Collaborative Visualization in the Grid Environment

by

Haoxiang Wang

Submitted in accordance with the requirements for the degree of Doctor of Philosophy.

The University of Leeds
School of Computing

September 2007

The candidate confirms that the work submitted is his own and that the appropriate credit has been given where reference has been made to the work of others.

Acknowledgements

I would like to thank all the people that have helped me get through the time spent on this PhD study.

Special thanks to Professor Ken Brodlie for encouraging, guiding me through all these years research. There are also other things besides research that I have learned from you. It is more than the word ‘indebted’.

Thanks also go to colleagues in the Visualization and Virtual Reality Research group, School of Computing. Especially, thank Dr. Jason Wood and Mr. John Hodrien for the help over the years and Dr. Mark Riding for the help in setting the Grid machine at Manchester.

Finally, a big thank you to my parents and my girlfriend, Feng, for all the love and support throughout all these years.

Abstract

Scientific visualization today usually involves scientists from multiple disciplines trying to tackle complex simulation problems or dealing with large datasets. These scientists are likely geographically distributed, so there is a cost in both finance and time, when they are required to collaborate or meet face-to-face. The complex problems and larger datasets require high performance computing facilities which are not always available from these scientists' desktops.

The objective of this research is to address these two issues. We aim to create a novel visualization framework which supports collaborative working amongst different users and utilizes high performance computational resources in a Grid environment.

The system is named as NoCoV, short for Notification-service-based Collaborative Visualization system. The system involves a number of visualization notification services which can be jointly provided by different developers and deployed on different Grid networks. A visualization notification service can subscribe to other services and receive notification messages delivered from them. By using this subscription/notification communication pattern, users can link suitable visualization services together through a centralized controller service to create customized visualization pipelines according to their different requirements. Collaborative working is supported in both creating and steering of visualization pipelines. Visualization information is recorded as XML-based descriptions and published as notification topics on a controller service. By subscribing to this controller service, collaboration can be achieved by sharing the visualization information amongst different participants. The NoCoV system accommodates users with different background knowledge and supports collaboration between them. Visualization services are jointly provided by different visualization service developers. Visualization experts can compose pipelines with the services created by service developers using their visualization expertise. Application scientists can use pipelines created by visualization experts and can set steering parameters according to their specialized knowledge.

A NoCoV prototype is implemented as proof of concept. The prototype has four visualization services and one controller service which are implemented as WSRF notification services deployed on GT4 containers. The prototype also provides a pipeline editor client GUI for visualization experts to create visualization pipelines and a parameter control client for application scientists to steer these pipelines.

The performance of the NoCoV prototype is evaluated by a test involving a pipeline distributed across different Grid networks. The limitations of the NoCoV system and the future work are also discussed at the end of this thesis.

The overall vision of the NoCoV system is to produce the next generation of visualization system, which has a worldwide repository of visualization services deployed on different Grid networks and supports collaboration amongst users on a global scale.

Declarations

Some parts of the work presented in this thesis have been published in the following articles:

- **Wang, H.; Brodlie, K.; Handley, J. and Wood, J.**, "Service-oriented approach to collaborative visualization", *Proceedings of the UK e-Science All Hands Meeting 2006*,(2006) 241-248. Now selected as one of the best papers from the conference and will be published in a special issue of *Concurrency and Computation: Practice and Experience*, (2007)
- **Wang, H.; Brodlie, K. and Wood, J.**, "Evolving visualization to service-oriented architectures on the Grid", *Proceedings of 1st Workshop of the Service-Oriented Software Research Network (SOSoRNet)*, (2006) 25-31.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The symbolic, which includes the literal and the numeric, is one of the most important ways to represent and understand the world. However, limited by the cognitive capabilities of the human brain, the symbolic is not easily understood when it is used to describe complicated or large-scale problems. "An estimated 50 percent of the brain's neurons are associated with vision" [63]. By using computer technologies to transform the symbolic into the geometric, visualization aims to help this neurological machinery to work, so that human can have a deeper insight of the world.

