UNIVERSITY OF SOUTHAMPTON

FACULTY OF ENGINEERING, SCIENCE & MATHEMATICS

School of Engineering Sciences

Grid approaches to Data-Driven Scientific and Engineering Workflows

by

A.Paventhan

Thesis for the degree of Doctor of Philosophy

June 2007

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE & MATHEMATICS
SCHOOL OF ENGINEERING SCIENCES

Doctor of Philosophy

GRID APPROACHES TO DATA-DRIVEN SCIENTIFIC AND ENGINEERING WORKFLOWS

A.Paventhan

Enabling the full life cycle of scientific and engineering workflows requires robust middleware and services that support near-realtime data movement, high-performance processing and effective data management. In this context, we consider two related technology areas: *Grid computing* which is fast emerging as an accepted way forward for the large-scale, distributed and multi-institutional resource sharing and *Database systems* whose capabilities are undergoing continuous change providing new possibilities for scientific data management in Grid.

In this thesis, we look into the challenging requirements while integrating data-driven scientific and engineering experiment workflows onto Grid. We consider wind tunnels that houses multiple experiments with differing characteristics, as an application exemplar. This thesis contributes two approaches while attempting to tackle some of the following questions: *How to allow domain-specific workflow activity development by hiding the underlying complexity? Can new experiments be added to the system easily? How can the overall turnaround time be reduced by an end-to-end experimental workflow support?* In the first approach, we show how experiment-specific workflows can help accelerate application development using Grid services. This has been realized with the development of MyCoG, the first Commodity Grid toolkit for .NET supporting multi-language programmability. In the second, we present an alternative approach based on federated database services to realize an end-to-end experimental workflow. We show with the help of a real-world example, how database services can be building blocks for scientific and engineering workflows.

Contents

List of Tables

List of Figures

Declaration of Authorship

I, A.Paventhan ...., declare that the thesis entitled "Grid approaches to Data-Driven Scientific and Engineering Workflows" .... and the work presented in it are my own.

I confirm that:

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

Signed: ....

Date: ....

Acknowledgment

It has been a wonderful experience working with my academic supervisor Dr Kenji Takeda. I would like to thank Kenji for providing me the opportunity, guidance, all the technical discussions, necessary help, and especially pointing me in the right direction to the people and the resources.

I would like thank my other academic supervisor Professor Simon Cox for giving me the opportunity to work with the Microsoft Institute for High Performance Computing team and providing much needed constructive inputs during the course of my PhD. I am also thankful to other members of the Microsoft Institute of HPC team - Dr.Denis Nicole, Dr.Gang Xue, Dr.Matt Fairman, Dr.Marc Molinari and Mr.Marc Scott for the lively discussions.

I have found inputs from other members of Kenji's PhD team extremely valuable. My special thanks go to Dr.Alistair Brizell for sharing his expertise on Laser Doppler Anemometry setup, data acquisition and processing, and Mr.Ben Fenech for sharing his expertise on Microphone array setup, data acquisition and matlab processing code.

I would like to thank Dr.Neil Bressloff for all the systems support during my stay in CEDC. My thanks also go to Winston Lau, Sourish Banerjee and other CEDC colleagues for all the support.

I am grateful to Mr.Paul Bickmore for providing all the software resources and he has been of great help throughout the course of my PhD. I am also thankful to the ISS support staff for the help during the GridFTP test between Southampton and Manchester.

My parents have always been the source of encouragement. I would like to dedicate this thesis to them.

Chapter 1

Introduction

Large scale computational applications have benefited enormously from the continuous hardware innovations in terms of faster microprocessors, cheaper and mass storage devices, and high-speed networks. The operating system components, language compilers and other development tools, platform-neutral runtimes, database systems, among others, provide advanced software infrastructures. Until the mid 1990s the services based on these hardware and software resources were from within a single organizational unit. But, since late 1990s there has been a significant shift in the way the resources are managed to support large scale data and computation. This change in perception has been due to the emergence of "Grid" [1] for enabling multi-institutional resource sharing and problem solving.

The initial ideas of Grid computing started from the need to link multiple geographically distributed High Performance Computing (HPC) installations into a virtual supercomputer known as a metacomputer [2]. For example, the I-WAY [3] project linked 17 sites (comprising supercomputers, mass storage systems, advanced visualization devices and databases) into local environments via high-speed Asynchronous Transfer Mode (ATM) networks. The I-WAY project experience is a precursor to many of the ideas and developments in present day Grids.

The evolution of Grid from an interoperable computing infrastructure linked by high-speed networks into a service-oriented architecture based on Web Services standards has gone through different phases. It is difficult to confine Grid into a rigid definition. But a widely accepted and a popular one was provided by Foster et al., which defines Grid as *"coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations"* [4].

The Grid infrastructure is aimed at providing a distributed computing environment for effective utilization of both hardware and software resources. The Grid approach is gaining acceptance from the scientific community and it is making inroads into enterprise and business computing as well. However, the application requirements from different scientific disciplines bring different sets of computational and data management challenges. It is important that the Grid research community is presented with different scientific application scenarios and their unique requirements, and approaches to integrating them onto Grid to ensure that the appropriate tools and technologies are developed.

This thesis first contribution is in extending Globus Grid services reach to new platforms by the development of MyCoG, the first multi-language commodity Grid toolkit for .NET and providing a scientific workflow approach leveraging MyCoG. The second contribution is in developing an end-to-end scientific workflow approach entirely based on database services. In the application scenario we consider experimental aerodynamics, in which multiple users can carry out different aerodynamic experiments at geographically distributed wind tunnel facilities with differing data and processing characteristics.

There are many workflow principles used in business process automation that are relevant to scientific workflows. Business workflows tend to be routine whereas scientific workflows are often exploratory, sometimes following a trial-and-error approach. The amount of data that are to be processed at each stage of the workflow are often greater in scientific workflows than business workflows. The Grid security policies required in a collaborative and resource sharing scientific environment are different from business environments. These differences often to lead to either customization of commercial workflow engines or development of new workflow engines tailored to scientific workflows.

Scientific workflow solutions can be grouped into two categories: 1) Workflow systems motivated by domain-specific requirements. 2) Generic workflow systems. The extensibility of a domain-specific workflow solution to another application domain is limited as it does not address the other sets of requirements. On the other hand, a generic workflow solution would require further customization to be applicable to a particular application domain. In specialized engineering facilities such as wind tunnels, researchers would be conducting several experiments with differing characteristics in terms of configuration settings, data size, data format, processing requirements and so on. In this scenario, it is important for a workflow approach to abstract the general requirements so that new experiments can easily be supported by means of extensibility and at the same time provide experiment-specific activities supporting customized experimental workflow development. This customized approach to workflow can help reduce the time taken for a complete workflow by automating data flow driven activities, supplementing or replacing manual user-driven steps.

In our first approach, we present a grid workflow architecture that supports wind tunnel users (engineers, researchers and industrial users) to compose sequential workflows and access Globus Grid services seamlessly. We show how customized workflows from experiment-specific activities, that would hide the underlying complexities in Grid access, can be developed and extended by users using their preferred programming languages. This has been achieved by the development of MyCoG.NET, the first Commodity Grid toolkit to leverage .NET platform and support multi-language programmability.

The capabilities of Relational Database Management Systems (RDBMS) from native XML support and procedural language stored procedures to publish/subscribe replication among others are increasing and their architectures are undergoing continuous change. The development of such new capabilities are driven by the business market and it has the potential to enable new approaches to scientific data management in Grid environment. We present an alternative approach to the previous one based on federated database services to realize an end-to-end experiment workflow. Both the experiment data and user algorithms for processing are managed inside federated databases. The individual database instances are autonomous and their interactions are by means of transactional replication and asynchronous messaging.

Even though, the above two approaches have been demonstrated with an experimental application example, their extension to computational simulations are possible.

1.1 Integrating Experiments onto Grid

Scientists and engineers conducting experiments often perform a sequence of tasks in a workflow pattern similar to that of a business process. The individual steps in business workflow systems are typically control flow driven or state driven, whereas scientific workflows may also be data and event driven. In a simplistic scenario, the steps in an experimental workflow might include setup, data acquisition, data movement, pre-processing, processing and visualization. The data acquisition instruments, storage systems and compute resources are often distributed within and across organizational boundaries. The user may have to deal manually with the complexity of the resource discovery, data movement and job scheduling, impacting the overall turnaround time and reducing time to insight.

The requirements on Grid to realize experimental workflows from different scientific domain are unique as their data, processing and functional characteristics vary to greater extent. At one end of the spectrum, experiments such as the Large Hadron Collider [5] produce large volumes of data at one location to be distributed to several sites for processing. At the other extreme, the wind tunnel process (our application example) is characterized by multiple experiments, different locations, changing parameters for each run, varying data formats and customized processing. Also, the approach taken by individual application areas are largely driven by their core functional requirement. For example, in the Network for Earthquake Engineering and Simulations grid (NEESgrid) [6] the important functional requirement is remote steering of experiments and hybrid testing, the myLEAD system in Linked Environments for Atmospheric Discovery (LEAD) [7] project is expected to respond to changing weather conditions in realtime, the engine fault diagnosis and prognosis requires large-scale data mining in Distributed Aircraft Maintenance Environment (DAME) [8] system, the emphasis in the UK e-Science project myGrid [9] is resource discovery and workflow enactment, and the Sloan Digital Sky Server [10] requires to implement effective search algorithms to locate cluster galaxies. The differing application characteristics is further discussed in section 2.5.

Some of the major challenges to be tackled while integrating scientific and engineering experiments onto Grid are:

Figure 1.1: Airbus high-lift wing test at University of Southampton $7' \times 5'$ wind tunnel

- Usability by scientists and engineers
- Reliable and timely data movement from an experiment site to storage and processing clusters
- Application-specific approach to managing experiment parameters, data management, processing and analysis support
- Providing frameworks supporting entire experimental workflows including data acquisition systems integration
- Managing disparate experimental data sources and formats (be it image data, numbers or text)

1.1.1 Application Example: Wind tunnels

In this thesis, the wind tunnel experiments (Figure 1.1) are considered as an application example to demonstrate our approaches. Wind tunnels are widely used to design, test and verify aerodynamics of aircraft, cars, yachts, and buildings, amongst others. The University of Southampton has three large wind tunnel facilities [11] (with working section measuring 11' $\times$ 8', 7' $\times$ 5' and 3' $\times$ 2') spread over the campus, housing specialized experiment hardware and software for academic and industrial research. These facilities are often used by researchers and customers from other organizations. The pre-test planning and post-test analysis can be done either on-site, or off-site at user's base location. This creates the need for virtual organization support within any Grid system in which the user may wish to aggregate their own application software with those at the wind tunnel site. Hence any workflow system should be able to support creation and execution of Grid applications in a multi-site/organization scenario.

The wind tunnel data generated during acquisition vary in terms of the number of data items, file size and format, depending on the experiment and user parameters. For example, the Laser Doppler Anemometry (LDA) experiment generate from hundreds to thousands of files each few kilo bytes in size whereas every run of the microphone array experiment generate thousands of files each several mega bytes in size. A day of testing consists of hundreds of runs. In many experiments, the data movement operations to the processing computer are manual due to interoperability issues between hardware, software and the acquisition systems. The nature of the processing algorithms also differ − LDA is a short-running sequential code while the microphone array processing is data intensive and can be run in parallel.

Wind tunnel operations are an interesting experimental workflow example (see detailed discussions in section 2.5), in that there are many wind tunnel experiments with differing data and processing characteristics. But, the general approach towards realizing the experiment workflow can be applied in other domains as well. For example, in our approach, the idea of having common grid workflow activities for data transfer and processing and annotating with metadata for specific experiments can be easily applied to other application areas.

1.2 Research hypotheses and methodology

We propose the following research hypotheses that form the basis of this thesis and develop workflow approaches to substantiate them in the following chapters.

- Customized and domain-specific workflow approach to scientific workflows may lead to rapid application development which is often associated with business workflows.
- The integration of Commercial Off-the-shelf (COTS) development frameworks into Grid ecosystem may provide a suitable environment for the scientific workflows.
- The recent database capabilities may provide the necessary building blocks for developing end-to-end data-driven scientific workflows.

The methodologies we adopt to the workflow approaches that support the above hypotheses are based on the wind tunnel experiments application example described in the previous section. In Chapter 3, we will present the requirements of three wind tunnel experiments to substantiate the first hypothesis. We will show how domain-specific customizations on data transfer and processing stages of experiment workflows are useful. This will be demonstrated further in Chapter 4 by means of a hierarchical model for workflow activities supporting ready-to-use customized activities for different experiments. The wind tunnel experiment workflows were presented in our publication items 3, 5 and 6 of Appendix C.

In support of the second hypothesis, we will describe Globus Grid services workflow approach in Chapter 4. We will present the design and development of MyCoG.NET that enables Globus Grid service access from many languages supported by the .NET Common Language Runtime. We will also show how Windows Workflow Foundation, a Commodity Off-the-shelf workflow component of .NET can be used with MyCoG Grid interfaces to support scientific workflows based on Globus Grid services. The design and development of MyCoG was reported in our publication items 1 and 4 of Appendix C.

Chapter 5 will provide support for the third hypothesis by presenting a workflow approach based on database services. Considering the microphone array experiment as a case study, we will demonstrate how federated database services can form the basis for workflow activities to support entire workflow steps - data import, data transfer, processing and visualization. This workflow approach was presented in our publication item 2 of Appendix C.

1.3 Scope of Work

There are many challenges to be addressed while integrating data-driven scientific and engineering experiment workflows onto the Grid. It is more profound when scientific workflows involve multiple experiments with differing characteristics as in wind tunnel experiments. Even though our workflow example is from the field of experimental aerodynamics, the approach can be easily adopted to scientific workflows from other domains as well. We take an approach that abstracts the general requirements and at the same time provides experiment-specific extensions. In the following sections, we will introduce the two workflow approaches that form the key contributions of this thesis.

1.3.1 Workflow Integration based on Globus Grid Services

Globus middleware is being increasingly utilized by many scientific and engineering applications for Grid integration. The Globus Toolkit [12] provides software infrastructure supporting Grid security, resource management, data management, monitoring and discovery. Commodity Grid Toolkits (CoG) provide sets of classes and APIs to access Globus Grid services from already existing rich development frameworks (e.g, Java and .NET). In order to fully realize the benefits of Grid computing within the existing user community it is important to provide toolsets that are usable by scientists, engineers and business [13]. CoG toolkits play a vital role in achieving this. The language-specific nature of existing CoG toolkits, however, restricts their use to a small number of languages (e.g. Java, Python, Perl). Many languages favored by scientists and engineers, such as FORTRAN, are not supported and the prospect of porting CoG toolkits to other languages require significant effort.

How do we hide the complexity of Grid service access and provide interfaces in their favorite programming languages? How would we allow experiment-specific workflow activities development which is essential for rapid application development?.

In this approach we tackle the above two questions. We first enable the Globus

Grid services access from an existing workflow framework and next is to leverage the workflow framework to address issues in integrating experimental workflow.

We present the user with MyCoG.NET, the first CoG toolkit targeted at the .NET Common Language Runtime (CLR) to access Globus Grid services. User will be able to program in many different .NET languages including, FORTRAN, C^{++}, $C^{\#}$ and Java. The user will also have the flexibility of mixed language programming. For example, one can develop a Grid application by writing graphical user interfaces in C^{++} and scientific computations in FORTRAN, based on coding preferences and/or language strengths. Further, the easy Grid integration of many legacy scientific application code written languages such as FORTRAN becomes possible.

By bringing Globus Grid service access to .NET, we are able to leverage Windows Workflow Foundation part of Microsoft .NET 3.0 [14] for scientific applications. Windows Workflow Foundation has predefined sets of activities and we extend them to provide Grid activities and experiment-specific workflow activities for application development.

The wind tunnel user can compose their experimental workflow based on Globus Grid services using the following workflow activities.

- Globus Grid activities *(e.g, GridFTP, GRAM, MDS)*
- Experiment-specific activities *(e.g, LDAUpload, LDAProcess)*
- Base Windows Workflow Foundation activities *(e.g, IfElse, While, Parallel, InvokeWebService and so on).*
- or any combination of the above

The benefits derived from the above customized approach to wind tunnel application workflow are:

- Reduction in the overall turnaround time to run an experimental workflow
- Rapid application development leveraging existing business workflow efforts
- Ability to focus on the problem at hand rather than worrying about the underlying complexity in Grid resource access

1.3.2 Workflow Integration based on Federated Database Services

The majority of scientific applications in the Grid rely on file systems for data management, with very limited use of Relational Database Management Systems (RDBMS). The RDBMS is often used as a query engine to retrieve metadata and/or results.

With computational applications becoming data-centric, providing effective storage and access to data are important [15]. File systems may provide better raw read/write performance as compared to database systems, but there are additional benefits database systems could bring in terms of transactions support to guarantee data integrity, query language, data security and other ever growing features such as high-level language functions and stored procedures, native XML types and web services, transactional messaging, publish/subscribe replication, data mining extensions and so on. Considering the changing database systems landscape, they may now be viewed as *database operating systems* [16] into which one can plug subsystems and applications. The issues relating to database integration into a Grid environment has been studied in [17]. In this thesis, by means of our application example, we highlight how database systems' features and its integration can benefit Grid applications and identify improvement areas.

When experiments are conducted at multiple sites, the data are to be moved from sites for archival and processing purposes. *Can the data movement operations from the site be handled asynchronously providing local autonomy to the site? Can new experiments be added to the system easily? Can the overall turnaround time be reduced for the entire experimental workflow?* In an experimental setup similar to wind tunnels, very often, the user runs different versions of processing code against the data. *How do we allow a user to maintain different versions of the processing code to run against the data?*

In our second approach, we address the above questions while supporting an end-to-end experiment workflow using federated database services. Database federation can help heterogeneous data produced at different geographical locations to be managed, and provides the user with a single logical view. The individual database instances in the federation are autonomous and any of them temporarily being unavailable does not affect their interactions. Although database federation as an approach to data integration [18] can support functions like query optimization, the issue we address in this approach is geographical separation of data sources, be it within campus or across organizations. With database systems supporting both X.509 certificate based authentication/authorization and XML Web Services security standards, the Grid systems security model between multiple virtual organizations can be realized.

The master database instance, together with a set of worker database instances create a federation responsible for the storage, access and processing of the data. Each experiment site maintains a local site database and publishes the data for which the master has subscribed. When a user completes an experiment, the raw data, together with parameters associated with the experiment, are imported at the site. Since push-style transactional replication is employed, the data is propagated to the master in near-realtime. The user is able to register their custom processing code for a particular application and maintain different versions of them. Both the application code and the data are managed inside the database, making the code run closer to data. We also show how an experimental workflow can be constructed with workflow activities leveraging database functionalities. Additional services to query, analyze and visualize the results are available.

1.4 Outline of this Thesis

The rest of the thesis is organized as follows.

In Chapter 2, we review related works in the context of this thesis in the areas of Grid, data management, workflows, and application examples from different domains.

In Chapter 3, we discuss the requirements for integrating scientific and engineering experiments onto Grid environment looking into a few specific wind tunnel experiments.

In Chapter 4, we present the architecture and implementation details of MyCoG, Commodity Grid toolkit for the .NET platform and our first approach to scientific workflow integration based on Grid services.

In Chapter 5, we show how database systems features can be exploited by presenting an alternate approach to support end-to-end scientific workflows based on federated databases.

In Chapter 6, discussions and further work are presented.

In Chapter 7, we present the thesis summary and conclusions.

Chapter 2

Data Grids and Workflows

During the last decade there has been a growing interest in Grid computing influencing the way large-scale scientific and engineering research is conducted from a tightly coupled computing environment to a geographically distributed resource sharing approach. Scientists continue to use High Performance Computing (HPC) systems such as cluster of shared-memory multiprocessors, parallel vector machines and cluster of workstations connected by high-speed communication stack (low-latency and high-bandwidth networks, OS-bypass protocols and message passing software libraries). With scientific projects becoming increasingly collaborative, and people and the resources drawn from multiple organizations, Grid computing has become the next step in the evolution of HPC. The collaborative effort at the national and international level resulted in building Grid infrastructures such as NSF's Tera Grid in the USA, e-Science Grid in the UK, European Union DataGrid, Large Hadron Collider (LHC) Computing Grid at CERN in Geneva and Asia Pacific Grids. Grid infrastructures are being utilized by many application areas, ranging from high-energy physics, life sciences and geosciences to astronomy, earthquake engineering experiments and aerospace engineering. The characteristics of these applications are also wide ranging with some being *data-intensive*, some are *compute-intensive* and some requiring *remote steering* of experiments. This makes the development of Grid middleware services supporting data movement, data access, processing and workflow support for a given application domain challenging.

In this thesis, we address the challenges in integrating experimental workflows onto Grid by providing two approaches considering wind tunnel experiments as an application exemplar. In the remainder of this chapter we will discuss the related works in the context of this thesis with Grid technologies in section 2.1, XML Web

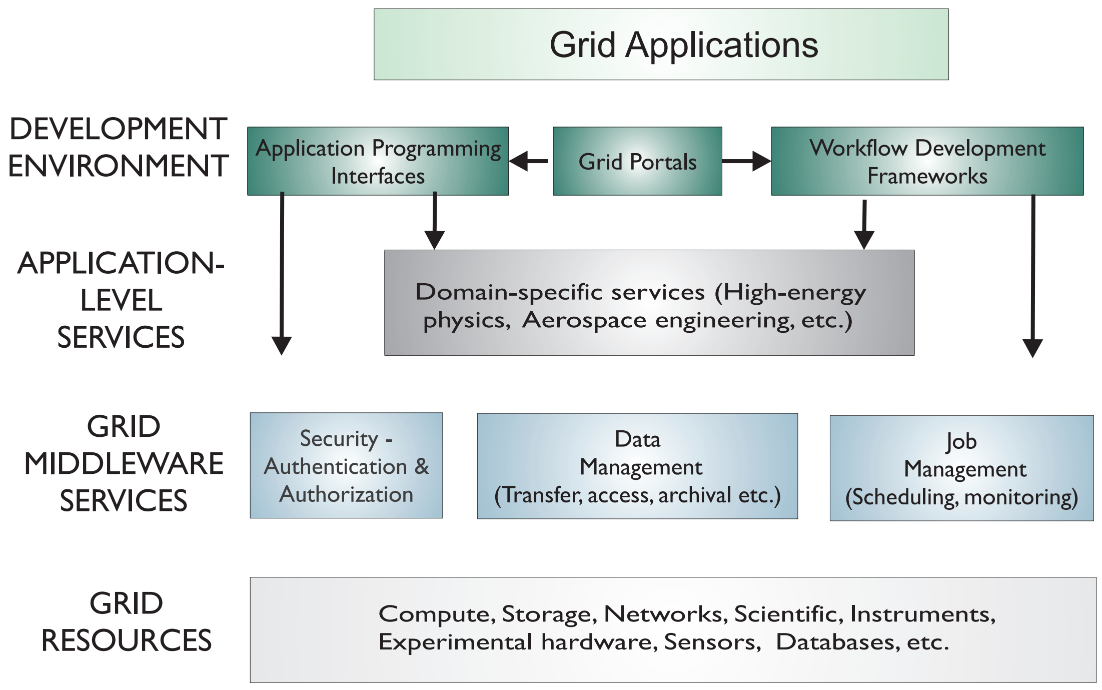

Grid Applications
DEVELOPMENT ENVIRONMENT
Application Programming Interfaces
Grid Portals
Workflow Development Frameworks
APPLICATION-LEVEL SERVICES
Domain-specific services (High-energy physics, Aerospace engineering, etc.)
GRID MIDDLEWARE SERVICES
Security - Authentication & Authorization
Data Management (Transfer, access, archival etc.)
Job Management (Scheduling, monitoring)
GRID RESOURCES
Compute, Storage, Networks, Scientific, Instruments, Experimental hardware, Sensors, Databases, etc.

Figure 2.1: Grid Architecture

Services in section 2.2, data management in section 2.3, workflow systems in section 2.4 and application examples in section 2.5.

2.1 Grid – Concepts and Technologies

The term *"Grid"* has been widely used since the late 1990s to refer to distributed resource access and sharing, analogous to an electric power grid, for advanced scientific collaborations. The essential characteristic of a typical Grid infrastructure is to support flexible, secure, coordinated resource sharing among virtual organizations [4].

Figure 2.1 shows the set of layers of services in a typical Grid architecture that we utilize for our discussion. The Grid resources are geographically distributed and they could be compute clusters, storage systems, networking hardware, scientific instruments, data acquisition systems amongst others. The Grid Middleware layer provides services ranging from authentication, instrument integration, data acquisition, data transfer and data archival to job scheduling and resource monitoring. This layer hides the underlying complexities in resource access for the upper layers while participating in a *virtual organization.* The application layer services are customized for a particular domain, such as experimental aerodynamics, which is the case-study considered in this thesis. The development environment may consist of APIs, specialized Grid portals and workflow frameworks supporting programmable interfaces for application development.

2.1.1 Globus

Globus has emerged as a *de facto* middleware standard for setting up computational and data Grids [19]. The Globus Toolkit version 2 (GT2) is based on Internet standard protocols while its current release GT4 leverages service-oriented architecture and XML Web services. Its components provide software infrastructure supporting Grid security, resource management, data management, monitoring and discovery.

2.1.1.1 Security

GT4 supports WS-Security and WS-SecureConversation to provide *Message-Level* security. But, GT4 services use *Transport-Level* security (TLS[1]) as a default security mechanism. This is due to the poor performance associated with *Message-Level* security [20]. *Authorization framework* allows authorization schemes (grid-mapfile or access control list service) and is based on Security Assertion Markup Language (SAML). The pre-WS components security part of GT2 is based on Grid Security Infrastructure (GSI). GSI supports X.509 certificate based mutual authentication between Grid service instances, dynamic creation of proxy-credentials and single sign-on. GSI conforms to Generic Security Services API (GSSAPI) [21] and uses Secure Socket Layer (SSL) protocol for authentication and message protection.

2.1.1.2 Data Management

GridFTP: Scientific applications in a Grid environment require secured and efficient movement of large amounts of data. GridFTP [22] extends the standard FTP protocol to support features such as GSI security, multiple data paths for parallel transfer, striping, third-party transfer, partial file transfer, data channel protection,

[1]TLS protocol is an IETF standard of GSSAPI

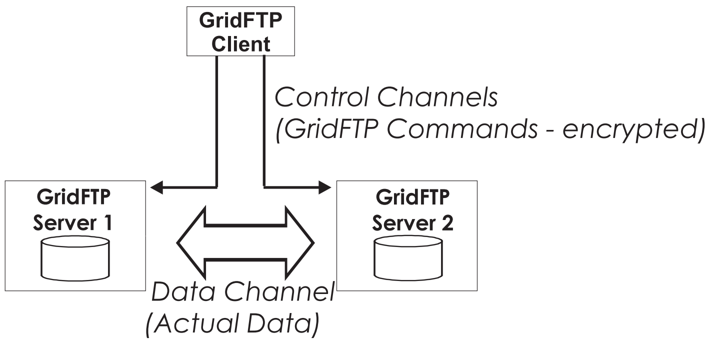


Figure 2.2: GridFTP - third-party file transfer

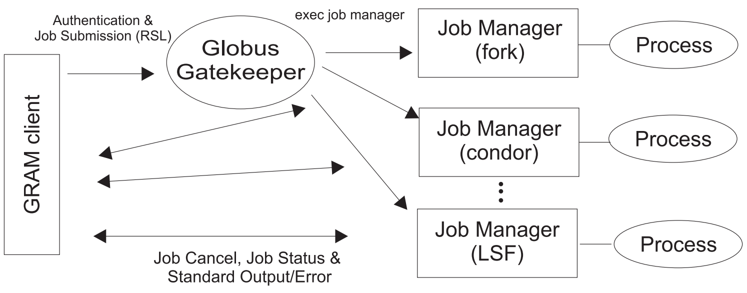


Figure 2.3: Globus GRAM Job Submission

TCP buffer size tuning and restartable transfer. The control channel is GSI authenticated and the GridFTP commands are exchanged through this securely. Since the control channel and data channel are separate network connections, the data transfer between two servers can be controlled from an intermediate machine as shown in Figure 2.2. This feature is known as third-party transfer in GridFTP.

RFT: Reliable File Transfer (RFT) [23] service provides an XML Web service interface, supporting multi-file transfers using the GridFTP third-party transfer mechanism. The request to RFT service consists of an array of source and destination URLs of the file locations. RFT service makes the control channel connection to the source and destination GridFTP servers on user's behalf, and performs the transfer. It stores the requests in the database so that the transfers can be restarted in the event of failure.

2.1.1.3 Resource Management

The Globus Resource Allocation Manager (GRAM) provides a standard interface for executing jobs on remote systems. The gatekeeper component of the GRAM service is responsible for GSI authentication of the user, parsing the job request written in Resource Specification Language and starting the appropriate job manager instance as shown in Figure 2.3. The job manager plug-in interface allows different local schedulers such as LSF, Condor or based on the *fork* system call. The job manager executes the user's job, handles job management requests from the user, provides a callback mechanism to communicate job state, and supports transfer of *stdout* and *stderr*. GRAM services are supported using both Web services based framework and pre-Web services approach in Globus Toolkit [24].

2.1.1.4 Monitoring and Discovery:

The Monitoring and Discovery Service (MDS) provides mechanisms for publishing and discovering resource status and configuration information. MDS in GT2 is a pre-Web services component based on LDAP (Lightweight Directory Access Protocol) and MDS4 is based on Web Services [25].

2.1.2 Other Grid Middlewares

The *gLite* middleware [26] development activity under Enabling Grids for E-sciencE (EGEE) programme involves many European countries. It provides five different sets of services to applications. They are *security services* supporting authentication, authorization and auditing, *information & monitoring services*, *data services* for metadata catalog, storage management and replica catalog, *job management services* supporting package management, job provenance, workload management and computing elements, and *Grid access services and APIs*. The Grid services in *gLite* follows a service oriented approach in order to achieve interoperability.

The UNICORE (UNiform Interface to COmputer REsources) [27] project was initiated in Germany in 1997 and was aimed at supporting the operations at supercomputing centers. It has now evolved into an open source Grid middleware and is being utilized in many European projects. The UNICORE architecture consists of user, server and target system tiers. The user tier provides interfaces to communicate to the server tier via the UNICORE Protocol Layer (UPL) based on SSL. The server tier gateway authenticates UPL requests and provides access to the site resources. The target tier interfaces with the local resource manager (Condor, LSF, PBS, etc.) at the site for actual scheduling.

The Open Middleware Infrastructure Institute (OMII) [28] part of the current UK e-Science Core programme will manage the release of open source Grid middleware tools supporting workflows, job submission & monitoring, reliable messaging, data access and integration services.

2.1.3 Grid Application Programming Interfaces

In order to fully realize the benefits of Grid computing within the existing user community, it is important to provide toolsets that are usable by scientists, engineers and business [13]. Commodity Grid (CoG) toolkits play an important role in achieving this objective. CoG toolkits provide a set of classes and APIs that enable access to Grid services from a commodity development environment/framework. The advantages Grid applications could derive from commodity environments such as .NET and Java, include platform neutral runtime, web-based deployment models, access to rich class libraries, web services tooling among others.

The Java CoG toolkit [29] part of the Globus project provides interfaces to access Globus Grid services. It comes with an exhaustive set of packages and classes supporting Grid security, GRAM, GridFTP, and MDS services. The language-specific nature of existing CoG toolkits, however, restricts their use to a small number of languages (e.g. Java [29], Python [30], Perl [31]). Many languages favoured by scientists, engineers and business end-users are not supported, such as FORTRAN, and the prospect of porting CoG toolkits to specialist user communities (e.g. Ada, Eiffel) is unlikely. A multi-language approach to CoG toolkits would enable a significantly larger community of end-users, across many sectors, to exploit the Grid. The architecture and implementation details of such an approach to access GT2 Grid services from .NET is presented and discussed in Chapter 4.

The Simple API for Grid Applications (SAGA) [32] research group within the Global Grid Forum released an API specification supporting several Grid functional requirements. Grid applications that are written using the SAGA API will be portable to run in a middleware-independent manner. The SAGA specification covers requirements derived from many application use-cases and builds on experiences from other projects such as CoG. Different language implementations (Java, C++, C, etc.) of the SAGA API complying with the specification are being developed.

2.1.4 Emerging Grid standards

The Global Grid Forum (after the merger with European Grid Alliance, now known as Open Grid Forum [33]) is a non-profit organization that consists of users, leading vendors, developers and researchers working in the standardization effort for Grid computing.

The basis for the standards emerging from GGF is a Grid service leveraging XML Web Services. A Grid service is a *stateful* Web Service that provides a set of well-defined interfaces and follows specific conventions [34]. All physical and logical resources that are part of the Grid can be modeled as a Grid service.

2.1.4.1 Web Services Resources Framework (WSRF)

The Open Grid Services Infrastructure (OGSI) [35] specification defines a mechanism for creating, managing and exchanging information among Grid Service entities by extending Web Services Description Language (WSDL) and XML Schema.

The OGSI document evolved into six independent WSRF specifications: WS-Resource Properties, WS-Resource Lifetime, WS-Notification, WS-Renewable References, WS-Service Group and WS-Base Fault. This set of specifications support modeling and managing state in a Web Services context. Although WSRF is now an OASIS (Organization for the Advancement of Structured Information Standards) standard [36], its adoption by industry, interoperability between different implementations etc. are still to be achieved, at the time of this writing. As can be seen from the discussion in section 2.2, there are competing and similar specifications to WSRF, and the industry is working towards a common Web Services Distributed Management (WSDM) family of specifications.

