned by different entities from the same Virtual Organization. None of them wants to give access to the data or the application to other members. Only GSI mechanisms are very hard to satisfy the requirement because of read and write permission problem among many data storages belonging to different identities. We introduce a third-party Credentials Management Model (CMM) to sort this problem out.

CMM is an end-to-end GSI based credentials management solution which is exposed for using via Grid portals. CMM consists of a front-end Grid portal and backend security services that provide secure management of the credentials.

Different roles delegate their proxy credentials to this agent. Following the user delegation, or stringent identity verification, CMM automatically generates and stores credentials for the users, which are allowed for subsequent use by the agent. The model organizes these credentials in a sophisticated manner to perform different sub-jobs of a Migrate Legacy Service job on behalf of different users. Using CMM, Migrate Legacy Service has the ability to authenticate to some data resource (GridFTP server, SRB [80] archive, etc.) as the user, meantime authenticate to some other resources using the service provider's credential. This means the service provider can retrieve the protected input data (accessed only by the user) on behalf of the user. It also can retrieve the

protected application executables and invokes remote service on the resource provider on behalf of itself.

CMM manages these credentials on behalf of the users without making them aware of their credentials' existence. The model also has functionalities to support credential renewal and revocation. By applying credentials of different users during different steps of a job procedure, CMA orchestrates these individuals to share responsibilities of the whole job.

5.6.3 Service Provider Account Model

Scenario b: The compute resource provider trusts the service provider, not the user. In a typical usage scenario, the user invokes the service provided by the service provider, and then the service provider accesses the compute resources on behalf of the user. This means the compute resources have to trust all the users trusted by the service provider, hence it brings account management burden to the resource provider. We introduce a Service Provider Account Model to avoid this problem. In this model, the user does not have direct access to the compute resources and is completely decoupled from the compute resources where jobs are effectively run. The user only has access to the service provider site. A special Grid user account is set up for the service provider, which is trusted by the compute resources. The service provider acts as an active agent between the user and the compute resources. On one side, the service provider authenticates and authorizes the users, and serves the requests of the users. On the other side, it accesses related compute resources on behalf of itself. The compute resources response the requests based on their trust of the service provider.

In GECEM, each use has a X.509 certificate and its Distinguished Name (DN) is included in a grid-mapfile on the service provider site in order to access the service. The service provider, as a special Grid user, has its certificate and is included in the grid-mapfile on the compute resource provider site.

This account management approach benefits from the following advantages:

i) It abstracts the resources used, allowing the resources to be used without the user's awareness. The user does not need to obtain multiple user accounts on different Grid sites to complete one job in the Grid environment. Instead of the user, the service provider will access the resources at cross-sites organisations.

ii) The Grid resource only knows about the service provider and is free from the task of access control for every Grid community user. This greatly reduces the account management burden and allows (for) scalability.

5.6.4 User Appointed Compute Resource Model

Scenario c: The compute resource provider trusts the user only, not the service provider.

In this scenario, the user is trusted not only by the service provider, but also by the compute resource provider. In this case, the service provider accesses the appointed compute resource on behalf of the user. This scenario allows verifying the identity of the user at the compute resource site, without having to be aware of the intermediate sites which forward the job.

5.6.5 Accounts Pool Model

Credential Management Agent makes the compute resource trust the service provider or the user, and avoid trusting all the users of all the service providers. But according to GSI, each trusted user needs to obtain a traditional user account on the compute resource and the user's Distinguished Name (DN) has to be mapped to this user account. This requires the compute resource to set up and maintain account for each trusted user; thus, making it a burden for large Grid communities. This is not a scalable long-term solution and it makes it difficult for Grid community to extend to many users. In GECEM context, this scalable issue is dealt with through applying the Accounts Pool Model (APM), which introduces a dynamic user accounts pool. We create a pool of user accounts at each site for all trusted users sharing. APM dynamically maps the trusted Grid user to a user account from the user accounts pool. This user account is blocked after it is allocated to a user and is released upon the job has been completed.