Starting from the historical hand-drawn diagrams, visualization is now more concerned with generating graphics from numerical data by using computers for data exploration and data presentation. Visualization is widely used for addressing scientific problems and itself is also recognized as a crucial component in e-Science.

As the growth in the size and the complexity of scientific problems, the research is in a trend of involving scientists from multiple disciplines and requiring powerful computational resources. The focus of this thesis is to present a novel visualization framework which supports different participants to collaboratively work on the same visualization and utilize heterogeneous computational resources in the Grid environment.

This chapter presents an overview of the research and an outline of the remaining chapters in this thesis.

1.1 Introduction

Richard Hamming observed in 1962 that "the purpose of computing is insight not numbers" [42]. Scientific visualization aims to allow users to have insight into the data and gain deeper understanding of the data, especially the increasingly large datasets which can be generated from simulations or experiments. By transforming numeric data into graphics, visualization provides scientists an opportunity to discover the unseen hidden in the data. For the scientific community, visualization has became a very important method for scientific discovery and research. In 1998 McCormick et al. even claim that "in many fields it (visualization) is already revolutionizing the way scientists do science." [55]

In the last decades, a number of widely used visualization tools have been developed, many of them following the dataflow paradigm. This dataflow paradigm is also known as the pipeline reference model which was proposed by Haber and McNabb [41] and Upson et al. [75]. In this model, a visualization pipeline consists of a set of processing steps where users can interact by setting steering parameters. These visualization systems are categorized as Modular Visualization Environments (MVEs). They provide users with a set of visualization modules and a visual pipeline editor tool allowing users to connect suitable modules. Some of these MVEs, such as Application Visualization System (AVS) [6], IRIS Explorer [45], IBM Data Explorer (DX) [27] and Amira [3], are introduced in detail in Chapter 2.

The nature of a visualization pipeline makes it straightforward to implement in a distributed paradigm with different parts of the pipeline running on different machines. There has been another trend in visualization systems to support collaborative work, allowing geographically distributed users to work on the same visualization synchronously or asynchronously. Distributed visualization utilizes different computational resources, whereas collaborative visualization offers the opportunity to utilize different human expertise.

To address the increasingly complex scientific problems, more and more computational resources are required for visualization to address these problems. The emergence of the Grid offers visualization systems the opportunity to utilize heterogeneous Grid resources. Some pilot projects of building Grid-based visualization systems are also presented in Chapter 2 in detail.

Service-Oriented Archtecture (SOA) is adopted as the fundamental design principle for many Grid implementations. SOA is an evolution of distributed computing. It provides computational prosesses and resources as loosely-coupled services.

This thesis introduces a SOA-based novel visualization framework in the Grid environment supporting collaborative work. The framework is named as NoCoV, standing for

Notification-service-based Collaborative Visualization and exploiting notification service as the basic concept in the design.

1.2 Scenario

The visualization framework proposed in this thesis aims to provide a worldwide collaboration between different users and using the Grid computing resources at a global scale. Imagine we have the following situation:

A meteorologist in Leeds wants to study an environmental dataset in collaboration with another meteorologist in London. To avoid the time and expense of travel, they plan to synchronously collaborate by using some visualization system over the network. Both of the scientists know well the target to be discovered in the dataset, however they do not have the visualization skills to build visualization pipelines in the visualization system.

Fortunately, they know a visualization expert in US, who is willing to help them to build a suitable visualization pipeline for them. As they are in different time zones, it is difficult for all of them to work at the same time, but they can collaborate asynchronously. The visualization expert can firstly create the pipeline, and then leave it for the two scientists to set parameters and create the visualization results they want.