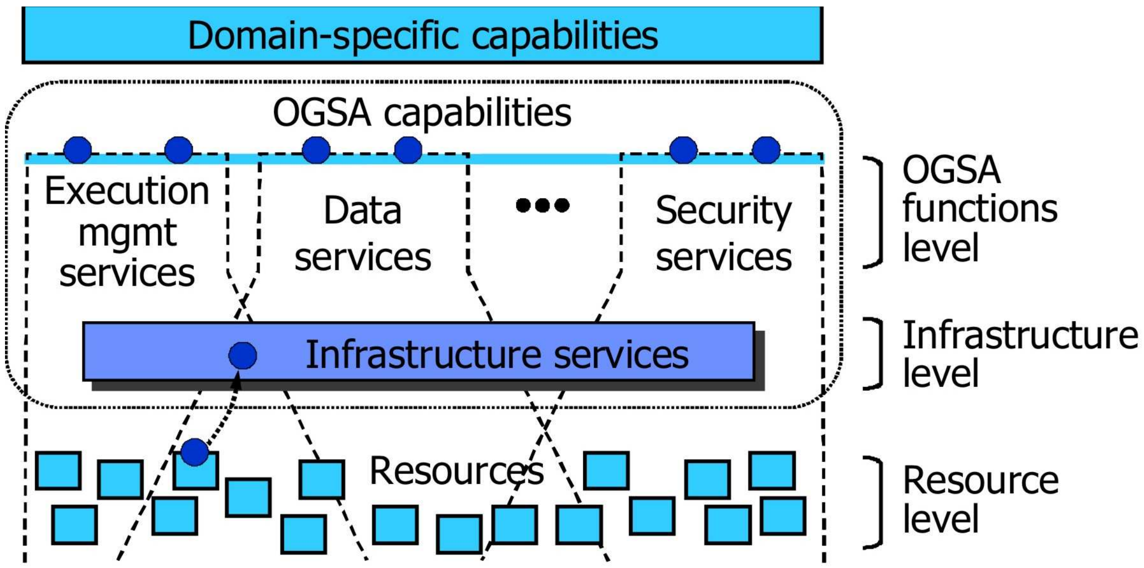


Figure 2.4: Levels of Management in OGSA [37]

2.1.4.2 Open Grid Service Architecture

The vision of Open Grid Services Architecture (OGSA) [37] is to provide a Web Services based platform to support seamless use and management of distributed, heterogeneous resources. The resources in Figure 2.4 represent the physical resources like CPUs, memory, disk etc. The OGSA services and the interactions among themselves, the interfaces they expose to the applications are all addressed at the core of the OGSA stack. The basis for OGSA Grid services is infrastructure services (also known as the Grid fabric) and their interfaces conform to specifications such as WSRF and WSDM. The higher level OGSA services include execution management services, data services, security services, resource management services and they are implemented using functional interfaces provided by the infrastructure services. The domain-specific implementations use underlying OGSA services to achieve applications functionality.

2.2 XML Web Services

XML Web services provide building blocks (Figure 2.5) [38] for developing distributed applications written in different languages on different platforms hosted by different organizations to communicate with each other in standards-based way.

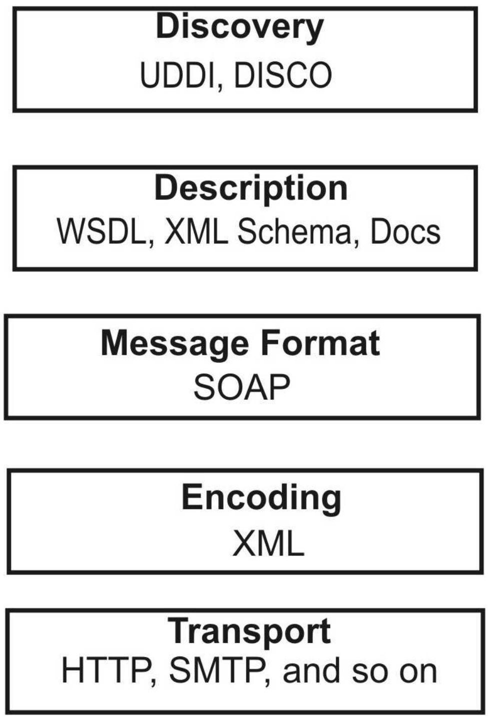


Figure 2.5: Web services building blocks

Protocols: The client application intending to access a Web service can discover it using Universal Description, Discovery and Integration (UDDI). The Web service endpoint and its behavior (operations) is described using Web Services Description Language (WSDL). In WSDL an abstract set of operations supported by one or more endpoints is designated as a *portType*. XML Schema provide a core set of built-in datatypes that can be used to describe the contents of the messages. Simple Object Access Protocol (SOAP) defines a standard format for serializing data into XML Messages that can be exchanged between peers. The messages encoded using Extensible Markup Language (XML) is transported over Hypertext Transport Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP). XML, XML Schema, SOAP and WSDL are specifications under the standards body World Wide Web Consortium (W3C) [39]. UDDI is ratified by OASIS.

Interoperability: There are many XML Web Service specifications (WS-*) defined on top of core XML and SOAP standards to ensure application level interoperability between different vendor platforms. For example, WS-Security [40] ratified by OASIS, describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. Additionally, WS-Security describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys. The different WS-Security implementations of Microsoft .NET Web Services Enhancements (WSE2.0), IBM WebSphere development environment and Sun J2EE will all be able to interoperate.

Tools: The .NET Framework *xsd* utility can produce high-level language classes ($C^{++}/C^{\#}$) from an XML Schema definition so that at runtime XML instances can be mapped to .NET objects. Another utility *wsdl* can generate proxy classes (client-side) or stub classes (server-side) from a WSDL definition. Similar tools are available on other development environments.

The Grid research community has leveraged XML Web Services for hosting Grid services. Despite the advantages of XML Web Services in being platform neutral, interoperable and in their ability to penetrate firewalls, there are many issues such as performance overheads still to be addressed. The majority of the Web Service implementations use HTTP (theoretically other transport protocols can be used) for their transport and due to the processing overhead associated with the Web Service pipeline, performance is significantly affected. In the Web Services model, the GridFTP data movement operations are only realized as third-party server-to-server transfers [23]. Although in reality, there are requirements for client-server style transfers. Also, due to competing specifications proposed by different vendors, many interoperability issues still remain in the Web Services world. For instance, WS-Transfer, WS-Eventing and WS-Management standards driven by Microsoft and backed by IBM, Sun and Intel are functionally similar to WSRF (see section 2.1.4.1). In the context of our work, Web Services are more suitable for remote access and monitoring, and its usage at the local wind tunnel site is limited due to near-realtime data movement and processing requirements.

2.3 Data Management in Grid

The sources of large quantities of scientific data range from sensors and advanced scientific experiments to long running computational simulations. The Grid research community is making a concerted effort in providing efficient storage and access frameworks to scientific data and associated metadata repositories. Further, the extraction of knowledge from geographically distributed data sources are also gaining importance.

2.3.1 Data Access and Integration

OGSA-DAI (OGSA Data Access and Integration) [41] provides a service oriented architecture to access data sources such as relational or XML databases. Some of the key features and motivations behind the OGSA-DAI project include:

- Providing uniform service-based interfaces to access multiple database management systems (MySQL, Oracle, DB2 etc). This approach contrasts with language-based solutions such as Java database connectivity (JDBC) by providing standard Web Services-based interfaces.

- Supporting document oriented interface in which a single document (known as *perform* document) may contain group of activities, for example, to update the database, query and deliver the result. This reduces the number of messages being exchanged between the consumer and the OGSA-DAI Grid Data Service (GDS). Each activity in the perform document is executed by an associated Java class.

- Supporting asynchronous requests where the query results are sent directly to a third-party (*DeliverTo* activity) for consumption as opposed to JDBC style synchronous connectivity used in client-server models. The delivery location for the result could be a GridFTP service, another OGSA-DAI service or a file location specified by an URL. Similarly, input for a query can come from these delivery locations using *DeliverFrom* activity.

We utilize Figure 2.6 to show how our database centric approach discussed in Chapter 5 is complementary to the OGSA-DAI work. In this model, OGSA-DAI services are responsible for data access, management and delivery while the other grid middleware services are the consumers of its data and responsible for long-running computations. In our approach end-to-end workflow activities are supported

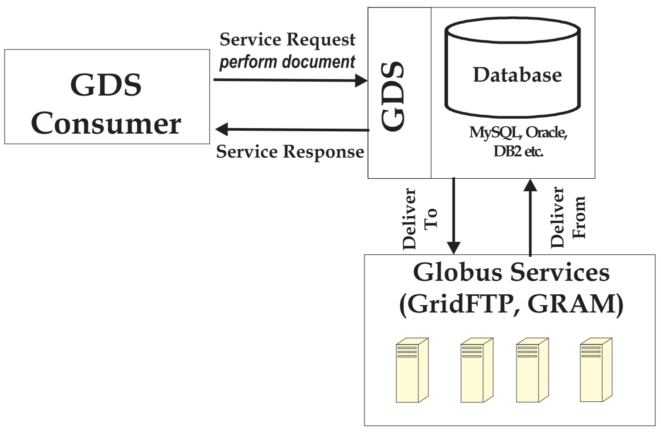


Figure 2.6: OGSA-DAI data delivery

based on federated database services responsible for data import, data transfer, computations and visualization. Both structured metadata (configurations, results etc) and raw experimental data are stored together in the database. The message exchanges between database instances are transactional as their interactions are based on SQL constructs. As XML Web Services are supported natively by many popular database systems, it is possible to wrap the database activities as Grid fabric services (based on WSRF, WSDM etc.) for Grid integration. Already OGSA-DAI interfaces are being redefined to a Web Services-Data Access and Integration (WS-DAI) [42] family of specifications and hence it would become possible in our approach to expose domain-specific WS-DAI services from database instances to provide access to structured data.

2.3.2 Other Data Grid Approaches

Storage Resource Broker (SRB) [43] is a middleware that combines different physical storage resources to provide seamless access to data sets. The three major components of SRB include metadata catalog (MCAT), SRB servers and SRB clients. The MCAT records are stored in an RDBMS. The SRB server does a logical to physical mapping by consulting MCAT to support multiple physical data store, replication and striping.

Grid-DB [44] is a data centric grid workflow system. Grid-DB submits the programs to grid middleware to execute in a compute cluster and provides the result to the user. It is discussed further in section 2.4.2.

In the case of both SRB and GridDB the computations are treated separately. Whereas the approach we discuss in Chapter 5 is different in that the code is run within the database system (supporting data parallel execution), the data movement is an integral part of the database and support for local autonomy for the experimental site data are explicitly addressed due to the nature of the application.

2.3.3 Knowledge Grid

The Knowledge Discovery in Databases (KDD) and data mining process can be logically grouped into four separate steps [45]: *data selection, data cleaning, data mining* and *evaluation*. The data mining step is used to extract interesting patterns from data using algorithms such as association rules, decision trees, clustering and similarity searches. The SQL/MM data mining extension of the SQL:1999 standard defines standard interfaces to data mining algorithms. IBM DB2's Intelligent Miner and Microsoft SQL Server's Analysis Services support these data mining extensions.

The Knowledge Grid uses basic Grid services to implement high-level distributed discovery services in order to mine data stored at multiple administrative domains. Some of the data mining efforts in Grid research include developing a knowledge Grid layer over Globus Grid services [46], knowledge flow networks in e-science [47] and data mining grid activities [48]. As the experimental raw data, metadata and results are all stored in database systems in our approach discussed in Chapter 5, it is possible to leverage the SQL/MM data mining extensions.

2.4 Workflow Systems

In this section, we will discuss workflow systems with specific emphasis on how business workflows compare with scientific workflow systems. We will also introduce Windows Workflow Foundation as one example of how a commercial workflow engine can offer an attractive execution environment for scientific workflows.

The Workflow Management Coalition (WfMC), with members from industries, formally defined workflow as "the automation of procedures where documents, information or tasks are passed between participants according to a defined set of

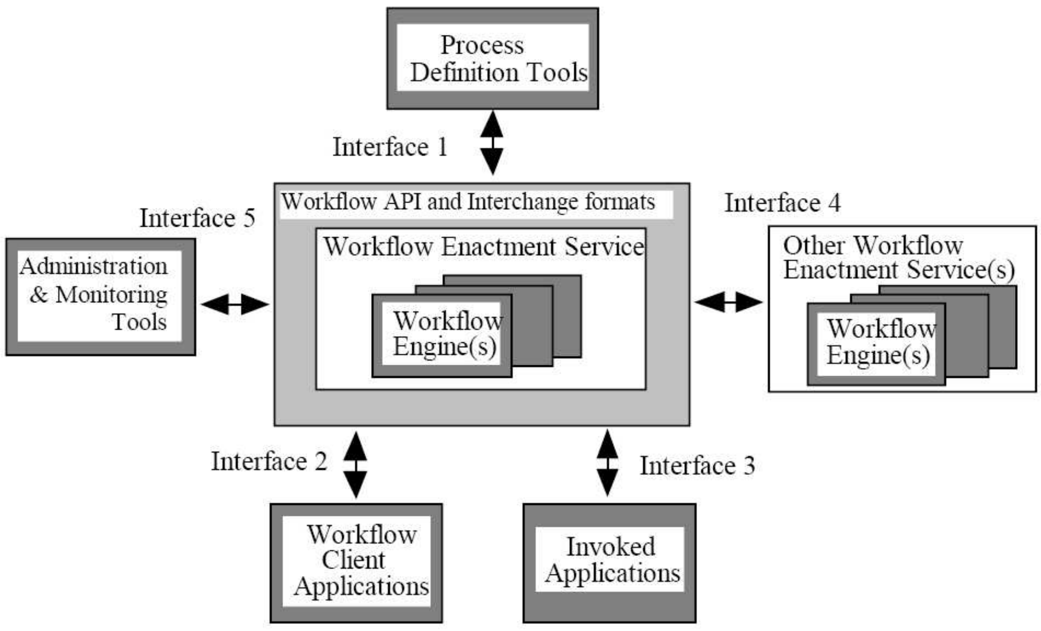

Process Definition Tools
Interface 1
Workflow API and Interchange formats
Workflow Enactment Service
Workflow Engine(s)
Interface 5
Administration & Monitoring Tools
Interface 4
Other Workflow Enactment Service(s)
Workflow Engine(s)
Interface 2
Interface 3
Workflow Client Applications
Invoked Applications

Figure 2.7: Workflow Reference Model © WfMC [49]

rules to achieve, or contribute to, an overall business goal" [49]. Towards realizing interoperable workflow systems, WfMC works on reference standards and specifications suggestive to vendors of workflow products. The workflow reference model shown in Figure 2.7 identifies the major components and their interactions. Central to this model is the workflow enactment service that may consist of one or more workflow engines in order to create, manage and execute workflow instances. The process definition tools help in creating workflow type definitions, business process activities, navigation rules, transition conditions, application invocations etc. The process definitions submitted to the enactment service are then interpreted at run-time by workflow engines, causing workflow instances to change state in response to external events or specific control decisions. The administration and monitoring interfaces support management functions such as process status monitoring, resource control operations, user/role management functions etc. Also the key objective of this reference model is to allow workflow activities from different vendor systems to interoperate. There is a clear separation between the workflow system responsible for the *management* (e.g., task ordering) of the business process and the application dealing with the actual *execution* of the business tasks. The theory behind

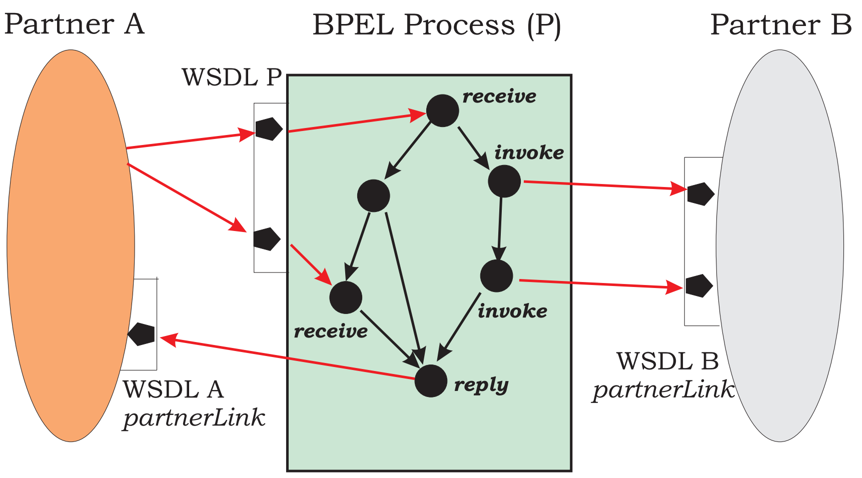


Figure 2.8: BPEL process interactions

the architecture of workflow systems, modeling and management of workflows are discussed in detail in [50].

2.4.1 Business Workflows

The business community has long been using some form of procedural automation for tasks that are routine in nature, such as purchase order management, internal audits, document lifecycle management, IT support services and ticket reservation systems. Enterprises increasingly conduct their business in partnership and often, in order to meet a business request, services from partners are to be integrated. Microsoft defined XLANG [51], a specification for business process automation supporting language constructs for message exchanges among partners. IBM also defined Web Services Flow Language (WSFL) [52], a similar specification supporting a graph-oriented view of business workflows. The convergence of both these specifications resulted in Business Process Execution Language for Web Services (BPEL4WS), supporting composition of services from different partners to achieve a business goal. BPEL is currently an OASIS draft specification under the name WS-BPEL [53] for public review.

The business process in BPEL is composed using a set of well-defined basic activities such as *receive, reply, invoke, assign, throw, compensate* and structured activities such as *sequence, while, if-else, repeat-until.* Figure 2.8 shows how typical

BPEL process activities interact with its partners. The *invoke* activity allows the business process to perform a one directional request or a bi-directional request-response operation on a WSDL portType (partnerLinks) offered by a partner. The *receive* activity allows the business process to wait for matching messages to arrive. WS-BPEL enables *"programming in the large"* that represents long-running stateful interactions involving multiple parties as opposed to *"programming in the small"* that are associated with simple fine grained execution of tasks within a single system [54, 55].

Some of the challenges and the requirements in adapting BPEL to scientific workflows as identified in [56] include *integration with legacy code, experimental flexibility, reuse and hierarchical composition, support for very long-running processes, access to Grid resources and most importantly Grid security integration.* Scientific workflows require access to large data sets; With many BPEL engines exchanging messages via SOAP over HTTP (Web Service Invocation Framework (WSIF) [57] partly addresses this issue with pluggable architecture), support for an alternative high-performance data transfer protocol is also a major requirement.

These specific requirements as listed above often lead to the development of new workflow engines tailored to scientific workflows or customization of commercial workflow engines. The suitability of three commercial workflow engines - Microsoft BizTalk® server [58], Oracle's BPEL Process Manager [59] and Windows Workflow Foundation [60] - for e-Science is studied in [61]. The Human Workflow Services components of the BizTalk® server has been utilized to support scientific workflows in the field of bioinformatics [62]. The Sedna [63] framework provides extensions to BPEL in terms of visual language and a modeling environment suitable for scientific workflows. The extensions in Sedna include a general purpose scientific Process Execution Language to increase expressiveness for scientific workflows and allows for domain-specific abstractions.

2.4.2 Scientific Workflows

Many fundamental workflow principles that are developed for business workflows are directly relevant to scientific workflows. But, the technical issues to be tackled can be different due to the nature of scientific workflows and the requirements from domain scientists as users. Business workflows are usually well-defined in advance and executed in a routine fashion whereas scientific workflows are exploratory and often follow a trial-and-error approach. A visually expressive workflow development environment is often a major requirement as scientific workflows are constructed by domain scientists; any requirement of BPEL expertise or major programming effort may not be an attractive option. The voluminous data that are to be acquired, moved, read and parsed at each stage of the scientific workflow can be far greater than in business workflows. Again the workflow system has to support appropriate data handling under the hood without expecting scientists to deal with it. Similarly, the security policies for user authentication and authorization for scientific collaborations are different from the business process interactions. The comparison between scientific versus business workflows has been discussed in excellent detail in [64].

We will discuss below some important workflow engines and their approach to scientific workflows.

The GriPhyN [65] project addresses the workflow requirements of physics experiments. When the user requests a data object, an abstract workflow DAG (Directed Acyclic Graph) that would generate the desired data object is constructed. The abstract workflow is then converted to a concrete workflow represented as Condor DAGman files [66] and submitted to the Condor-G scheduler.

In the KEPLER system [67], the workflow components are known as *actors* and their communications happen through interfaces called ports. The component interactions and their order of execution are controlled by an object known as a *director*. There are workflow component extensions supporting Web service invocation and Grid service access.

Grid-DB [44] is a data-centric grid workflow system. It provides a declarative language for the user to register code and data with the system. Further, a set of programs can be modeled into an abstract workflow. Grid-DB submits the programs to a Condor pool for execution.

The Triana system [68] exposes Grid Application Prototype (GAP) interfaces supporting different protocol bindings for peer-to-peer interactions between distributed services. The Triana user interface consists of toolboxes with sets of components and a work surface for users to compose workflows.

DiscoveryNet [69] addresses the need for knowledge discovery process in life-sciences. The components and workflows in DiscoveryNet are composed as Web and Grid services by sharing across teams.

The functional requirement of the application domain often influences what features are supported by a particular scientific workflow engine. For example, the workflow composition in GriPhyN and Grid-DB are language-based to support task dependencies. Scientific workflow environment often requires support for visual composition. The Triana and DiscoveryNet projects support a service-based composition model for sharing services across teams. The Web Services model does not explicitly address the requirements for high-performance and near-realtime data transfers. KEPLER system supports visual composition and the workflow activity extensions. But, its workflow composition model is more generic and requires more development effort for application-specific customization. In this context, we discuss the features of a commercial workflow engine in the next section that offer an attractive execution environment for scientific workflows.

2.4.3 Windows Workflow Foundation

The application example, such as the wind tunnel experiments we consider in this thesis run on various proprietary systems and their integration into a scientific workflow requires customized solution. Also, the data transfer and processing requirements are experiment-specific. The workflow framework is required to provide the user with ready-to-use experiment specific workflow activities hiding the underlying complexities and at the same time with the ability for user customization, if required. Over time the processes and systems available in these scenarios change significantly, so the ability to rapidly develop and customize workflows is crucial. In the rest of this section, we discuss Windows Workflow Foundation, a commercial workflow system which has been leveraged for our scientific workflow approaches presented in Chapter 4 and Chapter 5.

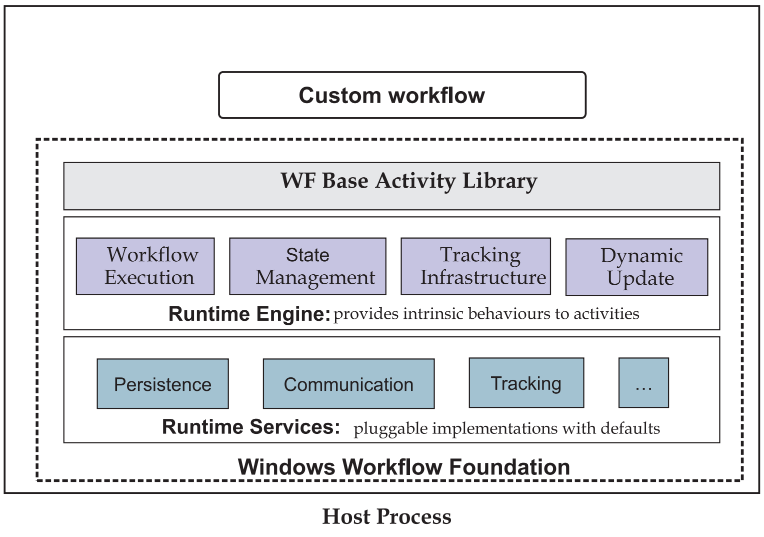


Figure 2.9: Windows Workflow Foundation components & services

Windows Workflow Foundation is an extensible framework that is part of the upcoming Microsoft's next version of .NET development Framework 3.0 [14]. Figure 2.9 shows Windows Workflow Foundation components and services. The workflow in Windows Workflow Foundation is composed from a set of *activities*, compiled to a .NET assembly. It can be executed under the Common Language Runtime (CLR) in a variety of container processes [60].

2.4.3.1 Workflow Model and Composition

Windows Workflow Foundation supports two models [60]: (1) Sequential workflow model - comprising activities that execute in a predictable sequential path; (2) State machine model - a flow driven by events triggering state transitions. In both these models the basic element of the workflow is called an *activity*. Some of the Windows Workflow Foundation's activity types include: control-flow (While, IfElse, Delay), exception (throw, exception-handler and Business Process Execution Language (BPEL) compensations), data handling (Update, Select), transactions (and compensations for long-lived "transactions" that cannot be directly unwound) and Communication (InvokeWebService, InvokeMethod). All *activities* are derived from the *System.Workflow.ComponentModel.Acitivity* base class. The Windows Workflow Foundation extensible development model enables creation of domain-specific *activities* which can then be used to compose workflows that are useful and understandable by domain scientists.

A workflow consists of metadata for the workflow definition, and the accompanying .NET classes that form the code file. The workflow can be composed using a visual workflow designer; this has a drag-and-drop interface and can be hosted in Visual Studio or a user-application. Alternatively, users can write declaratively in eXtensible Applications Markup Language (XAML), an XML dialect for writing workflows. The workflow can also be completely coded in any of the .NET languages. Workflows are compiled with the *wfc* workflow compiler into .NET assemblies before they can be run.

2.4.3.2 Workflow Runtime, Scheduling and Hosting

The workflow runtime layer is at the core of Windows Workflow Foundation and is responsible for execution, tracking, state management, scheduling and policies. The workflow engine runs inside a *hosting process* of the workflow application. This hosting layer is responsible for communication, persistence, tracking, transaction, timing and threading. It is possible to dynamically update the running workflows. The *hosting process* can reside on the client or server-side, depending on the user requirements and application. A long running workflow instance can be persisted, when it is faced with resource constraints, with all its state written into a database so that it can be restarted again.

2.4.4 Workflow Systems: Comparison

We have seen in section 2.4.1 some of the efforts evaluating commercial workflow engines and BPEL to meet scientific workflow requirements. The Windows Workflow Foundation is a generic workflow development framework supporting basic activities similar to BPEL, so a custom activity library for BPEL can be easily developed over it. This will also enable importing and exporting of BPEL workflow definitions into its environment. The lack of visual expressiveness in BPEL has been identified [63] as an issue to be addressed. Table 2.1 compares BPEL features with Windows Workflow Foundation in the context of WfMC reference model. With an extensible framework for workflow activities and a flexible approach to workflow

Table 2.1: BPEL and WF in relation to WfMC model

Functionality as in WfMC Reference model	BPEL	Windows Workflow Foundation (WF)
Process definition	• BPEL process written in XML • No formal visual representation - left to the vendors of workflow engine (e.g, BizTalk Orchestration designer [58]) • Activities - Basic, structured, partnerLinks, fault-handling	• Workflow design using XAML/ .NET languages/visual designer • Flexible workflow designer in IDE or can be part of user application • Built-in library with similar activities to BPEL and more, custom activities can be developed (e.g, Grid, Database activities discussed in Chapter 4 & 5). Export/import BPEL definitions.
Workflow engine	Many vendors support - IBM WebSphere, Microsoft BizTalk, Oracle BPEL Process Manager etc.	Workflow engine can be hosted as part of any process and it can be customized (e.g, persisting workflow into a database)
External interactions	BPEL processes interact via partnerLinks	InvokeWebService, WebServiceReceive, WebServiceResponse activities provide the same functionality
Administration & monitoring	• Explicitly not addressed in the specification - left to the BPEL engines. (e.g, Oracle BPEL Console [59])	• Workflow hosting process can customize the tracking & monitoring services or use the default ones. • Dynamic update of workflow possible (e.g, saving a running instance, modifying or inserting new activity etc.)
Advantages/Limitations:	Loosely coupled, platform independent services. Additional capabilities are needed to adapt for scientific workflows.	Generic and extensible framework. Workflows run on .NET platforms. But, with XML Web Services support, it can interact with other platforms.

hosting, most of the functionality of state-of-the-art scientific workflow systems [70] can be hosted on top of Windows Workflow Foundation.

2.5 Application Examples

The set of requirements on scientific workflow systems from different scientific disciplines, in terms of data size, formats, real-time requirements and computational complexities, bring different sets of challenges. In the following sections we will discuss some Grid applications from different scientific disciplines highlighting their data characteristics (acquisition methods, data format, data volume amongst others), the processing involved, the nature of the workflows, the networking demand and so on.

2.5.1 LHC - Particle Physics Experiments

The major high energy physics experiment Large Hadron Collider (LHC) consists of an accelerator and four associated detectors (ATLAS, CMS, LHCb and ALICE), is expected to begin its operations in 2007 at the European Center for Nuclear Research (CERN). The high energy proton-proton collisions will be studied in order to answer some of the fundamental scientific questions including the origin of mass in the universe. With approximately 800 million collisions per second, the experimental data will be generated at the rate 225 Mbytes/sec resulting in many petabytes per year [5]. Considering the number of physicists engaged at laboratories world-wide and the amount of computing power required, a hierarchical grid computing model (made of different tiers) to support the collection, storage and analysis is adopted. The data is stored and processed initially at a Tier-0 facility at CERN before distributing (requires network bandwidth of 5 Gbits/sec) to 7 Tier-1 national centers at USA, leading EU countries and elsewhere. Similarly, they are processed and further analyzed at Tier-1 and distributed (requires network bandwidth of 10 Gbits/sec) to approximately 25 Tier-2 regional centers, further analyzed and distributed (requires network bandwidth of 1 Gbits/sec or more) to smaller physics groups made of hundreds of Tier-3 data centers, and further to thousands of Tier-4 desktops.

Since the CMS (Compact Muon Solenoid) detector will not begin to acquire data until 2007, physicists from the collaborating institutions are currently taking part in compute intensive Monte Carlo simulation studies. The software infrastructure for this collaboration is built based on Grid technologies – the Globus Toolkit and Condor-G high throughput computing system job submission interface to Globus [71]. The physics simulation functionality has been adapted from the existing legacy CMS code without any rewriting for the Grid. The output of this study can be compared against the actual data, when the CMS begins its operation in 2007, to improve the detector calibrations and help in new scientific discoveries.

The following are the major components and services identified for the CMS computing system during the production run phase in 2007 [71]:

- Grid workload management system to access distributed computing resources

- Data management services supporting storage, data transfer, placement and
- Workflow management services (for data reconstruction, calibration activities, user analysis) tying other software components and shielding the physicists from the complexity of the system and services.

Other general requirements are minimizing the dependency of jobs on worker nodes (for reliability, fault-tolerance and to avoid single point of failures), keeping the local-site configuration autonomous without having to synchronize with other sites and interoperability between different Grid middlewares (LCG-2, EGEE, OSG, NorduGrid) utilized at multiple Grid implementations.

2.5.2 NEESgrid - Earthquake Engineering Experiments

Network for Earthquake Engineering and Simulations (NEES) program is funded by NSF in major earth quake engineering test facilities to study the effects of earthquake on buildings and other structures. The NEESgrid [6] software infrastructure is designed to support the following features.

- store and share data in a centralized repository (data management)
- enable collaboration between distributed researchers
- facilitate remote participation in experiments (telepresence)
- support simulation including hybrid testing
- securely enable authorized access to data and resources

The resources in the NEESgrid system include experimental facilities, data repositories and computers used for simulations. Each of these resources could be owned and controlled by different institutions. The different experimental platforms at NEESgrid sites are *shake table* for testing structures, *reaction wall* to measure static load, *centrifuge* to study soil mechanics, and *wave tanks* to study the behavior of tsunami waves. The data generated from these experiments are managed with file hierarchies based on four directory levels - project, experiment, trial and data. The experimental design metadata include geometry of the specimen and physical properties (material, simulation model) linked to the geometry. The time series data generated from the experiments could be data from sensors, still images, videos, simulation states amongst others. The telepresence component enables researchers to visualize the remote experiment by using a client application to observe the live or archived numerical data generated from acquisition devices, monitoring live video stream of the experiments or by writing their own custom clients using APIs. The remote interaction with the experiment is provided by NEESgrid Teleoperations Control Protocol (NTCP). Apart from experiments, NTCP can be used for Multi-site Online Simulations Test (MOST), enabling hybrid testing that combines physical experiments and simulations.

2.5.3 SDSS - Astronomical Data Collections

The Sloan Digital Sky Survey (SDSS) [10] is an astronomical survey aimed to provide detailed optical images covering more than a quarter of the sky, and a 3-dimensional map of about a million galaxies and quasars. The astronomical raw data is gathered from a 2.5 meter SDSS telescope at Apache Point observatory in New Mexico and stored into a tape. The tape is then physically transported to Fermilab via express courier [72]. The large amount of data from the tape is then passed through a set of data processing pipelines that include spectrographic pipeline, monitor pipeline, astrometric pipeline and photometric pipeline. There are around 400 attributes for each celestial object such as stars, galaxies and quasars are generated from the processing steps along with a 5-color image. The total amount of data at the time of project completion would be approximately 40 Terabytes made of 25 Terabytes of source data and 13 Terabytes of processed data.

The SkyServer is designed to provide internet access to the public Sloan Digital Sky Survey data for astronomers and for science educations. A cluster of Microsoft SQL Server has been utilized to manage the large astronomical database. The astronomical objects generated from the processing pipeline are assigned (partitioned) to different SQL Server based on their spatial dimensions (right ascension and declination). This enables the MaxBCG algorithm to be run in parallel to locate brightest cluster of galaxies in a catalog of astronomical objects. With efficient indexing, join and parallel query operations, a twenty times speedup was achieved [72] as compared to a file-based implementation.

2.5.4 GeWiTTS - Grid Enabled Wind Tunnel Test Service

The objective of GeWiTTS [73] system is to support remote users with different requirements to interact in a collaborative environment while performing wind tunnel tests. The virtual organization between BAE systems and its collaborating partners during the wind tunnel data acquisition and the prototyping stage of the design process has been demonstrated. While one member of the virtual organization is in control of the experiment, others will be able to connect to the wind tunnel to get the view of the experiment as it progresses and provide feedback to the controlling member. GeWiTTS focus is on experiment steering and realtime data sharing.