CHAPTER 6

Conclusions and Future Work

The purpose of this chapter is to summarize the thesis contribution and conclusions. We conclude by highlighting some future work that could stimulate future research in enabling legacy applications on distributed compute resources.

6.1 Summary of the Contribution

Adaptable and flexible integration frameworks are highly required to meet challenges encountered in the scientific problem solving environment. The main goal of the thesis has been the development of a framework for compute resources sharing. The contribution could be summarized as follows:

i) Providing an XML based language, Command-line Description Language (CoLDeL), to describe individual command-line applications, precisely. CoLDeL acts as a protocol so that different service providers can follow it to generate an Application Definition File for each scientific algorithm to be used by the Application Services. An XML schema has been defined which ensures strongly typed data exchanging among services. An initial number of types and a rich set of elements are declared.

ii) Providing Application Service Toolkit (ASToolkit) that is both cost-effective and simple to wrap a large number of applications as Application Services, without the problems of updating and maintaining the source codes and deployments of all the Application Services.

ASToolkit is an automatic toolkit that wraps scientific applications as Application services and deploys them on the grid. The Application Service is described by CoLDeL, presents a Web Service Description Language (WSDL) interface to potential clients and interacts with Grid resources via a component plug-in model.

iii) Employment of a component plug-in mechanism that allows the Application Service to be configured (at deployment time) with a Job Submission Component capable of interacting with the available compute resources.

The component plug-in mechanism makes the Application Service Grid-aware and renders the Application service the capability to provide a uniform submission layer on top of different heterogeneous execution environments. The Application Service provides a level of abstraction to the client that is much higher than services like Gram because it takes low level job submission details like environment variables and temporary file management out of the hands of the client.

iv) Providing a WS-Security based authorization mechanism by which service providers can control what users can invoke on their Application Services to run the applications.

v) Providing a mechanism to create the specific Application Service on demand in the event it is not kept persistent or is unavailable during the execution of a scientific workflow.

The unique contribution of this work is the design and implementation of this mechanism, which is termed Abstract Application Service (AAS). AAS can create specific Application Service instance on demand in a way that is completely transparent to the user and provides a high availability of Application Services without actually requiring them to be persistent. The novel aspect of the mechanism is that AAS creates Application Service by configuring itself on the fly to become a particular Application Service in need, not by instantiating the Application Service. This is achieved by the dynamic combination of the common abstraction for legacy applications and application description using specially designed Command-Line Description Language (CoLDeL). This combination allows AAS to dynamically configure itself to a particular Application Service just in time. An AAS may have several concrete instances running at the same time on the grid, and each concrete service instance may have a different legacy application associated with it.

AAS is a generic application service. Scalability of this AAS approach is achieved by delivering the applications through a dynamically reconfigurable AAS. This mechanism obviates the need to keep all the available applications wrapped as persistent Application Services.

vi) Providing an overall framework for enabling the legacy applications and data on Grid based and Service Oriented Architecture.

The framework has achieved four primary functional goals: to provide an ability to allow legacy algorithms to be accessed easily and run in the Grid environment; to allow existing data management procedures to be more accessible and interoperable; to provide graphical user interfaces to access a large number of Application services and federated database from a scientific portal, and yet keep the portal lightweight and manageable, and finally to provide a lightweight workflow composer to compose sequential workflows from Application Services.

vii) Exploring a Group Applications Service (GAS) approach to further optimize the execution time of a workflow. GAS merges several Application Services into a single service. It reduces the grid overhead induced by the Web service invocation, scheduling, and data transfers. GAS not only fulfils the tasks of a set of Application Services, but also takes over the role of workflow service to assemble the applications and orchestrate the data transfer and input/output. The contribution of this approach is to move the jobs assembling and orchestrating tasks from workflow level to Application Service level. Actually GAS acts in dual roles as both Application Services and workflow. This provides an opportunity for GAS to optimize and offer the most efficient performance based on its knowledge of not only applications, but also the workflow and job dependencies.

viii) Providing a mechanism to monitor and restart the job.

6.2 Future Work

There are a number of improvements that can be made to the framework in order to better support the applications that it wraps.