The environmental dataset is stored on a Grid network in Leeds, where there also exists a filter service which can dramatically reduce the size of the dataset by applying certain algorithms to refine the dataset according to the scientists' interest. Meanwhile there is a research group in China. They have already had many successful experiences in visualizing this kind of environmental dataset. They implemented the visualization process as a service. As the process is quite computationally demanding, they have deployed the service on a China Grid network.

The visualization expert in US therefore decides to link the filter service in UK and the visualization service in China into his visualization pipeline. In this scenario, the visualization pipeline is distributed across different Grid networks at a global scale, and the participants involved are also from different locations in the world.

1.3 Objectives

Reflecting the scenario provided in Section 1.2, the objectives of the research described in this thesis are:

- **Utilizing powerful computational resources for visualization.** For dealing with increasing large datasets efficiently, we aim to evolve visualization into the Grid environment with the support of accessing these heterogeneous computational resources from users' desktops.

- **Allowing collaboration.** Collaboration is one of the main objectives that this research aims to provide. Synchronous collaboration can be prompt and efficient for participants to share thoughts and reach agreements. However, for participants from different time zones, an asynchronous collaboration mechanism allowing them to work at different times can be useful.

- **Accommodating users with different background knowledge.** Visualization in the Grid environment requires skills of Grid computing, visualization skills and knowledge in particular scientific areas. This is a steep learning curve for users to master all this knowledge. Our research aims to provide different ways to access visualization for different kinds of users and avoid this steep learning curve for users.

- **Enabling extensibility and distribution.** The research explores the implementation of a visualization system using Service-Oriented Architecture. Visualization functions are delivered to users as loosely-coupled services, so that the system can be easily distributed and extended by adding new visualization services.

1.4 Major Contributions

The major contributions achieved by the research introduced in this thesis are:

- Design of a Service-oriented visualization framework;
- Use of notification services to implement visualization pipelines;
- Implementation of the framework in the Grid environment;
- Capability of distributing visualizations across different Grid networks;
- Support of both synchronous and asynchronous collaboration;
- Different interfaces for different users;
- Evaluation of the collaborative visualization framework;

Based on this research and implementations which have been done, we identify the areas of future work in order to achieve our ideal visualization system.

1.5 Thesis Structure

In order to get a clear vision of where our research started from, we look back at how technologies were developed and evolved until the present stage. In Chapter 2, we review several visualization systems and projects which worked in the areas of distributed and collaborative visualization. With an overview of Grid computing technologies, some pioneering projects in Grid-based visualization are also discussed in this chapter.

In Chapter 3, critical appraisals of related projects are made in order to refine the requirements of our research and lay out an overall vision.

Some early experiments of our research are presented in Chapter 4. With this initial work, some requirements to our research have been proved as feasible. The limitations reflected from these experiments also inspire some design features in our visualization framework.

Chapter 5 presents the detailed design of the visualization framework (NoCoV) in a three-layer architecture. Chapter 6 describes how we implemented this framework as a proof-of-concept prototype. Technical details about how the visualization pipeline is implemented as a set of linked notification services and how users can collaboratively work with the NoCoV system are illustrated in this Chapter.

The evaluation of the NoCoV framework is carried out in Chapter 7, including the comparison between NoCoV and other collaborative systems.

Finally, in Chapter 8 we conclude the research presented in this thesis and identify the future work need to be completed in order to achieve our final vision of Service-oriented collaborative visualization.

1.6 Summary

This chapter has laid out an overview of this thesis. We also presented the objectives of our research and the major contributions which we have achieved through this research.

Chapter 2

Background and Literature Review

2.1 Introduction

This chapter describes the background of this research. It is always important to look back at how the technologies were developed in order to inspire a vision of how to evolve them further. Meanwhile, by investigating relevant works and analyzing strengths and weaknesses of them, motivations and requirements can be produced for our research.