2.5.5 LEAD - Atmospheric Observations

The Linked Environments for Atmospheric Discovery (LEAD) [7] is aimed at supporting on-demand weather forecasting that would respond to changing weather conditions in real-time. The core feature of LEAD is MyLEAD services, based on a Web Services framework, to support dynamic workflow orchestration and data management. The user interaction with the services are through the LEAD portal to compose the workflow connecting application components. The MyLEAD agent works on user's behalf and is dedicated for a single workflow instance and manages the state of the workflow. MyLEAD consists of three major services - the metadata catalog service, notification service and workflow service [74]. The application-specific metadata catalog resides in a relational database and accessed through OGSA-DAI interfaces supporting domain-specific queries. The MyLEAD agent and the workflow engine subscribe to application-specific events published by the notification service based on WS-Eventing standard. The workflow state transition occurs when experiment status event arrive from the notification service. A slightly modified version of the standard Business Process Execution Language (BPEL) is used for workflow orchestration.

2.5.6 *my*Grid - e-Science Workbenches

The myGrid is an UK e-Science project to build middleware tools to support experiments in the field of molecular biology. The Taverna workbench environment part of myGrid enables biologists and bioinformaticians to compose and execute workflows written in Simplified conceptual workflow language (Scufl). These workflows represent *in silico* experiments composed from wide variety local or remote bioinformatics services (over 1000 bioinformatics Web Services are accessible to myGrid user). The data produced from the services can be in various formats, mostly textual or flat files. Taverna does not impose a common data model.

Taverna uses a three tier data model to describe resources and their interoperations [9]. The *Application Data Flow layer* provides a domain-specific view hiding the complexity of the underlying implementation styles of the services and their interoperations. The *Execution Flow layer* is responsible for the control flow assumptions made by the user. The *Process Invocation layer* interacts and invokes the concrete services. Taverna uses public registries such as UDDI to discover services. The discovered services then can be combined by the user into a workflow.

2.5.7 DAME - Aircraft Engine Maintenance

Distributed Aircraft Maintenance Environment (DAME) [8] is a collaborative project in the UK between industrial and academic partners to build a Grid-based aircraft engine Diagnostics and Prognostic (DP) and maintenance solution. The aircraft engine maintenance problem exhibit characteristics that are typical in Grid environment - data centric, distributed, multiple stakeholders, high dependability and so on.

The DAME system consists of some core Grid services. The *Engine Data Service* manages the data transfer from on-engine monitoring system to a ground station. The annual fleet data can run into many terabytes as each flight produces approximately 1 gigabyte of data. The *Data Storage and Mining Service* applies pattern matching algorithms to rapidly search both raw and archived engine data. The other services include *Engine Modeling Service, Case-Based Reasoning Support,* and *Maintenance Interface Service.* The DAME operation starts when an abnormal sensor data is detected by the on-engine system and cannot be classified locally, and passed to the ground-based system for storage and diagnosis. The data mining activity runs next to recover related events that matches the abnormal condition. The remedial action is selected from historical data or by executing appropriate signal processing step. The final step in the diagnostic process involves sharing of the results with the stakeholders for maintenance, repair and overhaul.

2.5.8 Application Characteristics - Comparison

Table 2.2 shows the comparison of some of the important characteristics of the application examples discussed in previous sections and wind tunnel experiments discussed in Chapter 3. The approaches taken to integrate experiments from different application domains are unique as their data, processing and functional characteristics vary to greater extent. At one end of the spectrum, experiments such as LHC produce large data sets with high performance computing and high-speed data transfer requirements. At the other extreme, the wind tunnels (discussed in next chapter) are characterized by different experiments, multiple locations, multiple runs, changing parameters, high volume data and customized processing. Also, the approach taken by individual applications are largely driven by their core functional requirement. For example, in both NEESgrid and GeWiTTS the important functional requirement is remote steering of experiments and hybrid testing, MyLEAD system is expected to respond to changing weather conditions in realtime, the engine fault diagnosis and prognosis requires large-scale data mining in DAME system and the Sloan Digital Sky Server has to implement effective search algorithms to locate cluster galaxies.

When compared to the above application examples, the different wind tunnel experiments we discuss in Chapter 3 run on some proprietary systems, shared by multiple users, generate data in various formats often with near-realtime data movement and customized integration requirements. Further, an aerodynamicist would be interested in customizing the processing algorithms. The approaches we present in Chapter 4 and 5 address the data-driven nature of this application scenario.

Table 2.2: Comparison of important application characteristics

Application	Domain	Raw data (format) characteristics	Important functional requirements	Workflow requirements	I/O or data or CPU -intensive
LHC	Particle Physics	numerical, time-series data	large volume of data is to be transferred efficiently to other sites and interoperability between Grid middleware	workflow management services for data calibration and user analysis	compute as well as data-intensive
NEESgrid	Earthquake engineering	numerical data and video stream (for monitoring)	remote participation (telepresence) and steering of experiment	none	I/O intensive (hardware integration)
SDSS	Astronomy	Astronomical objects/ images represented as numerical data	MaxBCG algorithm search on astro objects	no explicit requirements - preprocessing pipeline can be modeled as workflow	data-intensive
GeWITTS	Aerospace	video stream	remote view of experiment	none	I/O intensive
LEAD	Atmospheric sciences	numerical, sensor data from Doppler radars	Real-time response to changing weather conditions	event-driven workflow	compute-intensive
*my*Grid	Molecular Biology	semi-structured, heterogenous and bespoke formats representing gene sequences, protein sequences etc.	construction and editing of service-based workflows	workflow steps to take input and output in domain-specific formats	experiment -dependent, some data-, some compute-intensive
DAME	Aircraft Maintenance	Engine sensor data - real valued variables monitored over time, historical data and non- -declarative data such as procedures used	Data storage and mining - advanced pattern matching and data mining on large volume of data for engine fault diagnosis and prognosis	none	data-intensive, device integration
Wind tunnel grid system	Experimental Aerodynamics	experimental data - numerical time series data, image frames, velocities, pressure, turbulence etc. (bespoke data formats)	integrating multiple experimental data flow and custom processing step	experiment-specific data-driven approach to workflow	data-centric, integration of proprietary DAQ system and their data formats

2.6 Summary

In this chapter, we have reviewed different Grid technologies, middleware development efforts and emerging Grid standards. We have discussed some of the characteristics that distinguish scientific workflow systems from business workflows and the adaptation of commercial workflow engines to scientific environments. We presented application projects from different domains and highlighted their difference in approach to Grid integration. The sources of variability are due to various factors such as the data formats, the role of database management systems, the ability to compose workflows from loosely coupled services, the realtime response to changing events, the support for remote access to hardware, the computational requirements and the high-speed data transfer requirements. The unique set of requirements often lead to a specialized approach to individual application domains.

Chapter 3

Integrating Wind tunnel experiments on the Grid: Requirement analysis

Aerodynamicists use wind tunnels to design, test and verify aerodynamics of aircraft, cars, yachts, and buildings, amongst others. The wind tunnels are normally designed for a particular application and a speed range to support a variety of testing techniques and flow conditions. During the wind tunnel experiment, often the scale models are tested and the test conditions are matched against the actual flight conditions in order to acquire the meaningful data. Various techniques are then applied to study the airflow around the model and compare it with the theoretical and simulation results.

Similar to many other scientific workflows, the activities in a typical wind tunnel experiment consists of experiment setup, data acquisition, data transfer, processing, analysis and visualization. When compared to other application examples discussed in the last chapter, the wind tunnel experiments differ in the following manner:

- *Facility:* There are multiple wind tunnel facilities shared by users, operated often on a pay-per-use basis, distributed across campus, and users are often bound by time slots.
- *DAQ:* There are specialized data acquisition systems for each experiment with varying setting up procedures.
- *Data formats:* The data generated from experiments could be numerical time

series data, audio samples, image data etc.

Despite some of the unique characteristics of wind tunnel experiments, there are common requirements similar to other application workflows:

- *Data access:* Location-independent access to the experimental data and results
- *Data transfer/store:* On completion of the experiment the data is to be transferred for processing and archival.
- *Processing/Analysis:* Experiment-specific processing and analysis.

In the following sections we look at the requirements on integrating wind tunnel experiments onto Grid in general and three of the experiments in detail.

3.1 Wind tunnel experiments and Requirements

The University of Southampton wind tunnel facilities [11] are distributed across the campus housing a variety of specialized experimental hardware and software for academic and industrial research. The R J Mitchell wind tunnel is a large and low-speed wind tunnel ($11' \times 8'$ working section) with moving ground and supports a maximum wind speed of 50 meters/s. It is ideally suited for vehicle aerodynamics work. The $7' \times 5'$ wind tunnel supports wind speeds up to 55 meters/s. It is used extensively for undergraduate and postgraduate research projects. It has a moving ground for vehicle aerodynamics work (Figure 3.1), and a low speed section for wind engineering studies and work for the marine industry on racing yacht sails. The $3' \times 2'$ wind tunnel has an open circuit facility with a 0.9m $\times$ 0.6m $\times$ 4.5m working section. It is installed in a laboratory, and is equipped with a 3D computer controlled probe traversing system and dynamometer, and is used for a variety of aerodynamics research and student project work. There is a further laboratory equipped with a number of tunnels that are primarily used in the undergraduate courses.

Two of the aerodynamic forces that are of interest to aerodynamicists are *lift* and *drag*. The force that is perpendicular to the flow is called the *lift*; the force that is acting along the flow is called the *drag*. The resultant net force depends on the pressure variations (which in turn depends on the velocity changes) around the object. Aerodynamic forces also depend on the viscosity of the fluid with a parameter known as *Reynolds number (Re)*. The lower Reynolds numbers are characterized by boundary layers that are *laminar* (velocity changes uniformly) and for the higher Reynolds numbers, the boundary layers are *turbulent* (velocity changes unsteady). Aerodynamicists employ hot-wire anemometry to study the turbulent flows of the boundary layers.

For certain class of applications Computational Fluid Dynamics (CFD) can supplement or replace wind tunnel testing. But, in cases of flight conditions where external turbulent flow is present, CFD may not be practical and wind tunnel experiments are still the preferred way of testing.

In the wind tunnel environment, there are multiple experiments with varying degrees of data management and processing requirements. The data generated during acquisition vary in terms of the number of data items, file size and format, depending on the wind tunnel experiment and user parameters. Due to limited storage capacity on the acquisition system and multiple user's using the facility, the raw data should be transferred from the acquisition system to a network location where it can be managed and processed. This requires an experiment-specific approach to integrating the acquisition system. Data access from different network locations should also be supported as a researcher may be in office or home, a student user may be in public work station area and an industrial client may access from off-campus. In the following sections we discuss three of the experiments to highlight their requirements in integrating them onto Grid.

3.2 Laser Doppler Anemometry (LDA)

The Laser Doppler Anemometry (LDA) [75] technique is ideal for 1D, 2D and 3D point measurement of velocity and turbulence distribution in both free flows and internal flows. LDA systems are used to gain a clearer understanding of fluid mechanics. The measurement results are important steps in fine-tuning product designs to improve aerodynamic efficiency, quality and safety. The major advantages include non-intrusive measurement, high spatial and temporal resolution

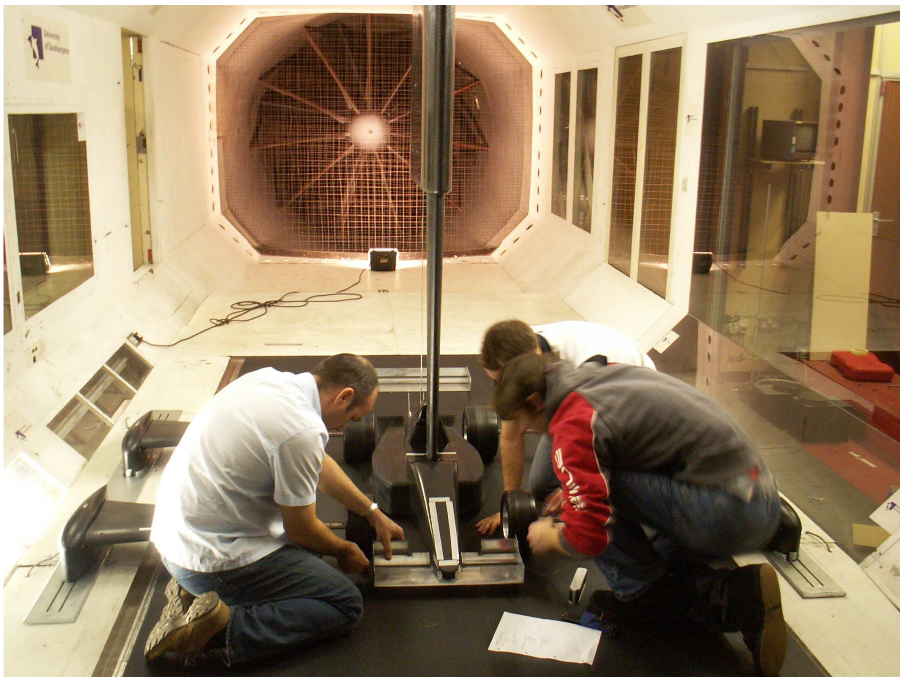
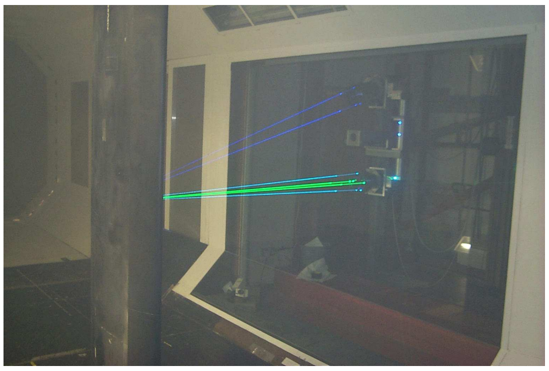


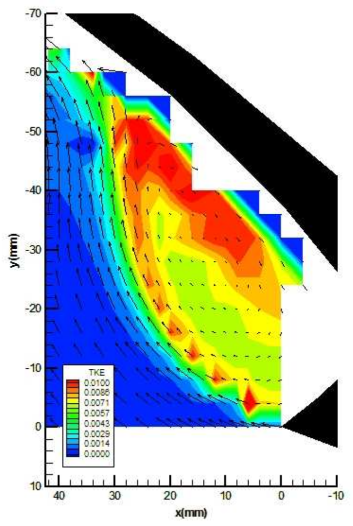
Figure 3.1: Setting up during a typical wind tunnel test

and the ability to measure in reversing flows. The accuracy of this technique is invaluable for measuring turbulence levels to understand flow physics at a detailed level. Figure 3.2 shows LDA data acquisition and a typical visualization.

3.2.1 Data Acquisition

For each experimental configuration, a calibration step must be performed and a transformation matrix must be derived. The transformation matrix is used to translate the raw data in laser coordinate system to the tunnel coordinate system during processing. LDA data acquisition software collects selected number of samples (typically thousands) at user programmed traverse positions of up to three velocity components (This value is also equal to number of Burst Spectrum Analyzers (BSA)). The collected data are stored in separate raw data files for each traverse position. The raw data filenames have a suffix 0, 1 or 2 to indicate the velocity component (u, v & w, in the laser coordinate system). The file extension represents the traverse position. There are essentially $n \times p$ raw data files for one experiment, n $(n=3)$ is the number of velocity component and p is the number of traverse positions. The user parameters for acquisition are stored in a separate flat


(a) LDA experiment in progress



(b) Result - kinetic energy contours [76]

Figure 3.2: Laser Doppler Anemometry Experiment

file.

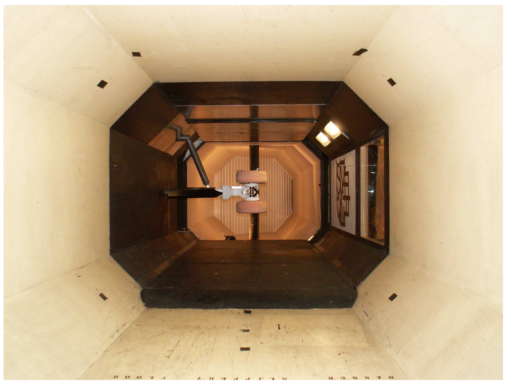
3.2.2 Data Transfer

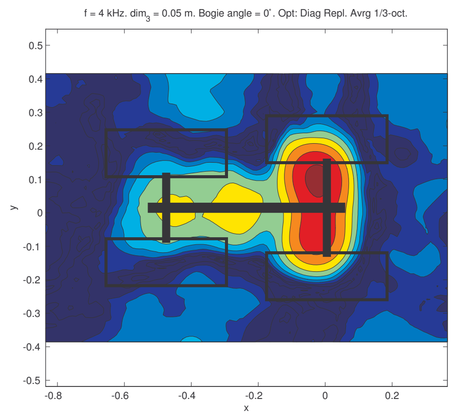
The upload activity of the workflow requires verification of the raw data files prior to the uploading to processing node. Since the number of velocity components and traverse positions are known from the parameter file, all the raw data files along with metadata (transformation matrix, user parameters) can be uploaded without any user intervention.

3.2.3 Processing

Basic LDA Processing codes/algorithms are available as a result of other projects [77, 78]. The major requirement here would be to integrate the available LDA processing logic into Wind Tunnel Grid framework - making them run in a Grid environment, exposing them as a Grid service to the user, interfacing with data management and visualization components.

Data Conversion: The raw data file contains BSA setup information (header) followed by series of *burst*. The *burst* holds Velocity, Transit time and Arrival time as integer numbers. These integer values are converted to physical units (floating


(a) Landing gear test



(b) Result - noise contour plot [79]

Figure 3.3: Microphone Phased Array Experiment

point numbers) using standard conversion formulas. This step would produce $n \times p$ converted data files with one file for every raw data file. Here n is number of BSA channels and p is number of traverse positions.

Coincidence Processing: Sorted data files can be generated as a result of coincidence processing with user-selectable coincidence interval. There is one sorted file created for every n converted data files, where n is number of BSA channels. The transformed velocity components are computed from measured velocity components using transformation matrix.

Moment Processing: In this step Mean value, Variance, RMS value, Skewness, Flatness factor, Turbulence intensity, Cross moments are computed and stored in Moments file.

Spectrum Processing: Spectral estimates are computed using FFT and stored along with the corresponding frequencies in Spectrum file.

Correlation Processing: Correlation coefficients are calculated from the inverse FFT of spectral estimates computed in Spectrum Processing.

We cover further details on LDA processing in Appendix A.1.

3.3 Microphone Arrays

The microphone array technique is used to measure noise of aircraft components (slats, landing-gears, flaps etc) to help aerospace engineers improve the aircraft design and to reduce the overall airframe noise. Microphone arrays consist of multiple, O(100), microphones that must be simultaneously sampled. The phase shift between channels is then used to derive acoustic source information [79, 80, 81].

The aeroacoustic researchers at University of Southampton use a National Instrument's NI4472-based data acquisition system designed for acoustic and vibration applications. This is able to sample multiple channels at up to 96kHz (48kHz anti-aliased), while remaining tightly synchronized in time. The system controller is driven by an NI's proprietary Labview system running Windows XP. The I/O slots of the controller can be populated with specialized data acquisition cards each supporting a number of channels. For example, a system with 7 cards and 8 channels would support an array of 56 microphones for the measurement. Figure 3.3 shows a landing gear test using 56 microphones and an associated noise contour plot.

3.3.1 Data Acquisition

The data acquisition in microphone technique involves simultaneous sampling of all the channels. A typical data acquisition event on a high channel count system would generate a large volume of data, running into hundreds of mega bytes per second [81]. In order to achieve realtime processing of the time-series data, efficient data storage and data transfer techniques must be employed. The raw data comprises blocks of samples received from individual microphones at a user-specified sampling rate.

3.3.2 Data Transfer

The metadata (number of microphones, microphone sensitivity data, sampling rate, block size etc) must be used for customized data upload.

3.3.3 Processing

The microphone array processing happens in two stages: cross spectral matrix computation and beamforming. The cross-spectral matrix (CSM) is an $M \times M$ matrix, where M is the number of microphones. The CSM computing steps involve data calibration of the raw samples, Frequency Fourier Transform (FFT) computation, block averaging of cross spectral components and background noise

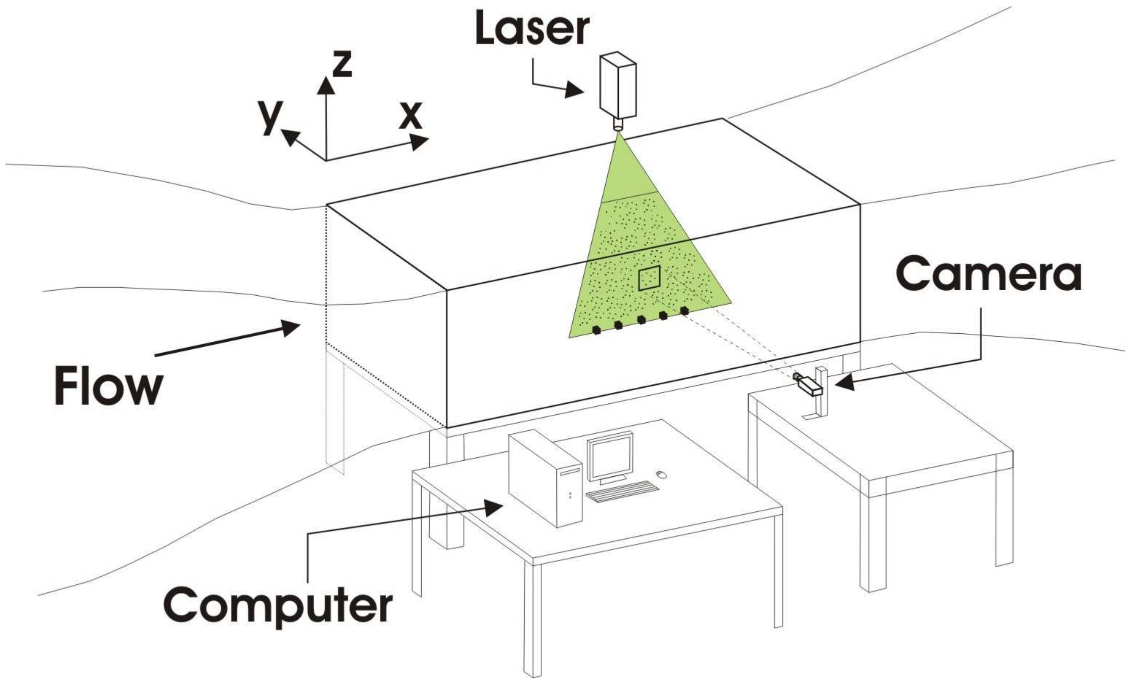


Figure 3.4: Particle Image Velocimetry Experiment

removal [79, 81]. By dividing the microphone samples, the CSM steps can be run in parallel as can be seen in Chapter 5. In the beamforming step, the beamforming expression is computed using the CSM. The grid coordinates are then generated for plotting. Similar to the data transfer, user processing step also requires customization so that different algorithms can be developed and used as the state-of-the-art advances.

Further details on microphone array processing can be found in Appendix A.2. A complete discussion on this topic can be found in [81].

3.4 Particle Image Velocimetry (PIV)

Figure 3.4 shows a schematic of the Particle Image Velocimetry experiment setup. PIV [82, 83] is a non-intrusive, field-based technique to measure fluid velocity. In contrast to the LDA system, which measures velocities at a single point in space at multiple times, the PIV system simultaneously measures velocities in the field of view of the sensor (a digital camera) at a single instant in time. 2-D PIV systems use a single camera, while 3-D PIV systems employ two CCD cameras, one on the

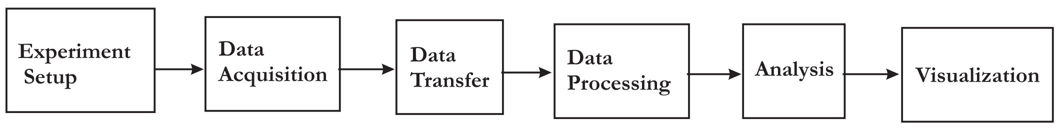


Figure 3.5: Typical wind tunnel experiment workflow

left and one on the right, to produce two 2-dimensional vector maps showing the instantaneous flow field as seen from each of the cameras. Using the calibration function obtained during camera setup, the true 3-D particle displacement can be calculated.

The requirements for PIV are: 1. An upload component to transfer image frames from acquisition system to processing cluster. 2. Timely processing response by means of high-performance implementation. This requires parallel implementation of cross-correlation computation between image frames.

3.5 Discussion

Figure 3.5 shows a typical wind tunnel experiment workflow steps which is similar to any other scientific workflow. Each activity in the wind tunnel workflow depends on the data produced from the previous workflow step, making them data-centric and often with near-realtime requirements. Since the workflow transitions are triggered by data, wind tunnel workflows fall under the categories of data-driven workflow systems, as opposed to control-flow driven [84]. The workflow framework should hence address the data-driven and near-realtime nature of the workflow.

Unlike many other scientific workflows, in the wind tunnel experiment scenario, multiple users share experiment facilities to run different experiments of interest. The wind tunnel experiments run on some proprietary system and their integration into a workflow environment requires customized solution.

Due to storage constraints on the acquisition system the experimental data are to be transferred in near-realtime to another system for processing/storage. The configuration metadata obtained during experiment setup can be utilized to perform a customized data transfer operation for a particular experiment. In typical wind tunnel processing, an aerodynamicist would be interested in changing and customizing the processing algorithms. User customization of the processing algorithm, together with an ability to use the default processing steps, is an essential requirement. Also, the user should be able to apply different processing parameters to any of his existing data set producing new result set.

An approach addressing the requirements discussed in the previous sections needs to abstract the general requirements and at the same time support experiment-specific extensions. This would make component reuse and new experiment additions possible.

3.6 Summary

In this chapter, we looked into wind tunnel experiment requirements in general and specific requirements of LDA, Microphone and PIV experiments. We can easily see from the requirements that wind tunnel experiment steps are data-centric. One step in the workflow is dependent on the data produced in the previous step; this requires workflow framework that addresses this data-driven workflow requirements. Also, the workflow steps such as data acquisition, data transfer and processing are to be specialized for individual experiments. Building abstract workflow activities to address the general requirements and customized activities to address the experiment-specific requirements enables new experiments to be easily added. Similarly, in pay-per-use facilities such as wind tunnel, ready-to-use workflow activities with visual workflow composition are important. The approaches to addressing these experimental workflow requirements are discussed in Chapter 4 and Chapter 5.

Chapter 4

Workflow Integration based on Grid Services

The ability of Grid technologies to pool resources from multiple organizations to solve large scale scientific problems is growing in importance. In the last few years, Globus has emerged as the *de facto* standard Grid middleware to be exploited in various application domains, especially in the context of academic research and development. The key concept it has brought about is in Grid security supporting *single sign-on* and *delegation*; together they play an important role in enabling *virtual organizations* [4]. Science and engineering research is becoming increasingly collaborative, with people from multiple organizations sharing different resources (hardware, software, data).

In order to fully realize the potential of Grid computing the end-user community (domain experts) should have access to appropriate software frameworks that makes the Grid easy-to-use so that they can do their science more effectively. We have seen from the last chapter, the different experimental workflow requirements and their integration challenges onto Grid. This leads us to some of the motivating questions: *How to extend Grid services reach to popular development frameworks with Grid security support? How do we hide the underlying complexity in Grid service access? Is it possible to support application workflow development from end-users favorite environment/languages leveraging their legacy code?*.

In this chapter, an approach to workflow integration based on Grid services is presented. Figure 4.1 shows a typical grid workflow development stack that we utilize to illustrate our approach. At the bottom of the stack are the Grid resources that include compute clusters, storage networks, data acquisition systems, licensed

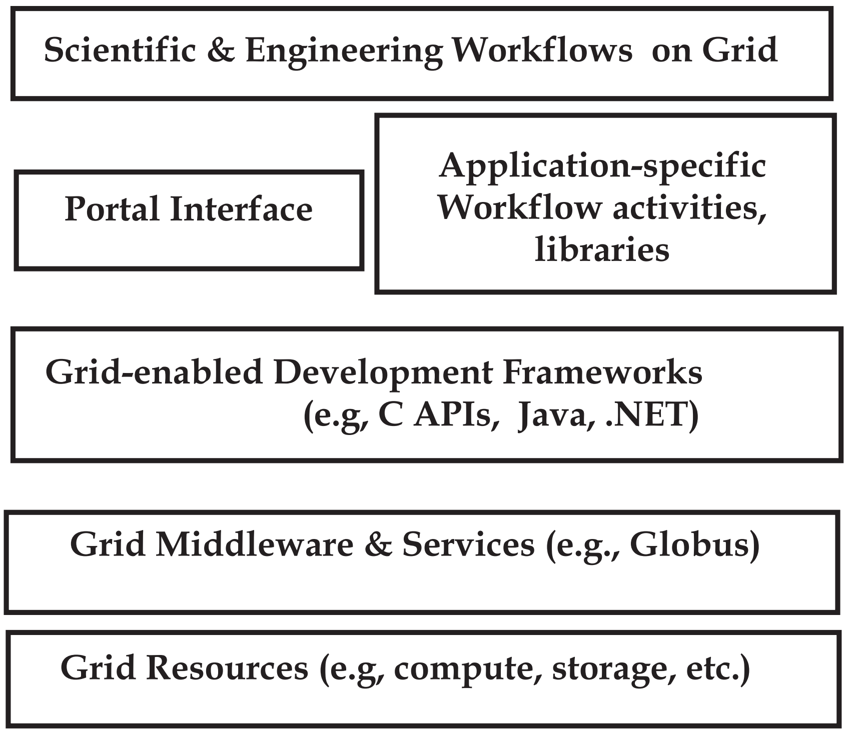

Scientific & Engineering Workflows on Grid
Portal Interface
Application-specific Workflow activities, libraries
Grid-enabled Development Frameworks (e.g, C APIs, Java, .NET)
Grid Middleware & Services (e.g., Globus)
Grid Resources (e.g, compute, storage, etc.)

Figure 4.1: Grid workflow development stack

softwares and so forth. The Grid middleware is responsible for services such as security, data management and job scheduling.

Abstracting Grid service access into a familiar development framework such as Java and .NET brings more opportunities for higher level applications. Grid application toolkits play a vital role in achieving this. This is demonstrated by means of MyCoG.NET, a multi-language Commodity Grid (CoG) toolkit to access GT2 Globus Grid services from the .NET framework. Existing CoG toolkits are available for a limited number of languages (Java [29], Python [30], Perl [31]). By enabling Grid access from the .NET framework, it will be possible to integrate legacy code and develop applications in languages such as FORTRAN which is still favored by many scientists. Further, Grid service access from any of the languages supported by the .NET Common Language Runtime (CLR) is possible.

Grid access from a commodity development environment enables applications to leverage continuously evolving and well-tested class libraries supporting text pro-

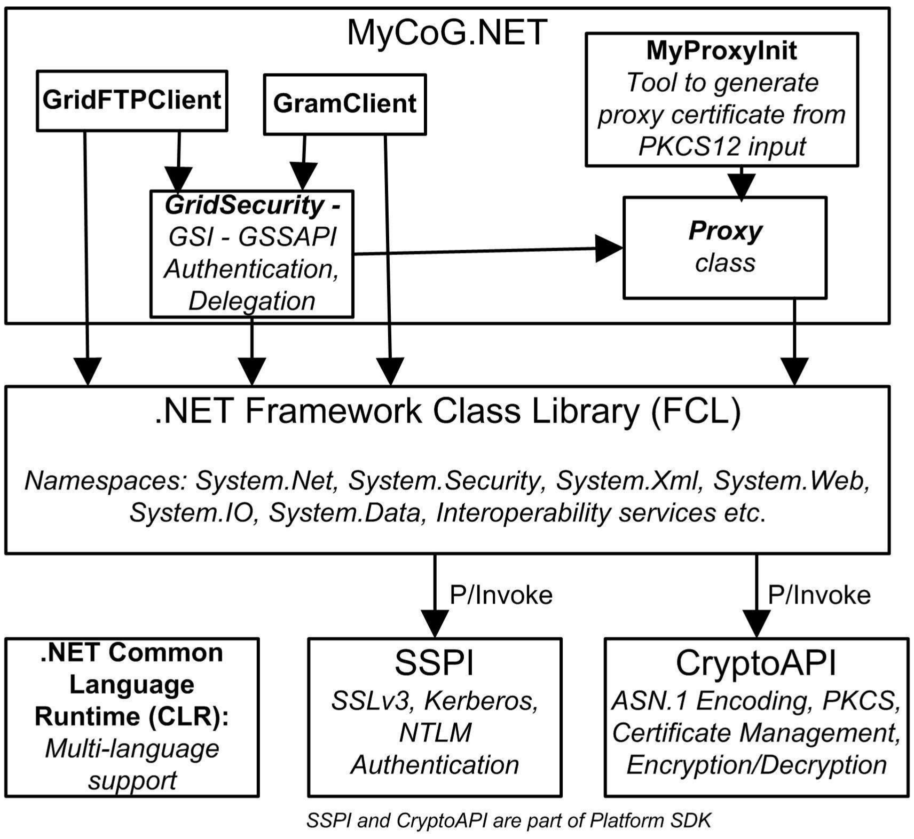

MyCoG.NET
GridFTPClient
GramClient
MyProxyInit
Tool to generate proxy certificate from PKCS12 input
GridSecurity - GSI - GSSAPI Authentication, Delegation
Proxy class
.NET Framework Class Library (FCL)
Namespaces: System.Net, System.Security, System.Xml, System.Web, System.IO, System.Data, Interoperability services etc.
P/Invoke
P/Invoke
.NET Common Language Runtime (CLR): Multi-language support
SSPI
SSLv3, Kerberos, NTLM Authentication
CryptoAPI
ASN.1 Encoding, PKCS, Certificate Management, Encryption/Decryption
SSPI and CryptoAPI are part of Platform SDK

Figure 4.2: MyCoG.NET architecture

cessing, database access, network programming, regular expressions, web service integration, portal development and so on. For instance, in our approach the workflow requirements of the wind tunnel experiments has been realized using the Windows Workflow Foundation component of .NET Framework 3.0.