6.2.1 Application Description File Generator

Based on CoLDel protocol, the service provider or application provider needs provide an Application Description File (ADF) for each application wrapped. Our experience shows that writing an ADF is not an easy task for a person who is not familiar with CoLDel. An Application Description File Generator, possible via Web interface, is needed to let the service provider enter the information required in an application description, and generate the ADF on the fly. The ADF is able to be pushed to the server side and registers with the ADFs pool. Application providers can also register the ADF file which they have already via the Web interface.

6.2.2 Batch Submission Optimization

At present, the batch submission capability is provided at workflow level, rather than Application Service level. This means the Application Service needs to be invoked a number of times to complete the batch submission. We plan to provide support for parameter sweeps [81] in the Application Services. This will allow users to run the same application a number of times using a "set" of values for each input parameter in one Application Service invocation.

6.2.3 Checkpointing and Monitoring Optimization

Currently one job is divided into following stages: input stage in, application computation, and output stage out. User can monitor the status of each job stage. The job specification is recorded for each job, which contains all the information regarding the parameters values, input files locations and others. The user can restart the job using the recorded job specification upon failures. But the job has to be resubmitted from the scratch, not from the failure point. A checkpointing capability is needed, which can be used to restart the jobs from failure point instead of from the very beginning.

6.2.4 Fault Detection

Currently we only provide an ability to recover from faults by restarting an application. We are unable to provide a capability to detect these faults, as they occur. We can provide automatic identification of causes of failure of applications. This will enable users to easily identify the reason why the application failed during its execution and take appropriate actions.

6.2.5 Asynchronous Communication

Presently, the communication between client and monitoring service is carried out via the client's requests for immediate, synchronous delivery. This is synchronous in nature. Asynchronous communication can be added, by using implementations of popular Web services based publish-subscribe systems such as Web Services Notification [82].

Bibliography

1. El hachemi M, Hassan O, Morgan K, Weatherill NP. 3D time domain computational electromagnetics using a H1 finite element method and hybrid unstructured meshes. Computational Fluid Dynamics Journal. 2004;13:55–66.
2. Rao A, Chandrashekara R, Sanchez-Ortiz GI, Aljabar P, Mohiaddin R, Hajnal JV, et al. Spatial transformation of motion and deformation fields using non-rigid registration. IEEE Transactions on Medical Imaging. 2004;23(9):1065-76.
3. Newcomer E, Lomow G, editors. Understanding SOA with Web Services: Addison Wesley; 2005.
4. [cited; Ant website]. Available from: http://ant.apache.org/index.html
5. Thain D, Tannenbaum T, M L. Distributed computing in practice: the Condor experience. Concurrency - Practice and Experience. 2005;17:323-56.
6. Gentzsch W. Sun Grid Engine: Towards Creating a Compute Power Grid. CCGRID, Proceedings of the 1st International Symposium on Cluster Computing and the Grid, Page: 35 IEEE Computer Society Washington, DC, USA; 2001.
7. Oinn T, Greenwood M, Addis M, Alpdemir MN, Ferris J, Glover K, et al. Taverna: Lessons in creating a workflow environment for the life sciences in Concurrency and Computation: Practice and Experience. Grid Workflow Special Issue. 2005;18(10):1067-110.
8. Churches D, Gombas G, Harrison A, Maassen J, Robinson C, Shields M, et al. Programming Scientific and Distributed Workflow with Triana Services. Concurrency and Computation: Pract and Exper. 2006;Special Issue: Scientific Workflows.
9. Ludaescher B, Altintas I, Berkley C, Higgins D, Jaeger-Frank E, Jones M, et al. Scientific Workflow Management and the Kepler System. Concurrency and Computation: Pract and Exper. 2006;Special Issue: Scientific Workflows.
10. Nadalin A, Kaler C, Monzillo R, Hallam-Baker P. Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). 2006 [cited; Available from: http://docs.oasis-open.org/wss/v1.1/
11. Welch V, Barlow J, Basney J, Marcusiu D, Wilkins-Diehr N. A AAAA model to support science gateways with community accounts. Concurrency and Computation: Practice and Experience.19(6):893 - 904.
12. Chin J, Harting J, Jha S, Coveney P, Porter A, Pickles S. Steering in computational science: mesoscale modelling and simulation. Contemporary Physics. 2003;44(5):417 - 34.
13. Foster I, Kesselman C, Nick JM, Tuecke S. The physiology of the grid: An open grid services architecture for distributed systems integration. 2002.
14. Jacob B, Berstis V. Fundamentals of Grid Computing. IBM Redpaper. 2002.
15. Cardoso J, Sheth AP, editors. "Foreword", Semantic Web Services, Processes and Applications: Springer; 2006.
16. Erl T, editor. Service-oriented Architecture: Concepts, Technology, and Design; 2005.
17. Channabasavaiah K, Holley K, Edward Tuggle J. Migrating to a service-oriented architecture. IBM DeveloperWorks; 2003.
18. [cited; Web Service from W3C]. Available from: http://www.w3.org/TR/wsa-reqs/
19. Graham S, Simeonov S, Boubez T, Davis D, Daniels G, Nakamura Y, et al. Building Web Services with Java. SAMS. 2002.