In Section 2.2 we explain the origins of visualization and a set of widely used visualization applications. Section 2.3 discusses why and how these applications were extended to enable distributed and collaborative visualization. Section 2.4 presents the development of the Grid and the technologies, especially the Service-oriented architecture, which can be used in the Grid environment. Some pioneering work concerning Grid-enabled and Service-oriented visualization is illustrated in Section 2.5, which enlightens the research introduced in this thesis. Section 2.6 mentions the formal description of visualization which becomes more important when we evolve visualization in the Grid environment and in a Service-oriented architecture.

2.2 Visualization

In this section, we begin with an introduction to the history of visualization and why it is important for scientific research. A number of popular visualization software systems are described, some of which can be classified as examples of Modular Visualization Environments (MVEs).

2.2.1 Origins of Visualization

Visualization is not a new paradigm, having been widely used in maps, scientific drawings, and data plots for thousands of years. One of the famous historical visualization examples is the map of Napoleon's invasion of Russia by Charles Joseph Minard (1781-1870), published in 1861 (Figure 2.1), in which the thickness of the line indicates the strength of the army, with numbers indicating strength at critical points. The line gives readers a straightforward impression of the reduction of the number in Napoleon's army, and the numbers explain the change in a precisely quantitative view. Other historical visualization examples can be found at Friendly's online Gallery of Data Visualization [36].

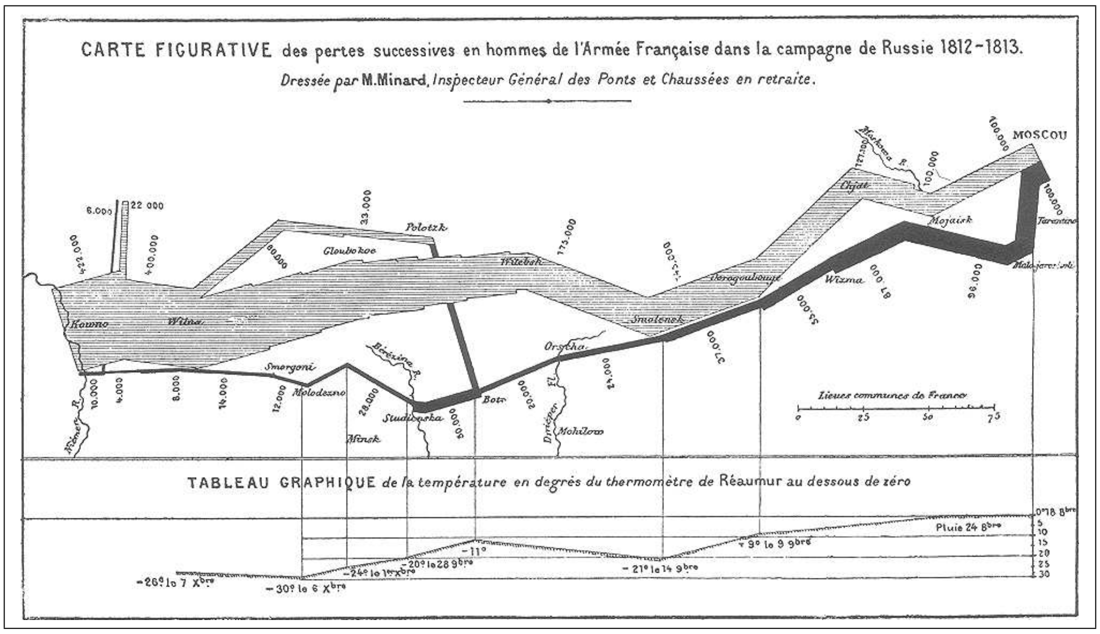

CARTE FIGURATIVE des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813.
Dressée par M.Minard, Inspecteur Général des Ponts et Chaussées en retraite.
MOSCOU
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro
Lieues communes de France

Figure 2.1: The map of Napoleon's invasion of Russia. (Charles Joseph Minard , 1861) (Resource from [36])

Today, visualization is recognized as the utilization of the computer graphics technologies as a tool for data exploration and data presentation. The origins of visualization lie in scientific and engineering disciplines, such as chemistry, biology, geography, astronomy, etc. The increasingly large datasets and the rapid growth of computer technologies (i.e. high speed network for data access, powerful computational resource for data processing, etc.) has drawn visualization more into the computer science area.