In the rest of this chapter, we discuss how Grid access has been enabled in the .NET framework by presenting the MyCoG.NET architecture and implementation details, we provide an approach to wind tunnel experiment workflow based on a web portal interface, and finally we discuss an experiment-specific workflow approach leveraging a commercial workflow system, Windows Workflow Foundation.

4.1 MyCoG.NET Architecture and Implementation

Figure 4.2 shows the MyCoG.NET architecture based on the .NET framework. MyCoG.NET consists of GridSecurity, GridFTPClient, GramClient and

Proxy classes. It also includes the MyProxyInit tool to generate an X.509 proxy certificate from a user's end-entity certificate or from another proxy. Underlying the MyCoG library is the .NET Common Language Runtime (CLR) providing a shared type system, an intermediate language and dynamic execution environment for the implementation, and inter-operation of multiple source languages. In the following sections we discuss the advantages MyCoG derives from .NET, implementation details, its usage from .NET languages, and its performance. The class diagrams from MyCoG namespaces are included in Appendix B.

4.1.1 Choice of .NET Framework

The .NET Framework consists of the Common Language Runtime (CLR) and Framework Class Library (FCL). The CLR operates on assemblies which are logical groupings of one or more managed modules or resource files. The assembly is the smallest unit of reuse, versioning and security. Assemblies can consist of types implemented in different programming languages. The .NET Common Language Infrastructure (CLI)(ISO/IEC 23271) and C$^{\#}$ (ISO/IEC23270) have been ratified as an ECMA standard [85]. More details on the architecture of the .NET platform and CLR can be found in [86]. Some of the advantages in using .NET as a development platform in a grid environment are discussed below.

Multi-language support: The CLR's standard set of types, self-describing type information (metadata) and common execution environment enables object-level interoperability between programming languages. Some of the already available language compilers targeting CLR include C^{++}, C$^{\#}$, Java, Pascal, COBOL, Eiffel, APL, FORTRAN, Perl, Python and Smalltalk. With mixed language support, for example, one can develop a grid application by writing graphical user interfaces in C^{++} and scientific computations in FORTRAN based on coding preferences and/or language strengths.

Performance: The .NET development environment compiles the high-level language code into Intermediate Language (IL). When a method is called for the first time the CLR's JIT (just-in-time) compiler verifies and converts IL code into native CPU instructions and caches it in memory at runtime. Subsequent calls to the

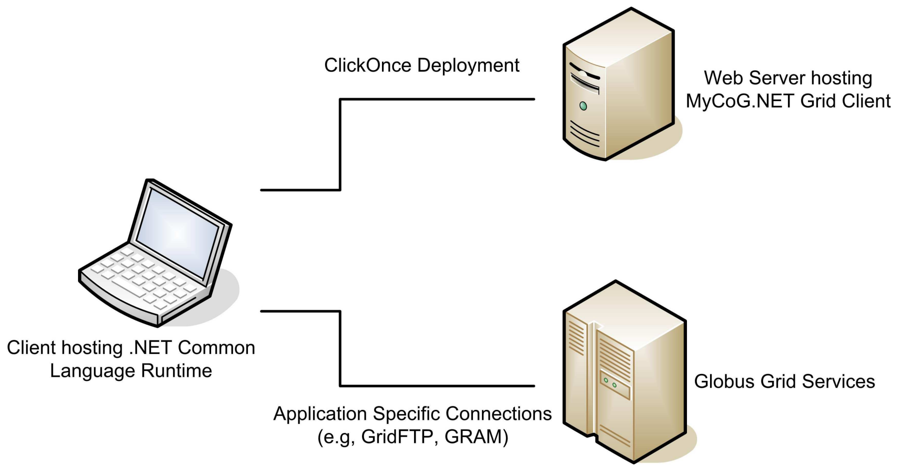

ClickOnce Deployment
Web Server hosting MyCoG.NET Grid Client
Client hosting .NET Common Language Runtime
Application Specific Connections (e.g, GridFTP, GRAM)
Globus Grid Services

Figure 4.3: ClickOnce deployment model

method execute directly using CPU instructions from the cache (avoiding the need for JIT compilation), thus improving the performance.

Portability: Since the Common Language Infrastructure (CLI) is ratified as an ECMA standard [85], there is growing interest in implementations on non-Windows platforms. The intermediate language is not tied to any specific CPU platform. A managed module containing intermediate language code can run on any CPU platform as long as the target operating system hosts the version of the CLR. For example, a .NET managed module compiled under Windows can be run under the Linux Mono [87] development environment. Apart from Microsoft's .NET just-in-time compilation support for the x86 platform, the Mono project sponsored by Novell supports the x86, SPARC and PowerPC architectures on Windows, Linux, Solaris, HP-UX, Mac OS and BSD operating systems.

ClickOnce - No Touch Deployment: Grid application deployment and providing user access to the latest version of the application are quite challenging for developers. The .NET Framework supports application deployment by means of *ClickOnce* [88] and it can be utilized in the context of the Grid as shown in Figure 4.3. The application developer publishes the application files on a Web Server using the *ClickOnce* Deployment procedure and provides the user with a URL. When the user clicks on the link the .NET runtime recognizes the application, downloads files onto the client desktop, and executes under a user specified security context.

Table 4.1: MyCoG GSI authentication, delegation and message security

MyCoG GridSecurity (SSPI)	Message Tokens	Globus Server (GSSAPI)
1. Authentication		**1. Authentication**
a. Get outbound credentials calling **AcquireCredentialHandle()**		a. Get inbound credentials calling **gss_acquire_cred()**
b. Generate SSLv3 authentication tokens calling **InitializeSecurityContext()** repeatedly and send to server.	$\Leftrightarrow$ Client-Server Hello $\Leftarrow$ Server Certificate $\Leftarrow$ Certificate Request $\Rightarrow$ Client Certificate $\Rightarrow$ Client Key Exchange $\Rightarrow$ Certificate Verify $\Leftrightarrow$ ChangeCipherSpec $\Leftrightarrow$ Finished	b. Parse client security tokens calling **gss_accept_sec_context()** repeatedly and respond to client.
2. Delegation		**2. Delegation**
InitDelegation()	$\Rightarrow$ Delegation Flag $\Leftarrow$ PKCS10 certificate request $\Rightarrow$ Proxy Certificate Chain	**gss_accept_delegation()**
3. Message Security		**3. Message Security**
EncryptMessage()	$\Rightarrow$ ApplicationData	**gss_unwrap()**
DecryptMessage()	$\Leftarrow$ ApplicationData	**gss_wrap()**

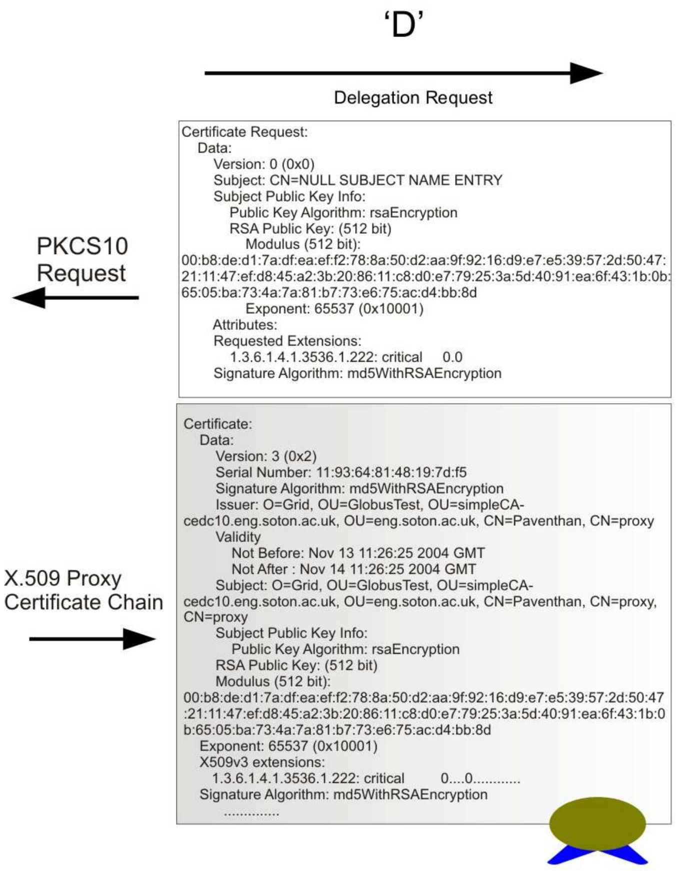
$\Leftrightarrow$ Both client and server exchange, $\Rightarrow$ Client to server message, $\Leftarrow$ Server to client message

ClickOnce provides a trustworthy deployment model for grid users to download and execute grid applications from centrally managed servers. This was demonstrated by our MyGridFTP tool [89], a *ClickOnce* Globus GridFTP client supporting grid security. The Jini downloadable proxies [90] that can hide the communication protocols necessary to talk with a remote service are analogous to applications deployed in the *ClickOnce* model.

Web Services support: The *System.Xml* namespace part of FCL provides W3C (World Wide Web Consortium) [39] standards-based support for processing XML. The W3C standards compliance ensures cross-platform inter-operability. The XML Schema Definition (XSD) tool can be used to convert XSD schemas to .NET language classes. The Web Services Description Language (WSDL) tool generates code for XML Web services and XML Web service clients from WSDL contract files and XSD schemas.

4.1.2 Grid Security

Grid security is crucial for authentication, authorization and delegation. The Generic Security Services Application Programming Interface (GSSAPI) [21] defines a portable API for client-server authentication. On the server side, the Globus

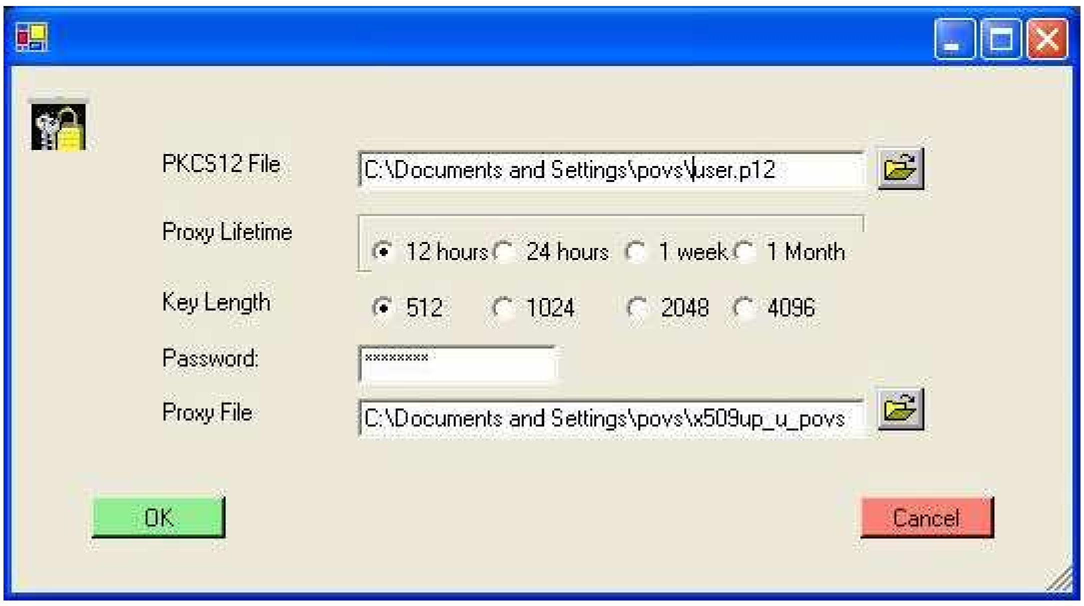


```
Certificate Request:
   Data:
      Version: 0 (0x0)
      Subject: CN=NULL SUBJECT NAME ENTRY
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
         RSA Public Key: (512 bit)
            Modulus (512 bit):
00:b8:de:d1:7a:df:ea:ef:f2:78:8a:50:d2:aa:9f:92:16:d9:e7:e5:39:57:2d:50:47:
21:11:47:ef:d8:45:a2:3b:20:86:11:c8:d0:e7:79:25:3a:5d:40:91:ea:6f:43:1b:0b:
65:05:ba:73:4a:7a:81:b7:73:e6:75:ac:d4:bb:8d
            Exponent: 65537 (0x10001)
      Attributes:
      Requested Extensions:
         1.3.6.1.4.1.3536.1.222: critical    0.0
      Signature Algorithm: md5WithRSAEncryption
```

```
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number: 11:93:64:81:48:19:7d:f5
      Signature Algorithm: md5WithRSAEncryption
      Issuer: O=Grid, OU=GlobusTest, OU=simpleCA-
cedc10.eng.soton.ac.uk, OU=eng.soton.ac.uk, CN=Paventhan, CN=proxy
      Validity
         Not Before: Nov 13 11:26:25 2004 GMT
         Not After : Nov 14 11:26:25 2004 GMT
      Subject: O=Grid, OU=GlobusTest, OU=simpleCA-
cedc10.eng.soton.ac.uk, OU=eng.soton.ac.uk, CN=Paventhan, CN=proxy,
CN=proxy
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
      RSA Public Key: (512 bit)
      Modulus (512 bit):
00:b8:de:d1:7a:df:ea:ef:f2:78:8a:50:d2:aa:9f:92:16:d9:e7:e5:39:57:2d:50:47
:21:11:47:ef:d8:45:a2:3b:20:86:11:c8:d0:e7:79:25:3a:5d:40:91:ea:6f:43:1b:0
b:65:05:ba:73:4a:7a:81:b7:73:e6:75:ac:d4:bb:8d
   Exponent: 65537 (0x10001)
   X509v3 extensions:
      1.3.6.1.4.1.3536.1.222: critical       0....0...........
   Signature Algorithm: md5WithRSAEncryption
      .............
```

Figure 4.4: X.509 proxy delegation

Toolkit implements Grid Security Infrastructure (GSI) providing single sign-on capability based on GSSAPI and Grid forum recommended extensions [91]. The MyCoG GridSecurity module encapsulates GSI client functionality by making calls to Windows native Security Support Provider Interface (SSPI) for Secured Socket Layer version 3 (SSLv3) handshake and implementing delegation and authorization message exchanges. The GSI Message specification [92] defines three types of GSI messages exchanged between client and server, as shown in Table 4.1. During context-establishment, the MyCoG client uses the GridSecurity module to exchange SSLv3 handshake messages for mutual authentication with the Globus server. Upon successful mutual authentication, the server makes an authorization decision based on the mapping of certificate subjects to local user names present in the grid-mapfile.



PKCS12 File: C:\Documents and Settings\povs\user.p12

Proxy Lifetime: [✓ 12 hours] 24 hours 1 week 1 Month

Key Length: [✓ 512] 1024 2048 4096

Password: ********

Proxy File: C:\Documents and Settings\povs\x509up_u_povs

Figure 4.5: MyProxyInit user interface

4.1.2.1 Delegation

During the delegation phase (Figure 4.4), the client can delegate its credentials to the server by sending a delegation flag. The server responds by sending a PKCS10 certificate request containing the valid public key and proxy certificate extension. The MyCoG GridSecurity module calls the Proxy class to create a new proxy certificate conforming to RFC 3820 [93]. The Proxy class encodes and signs the proxy certificate by making calls to CryptoAPI before sending it back to the Globus GSI server.

4.1.2.2 Proxy Support

User can create X.509 proxy certificates using the MyProxyInit tool shown in Figure.4.5 before running any MyCoG client. MyProxyInit takes the PKCS12 formatted user certificate file, pass phrase and proxy certificate name as inputs. It calls the Proxy class for RSA public key generation and proxy certificate creation. We decided in favour of PKCS12 rather than PEM-formatted user certificates and the *openssl* proprietary private key format due to inter-operability issues. Both the user certificate and key can easily be exported to PKCS12 format using an *openssl* command as below. This one time conversion step is required to use MyCoG clients if the user's X.509 certificate issued by the Certification Authority is in PEM format.

```
openssl pkcs12 -export -in usercert.pem -inkey userkey.pem -certfile cacert.pem -out usercert.p12
```

4.1.3 File Transfer

The MyCoG GridFTPClient class provides GridFTP file transfer features defined in [22] that include third-party file transfer, parallel streams, partial file transfer, TCP buffer size tuning and striped data transfer. It makes a call to the MyCoG GridSecurity module for GSI authentication and delegation. The code snippets in Table 4.2 show how the MyCoG GridFTPClient can be called from FORTRAN, C^{++}, $C^{\#}$ and Java programs. We have tested the FORTRAN code using the Lahey/Fujitsu Fortran compiler [94] for the .NET Framework, and other languages using Microsoft Visual Studio .NET compilers.

4.1.4 Job Submission

The MyCoG GramClient class supports job submission to the Globus gatekeeper service part of Globus Resource Allocation Manager (GRAM) after successful GSI authentication and delegation. Job requests with requisite parameters are constructed in Resource Specification Language (RSL) and submitted using *MyCoG.GramClient.request* (Table 4.3). The GramClient class supports Globus Access to Secondary Storage (GASS) server URL submission to transfer the *stdout* and *stderr* of the remote job. The method *MyCoG.Gram.GramClient.ping* can be used to find whether the user has adequate permissions prior to submitting jobs to the Globus gatekeeper.

4.1.5 Performance

In order to evaluate the performance of MyCoG.NET we have tested it on Local Area Network (LAN) and Wide Area Network (WAN) infrastructures and compared it with the Java CoG toolkit. The upload and download performance are shown for various file sizes in Figures 4.6, 4.7, 4.8 and 4.9.

The performance test was measured within two separate programs written in $C^{\#}$ and Java, respectively, using MyCoG APIs and Java CoG APIs. For the

Table 4.2: Usage of MyCoG GridFTP API from different .NET languages

Usage in FORTRAN

```
use MyCoG%GridFTP
use MyCoG%GridSecurity
use MyCoG%Proxy
.... !  Reading X.509 Certificate from store omitted for brevity
type(GridFTPClient),pointer ::  gftp
allocate(gftp, source=GridFTPClient(serverName, port))
gftp%DELEGATION = DelegationType%LimitedDelegation
call gftp%Authenticate(GetDefaultProxyLocation)
call gftp%Mode(GridFTPClient%ExtendedMode)
call gftp%ParallelUpload(localFile, remoteFile, numberOfStreams)
....
```

Usage in C^{++}

```
using namespace MyCoG::GridFTP;
using namespace MyCoG::GridSecurity;
using namespace MyCoG::Proxy;
.... // Reading X.509 Certificate from store omitted for brevity
GridFTPClient *gftp;
gftp = new GridFTPClient(serverName, port);
gftp->DELEGATION = DelegationType.LimitedDelegation;
gftp->Authenticate(GetDefaultProxyLocation);
gftp->Mode(GridFTPClient::ExtendedMode);
gftp->ParallelUpload(localFile, remoteFile, numberOfStreams);
....
```

Usage in Java

```
import MyCoG.GridFTP.*;
import MyCoG.GridSecurity.*;
import MyCoG.Proxy.*;
.... // Reading X.509 Certificate from store omitted for brevity
GridFTPClient gftp = null;
gftp = new GridFTPClient(serverName, port);
gftp.DELEGATION = DelegationType.LimitedDelegation;
gftp.Authenticate(GetDefaultProxyLocation);
gftp.Mode(GridFTPClient.ExtendedMode);
gftp.ParallelUpload(localFile, remoteFile, numberOfStreams);
....
```

Usage in C$^{\#}$

```
using MyCoG.GridFTP;
using MyCoG.GridSecurity;
using MyCoG.Proxy;
.... // Reading X.509 Certificate from store omitted for brevity
GridFTPClient gftp = null;
gftp = new GridFTPClient(serverName, port);
gftp.DELEGATION = DelegationType.LimitedDelegation;
gftp.Authenticate(GetDefaultProxyLocation);
gftp.Mode(GridFTPClient.ExtendedMode);
gftp.ParallelUpload(localFile, remoteFile, numberOfStreams);
....
```

Table 4.3: Usage of MyCoG GRAM API in C$^{\#}$

```
using MyCoG.Gram;
using MyCoG.GridSecurity;
using MyCoG.Proxy;
 ....
gram = new GramClient(serverName, port); //default port 2119
gram.DELEGATION = DelegationType.LimitedDelegation;
gram.authenticate(GetDefaultProxyLocation);
string gassURL = gram.GetGassURL(); //construct rsl string next
string rslString = ‘‘& (executable = /bin/ls) (arguments = -l)(stdout=’’
                + gassURL + ‘‘//C:/temp/list.txt)’’
gram.request(rslString)
....
```

Table 4.4: GSSAPI authentication

	Authentication Time		
	Minimum (ms)	Maximum (ms)	Average (ms)
JavaCoG (LAN)	1891	2563	2122
MyCoG (LAN)	734	1156	848
JavaCoG (WAN)	3192	5197	3554
MyCoG (WAN)	761	2704	1179

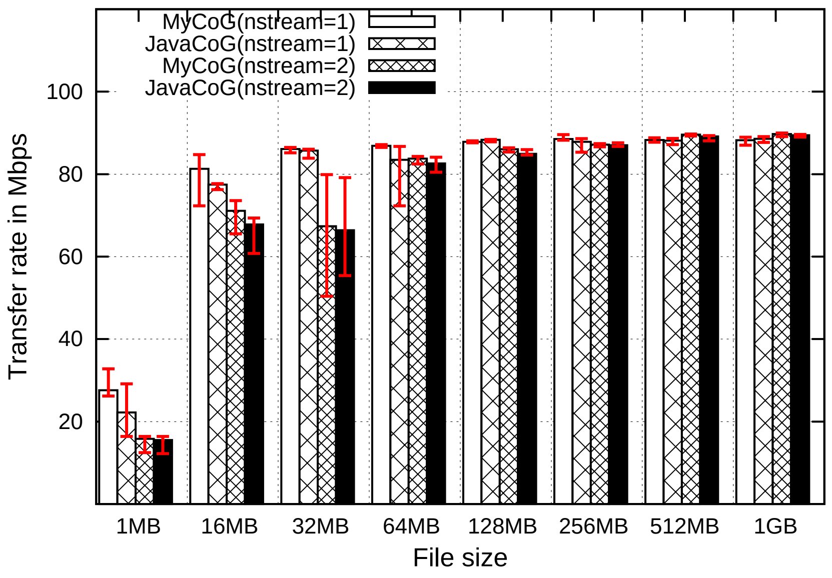


Figure 4.6: Upload performance (100 Mbps LAN)

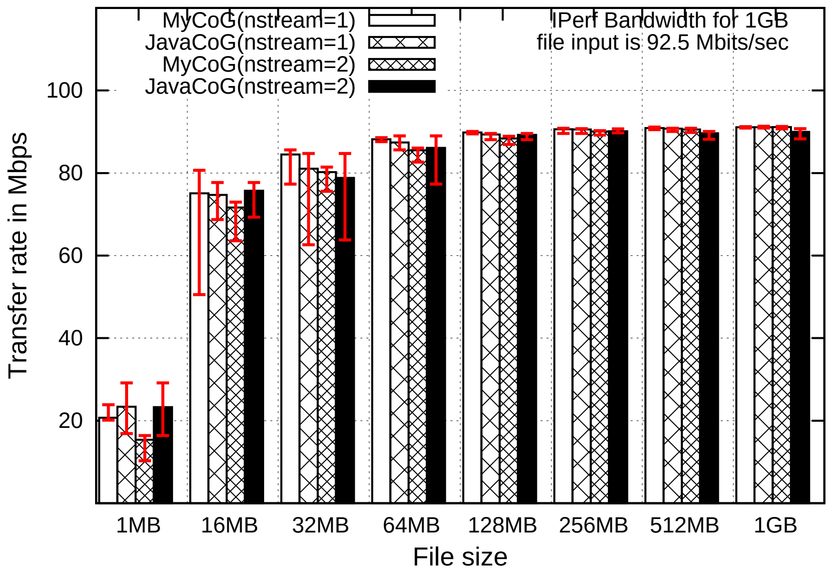


Figure 4.7: Download performance (100 Mbps LAN)

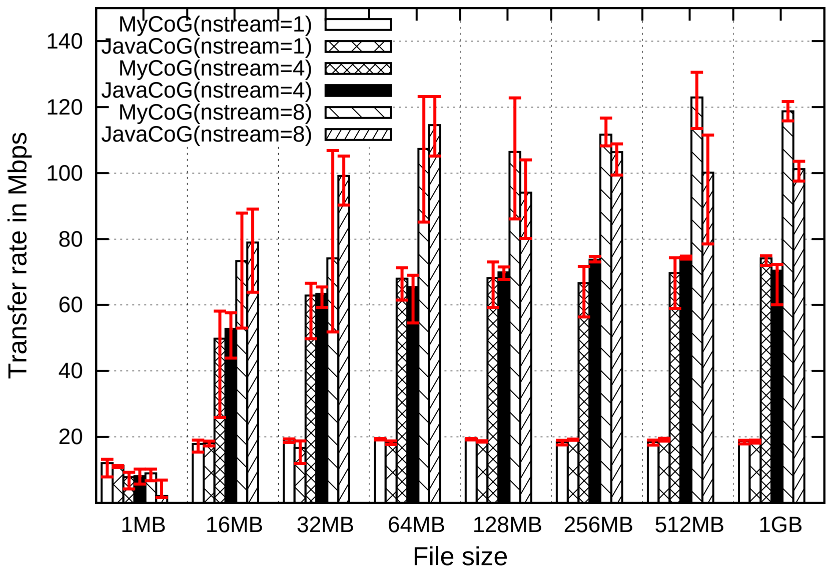


Figure 4.8: Upload performance (WAN)

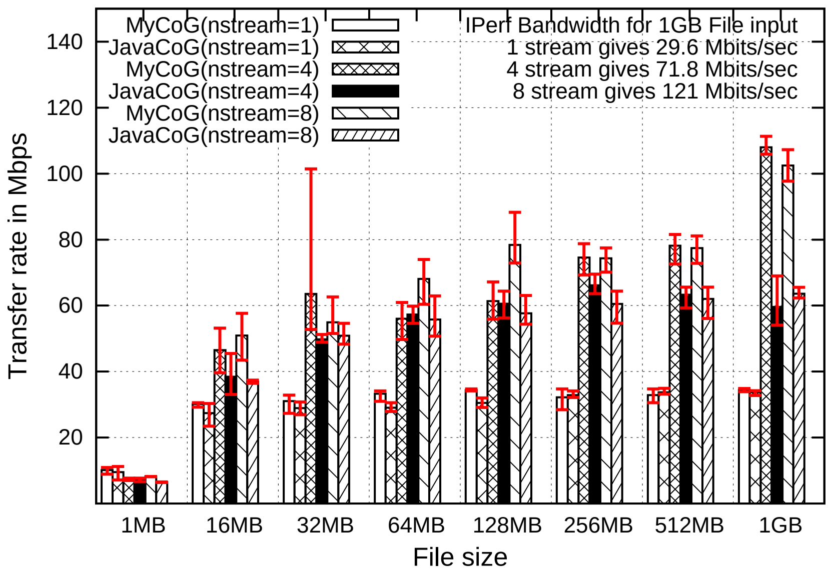


Figure 4.9: Download performance (WAN)

LAN test, the GridFTP clients were run on an Intel Pentium-IV 2.2 GHz Desktop running Windows XP Professional and the GridFTP server was configured on a Dual Intel Pentium-III 450 MHz, Linux system. The client and server are connected over a switched 100 Mbps Ethernet LAN. The WAN test was carried out between the University of Southampton and one of the UK National Grid facilities at the University of Manchester. The local access network bandwidth is limited to 1 Gbps, even though the backbone network runs at 2.5 Gbps. For the WAN test, the GridFTP clients were run on a Intel Pentium-III 733MHz/Windows 2003 machine and the GridFTP server at the University of Manchester on a Dual Intel Xeon 3.06GHz/Linux server. Each file was transferred between client and server twenty times and the minimum, average and maximum bandwidths were recorded. The TCP buffer size for the GridFTP data channel connection was suitably set during WAN experiments. It was calculated based on the bandwidth-delay product and number of streams.

As can be seen from the error bars, large file sizes have significantly lower bandwidth variability between different runs. Also, parallel streams give significantly improved performance in the WAN environment as compared to the LAN tests. In the LAN experiment, both the Java CoG GridFTP and the MyCoG.NET

GridFTP show a maximum download performance of 91 Mbps and upload performance of 89 Mbps for 1 GB file size. The aggregate bandwidth due to parallel streams improves as we increase the file size and number of streams in the WAN tests. As a comparison, iperf [95] tool bandwidth measurements for LAN and WAN are shown in Figures 4.7 and 4.9 respectively.

By default, the Windows XP operating system sets the TCP send and receive buffer size to 8K and can be increased only up to 64K. This value is limiting, especially for file transfers over a WAN. To overcome this limitation we have used Windows Server 2003 for our WAN tests, which allows up to 1 GB for the TCP buffers. The TCP buffer size can be increased in MyCoG programmatically subject to operating system upper limits. This allowed us to draw the maximum achievable bandwidth using parallel streams.

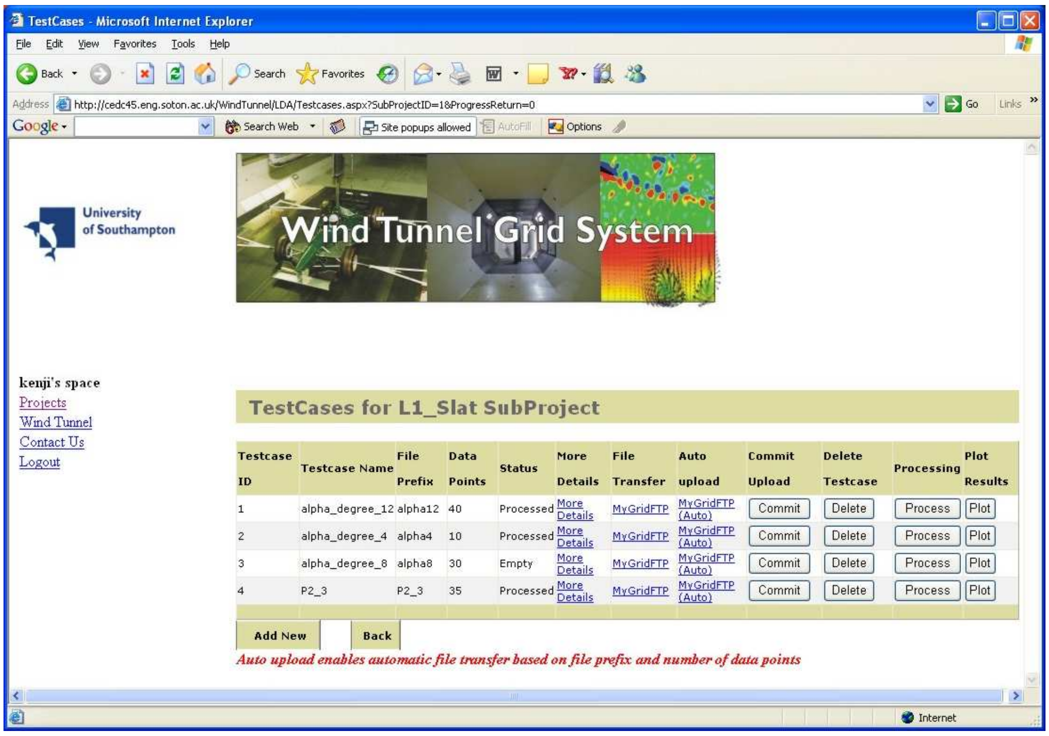
The time taken for Grid server authentication and authorization was measured separately during the runs to compare the Grid security implementations of MyCoG and Java CoG, as shown in Table 4.4. The X.509 user certificate for the LAN experiment is of 1024 bit key length and for the WAN experiment it is of 512 bit. The lesser authentication time of MyCoG could be attributed to its use of the native Windows runtime and security infrastructure (SSPI).

4.2 Wind Tunnel Experiment Workflow

In the following sections, we discuss the Laser Doppler Anemometry (LDA) experiment workflow based on web portal interfaces, followed by an LDA sequential workflow model using Windows Workflow Foundation. Both approaches utilize the MyCoG Grid toolkit for workflow integration onto the Grid.

4.2.1 Portal Solution

Figure 4.10 shows an LDA experimental workflow hosted by a portal providing access to Globus GridFTP and GRAM services. Users are able to login to create a project, create testcases, upload data, initiate processing and generate plots. The portal service is hosted using Internet Information Services (IIS) running on a Windows Server 2003 server. Below are the project metadata tables stored in SQL


TestCases - Microsoft Internet Explorer
http://cedc45.eng.soton.ac.uk/WindTunnel/LDA/Testcases.aspx?SubProjectID=1&ProgressReturn=0
University of Southampton
Wind Tunnel Grid System
kenji's space
Projects
Wind Tunnel
Contact Us
Logout

TestCases for L1_Slat SubProject

Testcase ID	Testcase Name	File Prefix	Data Points	Status	More Details	File Transfer	Auto upload	Commit Upload	Delete Testcase	Processing	Plot Results
1	alpha_degree_12	alpha12	40	Processed	More Details	MyGridFTP	MyGridFTP (Auto)	Commit	Delete	Process	Plot
2	alpha_degree_4	alpha4	10	Processed	More Details	MyGridFTP	MyGridFTP (Auto)	Commit	Delete	Process	Plot
3	alpha_degree_8	alpha8	30	Empty	More Details	MyGridFTP	MyGridFTP (Auto)	Commit	Delete	Process	Plot
4	P2_3	P2_3	35	Processed	More Details	MyGridFTP	MyGridFTP (Auto)	Commit	Delete	Process	Plot

Add New Back

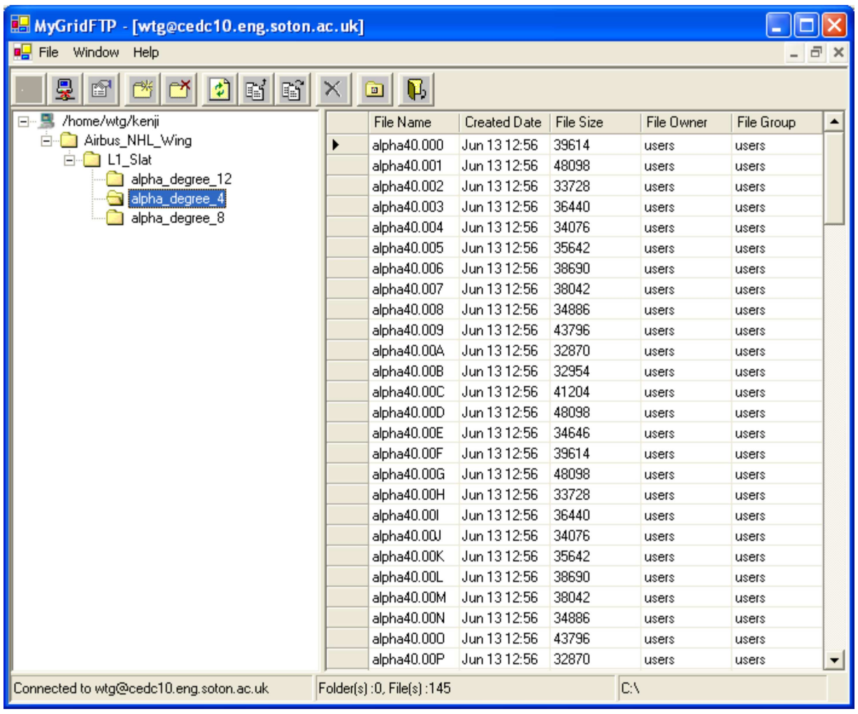
Auto upload enables automatic file transfer based on file prefix and number of data points

Figure 4.10: Wind tunnel grid portal hosting GridFTP, GRAM and PlotWS

Server RDBMS with a brief description of their contents. The application specific *Testcases* metadata vary depending on the Wind Tunnel experiment.