20. [cited; SOAP from W3C]. Available from: http://www.w3.org/TR/soap12-part1/
21. Christensen E, Curbera F, Meredith G, Weerawarana S. 2001 [cited; WSDL from W3C]. Available from: http://www.w3.org/TR/wsdl
22. Clement L, Hately A, Riegen C, Rogers T. [cited; UDDI from UDDI.org]. Available from: http://uddi.org/pubs/uddi_v3.htm
23. [cited; BPEL from IBM]. Available from: http://www.ibm.com/developerworks/library/specification/ws-bpel/
24. Library M. .NET Framework Conceptual Overview. 2007 [cited; Available from: http://msdn.microsoft.com/en-us/library/zw4w595w.aspx
25. Universal Description Discovery and Integration. [cited; Available from: http://en.wikipedia.org/wiki/UDDI
26. Bloomberg J. UDDI: Straw man or ugly duckling.
27. Microsoft, IBM, SAP To Discontinue UDDI Web Services Registry Effort. SOA WORLD MAGAZINE. 2005/12/18.
28. Common Object Request Broker Architecture. [cited; Available from: http://en.wikipedia.org/wiki/CORBA
29. Henning M. The Rise and Fall of CORBA. ACM queue
30. Foster I. Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing; 2005.
31. Senger M, Rice P, Oinn T. Soaplab - a unified Sesame door to analysis tools. UK e-Science, All Hands Meeting Editors - Simon J Cox, p.509-513; 2003.
32. Rice P, Longden I, Bleasby A. EMBOSS: The European Molecular Biology Open Software Suite. Trends in Genetics 2000;16((6)):pp276--7.
33. [cited; AppLab]. Available from: http://www.ebi.ac.uk/~senger/applab/
34. [cited; Tomcat Website]. Available from: http://tomcat.apache.org/index.html
35. [cited; CORBA]. Available from: http://www.corba.org/
36. Sanjeepan V, Matsunaga A, Zhu L, Lam H, Fortes J. A Service-Oriented, Scalable Approach to Grid-Enabling of Legacy Scientific Applications. International Conference on Web Services (ICWS-2005), pages 553–560; 2005.
37. Adabala S, Chadha V, Chawla P, Figueiredo R, Fortes J, Krsul I, et al. From virtualized resources to virtual computing grids: the In-VIGO system. Future Generation Computer Systems. 2005;21((6)).
38. Delaitre T, Goyeneche A, Kacsuk P, Kiss T, Terstyanszky GZ, Winter SC. GEMLCA: Grid Execution Management for Legacy Code Architecture Design. 30th EUROMICRO Conference; 2004.
39. Kacsuk P, Goyeneche A, Delaitre T, Kiss T, Farkas Z, Boczko T. High-level grid application environment to use legacy codes as OGSA grid services. Fifth IEEE/ACM International Workshop; 2004; 2004. p. 428 - 35.
40. Bresnahan J, Link M, Khanna G, Imani Z, Kettimuthu R, Foster I. Globus GridFTP: What's New in 2007. the First International Conference on Networks for Grid Applications (GridNets 2007); 2007.
41. Terstyanszky G, Delaitre T, Goyeneche A, Kiss T, Sajadah K, Winter SC, et al. Security mechanisms for legacy code applications in GT3 environment. Parallel, Distributed and Network-Based Processing, 2005 PDP 2005 13th Euromicro Conference; 2005; 2005. p. 220 - 6.
42. Geddes J, Lloyd S, Simpson A, Rossor M, Fox N, Hill D, et al. NeuroGrid: Collaborative Neuroscience via Grid Computing. In: COX S, editor. Proc UK e-Science All Hands Meeting 2004.