With the growth of high performance computing in 1980s, McCormick, DeFanti and

Brown wrote an ACM report which encouraged research funding agencies to invest in visualization [55]. While a number of visualization systems were developed following the inspiration of McCormick's ACM report, visualization scientists were also looking for a generic conceptual model for these different visualization systems. Haber and McNabb proposed a visualization reference model [41] to abstract visualization as a pipeline of processes which transform raw data (captured from experiments or simulations) step by step into a visual representation (graphic or animation). Figure 2.2 demonstrates the different types of process in this visualization pipeline: Data, Filter, Map and Render. The Data step feeds raw data into the pipeline, and the Filter process extracts the relevant data or refines the raw data by using some algorithm. The Map step generates a geometric representation using the filtered data, and then sends the geometric data to the Render step, which finally converts data into a visual image or animation.

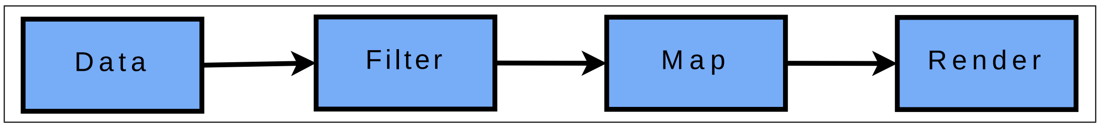


Figure 2.2: Visualization reference model [41]

A similar reference model was also introduced by Upson, et al [75] in the paper describing the initial work on the Application Visualization System (AVS) [6]. The reference model can be regarded as a milestone in the evolution of visualization. Subsequently, many powerful visualization software systems were developed with the influence of the reference model, most of which are still commonly used nowadays. These visualization systems are introduced in detail in section 2.2.2.

2.2.2 Widely Used Visualization Systems

Modular Visualization Environments (MVEs) can be regarded as typical implementations of Haber and McNabb's reference model. An MVE usually provides a set of encapsulated modules, each of which can be classified as a type of Data, Filter, Map or Render, corresponding to the steps in the reference model. Users can create visualizations by linking these modules to build up a visualization pipeline, according to their specialized requirement. Wood summarized six basic features of MVEs in his PhD thesis [81], namely:

- A graphical user interface (GUI) for constructing visualization pipelines.
- A library of routines for general visualization requirements.
- A set of internal data representations.

- A data/memory/control management mechanism.
- A command language for driving the system by means of predefined scripts.
- A mechanism for user extensibility.

Five visualization systems are introduced briefly in the following paragraphs to exemplify MVEs.

The **Application Visualization System** (**AVS**) is designed for developing interactive visualization applications quickly [6, 75]. The system is now named as AVS5. AVS was written in C++ following the Object-Oriented Programming paradigm and comes with a visual network editor, through which users can combine modules provided to create visualizations.

IRIS Explorer [45, 78] from NAG Ltd is another example of an MVE. It provides a large suite of specialized modules included in its library. A map editor is provided by IRIS Explorer, through which users can build up complicated visualization pipelines by choosing modules from the library and interconnecting them with wires. Users can also encapsulate their own code as modules and import them into IRIS Explorer by using the Module Builder tool.

Data Explorer [1, 27] from IBM is also implemented as a modular visualization system, based on the pipeline model. A Graphical User Interface is provided for users to build up their visualization pipeline easily and a wide variety of widgets are provided for users to manipulate images or control various aspects of visualization. It was renamed as OpenDX subsequently, when the software came to be open-source for the community.

Amira [3] and **MeVisLab** [56] are two MVEs both from Germany. They are both built with C++ class libraries and provide modules for users to combine together to generate complex image processing and visualization networks. Extensibility is also supported in both pieces of software by allowing a user to add customized modules.