UserAccounts	(username, login, password, roles, user's X.509 subject, homedirectory on the Linux GridFTP server)
Projects	(name, parameters used during data acquisition)
TestCases	(Application specific information for data management and processing)

The wind tunnel Grid portal hosts a MyGridFTP [89] client developed using the MyCoG toolkit for experimental data upload. Since MyGridFTP is hosted using the *ClickOnce* deployment model (discussed in section 4.1.1), it can be invoked from any laptop or desktop on the network. When the user selects the MyGridFTP *http* link, the application is downloaded from the web server and executed on the client machine seamlessly. The advantages include access to up-to-date versions of the



/home/wtg/kenji
Airbus_NHL_Wing
L1_Slat
alpha_degree_12
alpha_degree_4
alpha_degree_8

File Name	Created Date	File Size	File Owner	File Group
alpha40.000	Jun 13 12:56	39614	users	users
alpha40.001	Jun 13 12:56	48098	users	users
alpha40.002	Jun 13 12:56	33728	users	users
alpha40.003	Jun 13 12:56	36440	users	users
alpha40.004	Jun 13 12:56	34076	users	users
alpha40.005	Jun 13 12:56	35642	users	users
alpha40.006	Jun 13 12:56	38690	users	users
alpha40.007	Jun 13 12:56	38042	users	users
alpha40.008	Jun 13 12:56	34886	users	users
alpha40.009	Jun 13 12:56	43796	users	users
alpha40.00A	Jun 13 12:56	32870	users	users
alpha40.00B	Jun 13 12:56	32954	users	users
alpha40.00C	Jun 13 12:56	41204	users	users
alpha40.00D	Jun 13 12:56	48098	users	users
alpha40.00E	Jun 13 12:56	34646	users	users
alpha40.00F	Jun 13 12:56	39614	users	users
alpha40.00G	Jun 13 12:56	48098	users	users
alpha40.00H	Jun 13 12:56	33728	users	users
alpha40.00I	Jun 13 12:56	36440	users	users
alpha40.00J	Jun 13 12:56	34076	users	users
alpha40.00K	Jun 13 12:56	35642	users	users
alpha40.00L	Jun 13 12:56	38690	users	users
alpha40.00M	Jun 13 12:56	38042	users	users
alpha40.00N	Jun 13 12:56	34886	users	users
alpha40.00O	Jun 13 12:56	43796	users	users
alpha40.00P	Jun 13 12:56	32870	users	users

Connected to wtg@cedc10.eng.soton.ac.uk | Folder(s) :0, File(s) :145 | C:\

Figure 4.11: MyGridFTP graphical user interface

application (if the version to be downloaded is already present on user's computer, then .NET runtime executes the local version), no software installations on the client machine, access to local file systems, a rich interactive client experience and the ability to make network connections to any server. In this LDA example, an auto upload feature based on metadata is supported (*"MyGridFTP(Auto)" http* link in Figure 4.10). An LDA dataset consists of multiple data files with a specific naming pattern as discussed in section 3.2. The metadata driven data upload is useful as the user does not have to select the individual data files as the entire dataset can be uploaded automatically.

Similarly, the processing can be initiated using the "Process" link in the user's project page and the results get imported into database tables. The PlotWS [96] web service is used for generating X-Y plots (Figure 4.12) for user analysis.

A workflow integration approach leveraging Windows Workflow Foundation that supports reuse and hierarchical composition of activities is presented next.

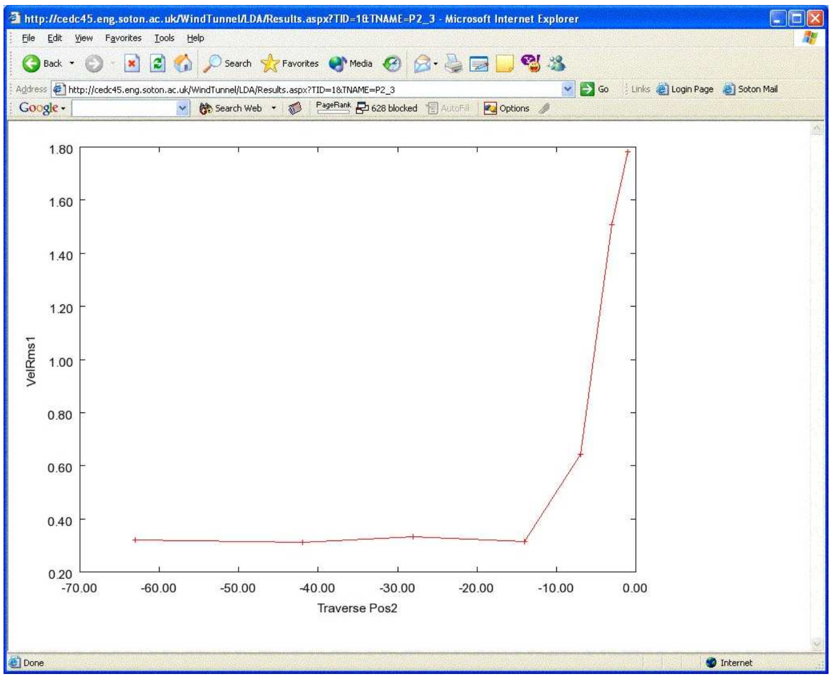

http://cedc45.eng.soton.ac.uk/WindTunnel/LDA/Results.aspx?TID=1&TNAME=P2_3 - Microsoft Internet Explorer
VelRms1
Traverse Pos2
1.80
1.60
1.40
1.20
1.00
0.80
0.60
0.40
0.20
-70.00
-60.00
-50.00
-40.00
-30.00
-20.00
-10.00
0.00

Figure 4.12: 2D LDA Plot using Plot Web Service

4.2.2 Workflow Architecture leveraging Workflow Foundation

We have seen the Grid integration requirements of wind tunnel experiments from Chapter 3. The workflow architecture addressing those requirements is shown in Figure 4.13. The workflow architecture is based on Windows Workflow Foundation - a component of .NET Framework 3.0 as described in detail in section 2.4.3.

In order to enable activities and workflow reuse, the architecture supports workflow composition based on three activity hierarchies: 1. Windows Workflow Foundation Activities. 2. MyCoG.NET-based Grid activities to access Globus services. 3. Experiment-specific activities for upload, processing, results and so on. The user can compose workflows from these activity sets or derive to override any of them depending on the application requirements.

The workflow can be hosted on the client or on the server. In client-controlled hosting, the workflow runtime runs as part of the *host process* on the user's PC. This requires the user to leave the *host process* running until the workflow finishes, but does not require access to a workflow server. The running workflow can be monitored from the client. In the second case, the user deploys their workflow

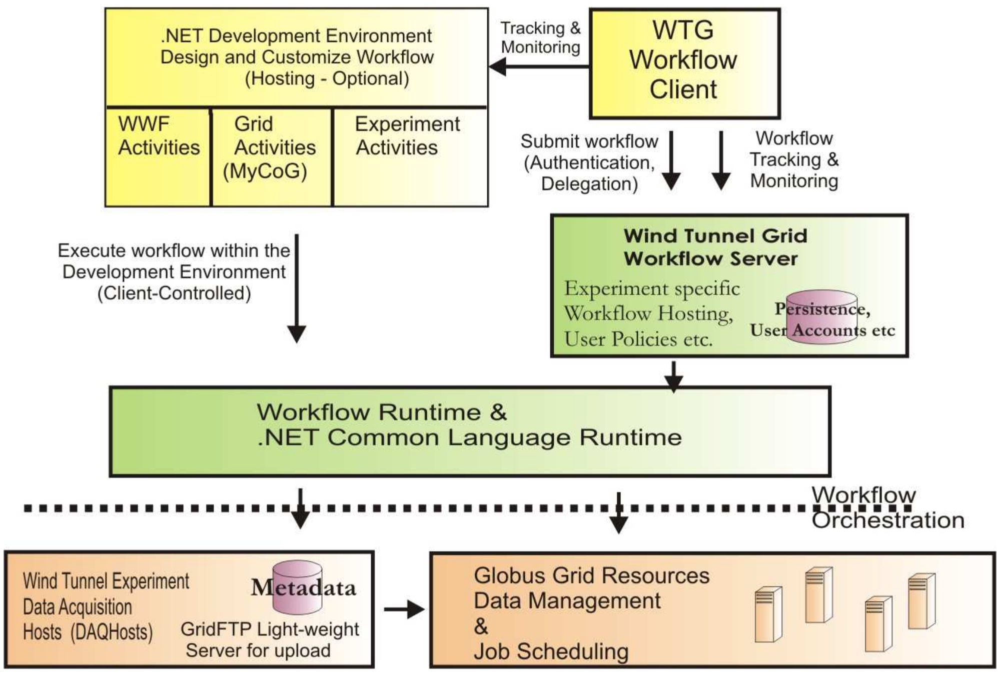

.NET Development Environment
Design and Customize Workflow
(Hosting - Optional)
WWF Activities
Grid Activities (MyCoG)
Experiment Activities
Tracking & Monitoring
WTG Workflow Client
Submit workflow (Authentication, Delegation)
Workflow Tracking & Monitoring
Execute workflow within the Development Environment (Client-Controlled)
Wind Tunnel Grid Workflow Server
Experiment specific Workflow Hosting, User Policies etc.
Persistence, User Accounts etc
Workflow Runtime &
.NET Common Language Runtime
Workflow Orchestration
Wind Tunnel Experiment Data Acquisition Hosts (DAQHosts)
Metadata
GridFTP Light-weight Server for upload
Globus Grid Resources
Data Management
&
Job Scheduling

Figure 4.13: Wind tunnel experimental workflow architecture

for hosting to the wind tunnel grid workflow server after successful Grid Security Infrastructure (GSI) [92] authentication and delegation of user credentials. The wind tunnel Grid workflow server maintains user account information. A separate *host process* is instantiated for the user's workflow and the runtime is started. The user is able to disconnect from the server without interrupting the running workflow, and can check its status at a later time.

The generic wind tunnel workflow activities are: *WTWInit* - initializes a wind tunnel workflow server hosting process for the user; this is the first activity in any wind tunnel experimental workflow. *UserNotification* - Customized user notification on the state of the workflow (workflow completion or failure).

4.2.3 Experiment-specific Activities Design

The importance of experiment-specific customization has been discussed as part of the experimental workflow requirements in chapter 3. Figure 4.14 shows the class hierarchy depicting how experiment-specific activities can be derived and customized by the user. New experimental workflows can be realized by adding

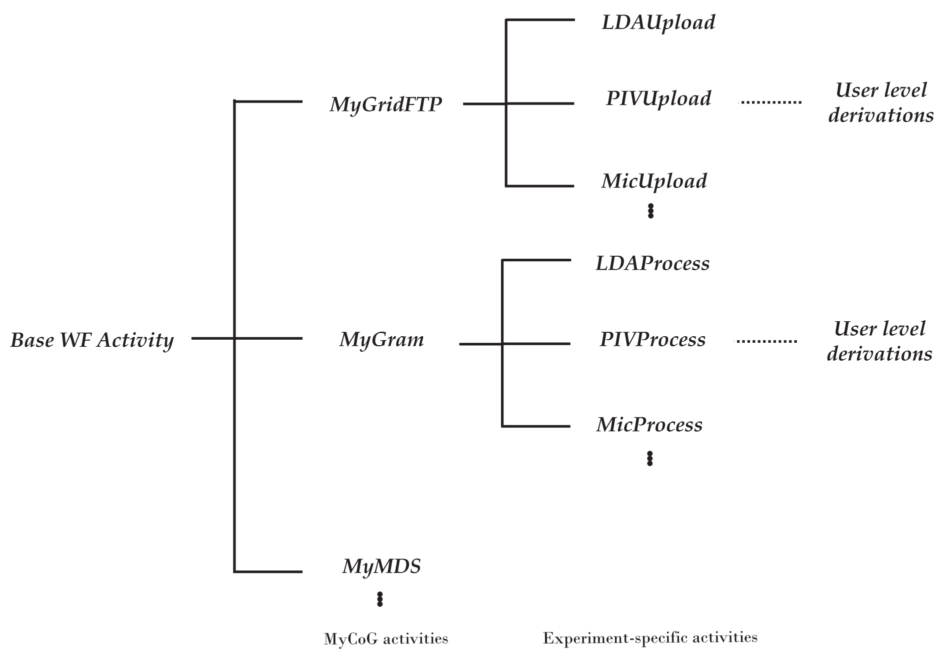


Figure 4.14: Activity class hierarchy

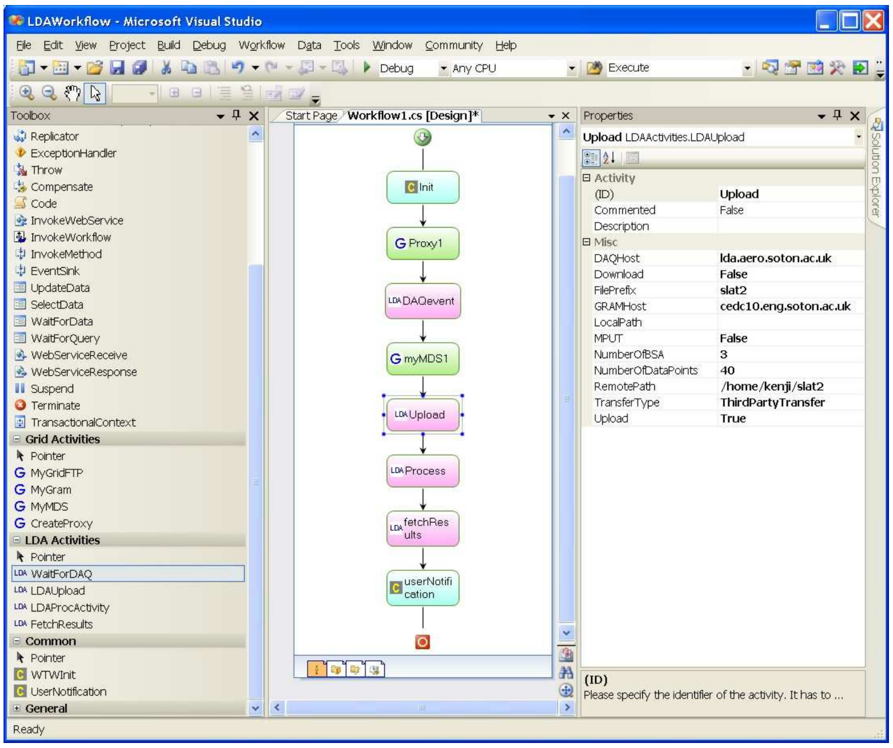
customized activities for that experiment.

Globus Grid activities: The *MyGridFTP, MyGram* and *MyMDS* activities use MyCoG to access Globus resources. These grid activities have generic properties such as server name, port number, user's X.509 certificate hash, and specific properties (e.g, *MyGridFTP* has TransferType, LocalPath, RemotePath, MPUT etc.). They are further customized for individual experiments.

4.2.3.1 Activities for LDA Experiment

LDAWaitForDAQ: The WaitForDAQ is an event driven activity customized for the LDA experiment. On completion of the data acquisition, this activity verifies the raw data files for completeness and transitions the workflow to the next activity.

LDAUpload: This activity is derived from *MyGridFTP* and has specific input properties for the experiment (data acquisition hostname, number of data points, number of burst spectrum analyzers). Some properties are initialized at workflow design time with default values and others received as input from the *host process*. The experiment specific properties enable automatic uploading of raw data files from the



Upload LDAActivities.LDAUpload

Activity	
(ID)	Upload
Commented	False
Description	
Misc	
DAQHost	lda.aero.soton.ac.uk
Download	False
FilePrefix	slat2
GRAMHost	cedc10.eng.soton.ac.uk
LocalPath	
MPUT	False
NumberOfBSA	3
NumberOfDataPoints	40
RemotePath	/home/kenji/slat2
TransferType	ThirdPartyTransfer
Upload	True

(ID)
Please specify the identifier of the activity. It has to ...

Figure 4.15: Wind tunnel experiment workflow

data acquisition host to a Gram server, as can be seen from section 3.2.

LDAProcess: This is a GRAM activity derived from *MyGram*. It has similar properties to LDAUpload, and supports scheduling the LDA processing code to the GRAM gatekeeper service.

FetchResults: This activity, derived from *MyGridFTP*, transfers results from the Gram host to the Windows Workflow server and to the user's desktop.

Similar experiment-specific activities for other experiments, such as the Particle Image Velocimetry (PIV) and microphone arrays experiment, can also be developed. This approach enables an aerodynamicist to compose the experimental workflow using customized activity sets, thereby reducing their application development time. The experiment activities themselves can be further customized by the user to change, for example, part of the processing logic.

Figure 4.15 shows a sequential wind tunnel application workflow designed using Grid workflow activities that are customized for an LDA experiment.

While the sample workflow presented here is in the context of a single organization, the ability to perform Globus authentication, GRAM job submission and GridFTP from within the workflow enables multi-site, multi-organizational scenarios to be successfully created and executed. Aggregation of experimental testing from different wind tunnel sites is well catered for within this implementation.

As can be seen from the two approaches described, the portal solution offers users with a significant capability compared to the previously existing bespoke, multi-platform, manual processing system, but it provides limited flexibility for customization. Also, the individual workflow steps are user driven. Whereas the workflow approach leveraging Windows Workflow Foundation can be sequential/event-driven and the activities are easily customizable.

4.3 Discussion

By leveraging .NET, MyCoG supports multi-language programmability as compared to other Commodity Grid toolkits. This enables many of the existing legacy scientific applications and libraries written in languages such as FORTRAN, to be integrated into the Grid. Existing Commodity Grid toolkits and Grid APIs provide interfaces to a particular Grid middleware such as Globus. The Simple API for Grid Applications (SAGA) [32] specification is an Open Grid Forum standardization effort to provide common interfaces to multiple Grid middleware from different programming languages. SAGA in a Grid environment is often compared to Message Passing Interface (MPI) in a cluster environment that is portable across platforms and network hardware. The SAGA effort does not include any workflow related APIs. It is possible to define abstract workflow APIs based on hierarchical activity model to be part of the SAGA specification. As shown with LDA workflow example in section 4.2.3, the hierarchical activity model enables easy domain-specific extensions.

The Karajan workflow framework [97] that is currently under development provides a task library using the Java CoG Kit to interface to Globus Grid middleware. The approach we have discussed in this chapter differs by providing a hierarchy of workflow activities, enabling reuse of the workflow components. *This* *allows new experiments to be easily added to the system or an existing one to be customized, which is important for domain-scientists so they can concentrate on their science rather than the underlying plumbing.*

While MyCoG.NET and the workflow activities are supported mainly under the Windows platform, their extension to other .NET platforms can be achieved using the Mono .NET environment that is available on Windows, Linux, Solaris, HP-UX, Mac OS and BSD operating systems. Also, the Grid workflow activities are currently supported for Globus Toolkit Version 2 (GT2) services. The future development would require Web services based activities to access GT4 services.

4.4 Summary

In this chapter, we have described the motivation, architecture and performance of MyCoG.NET, a multi-language CoG Toolkit for the .NET Framework to access Globus (GT2) Grid services. The seamless integration of multiple source languages by the .NET runtime makes MyCoG usable from many programming languages. We have used MyCoG APIs for .NET within FORTRAN, C^{++}, $C^{\#}$ and Java programs to demonstrate the multiple language support. Performance of MyCoG.NET is comparable to the Java CoG, and shows improved authentication performance by using the native client-side security infrastructure.

The ability to compose and execute workflows in a Grid context is important to end users. We highlight how the use of a generic off-the-shelf workflow framework, in conjunction with the MyCoG.NET toolkit, can support varied application requirements. We have demonstrated MyCoG.NET usage by implementing real-world scientific workflows in wind tunnel applications, using Windows Workflow Foundation. We have also discussed how customized workflow activities can accelerate Grid application development. The application example we have chosen to demonstrate the customized workflow approach is an experiment one. But this approach can be utilized in long running computational simulations as well.

In summary, the following are the key contributions of the workflow integration approach discussed in this chapter.

- *Extending Grid service access to new platforms:* One of the main objectives of the Grid is to integrate heterogeneous resources. MyCoG.NET is the first commodity Grid toolkit enabling Windows applications running under the .NET platform to become an integral part of the global Grid. This also involves extending *Grid security* for authentication and authorization.

- *Multi-language programmability:* By leveraging .NET, MyCoG enables access to Globus Grid services from many languages. For example, an existing legacy scientific application written in FORTRAN can be easily integrated into the Grid.

- *Hierarchical activity model for workflow composition:* The application workflow activities follow an object oriented hierarchical model with basic activities specialized for the Grid, and customized activities for experiments. There are two possible development roles in this scenario - A Grid workflow developer writes new Grid/experiment activities, and domain-scientists compose application workflows and customize the experiment-specific activities. This model is of importance in an environment such as wind tunnels to support multiple experiments where the addition of new experiments with experiment-specific customizations is required.

- *Demonstration of real-word scientific workflow:* The approach has been demonstrated with a Laser Doppler Anemometry (LDA) experiment utilizing workflow activities customized for LDA.

Chapter 5

Workflow Integration based on Federated Database Services

The majority of scientific applications in the Grid rely on file systems for data management with very limited use of Relational Database Management Systems (RDBMS). In cases where RDBMS is used, it is often exploited as a query engine to retrieve metadata and results. In this chapter, we present an approach based on federated database services and show how database systems capabilities can be leveraged to realize an end-to-end wind tunnel experiment workflow. We also highlight the advantages in database-centric approach to scientific data management.

Storing large volumes of scientific data using flat files may offer performance advantages over databases at present. But, some of the database systems unique capabilities listed below may offer more benefits, flexibility, robustness and greater control in large-scale scientific data management.

- procedural language stored procedures and functions
- transactional messaging
- native XML types and XML Web Services
- publish/subscribe replication
- and other traditional strengths – SQL interface to select/insert/delete/update, transactions support to guarantee data integrity, backup/recovery, fine-grained secure access to tabular data (row-level or column level) and so on.

The development of some of these new database capabilities are driven by the business market, and the ability to take advantage of this ongoing development effort is important to enable sustainable approaches to scientific data management in the Grid environment.

When the raw data and metadata are managed together in an RDBMS, it becomes easier to keep them synchronized. Further, multiple versions of the processing code can be managed inside the database just as the tabular data as cataloguing custom high-level language libraries/assemblies are supported. Database federation can help heterogenous data produced at different geographical locations to be managed, and provide the user with a single logical view. The individual database instances are autonomous and any of them temporarily being unavailable does not affect their interactions. Our architecture takes this approach while integrating the geographically distributed wind tunnel sites. There are three logical databases – *sites*, *master*, and *workers*. The *site* databases operate independently in the federation importing the raw data at sites and sharing through publish/subscribe replication with the *master*. The *master* along with a set of *workers* manages user's custom data processing code and are responsible for data management, processing and analysis. They communicate by means of reliable transactional messaging, which enables fault-tolerance to the communication in the event of site node, worker node or even the master temporarily being offline due to various reasons. Although database federation as an approach to data integration [18] can support functions such as query optimization, the issue we address in our approach is geographical separation of data sources, be it within campus or across organizations.

Considering the changing database systems landscape, they may be viewed as *database operating systems* [16] into which one can plug subsystems and applications. The importance and the issues in integrating database systems into the Grid environment has been studied in detail in [17]. Different scientific applications in fields such as High Energy Physics [98], Earth Sciences [99] and Geosciences [100] have already been demonstrated using database-centric approach in a Grid environment. The work presented in this chapter differs from previous approaches by providing an end-to-end workflow exclusively using the database capabilities.

With the growing interest in XML Web Services in scientific Grids, and with databases beginning to support native XML types and XML Web services, we can expect the role of databases in Grid environments to grow in importance.

The reference implementation we discuss in this chapter is based on Microsoft SQL Server 2005 for database activities and Windows Workflow Foundation for workflow integration. With similar database capabilities supported by popular database systems, we believe it is possible to implement using other database systems as well.

The remainder of the chapter is organized as follows. Section 5.1 covers some of the recent developments in databases that are relevant to our approach. In Section 5.2, we describe a generic design approach based on database systems to meet scientific workflow requirements. In Section 5.3, we present the federated database architecture for wind tunnel experimental workflow. In Section 5.4, we discuss the implementation details of the database activities that enable workflow integration. In Section 5.5, we show how user could compose, run and monitor workflows based on database activities. Section 5.6 presents discussions in the context of related works and other applications.

5.1 Recent Database Trends

The capabilities of database systems are increasing and their architectures are undergoing continuous change. Some of the features that provide new possibilities to scientific application development are discussed below.

- **Language runtime:** Many popular database systems now host language runtimes supporting high-level language stored procedures, functions, triggers, and user-defined data types. For example, Microsoft SQL Server 2005 hosts the Microsoft .NET Common Language Runtime (CLR) [101]; the Java Virtual Machine (JVM) and .NET CLR are supported in Oracle [102] and IBM DB2 [103]. This enables scientific applications to manage both data and the processing code in databases. The implementation approach discussed in section 5.4 leverages SQL Server CLR integration feature enabling user to register compute-intensive code written in any of the CLR languages (C++, Java, C#, and so on).

- **Native XML Support:** With XML becoming a data type, storing XML documents, validating them against a schema, and querying based on XQuery expressions are all part of the core XML functionalities built into popular database systems [104, 105, 106]. This feature is useful in processing XML message exchanges between Grid services and to store semi-structured scientific data in XML format.

- **XML Web Services:** With the increasing interest in XML Web Services, database systems [107, 108, 109] are beginning to support web services hosting inside the databases, eliminating the need for external hosting containers or web servers. This would enable Web Services Resource Framework (WSRF) [36] based, or similar, Grid services to be exposed directly from the databases. The WS-Security [40] is an OASIS standard supporting PKI, Kerberos and SSL security models. As WS-Security is already supported [110, 111, 112] by database systems, it will be easy to realize the adaptation and interoperability of the emerging Grid security models based on WS-Security inside database systems.

- **Transactional messaging:** Asynchronous and reliable messaging between database instances are possible in present day database systems(SQL Service Broker [113] or Oracle streams [114]). We have utilized Microsoft SQL Server 2005 Service Broker for service level interactions which is discussed in section 5.4. Service Broker objects include queues, dialogs, message types, contracts and services. These objects can be created using regular CREATE, ALTER and DROP Data Definition Language (DDL) commands. The messages from the transmit queue of the local database instance to the receive queue of the remote database instance can be transferred inside a transaction making the message transfer reliable. This database feature can be exploited for developing reliable Grid services.

- **Replication:** The publish/subscribe model in replication allows tables, stored procedures or any other database objects to be published. Different replication styles determine when and how the data reaches the subscriber. For

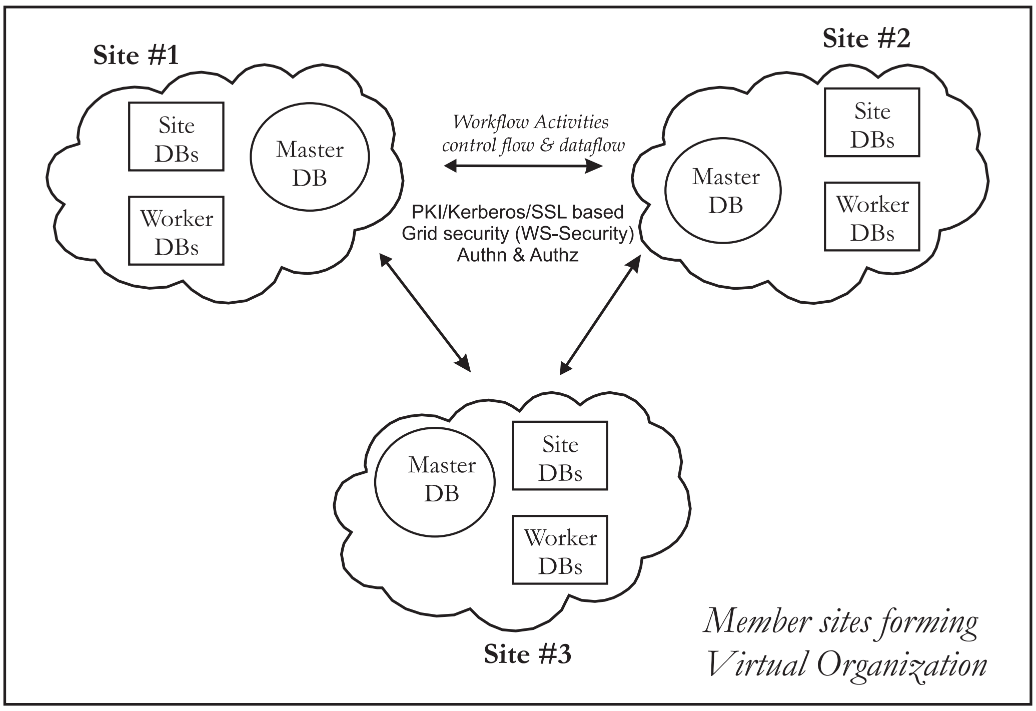

Site #1
Site DBs
Master DB
Worker DBs
Workflow Activities control flow & dataflow
PKI/Kerberos/SSL based Grid security (WS-Security) Authn & Authz
Site #2
Site DBs
Master DB
Worker DBs
Master DB
Site DBs
Worker DBs
Site #3
Member sites forming Virtual Organization

Figure 5.1: Federated database workflow approach in the Grid

example, transactional push-style replication moves data to the subscriber in near-realtime. Database replication can be utilized when scientific applications have to deal with distributed data and the availability of data is to be ensured in more than one location.

5.2 Design Approach

Figure 5.1 presents the high level design of the federated database workflow approach that meets the data-driven scientific workflow requirements by leveraging the database features discussed in the previous section. Each member sites in a virtual organization (VO) hosts a *master* database, one or more *site* and *worker* databases. The site DBs are the origin of the data flows and they publish their data to all the subscribed master DBs in the VO using *database replication* for further processing. The master runs workflow services supporting execution of application workflow steps. The user's custom processing code is registered with the master and workers as a *high-level language stored procedures*. The master load balances the workflow activities by submitting requests to workers through *transactional mes-*

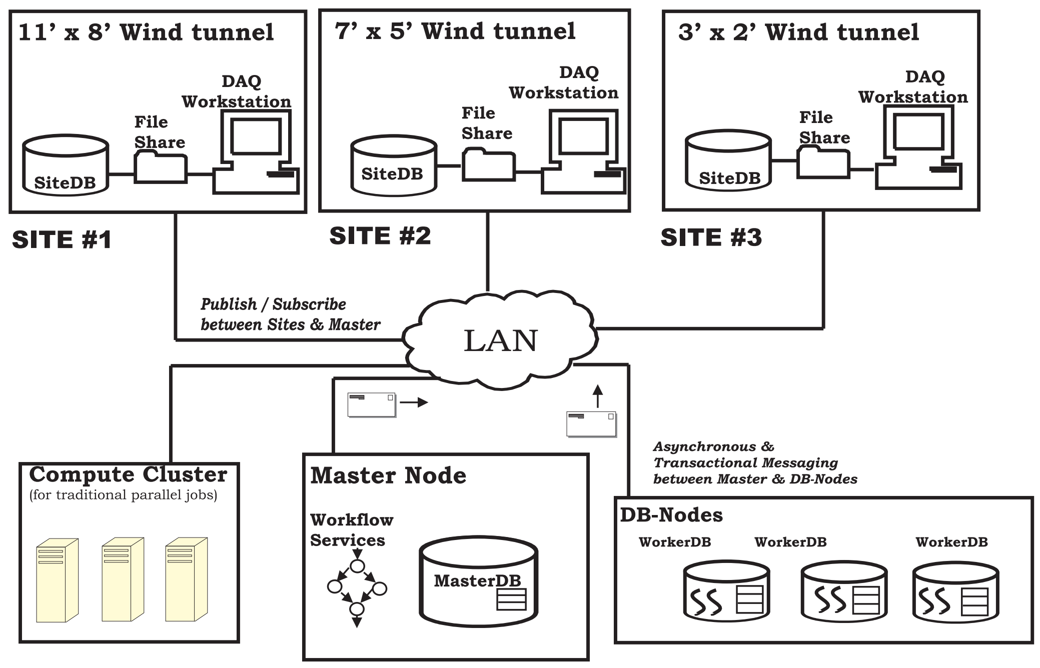


Figure 5.2: Federated database architecture for wind tunnel workflow

saging. The results from workers are combined at the master as a tabular data for further analysis and user queries. Grid security is crucial for virtual organizations support. With native *XML Web Services*, WS-Security support is already part of database systems as discussed in the previous section. Hence, it will be possible to host the emerging WS-Security based OGSA Authentication and Authorization [115] security framework inside database systems.