43. Geddes J, Lloyd S, Simpson A, Rossor M, Fox N, Hill D, et al. NeuroGrid: Using Grid Technology to Advance Neuroscience. Proceedings of the 18th IEEE Symposium on Computer-Based Medical Systems; 2005; 2005.
44. Geddes J, Mackay C, Lloyd S, Simpson A, Power D, Russel D. The Challenges of Developing a Collaborative Data and Compute Grid for Neurosciences. Computer-Based Medical Systems, 2006 CBMS 2006 19th IEEE International Symposium. p. 81-6.
45. Graham S, Davis D, Simeonov S, Daniels G, Brittenham P, Nakamura Y, et al. Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI Sams Publishing.
46. KEI O, Koich O, Yoshiaki F. A Method to Improve Modularity on Component-Oriented Web Application Frameworks. IEIC Technical Report (Institute of Electronics, Information and Communication Engineers).105(229):13-8.
47. Ferris C, Karmarkar A, Yendluri P. Basic Profile Version 2.0. 2007 [cited; Available from: http://www.ws-i.org/Profiles/BasicProfile-2_0(WGD).html
48. Adams C, Farrell S. Internet X.509 Public Key Infrastructure: Certificate Management Protocol. 1999.
49. Griffin P. Introduction To XACML. Web Services Journal 2004 [cited; Available from: http://dev2dev.bea.com/pub/a/2004/02/xacml.html
50. Bergsten H. JavaServer Pages. O'Reilly Media. 2003.
51. [cited; JWSDP website]. Available from: http://java.sun.com/webservices/reference/apis-docs/jwsdp2.0.jsp
52. Mahmoud QH. The New Java Web Services Developer Pack 1.4 (Java WSDP 1.4). Sun Developer Network; 2004.
53. GT 2.4: The Globus Resource Specification Language RSL v1.0. [cited; Available from: http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html
54. Burns M, Rowland AL, Rueckert D, Hajnal JV, Hill D. A Grid Infrastructure for Image Registration and Segmentation. In: Cox CJ, editor. Proc UK e-Science All Hands Meeting 2004; 2004.
55. [cited; DAGMan website]. Available from: http://www.cs.wisc.edu/condor/manual/v7.0/2_10DAGMan_Applications.html
56. Wilkinson MD, Links M. BioMOBY: An open source biological web services proposal. BRIEFINGS IN BIOINFORMATICS. 2002;3(4):331–41.
57. Taylor I, Shields M, Wang I, Philp R. Distributed P2P Computing within Triana: A Galaxy Visualization Test Case. IPDPS 2003 Conference; 2003.
58. Taylor I, Shields M, Wang I, Philp R. Grid Enabling Applications Using Triana. Workshop on Grid Applications and Programming Tools, Seattle In conjunction with GGF8 jointly organized by: GGF Applications and Testbeds Research Group (APPS-RG) and GGF User Program Development Tools Research Group (UPDT-RG); 2003.
59. Yoo TS, Ackerman MJ, Lorensen WE, Schroeder W, Chalana V, Aylward S, et al. Engineering and Algorithm Design for an Image Processing API: A Technical Report on ITK - The Insight Toolkit. Medicine Meets Virtual Reality, J Westwood, ed. 2002:586-92.
60. Zhang Y, Brady M, Smith S. Segmentation of brain MR images through a hidden Markov random field model and the expectation maximization algorithm. IEEE Trans on Medical Imaging. 2001;20(1):45-57.
61. Denton ERE, Sonoda LI, Rueckert D, Rankin SC, Hayes C, Leach M, et al. Comparison and evaluation of rigid and non-rigid registration of breast MR images. Journal of Computer Assisted Tomography. 1999;23:800-5.