Here, we introduce the Visualization Toolkit (VTK) as a contrast to the above MVEs. VTK [68, 77] is an open source visualization toolkit, which also reflects Haber and McNabb's reference model, but not an MVE. VTK consists of a C++ class library and several interfaces for Tcl/Tk, Java, and Python. Users can produce customized visualizations by writing their own code to invoke appropriate APIs provided by VTK. Compared with MVEs, VTK requires users to use a computer programming language, whereas, as a reward to these users, they have the full freedom to build up their own visualization. VTK's high customizability and open-source feature are largely responsible for its wide adoption in visualization.

These traditional visualization systems are very successful and have been widely adopted in both academia and industry, but drawn by the increasing requirements and the emergence of new technologies and hardware infrastructures, these visualization systems have kept evolving forward.

2.3 Distributed and Collaborative Visualization

Distributed visualization and collaborative visualization are two distinct, but interlinked concepts. Distributed visualization, as implied by the name, means that different parts of the visualization processes are executed on different host machines, while collaborative visualization focuses on the cooperation at the human level, with the visualization processes involving a number of participants. Some visualization systems involve both distributed and collaborative features.

2.3.1 Distributed Visualization

Distributed visualization allows different parts of the visualization pipeline to be run on different machines. Therefore, we can optimise the use of different hardware resources in order to achieve a good visualization result.

As classified by Haber and McNabb, usually a whole visualization process involves four types of steps with distinctive characteristics: Data, Filter, Map, and Render. We illustrate the distributed visualization by using an example visualization pipeline. In this pipeline, the Data step is a simulation which provides large-scale raw data every time step and also requires high computational capability on the machine where it is run. The Filter step can dramatically reduce the size of the data by applying some refining algorithm. Therefore, in the distributed visualization as the data needs to be transferred between different locations, the Filter step can largely reduce the time cost on data transfer. Computational capability is also essential in the Map step, which is the process of converting data into geometric representation. To gain high quality graphics at the Render step, powerful graphical processing hardware is needed. For end-users, a high resolution display screen will be important in order to have a clear view of the visualization result. However, without the distributed visualization, it is difficult and expensive to build up a machine which meets all the requirements for this example visualization pipeline.

The optimised distribution of the visualization pipeline is displayed in Figure 2.3. As both the Data and the Map require powerful computational resources, and there is large data transfer between the Data (simulation) and the Filter, they are allocated on the same supercomputer, while Render is run on a Graphics workstation and display the result on a high resolution screen.

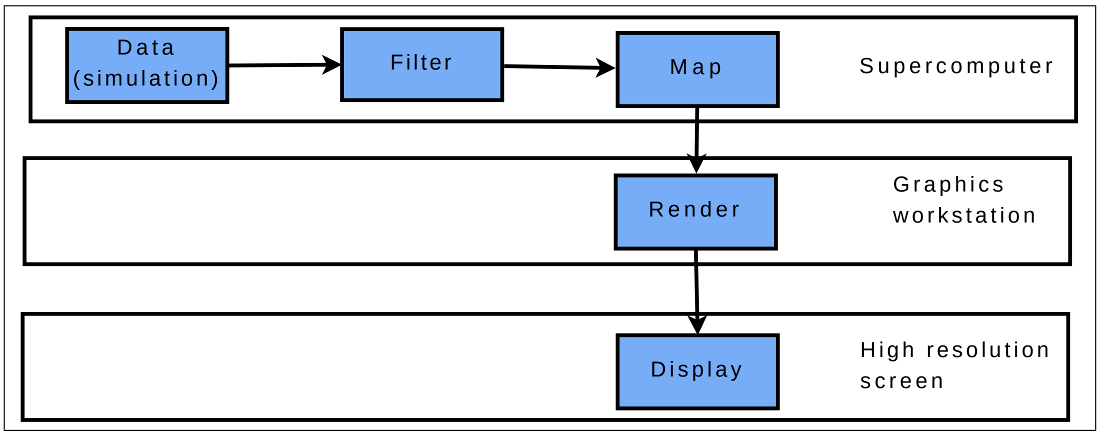


Figure 2.3: An example of distributed visualization

Taking advantage from their modular feature, most MVEs (i.e. IRIS Explorer, AVS, etc) enable users to distribute different modules onto different machines. More details and examples about these systems will be presented later in Section 2.3.3