5.3 Architecture

In this section, We discuss the detailed architecture of our wind tunnel experiments application example based on the design approach presented in the previous section. Figure 5.2 shows a federated database architecture for a typical multi-site wind tunnels which is deemed as mission critical system. The University of Southampton has three main wind tunnel facilities ($11' \times 8', 7' \times 5'$ and $3' \times 2'$) spread over the campus, housing heterogenous, specialized experimental hardware and software for academic and industrial research as discussed in Chapter 3. The high volume of data generated from multiple experiments are to be transferred from the data acquisition system to a suitable network location where users can carry out further processing and analysis.

There are three logical database instances participating in the federation:

- *Site databases (SiteDB)*: Considering the importance of timely data movement and near-realtime requirements, the SiteDB publishes the experiment data tables to the MasterDB using transactional and push-style replication. This ensures immediate transfer of experimental data to the master as soon as the data imported into SiteDB from the data acquisition system.

- *Master database (MasterDB)*: This maintains user and application tables, and publishes them to sites and worker databases. It subscribes to experimental data from all the sites. The master node also runs workflow services for users to register, run and monitor their application workflow.

- *Worker databases (WorkerDB)*: This set of database instances is managed as a cluster of nodes. It carries out the processing work assigned by the master. It also manages different versions of custom user code for processing.

The database instances in the federation enable a complete end-to-end wind tunnel experimental workflow to be created and executed by hosting a set of database services (*activities*) with master node providing additional workflow services. The master schedules the processing activities from multiple user workflows onto worker nodes for load balancing. Access to other Grid resources, such as compute clusters, is also supported based on Grid workflow activities described in Chapter 4.

Figure 5.3 shows the sequence of messages and data exchanges between different database instances and the user's wind tunnel grid client. The actions labeled with letters A, B, C and D are independent of a particular workflow instance and they can happen at any stage. The users can compose workflows based on database activities, compile into a workflow assembly and submit to MasterDB using workflow services for scheduling (step A). They can also monitor the status of their currently running workflows (step B). They can compile a customized assembly and register it through the assembly management services running in MasterDB (step C1). The master in turn makes the assembly available to WorkerDB for registering and subsequent load balancing of users jobs (step C2). Each assembly is registered with

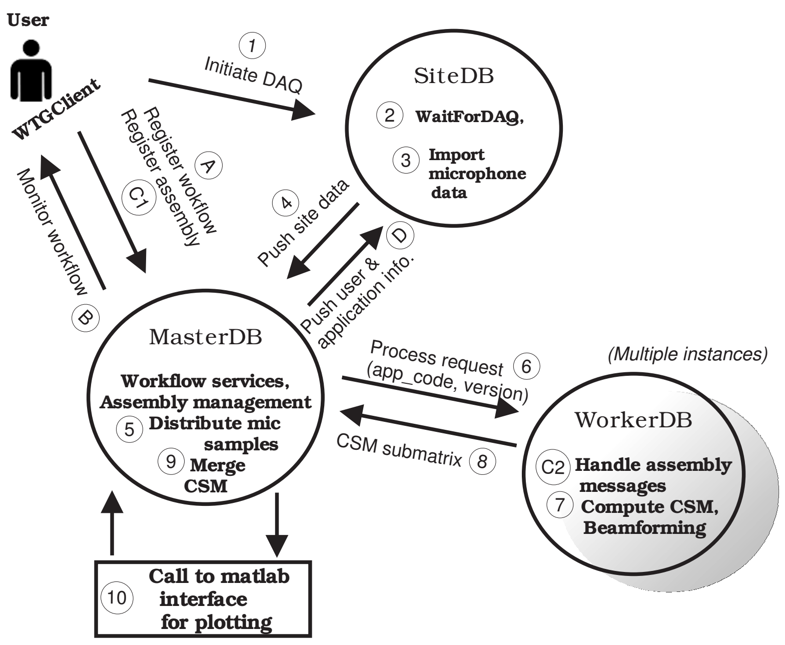


Figure 5.3: Data and message flow for microphone array application

a unique name derived from username, application type and user specified version number. This unique name registration enables user to maintain different versions of algorithms to process the experimental data.

The actual workflow execution starts when the user initiates the data acquisition during an experimental run (step 1). When the data acquisition is over, the service waiting for acquisition to complete (step 2) would change the state of the current experimental run from *"Waiting for DAQ"* to *"DAQ over"*. As the workflow is based on a state machine model, this state change transitions the workflow to the next stage, triggering an import data activity (step 3). Since the application data tables are subscribed at MasterDB and published by means of transactional push publication in sites, the newly imported data is transferred to MasterDB in near-realtime (step 4). Now, with data available at MasterDB and the user's application code registered with master and workers, the data can be distributed for processing (step 5). The processing request to workers comprise user name, application code and version to uniquely identify the assembly for processing (step 6). On receiving the processing request, the worker either invokes the default processing or customized method registered by the user (step 7). The worker node sends the computed results and the status of the processing to master (step 8). The master receives, consolidates and records the results (step 9). The final step involves call to Matlab interface for generating plot and saving it into the results table (step 10).

5.4 Implementation

The implementation details we discuss in this section are based on Microsoft SQL Server 2005, leveraging SQL Service Broker [113] and .NET integration [101] features. We briefly cover these features and discuss the implementation details for the LDA and Microphone array applications.

5.4.1 SQL Server

SQL Service Broker provides asynchronous, reliable and transactional messaging support. Service Broker objects include queues, dialogs, message types, contracts and services. These objects can be created using regular CREATE, ALTER and DROP Data Definition Language (DDL) commands. The messages from the transmit queue of the local database instance to the receive queue of the remote database instance can be transferred inside a transaction making message transfer reliable. The messages can also be routed through an intermediary machine.

The .NET Common Language Runtime (CLR) is integrated into the recent release of Microsoft SQL Server 2005. It enables user to write stored procedures, triggers, functions in any of the CLR languages (C^{++}, Java, $C^{\#}$, and so on). The compute intensive code can be easily written in a high-level language rather than being restricted to Structured Query Language (SQL). The power of SQL can be realized, however, in set oriented queries (SELECT, UPDATE, INSERT, DELETE and so on). An application can take advantage by writing high-level language code for procedural logic and SQL code for queries.

The above features are typical as discussed in section 5.1, and we believe the implementation approach is applicable to other database systems as well.

5.4.2 Wind tunnel database

Figure 5.4 shows the database schema used for the wind tunnel experimental data management.

The list of database objects and their functional descriptions are as follows.

- *Users table:* This table holds the user name, user's role, user's X.509 certificate subject among other user specific details.

- *Applications table:* When the new application is created, the applications table is added with dedicated data, results and run table names.

  Both user table and applications table are published by the master and subscribed by sites and worker nodes.

- *Data tables:* These are application specific tables to store the raw data and configuration information of the experiment. For example, *LDAData* and *MicArrayData* hold data for LDA experiment and microphone arrays experiment respectively. The data tables are maintained at wind tunnel sites and published to master by transactional replication. At sites, the importing and update of the raw data are independent and at the same time the data gets propagated to the master in near-realtime.

- *Run tables:* Every experimental run has an associated data entry in the data tables. The relationship between runs and data is many-to-one. This allows multiple runs with different processing parameters to be associated with a single dataset.

- *Results table:* The results table contains one or more rows for each experimental run. In the case of the LDA experiment, there will be one row for each data point (the total number of data points is represented by *NPoints* column in *LDAData* table), and in the case of the microphone array experiment there will be one row for each beamforming frequency (*FreqArray* column in *MicArrayRuns* table is an array of different frequencies to be used for beamforming).

  The run and results tables are maintained in the master.

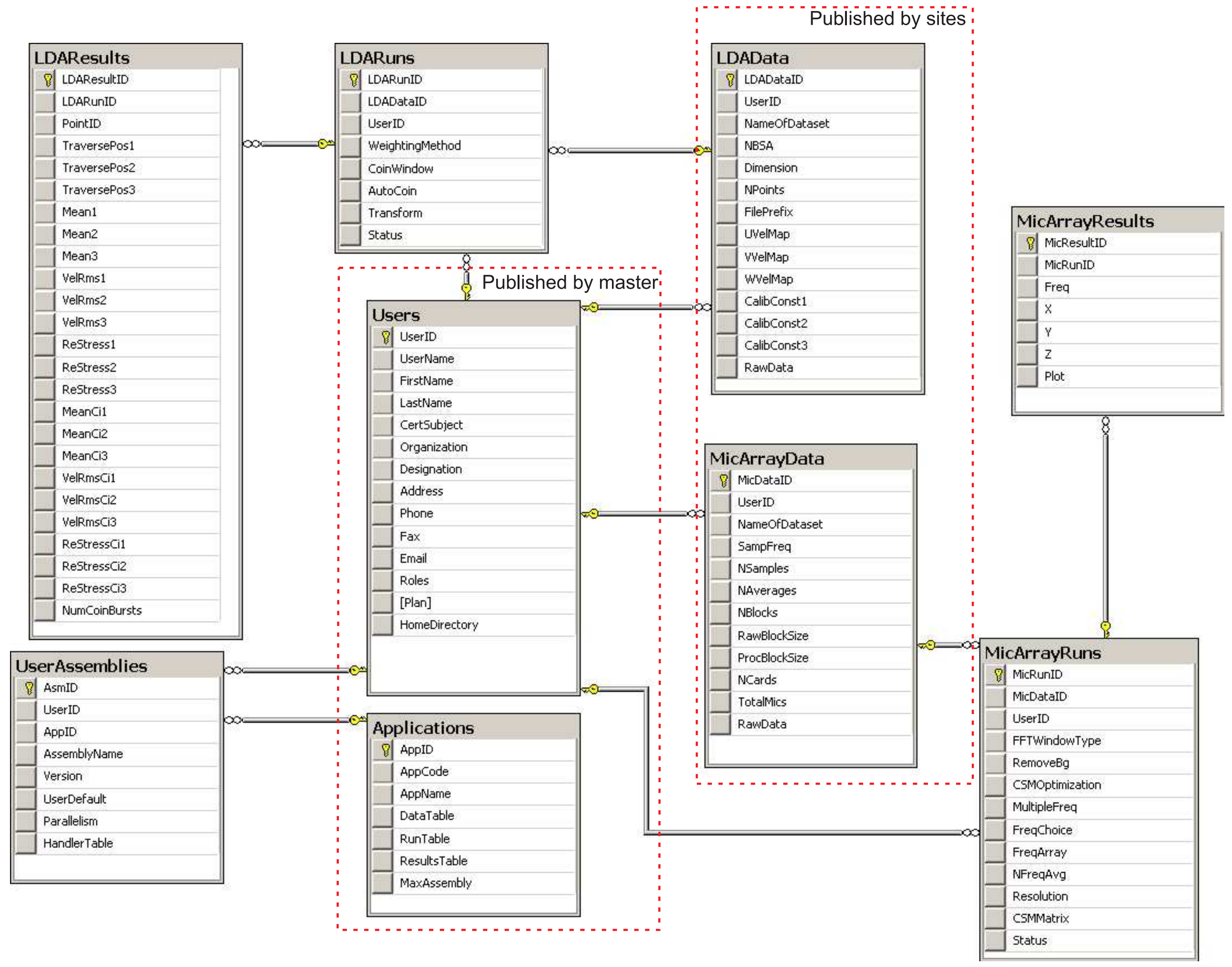


Figure 5.4: Database schema for wind tunnel data management

Table 5.1: An illustration of a transactional message exchange

Master database	Worker database
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`
`SET @Message = <CSMComputeRequest>...`	`WAIT FOR(`
`BEGIN DIALOG @conversationHandle`	`RECEIVE TOP(1)`
`  FROM SERVICE [MasterService]`	`  @mesg_type = message_type_name,`
`  TO SERVICE [WorkerService]`	`  @Message = message_body`
`  ON CONTRACT [WorkerContract]`	`  FROM [WorkerQueue]`
`SEND ON CONVERSATION @conversationHandle`	`  WHERE conversation_handle =`
`  MESSAGE TYPE [CSMComputeRequest]`	`       @ConversationHandle`
`  (@Message)`	`);`
`COMMIT`	`COMMIT`

- *User assemblies table:* This table is maintained at master and worker nodes. It has one entry for each assembly registered by the user. As will be discussed in section 5.4.3.1, the user can register different versions of an assembly and the signature of their custom processing methods are stored in a lookup table (*HandlerTable* column) for later invocation.

- *Stored procedures:* The service code for database activities are written as CLR stored procedures in three different assemblies, namely, site assembly, master assembly and worker assembly and they are registered in *SiteDB, MasterDB* and *WorkerDB* respectively. The classes that are part of these assemblies are shown in Figure 5.5. The main stored procedure in *SiteDB* is *ImportMicData.* The stored procedures that run at *MasterDB* are *Register / Remove / ReinstallAssembly, AddMicArrayRun, MasterCSMScheduler* and *Beamforming.* The *MasterCSMScheduler* instantiates *ParallelCSMScheduler* when the cross spectral matrix computation is performed in parallel. At the worker the main stored procedures are *WorkerCSMScheduler, Register / Remove / ReinstallAssembly* and *Beamforming.* The LDAConfig and LDAData together are serialized and stored as Binary Large Objects (BLOBs) into the LDAData table (*RawData* column in LDAData table). Similarly, MicConfig and MicData are also serialized and stored into the MicData table.

- *Service broker objects:* The service broker objects include messages, contracts, queues and services and they are created using regular SQL DDL commands.

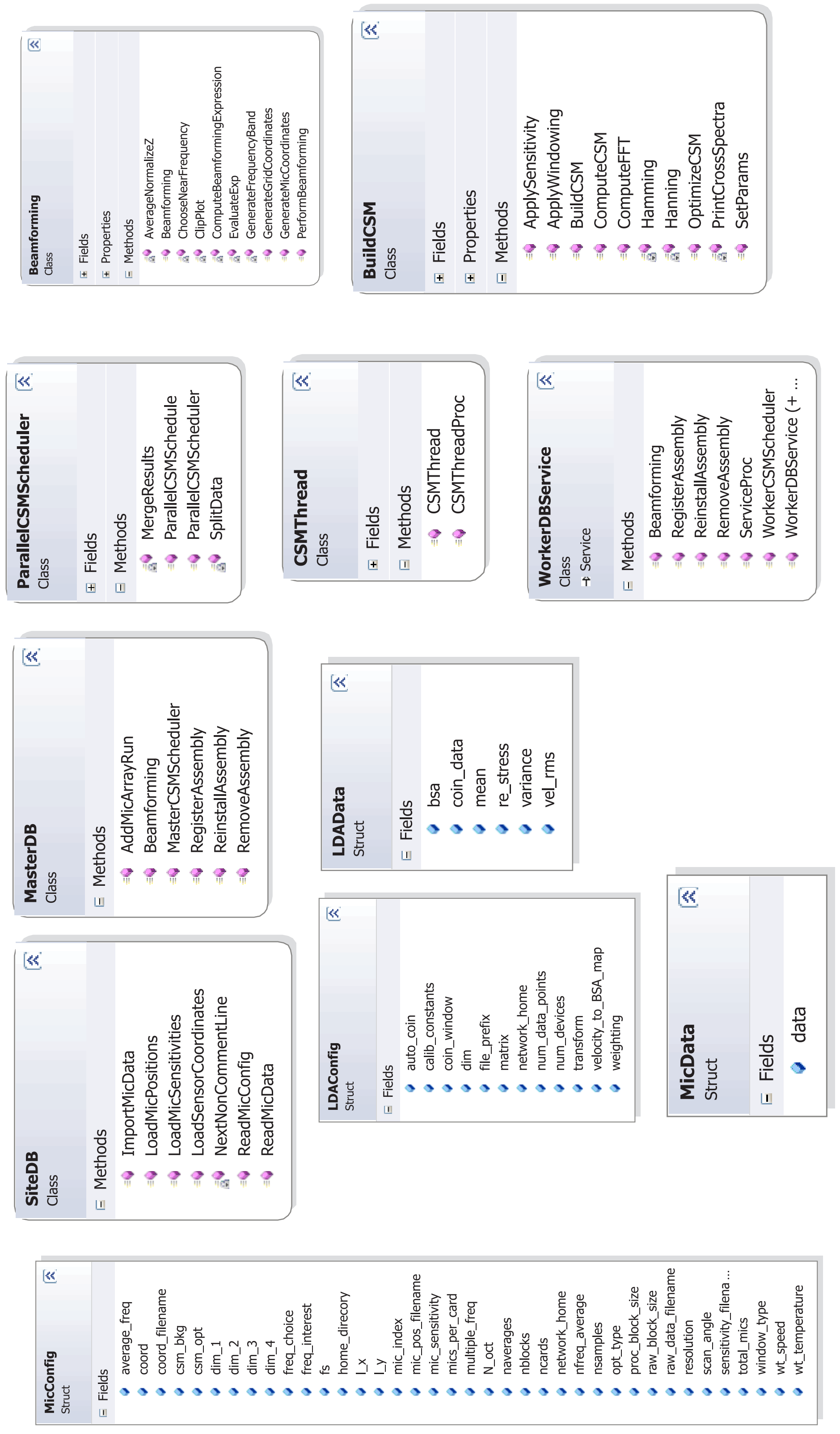


Figure 5.5: Classes managed at sites, master and worker

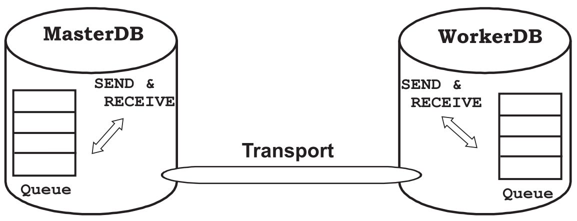


Figure 5.6: Transactional messaging

The message types can be binary or an XML with a validation schema. The contract specifies the types of messages that can flow between a sender and receiver. As shown in Figure 5.6, queues are placeholders for messages and central to *reliable, asynchronous and transactional messaging*; on new message arrival the activation stored procedure (for example, WorkerServiceProc in the following *Queue* declaration) is invoked. A service is an endpoint that participates in a conversation. The master and worker communicate with the help of broker objects. Table 5.1 shows how master would send a CSMComputeRequest message to worker. The following SQL commands show how different service broker objects are created in worker databases.

Message types:

```
CREATE MESSAGE TYPE RegisterAssemblyRequest VALIDATION = WELL_FORMED_XML;
CREATE MESSAGE TYPE ReinstallAssemblyRequest VALIDATION = WELL_FORMED_XML;
CREATE MESSAGE TYPE RemoveAssemblyRequest VALIDATION = WELL_FORMED_XML;
CREATE MESSAGE TYPE AssemblyReply VALIDATION = WELL_FORMED_XML;
CREATE MESSAGE TYPE CSMComputeRequest VALIDATION = NONE;
CREATE MESSAGE TYPE CSMComputeReply VALIDATION = NONE;
CREATE MESSAGE TYPE BeamformingRequest VALIDATION = NONE;
CREATE MESSAGE TYPE BeamformingReply VALIDATION = NONE;
```

Contract:

```
CREATE CONTRACT WorkerContract(
       RegisterAssemblyRequest SENT BY INITIATOR,
       ReinstallAssemblyRequest SENT BY INITIATOR,
       RemoveAssemblyRequest SENT BY INITIATOR,
       AssemblyReply SENT BY TARGET,
       CSMComputeRequest SENT BY INITIATOR,
       CSMComputeReply SENT BY TARGET,
       BeamformingRequest SENT BY INITIATOR,
       BeamformingReply SENT BY TARGET
);
```

Queue:

```
CREATE QUEUE WorkerQueue
        WITH STATUS = ON,
        RETENTION = OFF,
        ACTIVATION(
               STATUS = ON,
               PROCEDURE_NAME = WorkerServiceProc,
               MAX_QUEUE_READERS = 4,
               EXECUTE AS SELF
        );
```

Service Procedure:

```
CREATE ASSEMBLY WorkerAssembly FROM 'path/to/assembly' WITH PERMISSION_SET = SAFE;
CREATE PROC WorkerServiceProc
        EXTERNAL NAME WorkerAssembly.[WTG.DBLibrary.WorkerDBService].ServiceProc
```

Service:

```
CREATE SERVICE WorkerService ON QUEUE WorkerQueue (WorkerContract);
```

- *Replication Configuration:* Considering the near-realtime requirements of wind tunnel experiments, transactional replication model [116] as shown in Figure 5.7 is used. The incremental changes to the data at sites are propagated to the master as they occur. Replication configuration uses standalone programs called agents to distribute data. The log reader agent that runs at a distributor moves the transactions marked for replication from the transactions log on the publisher to the distribution database. The distributor in our test is configured on the same machine as the publisher (but it can be configured to run on a different machine as multiple publication sites can have a common distributor). The distribution agent makes the periodic changes available at the publisher flow to the subscriber. The following SQL code example

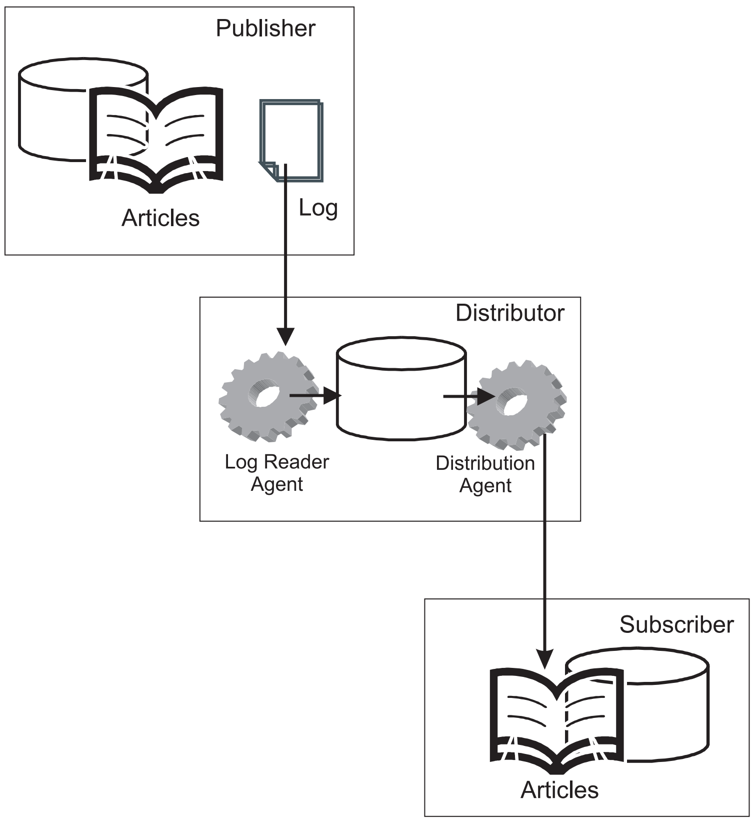


Figure 5.7: Transactional replication model

shows how site publications are created and an article (*MicArrayData* table) is added to the publication using system stored procedures *sp_addpublication* and *sp_addarticle*.

```
SET @PubName = N'SiteData';
EXEC sp_addpublication
       @publication = @PubName,
       @description = N'Publication of raw data from wind tunnel sites to master',
       @status = N'active',
       @allow_push = N'true';

SET @table = N'MicArrayData';
EXEC sp_addarticle
       @publication = @PubName,
       @article = @table,
       @source_object = @table,
       @source_owner = @schemaowner,
       @type = N'logbased';
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<RegisterAssembly>
  <UncPath>path_to_assembly</UncPath>
  <AssemblyName>name_of_assembly</AssemblyName>
  <UserName>user_name</UserName>
  <AppName>application_code</AppName>
  <Version>assembly_version</Version>
  <UserDefault>yes_or_no</UserDefault>
  <Parallel>number_of_workers</Parallel>
</RegisterAssembly>
```

Figure 5.8: Register assembly request in XML

5.4.3 Database activities

The following are the database activities that form the basis for the development of customized wind tunnel experiment workflows.

5.4.3.1 Assembly Activities

Assemblies are libraries containing a user's application-specific processing code, which can be registered with the master and worker databases. The user can choose the default processing functions available or write a custom one, register with the master, and in turn, with the worker. Once the assembly is registered, the public interfaces (public class, static functions, data) are available for access from service stored procedures.

The assembly activities RegisterAssembly, ReinstallAssembly and RemoveAssembly will internally translate into SQL DDL statements CREATE ASSEMBLY, ALTER ASSEMBLY and DROP ASSEMBLY, respectively. The essential properties associated with these assembly activities are user, application type (LDA or Microphone) and version.

Figure 5.8 shows the XML instance of the register assembly request message type. On receiving this message, the service procedure at the worker invokes the register assembly message handler. The handler function catalogs the assembly with a unique name derived from user name, application code and version. The user's custom processing methods are identified and a lookup table (*HandlerTable* column in *UserAssemblies* table) comprising the metadata for function invocation is also recorded into *UserAssemblies* table. When a processing message is sent from the master, the lookup table for this message is selected from the *UserAssemblies* table based on user name, application code and version. Since the user functions are declared *public*, they can be called directly using the metadata available in the lookup table.

5.4.3.2 Process Activities

The process activities execute application-specific code either from the default assembly or from a user registered one. When a processing message is sent from the master, the corresponding lookup table is retrieved to invoke the user's custom processing function. For example, the microphone array processing involves a cross spectral matrix (CSM) computation and beamforming step. The CSM computation step can be executed in parallel. The blocks of raw microphone array samples can be split among multiple threads equally (Figure 5.9) to improve the performance. During data acquisition, the samples are acquired in blocks each having 2^n samples. For example, if the total number of blocks is 100 and the block size is 2048, there would be 204800 samples to be partitioned. The partitioned data is sent to the workers depending on the number of processors in individual worker nodes. A dual CPU worker would receive two partitions of the data as it can schedule two threads. Considering the example again for a 4 thread case, a dual CPU worker node would receive 102400 samples with each thread working on 51200 samples or 25 blocks. The threads running on worker nodes compute the cross spectral matrix in parallel on part of the data. The computed cross spectral matrices are sent to the master and it does an averaging operation to form a final cross spectral matrix and stores in the *MicResults* table. The cross spectral matrix is used during beamforming.

Once the CSM is computed, the beamforming step is executed once for each frequency. In the case of multiple frequencies, different frequency values can be sent to worker nodes to generate the beamforming plot in parallel. The output of individual beamforming step are three square matrices X, Y and Z of *resolution* size with grid point values for plotting.

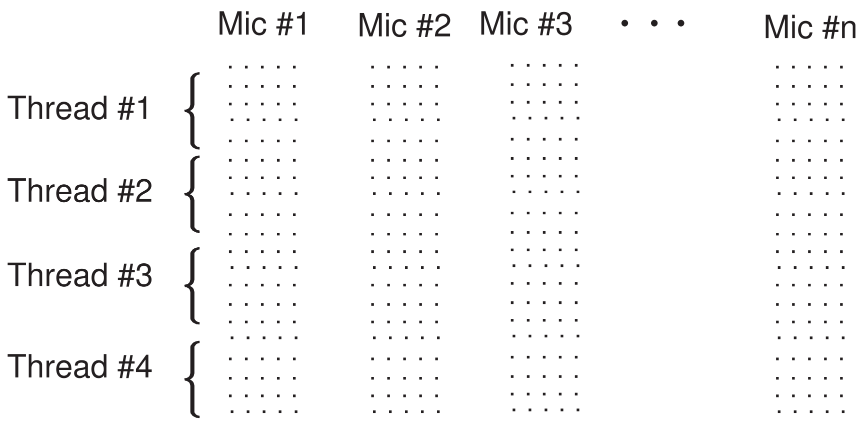


Figure 5.9: Partitioning of microphone samples for parallelism

Table 5.2: Microphone array processing (CSM timings)

Dual P-III 1GHz CPU; 1GB RAM	Load	Split	Merge	CSM step	CSM Total	Beamforming step (single frequency)
Sequential (Command line)	30.231	-	-	89.400	89.400	99.043 (288.149*)
2 threads (Command line)	28.587	0.996	1.499	51.287	53.782	99.043 (288.149*)
Sequential (SQL CLR)	12.253	-	-	89.635	89.635	292.553*
2 threads (SQL CLR)	12.529	1.321	1.604	54.423	57.348	292.553*
4 threads (SQL CLR) on two nodes	11.850	13.971†	84.496‡	30.899	129.366‡	292.553*

** Without using matrix-vector multiplication optimizations* *All timings in seconds*
† Time to split, serialize & send
‡ Time due to queuing delay & merge

5.4.3.3 Plot Activity

The X, Y and Z arrays computed during the beamforming step are used for plotting. In the case of multiple frequencies, a separate plot for each frequency is created and stored. Matlab has been chosen in order to generate publication-quality scientific plots. The plot function is written in Matlab which takes various arguments for plotting. The invoking of Matlab code from .NET was achieved using the Matlab .NET builder tool [117] which wraps the Matlab function into a .NET class. The plot activity uses the wrapper class to generate the noise contour plot as a *jpeg* or *eps* image and stores it inside the database for user download/visualization.

5.4.4 Microphone Processing: Performance

Table 5.2 shows the cross spectral matrix processing timings for a C$^{\#}$ command line application and for the same code hosted inside SQL Server as a Common Language Runtime (CLR) stored procedure. The two nodes utilized for this test are Dual Pentium III 1 GHz with 1 GB RAM running Windows Server 2003 and SQL Server 2005, and connected over a 100 Mbps LAN. The SQL Server hosts the .NET 2.0 runtime. The raw data samples used for the test were acquired from 56 microphones consisting of 100 blocks each of size 2048 (total samples = 204800) [79]. The load time is the time taken to read the microphone samples into memory for processing. In the command line case, the samples are read from a delimited text file and in SQL CLR case, they are read from the *RawData* BLOB (Binary Large Object) column in *MicArrayData* table. As can be seen from the table, the parsing of samples from text file takes more than double the time of deserializing the raw data BLOB into memory. The SQL CLR cross spectral matrix processing timing is comparable and the overhead due to processing inside database is marginal as can be seen from the Figure 5.10. The split time is the time taken to partition the samples among the threads and the merge time is the time taken to combine the cross spectral matrix received from threads by an averaging operation. The split and merge time for 2 threads of SQL CLR case on a single node is again comparable with the command line timings.

Computation of the beamforming expression involves multiplying the cross spectral matrix with a weight vector for each grid point of the plot. This matrix-vector multiplication can be optimized with specialized Intel Streaming SIMD (Single Instruction Multiple Data) Extension [118] instructions. The optimized command line beamforming timing is obtained using the NMath Core [119] C$^{\#}$ library (which in turn uses Intel's Math Kernel Library). This could not be registered into the database due to the SQL CLR strict versioning policies (we expect this to be resolved in the future). In order to provide a fair comparison, we measured the CSM timings without optimizations (marked with $\star$ in Table 5.2); the SQL CLR timings for beamforming step without the matrix-vector multiplication optimization is comparable to the same command line version.

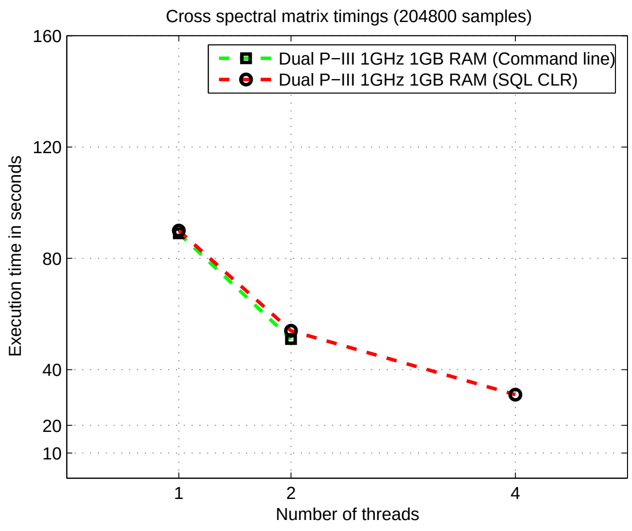


Figure 5.10: Microphone cross spectral matrix performance

The four threads case uses SQL service broker messaging to communicate the part data to the worker node. During the send operation the data is split and serialized. Since the service broker queue supports asynchronous operations, on receiving the reply from the worker the master service procedure merges the results to produce the cross spectral matrix. The overhead due to queueing of the raw data is noticeable. But, this particular test is more to illustrate the advantage of reliably partitioning and load balancing a service using database messaging, than to show any speedup. With multiple users running different experiments producing high volume data, a reliable service to the wind tunnel experimental environment is more important. This is discussed further in Section 5.6. Also, for cases, where the experimental processing is long running due to data volume or the nature of the processing, the queueing overhead can be amortized. Further, the database messages can be routed through a low latency and high bandwidth network technologies, such as, InfiniBand and other high speed interconnects, to improve the performance.