62. Jenkinson M, Smith SM. A global optimisation method for robust affine registration of brain images. Medical Image Analysis. 2001;5(2):143-56.
63. Sled JG, Zijdenbos AP, Evans AC. A non-parametric method for automatic correction of intensity non-uniformity in MRI data. IEEE Transactions on Medical Imaging. 1998;17:87-97.
64. Chen Y, Hassan O, Jones† JW, Weatherill NP, Wang X, Walker DW. The GECEM: Applying Grid Technology for CEM Research. International Conference on Data Management, ICDM 2008; 2008.
65. Lin M, W. WD. A portlet service model for GECEM. UK e-Science All Hands Meeting 2004,: S. J. Cox, editor, Proc.; 2004.
66. Lin M, Walker DW, Chen Y, Jones JW. A web service architecture for GECEM. UK e-Science All Hands Meeting 2004: S. J. Cox, editor, Proc.; 2004.
67. Lin M, Walker DW, Chen Y, Jones JW. A Grid-based Problem Solving Environment for GECEM. 2005 IEEE International Symposium on Cluster Computing and the Grid; 2005.
68. Chen Y, Hassan O, Jones† JW, Weatherill NP, Wang X, Walker DW. A Service-oriented framework on GECEM Grid. International Conference on Data Management, ICDM 2008; 2008.
69. Feller M, Foster I, Martin S. GT4 GRAM: A Functionality and Performance Study.
70. Tivoli Workload Scheduler LoadLeveler. [cited; Available from: http://www-306.ibm.com/software/tivoli/products/scheduler-loadleveler/
71. [cited; OpenPBS website]. Available from: http://www.pbsgridworks.com/Default.aspx
72. Laszewski GV, Gawor J, Lane P, Rehn N, Russell M, Jackson K. Features of the Java Commodity Grid Kit. Concurrency and Computation: Practice and Experience. 2002;14:1045-55.
73. Zhang X, Schopf J. Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2. Proceedings of the International Workshop on Middleware Performance (MP 2004), part of the 23rd International Performance Computing and Communications Workshop (IPCCC); 2004.
74. Novotny J, Tuecke S, Welch V. An Online Credential Repository for the Grid: MyProxy. the Tenth International Symposium on High Performance Distributed Computing (HPDC-10): IEEE Press; 2001.
75. Menezes A, Oorschot PV, Vanstone S, editors. Handbook of Applied Cryptography: CRC Press; 1996.
76. Butler R, Engert D, Foster I, Kesselman C, Tuecke S, Volmer J, et al. A National-Scale Authentication Infrastructure. IEEE Computer. 2000;33(12):60-6.
77. Introduction to JSR 168—The Java Portlet Specification. [cited; sun.developer]. Available from: http://developers.sun.com/portalserver/reference/techart/jsr168/pb_whitepaper.pdf
78. Shankar A. A General Introduction to (Grid) Portals/Gateways. [cited; Available from: http://dhruv.uits.indiana.edu/portals/portals-101.pdf
79. Sarkar B. Invoking Assembly Language Programs from Java. 2006 [cited; Available from: http://today.java.net/pub/a/today/2006/10/19/invoking-assembly-language-from-java.html
80. Rajasekar A, Wan M, Moore R, Schroeder W, Kremenek G, Jagatheesan A, et al. Storage Resource Broker - Managing Distributed Data in a Grid. Computer Society of India Journal. 2003;33(4):42-54.

81. Prodan R, Zenturio FT. A grid service-based tool for optimising parallel and grid applications. Journal of Grid Computing. 2004;2:15-29.
82. Niblett P, Graham S. Events and service-oriented architecture: The OASIS Web Services Notification specifications. IBM Systems Journal. 2005;44(4).