5.5 Workflow Integration

The workflow integration is achieved using Microsoft Windows Workflow Foundation that is part of the upcoming Microsoft .NET development framework 3.0 [14]

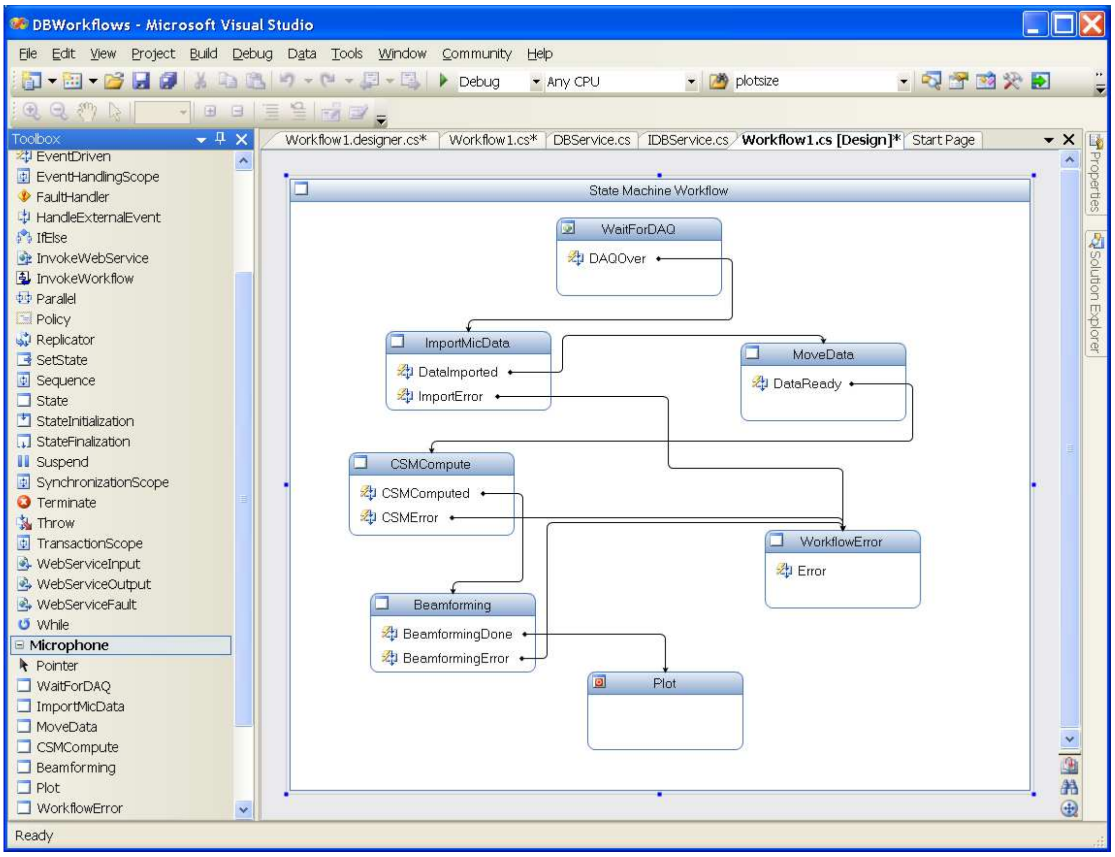


Figure 5.11: Microphone experiment workflow based on database activities

discussed in section 2.4.3. The database activities discussed in section 5.4.3 are wrapped into an experiment-specific workflow activity library for users to compose their experimental workflow and submit to master node for hosting. Figure 5.11 shows the composition of custom microphone workflow based on experiment-specific activities.

The state machine workflow model of Windows Workflow Foundation has been adopted for the development of customized microphone experiment workflows. The initial state of the microphone workflow is *WaitForDAQ* and the final state on success is *Plot* or *WorkflowError* in case of any error during workflow execution. The experiment-specific activities are derived from the *State* activity. The *State* activity is composed of one more event driven activities. For example, the *ImportMicData* state has the *DataImported* event transitioning to the *MoveData* state and *ImportError* event transitioning to *WorkflowError* state. On completion of an event, the *SetState* as part of the event-driven activity sequence transitions the workflow to the next state. The *CSMCompute* and *Beamforming* are workflow state representing processing.

The user composes the workflow, compiles it into an assembly and submits to the master node for hosting. The workflow is scheduled and run at the master node. The workflow activities connect to the master database to execute the corresponding database activities. The state transitions of the workflow are recorded into the run tables of the master database. The user can monitor the submitted workflow instances for their status, to find out whether they have completed successfully, are still running or terminated with an error.

This is similar to the customized workflow approach to Laser Doppler Anemometry (LDA) experiment based on sequential workflow model presented in Section 4.2.2.

5.6 Discussions

The nature and degree of use of Relational Database Management Systems (RDBMS) in scientific data management has been variable. Some of the usage has been to stream data near-realtime, to host scientific services by means of SQL stored procedures, to partition data to improve query performance, to store results, to store metadata and so on. In this section, we discuss related scientific projects highlighting the degree of database systems usage and provide our argument in favor of keeping databases central to the entire experimental workflow.

The NEESgrid framework [6] part of the Network for Earthquake Engineering Simulation project supports instrument integration and exposes domain-specific Grid services for conducting and monitoring distributed earthquake engineering experiments. In terms of the experimental facilities, NEESgrid has some close similarity with wind tunnel experiments, but the emphasis is more on remote access to instruments in a multi-site environment. NEESgrid uses databases for metadata management only. The myLEAD [74] tool part of the Linked Environment for Atmospheric Discovery (LEAD) project provides specialized services for atmospheric scientists to search, store and catalog data objects generated during their investigations. The metadata catalog is managed in RDBMS along with a set of database-stored procedures to expose persistent Grid services. It uses OGSA-DAI as a middleware for client interactions. OGSA-DAI [41] provides Grid service interfaces to different data sources (relational, XML, flat files). The advantages of database management systems in real-world scientific application have been demonstrated in Sloan Digital Sky Survey (SDSS) [120] project. With efficient indexing, join and parallel query operations, a twenty times speedup was achieved as compared to a file-based implementation. MySQL's streaming support has been utilized while hosting archival and real-time Geographical Information Systems (GIS) Grid services in [121]. The BioSimGrid project [122] manages large-scale biomolecular simulation data in flat files and associated metadata in RDBMS (Oracle). It supports simulation data to be deposited into a repository which is then replicated to different sites for retrieval and analysis.

As can be seen from the above applications, the major factors that influence how database systems are utilized in a scientific environment include data characteristics, nature of acquisition, processing requirements and performance. In the case of multi-user facilities such as wind tunnels where different experiments, multiple locations, multiple runs, changing parameters, high volume data and customized processing are the order of the day, it requires an approach that meets this set of demanding requirements.

The emphasis in the federated database approach [123] is on the ability of the local database instances to continue to support local operations autonomously, while they are part of the federation, to provide a set of global operations. Our architecture takes this approach while integrating the geographically distributed sites. The site databases operate independently in the federation sharing information through publish/subscribe replication with the master. Also, the communication between the master and the worker nodes are by means of reliable transactional messaging. Any site node or worker node or even the master not being available temporarily will not affect the global operations. This is of particular importance in wind tunnel operations, which are deemed mission critical. A typical industrial scenario would be for a Formula One racing team. Sites would include the factory, multiple wind tunnel sites, testing and race tracks in different countries, where the network bandwidth and quality of service cannot always be guaranteed. Any of these sites could be offline for a number of reasons, and many are required to operate around the clock. Hence local autonomy and reliability are important for such an application.

The Grid security integration in database systems is crucial for enabling virtual organizations (VOs) support. Some of the key requirements for authentication and authorization [124] as identified by the OGSA security road map include support for different authentication mechanism based on Public Key Infrastructure variants (X.509, PGP [2], AADS [3], SPKI [4] etc.) and Kerberos, support for fine grained (for example, based on X.509 subject) and coarse-grained (for example, based on groups, sites etc.) authorizations, allowing actions based roles/membership of an end entity within a VO and so on. As the building blocks for the emerging Grid services are based on Web Services, the interoperability of the WS-Security implementation between different platforms and vendor systems is an important issue to be resolved to enable VOs from heterogenous sites. The WS-I Basic Security Profile [125], for example, is aimed at promoting interoperability across platforms. WS-Security interoperability between different vendor implementations such as Microsoft Web Services Enhancements (WSE 2.0) and Java Web Services Developer Pack (JWSDP 1.5) has already been demonstrated [126]. Popular database systems already support WS-Security standards, but, implementing interoperable OGSA authentication and authorization using native XML Web Services is a challenging and an important undertaking for the Grid and database research community.

In general, scientific projects keep their non-relational and binary data, for example matrix or image frames, in flat files. If the binary data is stored in tabular form using elementary SQL types, recreating by combining from basic elements is a costly operation. If the data is to be accessed programmatically, say for processing, it is advantageous to store them as binary objects in database systems. In our approach, the wind tunnel raw data is imported as Binary Large Objects (BLOBs)

[2]Pretty Good Privacy
[3]Account Authority Digital Signatures
[4]Simple Public Key Infrastructure

into the database at experiment sites and they get replicated to master node for processing. As can be seen from this study [127], the load performance of read-only BLOB objects are comparable to file systems, whereas write/update operations that result in fragmentation of the BLOB affects the load time. Also, it shows database performance for objects of few mega bytes size is faster than file systems. In our microphone experiment example, the raw data is a read-only object, the BLOB structure is accessed before *CSMCompute* and never gets updated. Similarly the CSM matrix BLOB, once stored, is always read before *Beamforming* step. In this usage scenario, deserializing the BLOB is more advantageous as can be verified by the load time in Table 5.2 when compared to loading the raw data from text file stored in the file system. There is an additional overhead in parsing the floating-point samples in case of a text file. The other advantage in storing the raw data in BLOBs is the ability to support database operations such as backup and recovery. If the raw data is to be searched or only part of it is to be retrieved, for example as shown in the *BioSimGrid* scenario [128], BLOBs are not the suitable option as the whole BLOB is to be deserialized into memory before anything can be achieved. This limitation can be overcome by either storing the data in multiple BLOB columns with additional metadata or in pure tabular form. Also, in cases where the raw data size is more than the maximum BLOB size (typically 2GB) limit in database systems, it has to be either split into multiple BLOBs or preferably stored in file systems. Database systems have the limitation of handling extremely large objects in binary form; with native language runtime hosting, performance improvements in BLOB storage is an important requirement on database systems from scientific applications. The decision of whether to store the binary data in a database or in file systems depends on the data size and access pattern, and it is also application dependent [128].

In typical wind tunnel processing, an aerodynamicist would be interested in changing and customizing the processing algorithms. This is particularly true in a research and development environment. User customization of the processing algorithm, together with an ability to use the default processing steps, is an essential requirement. By taking advantage of the language runtime support inside databases, managing the user customized algorithm is possible in our approach. The Microsoft .NET development environment has an extensive support for languages such as Java, C^{++} and $C^{\#}$, with other language compilers such as Python and FORTRAN also being available, users can program in their language of choice. The user code is registered to run under the user's security credentials to gain authorized access to the dataset and results.

While an aerospace engineer works with an experimental model, he may be interested in comparing and analyzing against results generated from a parallel Computational Fluid Dynamics (CFD) code. The database architecture discussed in section 5.3 and the database schema presented in section 5.4.2 supports this. The parallel CFD code written in MPI (*Message Passing Interface*) or matlab can be submitted to a Globus GRAM Gatekeeper service running on a compute cluster using the *MyGram* workflow activity described in the last chapter. The CFD results can be imported to the appropriate *Results table* (*LDAResults, MicResults, PIVResults* etc.). It is also possible to maintain a separate table to hold computational results for each experiment and link it to the *Run tables* via foreign key relationship. This would allow the engineer to make effective comparison of the computational and experimental results using SQL query language and to have easy access to both of them. Even though our case study is from the field of aerospace engineering, similar requirements exists in other application areas. For example, in LHC experiment, Monte Carlo simulation results will be compared against the experiments (discussed in Section 2.5.1) and NEESgrid requires support for hybrid testing supporting comparison of simulation model and experimental design (discussed in Section 2.5.2). By exposing an application independent schema design interface, the approach to storing experiment data and simulation data together can be generalized to support other application areas.

5.7 Summary

In this chapter, we presented an approach supporting an end-to-end engineering workflow based on federation of databases. The transactional publish-subscribe replication model of the database system has been utilized to achieve site autonomy, and near-realtime and asynchronous data movement operations from experiment sites. The user's customized processing code in addition to the raw data and metadata are managed inside the database. The database instances host database services which are invoked from a state machine model workflow. We have demonstrated this with a reference implementation leveraging the features of Microsoft SQL Server 2005 and Windows Workflow Foundation. The architecture is generic and can be implemented using database systems other than SQL Server, such as ORACLE or IBM DB2 as well.

The resulting benefits are a reduction of the overall turnaround time by providing an easy-to-use, extensible workflow framework, relieving the user of data management issues, and providing a robust and reliable system by using the features typical of commercial database systems, such as replication and transactional messaging. Even though the approach and the implementation discussed in this paper are with reference to wind tunnel experiments, it can be easily extended to other scientific and engineering application with similar characteristics.

The following summarizes the key contributions of the workflow integration approach discussed in this chapter.

- *Database integration in Grid:* How database systems can play an an integral role in a Grid environment to support scientific workflow requirements has been analyzed and an end-to-end real-world workflow leveraging database services has been demonstrated.

- *Federated database services:* How different logical instances of databases (e.g, master, sites and worker) can work together in a federation to address geographical separation of data sources has been shown.

- *Parallelism and Load balancing:* In cases where the processing code is data parallel in nature, how the data can be reliably partitioned for load balancing on available database services has been demonstrated.

Chapter 6

Discussion

In this chapter we will discuss this thesis contributions and how it compares with similar efforts, and present some of the possible further works.

6.1 Contributions

Scientific and engineering applications from different domains ranging from high-energy physics, genomics, computational biology to geophysics, astronomy and aerospace engineering are becoming increasingly data-centric. The earlier efforts, before data management was considered seriously, have been focused on high performance computing aspects of the applications. Now, scientists and engineers have considered Grid as an important platform in order to build an end-to-end system addressing, not just the data management or compute requirements, but the complete application requirements.

In this thesis, we looked into the previous experiment workflow efforts in Grid from various application domains and highlighted the differences between them due to their data, processing and functional characteristics. In multi-user facilities such as wind tunnels, where different experiments with changing experimental hardware, software and parameters are the order of the day, experiment-specific integration is important. The requirements from wind tunnel experiments presented in Chapter 3 support the customized workflow approach which is the basis of the first hypothesis (see section 1.2) . We provided two approaches addressing the unique application requirements to wind tunnel experiments. Our first approach realizes an end-to-end wind tunnel experiment workflow based on Globus Grid services. This approach may be seen as *bottom-up* in that an existing Grid middleware has been customized to meet the application requirements. The second one presents an alternate approach to realize end-to-end wind tunnel experiment workflow completely based on federated database services. In this case, the system is built *top-down* considering the application requirements first.

The two workflow approaches have been demonstrated with two different applications - Grid services approach for LDA experiment and database services approach for Microphone arrays experiment. LDA experiment produces large number of small flat files with shorter and sequential processing code while Microphone experiment produces comparatively more data to be processed by a data parallel algorithm. Hence, the following discussions comparing the two workflow approaches are based on the core functional strengths they derive respectively from Grid and database environment rather than their end-to-end workflow timings.

The contributions from the Grid services approach is in enabling application workflow development based on experiment-specific workflow activities. The experiment-specific workflow activities are derived from the base Grid activities allowing easy customization for new experiments. The extended activities hold enough experimental metadata to help automate the workflow steps and hide the underlying complexity in Grid access from the user. Many of the existing legacy scientific applications and libraries are written in languages such as FORTRAN. The successful integration and interoperability of these legacy scientific application code requires multi-language approach to Grid access toolkit. The MyCoG Commodity Grid Toolkit presented in this thesis enables access to Globus Grid Services from multiple languages (FORTRAN, Java, C^{++}, $C^{\#}$ etc.) under the .NET Framework. The development of MyCoG.NET and the workflow approach based on that supports the second hypothesis (see section 1.2).

Existing workflow solutions are either developed keeping the target application domain in mind [69, 65] or their functional extension to suit a particular application domain requires more work [27, 44, 67]. This contrasts the hierarchical workflow activity design approach to wind tunnel experiments we have presented. Even though our workflow example is specific to the wind tunnel environment, the approach can be easily adopted in other branches of experimental and computational sciences as well.

The advantages of the Grid services approach is open standards, leading to interoperability between different middlewares and flexibility on protocols; for example, GridFTP for bulk data transfer. In the pre-Web Services model, the security and other message exchanges are based on custom protocols that are suited for high performance. XML Web Services has emerged as an accepted practice to build interoperable OGSA-based Grid services. But, there are still challenges to be overcome in making XML based protocols work for scientific computing. For example, the importance of addressing the performance bottlenecks associated with XML processing pipeline due to SOAP/HTTP have been recognized [129, 130]. The communication performance is crucial to certain class of applications with requirements for near-realtime data movement similar to wind tunnel experiments discussed in Chapter 3. Similarly in the Grid approach, the management of metadata generated from the scientific workflow steps is often ad hoc. An effective metadata capture and organization would help in automating the workflow steps reducing the manual intervention as far as possible.

Database systems can offer many advantages to Grid applications: powerful yet simple query language, ability to store raw data, metadata and the processing code, transactional model to ensure data integrity, data mining extensions, transactional messaging and so on. The continuing database architectural improvements [16] are taking the database systems a step closer to meeting the requirements [17] in integrating them onto Grid.

The contributions from our second approach is in supporting a complete end-to-end wind tunnel experimental workflow by hosting a set of services running on database instances in the federation. The transactional publish-subscribe replication model of the database system has been utilized to achieve the local autonomy, the near-realtime and asynchronous data movement operations from experiment sites. The user's customized processing code in addition to the raw data and metadata are managed inside the database. By taking advantage of the native language runtime support, the storing and invoking of different versions of application processing code from multiple user's is also supported. The asynchronous and reliable messaging between database services has been achieved using native SQL constructs supporting

Table 6.1: Comparing the two workflow approaches

	Globus Grid services approach	**Database services approach**
Data management	Flat file store, associations between raw-data and metadata are handled by the application code.	Relational database store, explicit data associations ensuring data integrity, easy search/selection
Data transfer	GridFTP high performance data transfer driven by workflow activity.	Transactional publish/subscribe model data transfer - near-realtime movement.
Processing	Long running jobs can be scheduled using GRAM activity.	Processing threads run inside DB - exploits data parallelism.
Security	Grid security based on GSSAPI.	WS-Security support. Any further work depends on emerging OGSA security model.
Reliability/ Fault-tolerance	Limited support. Workflow state can be persisted. But, workflow activity can fail due to service failures.	Database services are asynchronous and transactional ensuring reliability in the event of system failures.
Extensibility	New workflow activities can be derived from basic Grid activities and customized for the application.	New application schema needs to be added to sites and published to the master.

transactional messaging. The workflow approach based on federated database services and the demonstration of the microphone experiment workflow supports the third hypothesis (see section 1.2).

Table 6.1 compares the two workflow approaches presented in this thesis based on important functionalities. Both approaches have their advantages and limitations. Globus GRAM supports scheduling of compute- and communication-intensive MPI based parallel jobs. Hence, the Grid services approach is preferred for applications with long running processing characteristics with less data management issues. Since the task scheduling is internal to the database systems, the performance of concurrent tasks depends on the database systems scheduling policies and how well it can exploit the underlying hardware architecture. For example, SQL Server 2005 supports mapping of database thread to an operating system thread and it can also take advantage of non-uniform memory access (NUMA) architecture to reduce the memory latency [131]. On the other hand, for applications with high degree of data management and reliability requirements, the database approach is more appropriate. The Grid services and database services approach target different classes of applications; hence, we have based the comparisons between both the approaches on functional issues rather than workflow execution timings.

The GridAnt [132] is an extension of apache ant build system and provides

<gridAuthenticate>, <gridTransfer> and *<gridExecute>* Grid tasks, respectively for GSI authentication, GridFTP file transfer and GRAM job submission using Java CoG APIs. The ant targets are executed sequentially with no support for concurrent execution of workflow activities. The lack of iteration, condition evaluation, management of workflow state, persistence etc. have led to the Karajan workflow framework [97] based on the Java CoG Kit. Karajan, which is currently under development, aims at providing declarative style language in XML syntax for workflow definition and more streamlined approach towards workflow support. The Grid workflow approach we have presented in Chapter 4 supports an extensible model for workflow activity development with specialized activities for Grid which can be further customized for experiments. Also, the workflow can be composed using visual designer, written in XAML or completely coded in high-level languages.

OGSA-DAI [41] provides a service oriented architecture to access data sources such as relational or XML databases. In the OGSA-DAI model, its services are responsible for data access, management and delivery while the other grid middleware services are the consumers of its data and responsible for long-running computations. Some of the previous database-centric approaches in Grid environment have been in fields such as High Energy Physics [98], Earth Sciences [99] and Geosciences [100]. The work described in Chapter 5 differs in that we provide an end-to-end experiment workflow solution exclusively using the database capabilities. For example, integrating geographically separated data sources, metadata driven experiment workflow, storing application-specific data types, ability to run processing/analysis against them, storing results and the support for traditional visualization packages integration.

Despite the success of database systems in business applications its use in scientific environments is still limited. We believe the experience from our work on database systems has provided some insight into how present day database system with its changing capabilities can be integrated into a Grid environment. Similarly, some of the areas database systems need improvements to become more applicable in scientific environments include support for scientific data types (numeric arrays, matrices etc), support for long-running computations (MPI style jobs), fine-tuning bulk data movement operations, interoperability at all components between different vendor database systems (not just at the data import/export or replication operations), scripting support for all possible database operations (essential for workflows) and support for emerging Grid/Web Services security models to enable virtual organizations.

6.2 Further Work

There are many interesting opportunities for further work in both the approaches. In order for workflows to support different Grid middleware framework, an abstract set of workflow APIs independent of application domains and underlying workflow engines are important. This can be developed adhering to GGF/OGF Simple API for Grid Applications specification. The MyCoG Grid APIs have been developed for pre-WS components of Globus Grid services and it can be extended to include appropriate tools and APIs for the emerging Web Services based Globus components.

Unlike business computing, the message exchanges in scientific applications are often data intensive. This requires support for routing messages through low-latency and high bandwidth networks. In the federated database approach discussed, a high performance messaging layer can be developed so that fine tuning the performance of message exchanges between database instances would be possible. Similarly, compute intensive applications require support for task level parallelism and a communication model that can support broadcast, group communication etc; if compute/communication intensive application code is to be run under database systems, an effective framework supporting the task scheduling and communication model suitable for parallel computing are to be developed. One possible approach, for example, would be to utilize operating system threads external to database instances to handle long-running compute messages.

The SQL/MM data mining extension of the SQL:1999 standard defines standard interfaces to data mining algorithms. Since the experimental raw data, metadata and results are all stored inside database systems, it is possible to host data mining services exploiting data mining extensions.

In order for heterogenous databases to interoperate in a Grid and between virtual organizations, the services hosted inside the databases are to be compliant to emerging standards for the Grid fabric layer. *What is the best model to host web services inside the database? How can the database services wrapped effectively into XML Web Services? How do database services publish and discover?* The answers to some of the above research questions will help database systems better integrate in a Grid environment. A synergy between emerging Grid computing standards and database systems would be of great mutual benefit for the two communities.

Chapter 7

Summary and Conclusions

Scientific and engineering experiments involve people and facilities that are distributed across organizations. During the last decade there has been a growing interest in Grid computing and it is increasingly being adopted in many scientific projects. The Grid integration requirements of experiment workflows from different application domains vary due to their data, processing and functional characteristics. In this thesis, we looked into some of the experiment workflows and their integration challenges onto Grid, and presented two approaches to supporting end-to-end workflows considering wind tunnels as an application example.

In Chapter 2, we have reviewed the related works in the field of Grid and databases, and we have also discussed some of the experiment workflows from different application domains and highlighted the differences in their integration approaches. We have presented some of the unique integration requirements of wind tunnel experiments in Chapter 3. The application specific workflow steps for wind tunnel experiments substantiates the first hypothesis.

The first contribution from this thesis is the approach presented in Chapter 4 to an experimental workflow based on Globus Grid services. The experiment-specific activities are derived from base Grid activities enabling new experiments to be easily added. The workflow activities hold enough metadata to automate the workflow steps. The multi-language approach of the MyCoG Grid toolkit enables integration of many of legacy scientific application code written languages such as FORTRAN. The development of MyCoG under .NET and the workflow approach leveraging MyCoG demonstrates the suitability of commodity environment to scientific workflows as identified in second hypothesis.

The other contribution from this thesis is the federated database services approach to experimental workflows presented in Chapter 5. This approach shows how present day database systems can host set of services to realizing end-to-end experiment workflows. Federated database instances host services providing support for local experiment site autonomy, data movement, running of user's customized processing algorithm and plotting/visualization. This workflow approach supports the third hypothesis by realzing an end-to-end scientific workflows based on database services.

Both the experiment workflow approaches can be easily extended to other domains. The idea of having common grid workflow activities for data transfer and processing, and annotating with metadata for specific experiments can be easily applied in other application workflows as well.

The metadata capturing and their association with raw data, analysis and visualization of results, searching and mining for patterns will all be easy if widespread adoption of database management systems is achieved in scientific Grids. This requires more integration efforts similar to OGSA-DAI from the Grid communities. Similarly, there are many file systems features scientist's might prefer over database systems until those short-comings are addressed. Some of them include easy-to-use language APIs supporting SQL queries, support for scientific and user-defined data types (arrays, matrices, complex numbers etc.), support for parallelism, support for workflow system integration and simple command line interfaces to the whole system.

Appendix A

Processing details

A.1 Laser Doppler Anemometry

The LDA raw data files contain samples at a single traverse position for a particular Burst Spectrum Analyzer (BSA) channel. It follows the following format:

```
BSA_STATUS & BSA_PARAMS header for channel 1
burst
burst
...
...
```

Each of the i^{th} burst encodes the arrival time (AT), transit time (TT) and the velocity/frequency of the i^{th} sample. The untransformed velocity component along x direction is given by

$$U_x = f_{D1} \frac{\lambda}{2 \sin \theta/2} \tag{A.1}$$

where λ is the laser wave length, θ is half the angle between the laser beams and f_{D1} is the Doppler frequency for BSA channel 1. Similarly, the untransformed velocity components U_y and U_z can also be given for y and z directions respectively.

The transformed velocity components are calculated by using the transformation matrix as below.

$$\begin{pmatrix} U \\ V \\ W \end{pmatrix} = \begin{pmatrix} \frac{1}{2\cos\psi/2} & 0 & \frac{1}{2\cos\psi/2} \\ 0 & 1 & 1 \\ \frac{1}{2\sin\psi/2} & 0 & \frac{1}{2\sin\psi/2} \end{pmatrix} \begin{pmatrix} U_x \\ U_y \\ U_z \end{pmatrix} \tag{A.2}$$

where U, V and W are the transformed components, U_x, U_y and U_z are the measured components, and $\psi/2$ is the angle between z axis and the first axis.

The following are some of the output values due to LDA processing.

Mean value:

$$\overline{U}_{weighted} = \frac{\sum U_i \Delta t_i}{\sum \Delta t_i} \qquad \overline{U}_{unweighted} = \frac{\sum U_i}{N} \qquad i = 1, 2, 3, \cdots, N \tag{A.3}$$

where Δt_i is the transit time of i^{th} particle inter arrival time $(t_i - t_{i-1})$ and N is the number of samples.

Variance:

$$\sigma^2_{weighted} = \overline{u^2} = \frac{\sum_i (U_i - \overline{U})^2 \Delta t_i}{\sum_i \Delta t_i} \qquad \sigma^2_{unweighted} = \overline{u^2} = \frac{\sum_i (U_i - \overline{U})^2}{N} \tag{A.4}$$

RMS Value:

$$\sigma = \sqrt{\overline{u^2}} \tag{A.5}$$

The variance values $\sigma^2_{weighted}$ and $\sigma^2_{unweighted}$ are utilized to calculated the weighted and unweighted rms values respectively.

Cross moments:

$$\overline{uv}_{weighted} = \frac{\sum_i (U_i - \overline{U})(V_i - \overline{V}) \Delta t_i}{\sum_i \Delta t_i} \tag{A.6a}$$

$$\overline{uv}_{unweighted} = \frac{\sum_i (U_i - \overline{U})(V_i - \overline{V}) \Delta t_i}{N} \tag{A.6b}$$

A.2 Microphone Arrays

The first part of the microphone array processing is the Cross Spectral Matrix (CSM) computation. CSM is an $N \times N$ hermitian matrix A, where N is the number of microphones and each element $A_{nn'}$ represent cross-spectral component for microphones n and n'. Since $A_{nn'} = A^*_{n'n}$ only the upper or lower triangular matrix needs to be computed. The diagonal elements A_{nn} are auto spectral components of the microphone. The Cross Spectral Matrix A is given by,

$$A = \langle \vec{u}(t) \vec{u}^*(t) \rangle \tag{A.7}$$

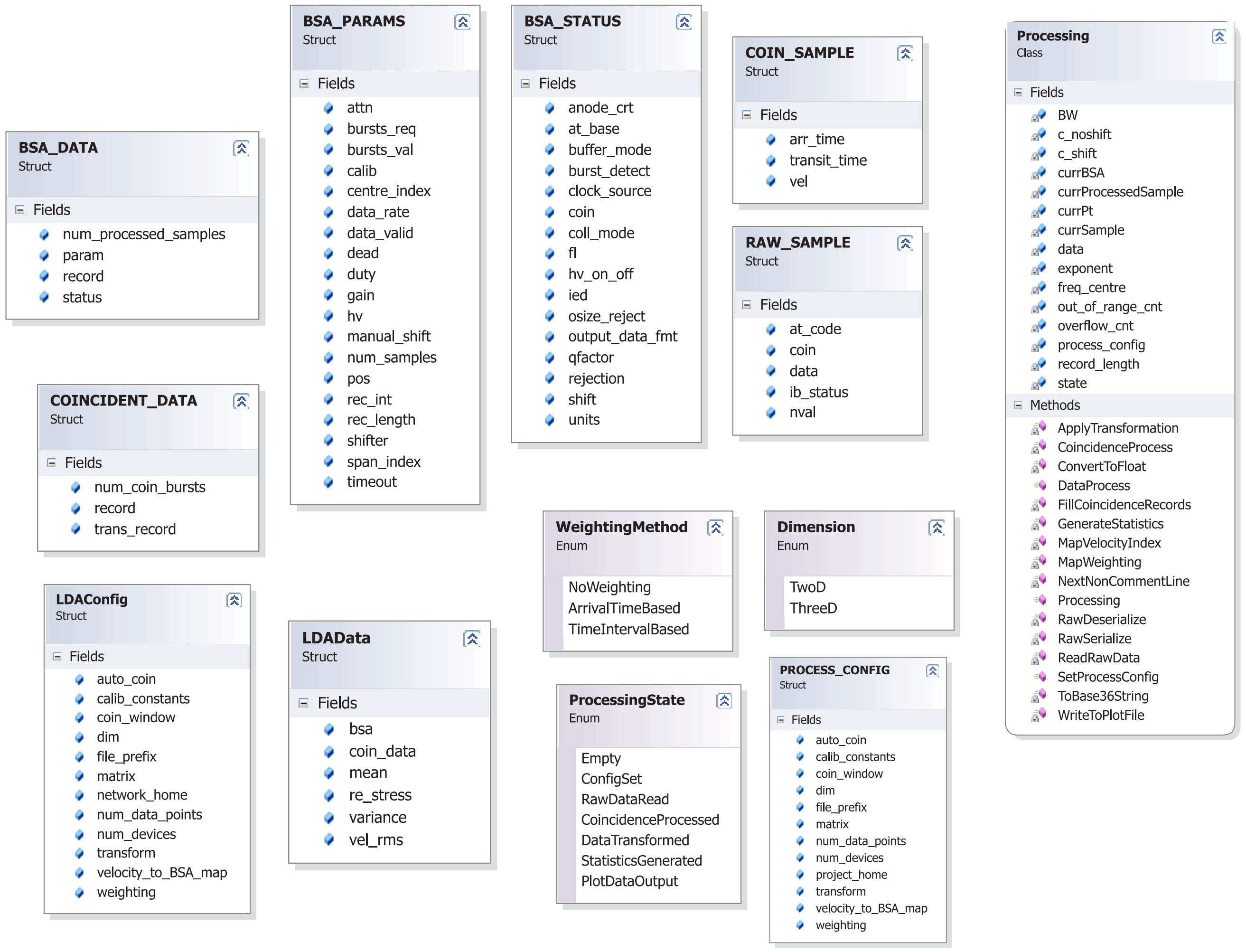


Figure A.1: Software structures of LDA Processing module

with elements

$$A_{nn'} = \frac{1}{T} \int_0^T u_n(t) u^*_{n'}(t) dt, \qquad n, n' = 1, 2, \cdots, N \tag{A.8}$$

The $u_n(t)$ represent the composite signal of microphone n and $u^*_n(t)$ is the complex conjugate transpose of it.

The next step in the processing is called beamforming. During the microphone array testing, beamforming is used to successively steer the phased array in a grid selecting the regions of interest to ascertain noise source distributions.

The beamforming expression for each point in the grid is given by

$$b(x_b) = \langle |\vec{w}^*(x_b)\vec{u}(t)|^2 \rangle = \vec{w}^*(x_b) A \vec{w}(x_b) \tag{A.9}$$

where $\vec{w}(x_b)$ is a microphone weight vector defined for each source point x_b of interest and A being the pre-computed CSM.

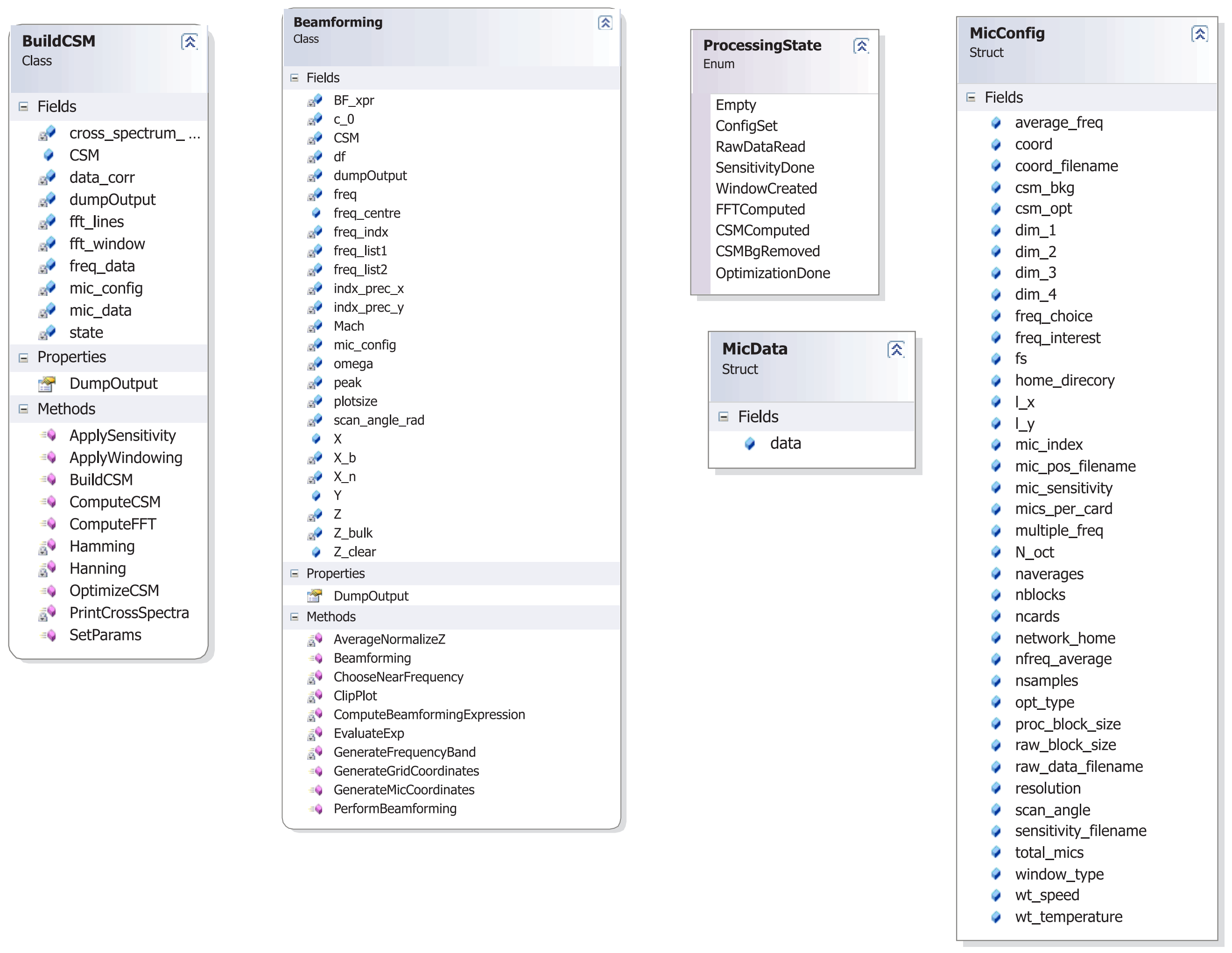


Figure A.2: Software structures of Microphone processing module

Appendix B

MyCoG.NET classes

B.1 MyCoG.GridSecurity and MyCoG.Proxy namespace

Figure B.1 shows classes in *MyCoG.GridSecurity* and *MyCoG.Proxy* namespace supporting authentication and delegation functionality. The *GridSecurity* class implements Secure Sockets Layer (SSL) mutual authentication protocol. It uses *Proxy* class to create X.509 proxy certificate and delegate client's security credentials to the server. The *Proxy* class in turn uses *PKCS10Decode* to decode the certificate request received from the server.

B.2 MyCoG.GridFTP namespace

Figure B.2 shows classes in *MyCoG.GridFTP* namespace. *GridFTPClient* class implements the standard FTP protocol and a set of GridFTP extensions; the GridFTP extensions include GSI security on control and data channels, parallel file transfers, partial file transfers and third-party transfers. For example, the *parallelUpload* and *parallelDownload* member functions of the *GridFTPClient* class support parallel data transfer in extended block mode as defined in GridFTP protocol extensions [22]. FTP is a request-response protocol with two types of network connections between client and server - *control* and *data*. *GridFTPClient* utilizes

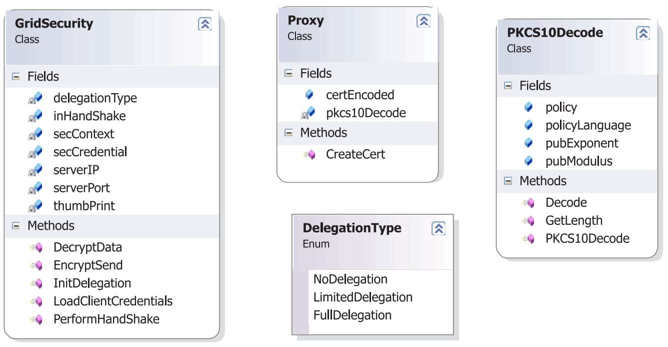


Figure B.1: Grid security classes

FTPControlChannel to exchange client commands and server replies for the entire duration of the FTP session (*GridFTPSession* class). The actual FTP commands for the server are generated using *Command* and *FeatureList* classes. The replies from the server are parsed using the *Reply* class. Many data channel connections are created during the session to perform the actual data transfer. The data channel host and port parameters are captured in *HostPort* and *HostPortList* (in case of striping) utility classes.

B.3 MyCoG.Gram namespace

Figure B.3 shows classes in *MyCoG.Gram* namespace. The *GramClient* class provides interfaces to Globus GRAM gatekeeper service for submitting, monitoring and terminating jobs. The GRAM protocol is a subset of HTTP; GRAM request messages are sent as HTTP POST requests while the GRAM reply messages are received in the form of HTTP status codes. The job requests to gatekeeper service are expressed in Resource Specification Language (RSL). The *GRAMProtocol* class is used for framing request messages and *HttpResponse* class is used for parsing gatekeeper responses. In cases of asynchronous job submissions, a callback handler (*GramCallbackHandler*) can be registered with the job manager that communicates state changes of the job to client's callback. The *stdin*, *stdout* and *stderr* for a job can be expressed as GASS URLs which are obtained subsequent to starting *GassServer* on the client machine.

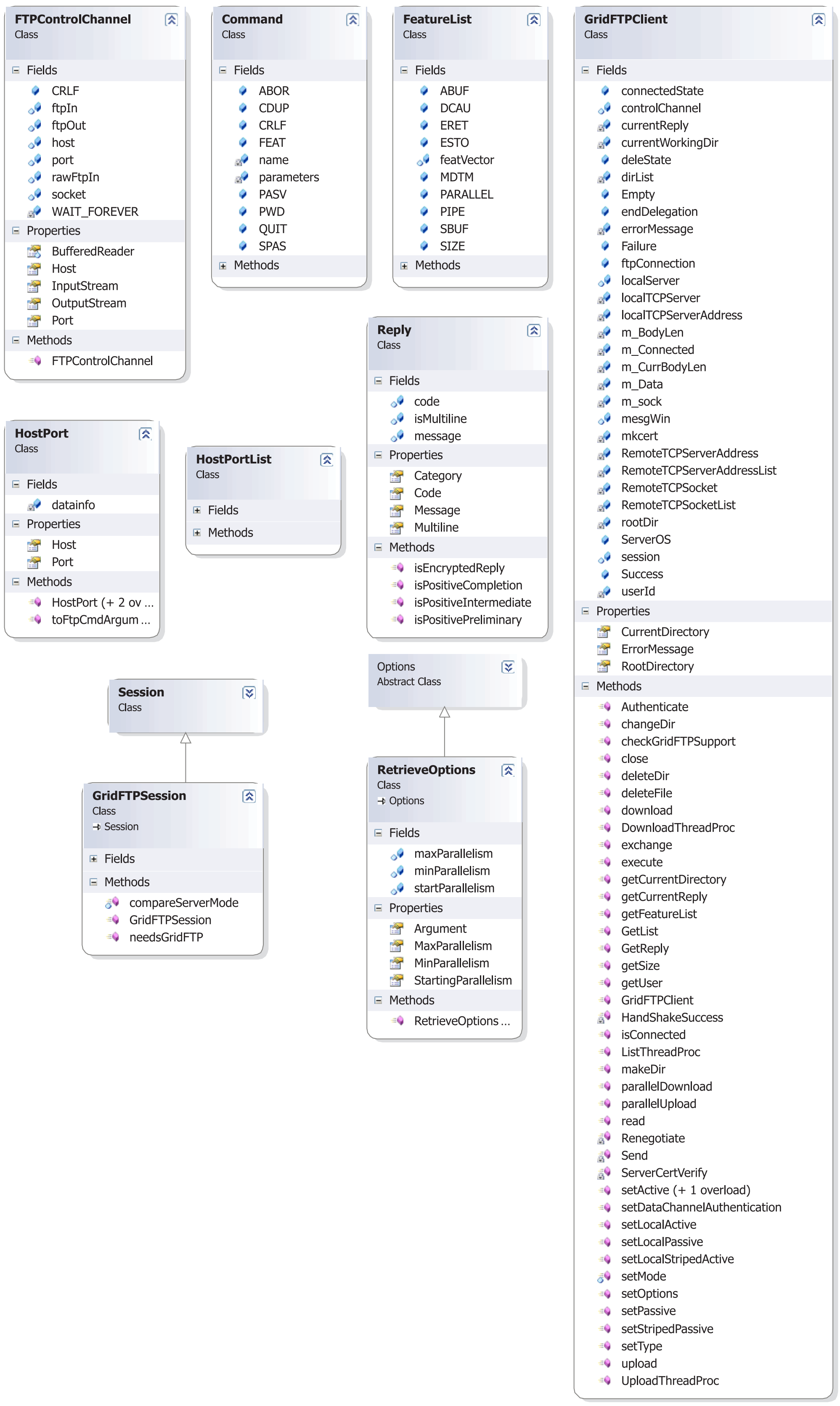


Figure B.2: Classes under MyCoG.GridFTP namespace

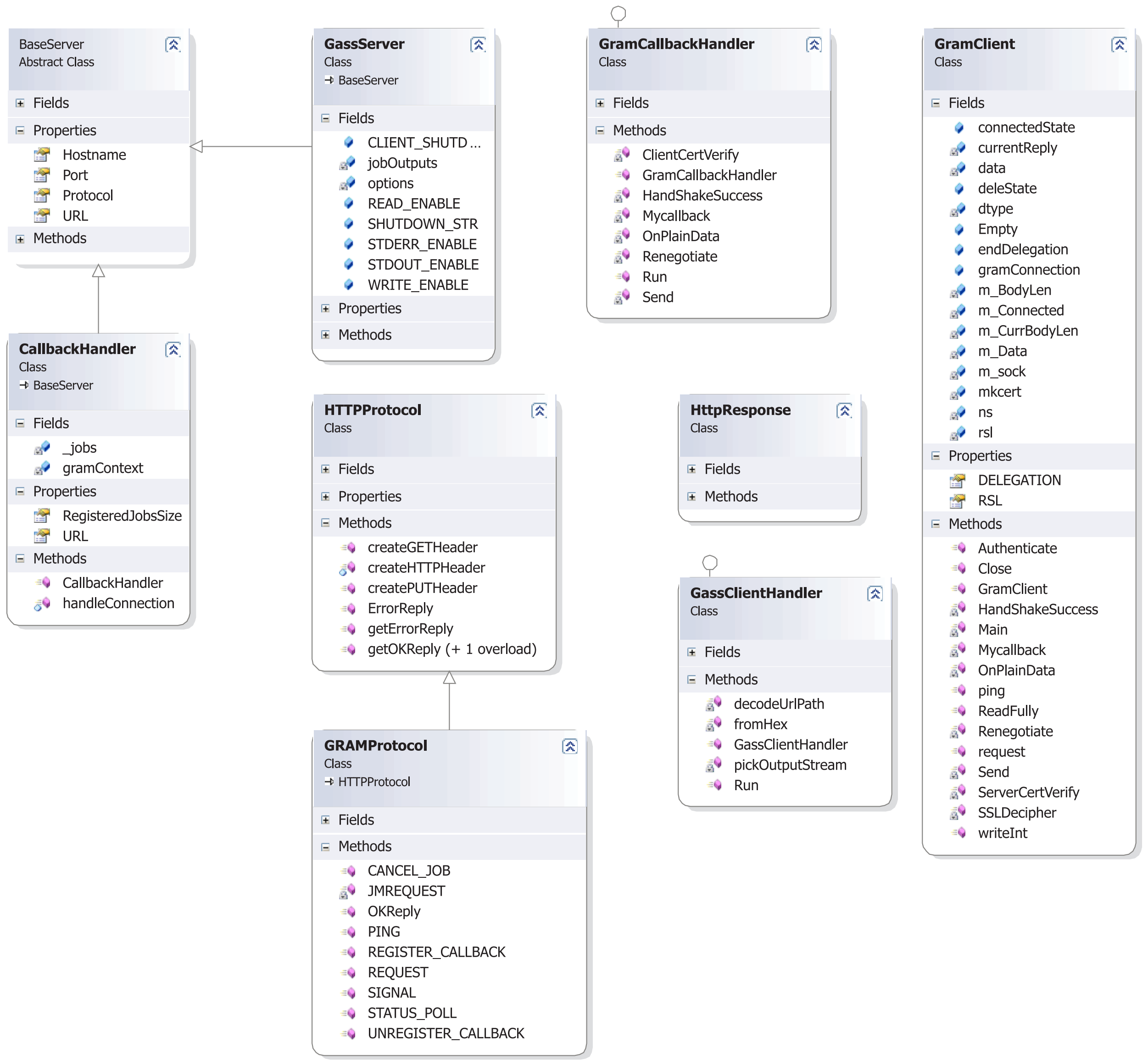


Figure B.3: Classes under MyCoG.Gram namespace

Appendix C

List of Publications

Journal:

1. A.Paventhan, K.Takeda, S.J.Cox, D.A.Nicole: "MyCoG.NET: a Multi-language CoG Toolkit", *Concurrency and Computation: Practice and Experience, DOI: 10.1002/cpe.1133*.

2. A.Paventhan, K.Takeda, S.J.Cox, D.A.Nicole: "Federated Database Services for Wind Tunnel Experiment Workflow", *Scientific Programming, 14(3-4):173-184, 2006*.

Conferences/Workshops:

3. A.Paventhan, K.Takeda: "MyGridFTP: A Zero-deployment GridFTP Client using the .NET Framework", LNCS Vol 3470, *European Grid Conference (EGC 2005)*, Amsterdam, Netherlands.

4. A.Paventhan, K.Takeda: "MyCoG.NET: Towards Multi-Language CoG Toolkit", *Middleware for Grid Computing (MGC'05), co-located with ACM/USENIX Middleware 2005*, Grenoble, France.

5. A.Paventhan, K.Takeda, S.J.Cox, D.A.Nicole: "Leveraging Windows Workflow Foundation for Scientific Workflows in Wind Tunnel Applications", *IEEE International Workshop on Workflow and Data Flow for Scientific Applications (SciFlow 2006), co-located with ICDE 2006*, Atlanta, USA.

6. A.Paventhan, K.Takeda, S.J.Cox, D.A.Nicole: "Workflows for Wind Tunnel Grid Applications", LNCS Vol 3993, *1st International Workshop on Workflow systems in e-Science (WSES 06), co-located with ICCS 2006*, Reading, UK.

References

[1] I Foster and C Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 2^{nd} edition, November 2003.

[2] C E Catlett and L Smarr. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.

[3] I Foster, J Geisler, B Nickless, W Smith, and S Tuecke. Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment. In *Proceedings of the 5th IEEE Symbosium on High Performance Distributed Computing (HPDC-5)*, pages 562–571, 1996.

[4] I Foster, C Kesselman, and S Tuecke. The anatomy of the Grid:Enabling scalable in virtual organizations. *International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[5] G Graham, R Cavanaugh, P Couvares, A D Smet, and M Livny. *Distributed Data Analysis:Federated Computing for High-Energy Physics, in "The Grid: Blueprint for a New Computing Infrastructure", I Foster and C Kesselman (Eds)*. Morgan-Kaufmann, 2003.

[6] L Pearlman, C Kesselman, et al. Distributed Hybrid Earthquake Engineering Experiments:Experiences with a Ground-Shaking Grid Application. In *Proceedings of the 13th IEEE Symbosium on High Performance Distributed Computing (HPDC-13)*, 2004.

[7] Linked Environments for Atmospheric Discovery. http://portal.leadproject.org [January 2007].

[8] J Austin et al. *"Predictive Maintenance: Distributed Aircraft Engine Diagnostics" in The Grid: Blueprint for a New Computing Infrastructure by I Foster and C Kesselman (Eds)*, pages 69–79. Morgan-Kaufmann, 2003.

[9] T Oinn et al. Taverna: Lessons in creating a workflow environment for the lifesciences. *Concurrency and Computation: Practice & Experience*, 18(10), 2006.

[10] Sloan Digital Sky Survey. http://www.sdss.org [January 2007].

[11] University of Southampton Windtunnels. http://www.windtunnel.soton.ac.uk [January 2007].

[12] Globus Toolkit - http://www.globus.org [January 2007].

[13] D E Post. The coming crisis in computational science. Los Alamos National Laboratory Report (LA-UR-04-0388), February 2004.

[14] Introducing the .NET Framework 3.0. http://msdn2.microsoft.com/, [January 2007].

[15] G Bell, J Gray, and A Szalay. Petascale Computational Systems: Balanced Cyberinfrastructure in a Data-Centric World. *IEEE Computer*, 39(1):110–112, January 2006.

[16] J Gray. The Revolution in Database Architecture. Microsoft Research Technical Report, MSR-TR-2004-31.

[17] P Watson. *Databases and the Grid, in "Grid Computing: Making the Global Infrastructure a Reality", F Berman and GC Fox and T Hey (Eds).* Wiley, 2003.

[18] L M Haas, E T Lin, and M A Roth. Data integration through database federation. *IBM Systems Journal*, 41(4):578–596, 2002.

[19] I Foster and C Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[20] A Globus Primer - Describing Globus Toolkit Version 4. http://www.globus.org/toolkit/docs/4.0/key [January 2007].

[21] J Linn. Generic Security Service Application Program Interface. In *RFC 2743*. IETF, January 2000.

[22] W Allock. GridFTP: Protocol Extensions to FTP for the Grid. *Global Grid Forum*, April 2003.

[23] R K Madduri, C S Hood, and W E Allcock. Reliable File Transfer in Grid Environments. In *LCN*, pages 737–738, 2002.

[24] GRAM: Execution Management. http://www.globus.org/toolkit/gram/ [December 2006].

[25] MDS: Information Services: Monitoring & Discovery System. http://www.globus.org/toolkit/mds/ [December 2006].

[26] E Laure et al. Programming the Grid with gLite. EGEE Technical Report (EGEE-TR-2006-001), March 2006.

[27] A. Streit et al. *GRID COMPUTING: THE NEW FRONTIER OF HIGH PERFORMANCE COMPUTING*, volume 14 of *Advances in Parallel Computing*, chapter UNICORE – from Project results to Production Grids, pages 357–376. Elsevier, 2005.

[28] Open Middleware Infrastructure Institute, UK. http://www.omii.ac.uk [January 2007].

[29] G Laszewki, I Foster, J Gawor, and P Lane. A Java commodity grid kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):643–662, 2001.

[30] K R Jackson. pyGlobus: A Python interface to the Globus Toolkit. *Concurrency and Computation: Practice & Experience*, 14(13-15):1075–1083, 2002.

[31] S Mock S, M Thomas, M Dahan, K Mueller, C Mills, and G Laszewski. The Perl Commodity Grid Toolkit. *Concurrency and Computation:Practice & Experience*, 14(13-15):1085–1095, 2002.

[32] T Goodale et al. SAGA: A Simple API for Grid Applications, High Level Application Programming on the Grid. *Computational Methods in Science and Technology*, (accepted for publication), 2006.

[33] Open Grid Forum. http://www.ogf.org [January 2007].

[34] I Foster, C Kesselman, J Nick, and S Tuecke. The Physiology of the Grid:An Open Grid Services Architecture for Distributed Systems Integration. *Global Grid Forum*, June 2002.

[35] S Tuecke et al. Open Grid Services Infrastructure (OGSI) version 1.0. *Global Grid Forum*, June 2003.

[36] Web Services Resources Framework v1.2 OASIS Standard. http://www.oasis-open.org/, [April 2006].

[37] I Foster et al. Open Grid Services Architecture (OGSA) version 1.0. *Global Grid Forum*, January 2005.

[38] S Short. *Building XML Web services for the Microsoft .NET Platform*. Microsoft Press, 2002.

[39] World Wide Web Consortium (W3C) - http://www.w3.org/ [January 2007].

[40] Web Services Security version 1.1, OASIS Standard. http://docs.oasis-open.org/wss/v1.1/, February 2006.

[41] M Antonioletti, M P Atkinson, R Baxter, et al. The design and implementation of Grid database services in OGSA-DAI. *Concurrency and Computation - Practice and Experience*, 17(2-4):357–376, 2005.

[42] M Antonioletti, A Krause, N W Paton, A Eisenberg, S Laws, S Malaika, J Melton, and D Pearson. The WS-DAI Family of Specifications for Web Service Data Access and Integration. *ACM SIGMOD Record*, 35(1):48–55, March 2006.

[43] M Wan, A Rajasekar, R Moore, and P Andrews. A Simple Mass Storage System for the SRB Data Grid. In *Proceedings of the 20th IEEE Conference on Mass Storage Systems and Technologies (MSS'03)*, pages 358–369, 2003.

[44] D T Liu. The design of GridDB: A Data-Centric Overlay for the Scientific Grid. In *Proceedings of the International Conference on Very Large Data Bases*, pages 600–611, 2004.

[45] R Ramakrishnana and J Gehrke. *Database Management Systems*. McGraw-Hill, third edition, 2003.

[46] M Cannataro and D Talia. The Knowledge Grid. *Communications of the ACM*, 46(1):89–93, January 2003.

[47] H Zhuge. Discovery of Knowledge Flow in Science. *Communications of the ACM*, 49(5):101–107, May 2006.

[48] Data mining Grid project. http://www.dataminінggrid.org [December 2006].

[49] D Hollingsworth. The Workflow Reference Model. *Workflow Management Coalition, Document Number TC00-1003*, http://www.wfmc.org, 1995.

[50] W van der Aalst and W van Hee. *Workflow Management: Methods, Models and Systems.* MIT Press, 2002.

[51] S Thatte. XLANG Specification. *http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm [December 2006].*

[52] F Leymann et al. Web Services Flow Language (WSFL1.0). *http://xml.coverpages.org/WSFL-Guide-200110.pdf [December 2006].*

[53] Web Services Business Process Execution Language version 2.0, Public review draft. http://docs.oasis-open.org/wsbpel/2.0/, August 2006.

[54] F DeRemer and H H Kron. Programming-in-the-large versus programming-in-the small. *IEEE Transactions on Software Engineering*, 2(2):80–86, June 1976.

[55] M P Singh, A K Chopra, N Desai, and A U Mallya. Protocols for processes: programming in the large for open systems. In *OOPSLA*, October 2004.

[56] A Slominski. *"Adapting BPEL to Scientific Workflows" in Workflows for e-Science, IJ Taylor, E Deelman, DB Gannon and M Shields (Eds)*, chapter 14. Springer, 2007.

[57] Web Services Invocation Framework (WSIF). http://ws.apache.org/wsif/overview.html [January 2007].

[58] Microsoft BizTalk Server. http://www.microsoft.com/biztalk [January 2007].

[59] Oracle BPEL Process Manager, Developers Guide. http://download.oracle.com/otndocs/products/bpel/bpeldev.pdf, October 2005.

[60] P Andrew, J Conard, and S Woodgate. *Presenting Windows Workflow Foundation (Beta Edition).* Sams, September 2005.

[61] S Eswaran, DD Vecchio, G Wasson, and M Humphrey. Adapting and Evaluating Commercial Workflow Engines for e-Science. In *2^{nd} IEEE International Conference on e-Science and Grid Computing*, 2006.

[62] A Rygg, M Scott, P Roe, and O Wong. Bio-Workflows with BizTalk: Using a Commercial Workflow Engine for e-Science. In *1^{st} IEEE International Conference on e-Science and Grid Computing*, 2005.

[63] B Wasserman, W Emmerich, B Butchart, N Cameron, L Chen, and J Patel. *"Sedna: A BPEL-based environment for visual scientific workflow modelling" in Workflows for e-Science,, IJ Taylor, E Deelman, DB Gannon and M Shields (Eds)*, chapter 26. Springer, 2007.

[64] R Barga and D Gannon. *"Scientific versus Business Workflows" in Workflows for e-Science,, IJ Taylor, E Deelman, DB Gannon and M Shields (Eds)*, chapter 2. Springer, 2007.

[65] E Deelman, J Blythe, Y Gil, and C Kesselman. *"Workflow Management in GriPhyN" in Grid Resource Management J. Nabrzyski, J. Schopf, and J. Weglarz (Eds)*. Kluwer, 2003.

[66] Condor DAGman. http://www.cs.wisc.edu/condor/dagman/ [January 2007].

[67] B Ludscher et al. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice & Experience*, 18(10):1039–1065, 2006.

[68] D Churches, D Gombas, et al. Programming scientific and distributed workflow with Triana services. *Concurrency and Computation: Practice & Experience*, 18(10):1021–1037, 2006.

[69] V Curcin et al. IT Service Infrastructure for Integrative Systems Biology. In *IEEE International Conference on Services Computing,(SCC'04)*, September 2004.

[70] J Yu and R Buyya. A Taxonomy of Scientific Workflow Systems for Grid Computing. *ACM SIGMOD Record*, 34(3):44–49, September 2005.

[71] CMS The Computing Project, Technical Design Report, CERN-LHCC-2005-023. http://csmdoc.cern.ch/cms/cpt/tdr, June 2005.

[72] M A Nieto-Santisteban, J Gray, A S Szalay, J Annis, A R Thakar, and W J O'Mullane. When Database Systems Meet the Grid. In *Proceedings of the 2005 CIDR Conference*, January 2005.

[73] Grid Enabled Wind Tunnel Test Service. http://www.nesc.ac.uk/talks/sc2004talks/Tuesday/SC2004_gewitts.ppt [January 2007].

[74] B Plale, D Gannon, J Alameda, B Wilhelmson, S Hampton, A Rossi, and K Droegemeier. Active Management of Scientific Data. *IEEE Internet Computing*, pages 27–34, February 2005.

[75] Dantec Dynamics, http://www.dantecdynamics.com [January 2007].

[76] K Takeda, G B Ashcroft, X Zhang, and P A Nelson. Unsteady aerodynamics of slat cove flow in a high-lift device configuration. *AIAA Paper 2001-0706, AIAA Aerosciences Meeting*, 2001.

[77] D R M Jeffrey. *An investigation into the aerodynamics of Gurney flaps*. PhD thesis, School of Engineering Sciences, University of Southampton, 1998.

[78] A Brizell. *Aspects of brake system aerothermodynamics*. PhD thesis, School of Engineering Sciences, University of Southampton, 2006.

[79] M G Smith, B Fenech, et al. Control of noise sources on aircraft landing gear bogies. In *12th AIAA/CEAS Aeroacoustics Conference, AIAA Paper 2006-2626*, May 2006.

[80] K Takeda, X Zhang, and P A Nelson. Unsteady aerodynamics and aeroacoustics of a high-lift device configuration. In *40th AIAA Aerospace Sciences Meeting and Exhibit, AIAA Paper 2002-0570*, 2002.

[81] R J Underbrink. *Aeroacoustic Measurements*. Springer, 2002.

[82] S Webb and I P Castro. PIV, LDA and HWA comparative measurements in various turbulent flows. Technical Report AFM-03/01, Aerodynamics & Flight Mechanics , University of Southampton, UK, 2003.

[83] S Webb. PIV User's guide. Technical Report AFM-03/02, Aerodynamics & Flight Mechanics, University of Southampton, UK, 2003.

[84] M Shields. *"Control- Versus Data-Driven Workflows" in Workflows for e-Science,, IJ Taylor, E Deelman, DB Gannon and M Shields (Eds)*, chapter 11. Springer, 2007.

[85] ECMA C# and Common Language Infrastructure Standards. http://www.ecma-international.org/publications/standards/standard.htm [January 2007].

[86] J Richter. *Applied Microsoft .NET Framework Programming.* Microsoft Press, January 2002.

[87] Mono Project - http://www.mono-project.com [January 2007].

[88] B Noyes. *ClickOnce*: Deploy and Update Your Smart Client Projects Using a Central Server. Microsoft Developer Network, May 2004.

[89] A Paventhan and K Takeda. MyGridFTP: A Zero-Deployment GridFTP Client Using the .NET Framework. *European Grid Conference*, LNCS 3470, February 2005.

[90] W Keith Edwards. *Core Jini.* Sun Microsystems Press, 2000.

[91] S Meder, V Welch, S Tuecke, and D Engert. GSS-API Extensions. Global Grid Forum Document, June 2004.

[92] V Welch. Grid Security Infrastructure Message Specification, Global Grid Forum Document. February 2004.

[93] S Tuecke. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. IETF, http://www.ietf.org/rfc/rfc3820.txt, 2004.

[94] Lahey/Fujitsu Fortran Compiler for .NET. http://www.lahey.com/ [January 2007].

[95] IPerf bandwidth measurement tool - http://dast.nlanr.net/Projects/Iperf/ [January 2007].

[96] G-Hydroflex project, http://www.soton.ac.uk/∼ghydflex/ [January 2007].

[97] G von Laszewski, M Hategan, and D Kodeboyina. *"Java CoG Kit Workflow" in Workflows for e-Science,, IJ Taylor, E Deelman, DB Gannon and M Shields (Eds)*, chapter 11. Springer, 2007.

[98] Heinz Stockinger. Distributed Database Management and the Data Grid. In *Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*, pages 1–12, 2001.

[99] S Narayanan, T Kurc, U Catalyurek, and J Saltz. Database Support for Data-driven Scientific Applications in the Grid. *Parallel Processing Letters*, 13(2):245–271, 2002.

[100] S Pallickara, B Plale, S Jensen, and Y Sun. Structure, sharing and preservation of scientific experiment data. *To appear in IEEE 3rd International workshop on Challenges of Large Application in Distributed Environments (CLADE)*, July 2005.

[101] A Acheson, M Bendixen, et al. Hosting the .NET Runtime in Microsoft SQL Server. In *ACM SIGMOD Conference*, pages 860–865, 2004.

[102] M A Williams. *Pro .NET Oracle Programming.* Apress, 2004.

[103] G Evans. A detailed look at DB2 Stinger .NET CLR Routines, IBM developerWorks article. June 2004.

[104] M Rys. XML and Relational Database Management Systems: Inside Microsoft SQL Server 2005. In *ACM SIGMOD Conference*, pages 958–962, 2005.

[105] Z H Liu, M Krishnaprasad, and V Arora. Native XQuery processing in Oracle XMLDB. In *ACM SIGMOD Conference*, pages 828–833, 2005.

[106] M Nicola and B Van der Linden. Native XML Support in DB2 Universal Database. In *Proceedings of the 31st VLDB Conference*, pages 1164–1174, 2005.

[107] Using Native XML Web Services in SQL Server 2005. http://msdn2.microsoft.com, [July 2006].

[108] K Mensah. Virtualize Your Oracle Database with Web Services, Oracle Technical Article. November 2005.

[109] G Hutchison. DB2 Web Services: The Big Picture, IBM developerWorks article. August 2002.

[110] SQL Server Authentication over SOAP. http://msdn2.microsoft.com [January 2007].

[111] D Wollscheid. DB2 Web Service Provider Security, IBM developerWorks article. April 2003.

[112] M Lehmann. Securing Web Services, ORACLE Magazine. January/February 2005.

[113] R Walter. *The Rational Guide To SQL Server 2005 Service Broker.* Rational Press, 2005.

[114] Sharing Information with Oracle Streams. An Oracle White paper, [May 2005].

[115] V Welch et al. Using SAML for OGSI Authorization. *Global Grid Forum*, March 2006.

[116] Replication Publishing Model Overview, SQL Server Books Online. http://msdn2.microsoft.com [January 2007].

[117] MATLAB Builder for .NET User's Guide Version 2. http://www.mathworks.com [January 2007].

[118] Intel Corporation. *IA-32 Intel Architecture Software Developer's Manual, Programming with Streaming SIMD Extensions 3 (SSE3)*, June 2006.

[119] .NET Numerical Applications with NMath Core, white paper,. http://www.centerspace.net, [July 2006].

[120] A S Szalay, J Gray, A R Thakar, P Z Kunszt, T Malik, J Raddick, C Stoughton, and J vandenBerg. The SDSS SkyServer – Public Access to the Sloan Digital Sky Server Data. *ACM SIGMOD*, June 2002.

[121] G Aydin, G C Fox, H Gadgil, and M E Pierce. Streaming Data Services to Support Archival and Real-Time Geographical Information System Grids. In *Sixth Annual NASA Earth Science Technology Conference*, June 2006.

[122] BioSimGrid project. http://www.biosimgrid.org, [September 2006].

[123] A P Sheth and J A Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[124] D Chadwick. Authentication and Authorization in the GGF, UK e-Science Security Meeting. *http://www.nesc.ac.uk/events/townmeeting0405*, April 2005.

[125] Basic Security Profile Version 1.1. http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1-2006-10-19.html [October 2006].

[126] S Guest. WS-Security Interoperability Using WSE 2.0 and Sun JWSDP 1.5. *http://msdn2.microsoft.com*, May 2005.

[127] R Sears, C van Ingen, and J Gray. To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem? *Microsoft Research, Technical Report, MSR-TR-2006-45*, June 2006.

[128] S Johnston. *Encouraging collaboration through a new data management approach.* PhD thesis, School of Engineering Sciences, University of Southampton, 2006.

[129] M R Head, M Govindaraju, R Engelen, and W Zhang. Benchmarking XML Processors for Applications in Grid Web Services. In *Supercomputing (SC06)*, November 2006.

[130] K Chiu, M Govindaraju, and R Bramley. Investigating the Limits of SOAP Performance for Scientific Computing. In *Proceedings of The Eleventh International Symposium on High Performance Distributed Computing*, pages 246–254, 2002.

[131] SQL Server 2005 Books Online, Thread and Task Architecture. http://msdn2.microsoft.com/en-us/library/ms176043(SQL.90).aspx, [June 2007].

[132] GridAnt - Client side Grid workflow management with Ant, White paper. http://www.mcs.anl.gov/∼gregor/papers/vonLaszewski-gridant.pdf [January 2007].