2.3.2 Collaborative Visualization

Large research projects usually involve scientists who are from different disciplines and geographically distributed. Distributed visualization can spread visualization pipeline across multi-sites, but when these dispersed scientists need to analyse the visualization result together, they have to travel to the same location. Collaborative visualization can address this problem and avoid the travel by allowing scientists to work together cross the network.

Applegate's reference model [4] from the field of Computer Supported Co-operative Working (CSCW) is widely used to characterise different types of collaboration. As shown in Figure 2.4, the two coordinate dimensions stand for time and place. 'Synchronous' on the time axis means participants in the collaboration cooperate at the same time. Vice versa, 'Asynchronous' supports participants to collaboratively work at different times. It is notable that most CSCW projects and applications focus on the synchronous collaboration and asynchronous collaboration at different places. White board systems, Instant Messenger applications and video conference systems can be regarded as tools which support collaboration from different places at the same time. Asynchronous collaboration is the characteristic reflected by systems such as e-mail, newsgroups, Bulletin board systems (BBS), etc.

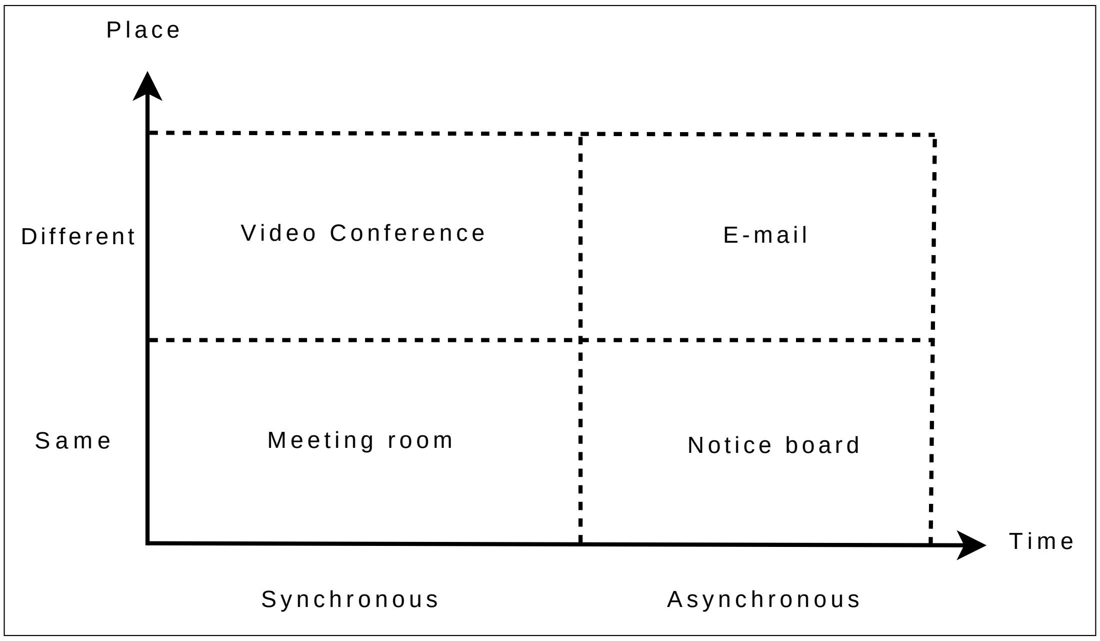


Figure 2.4: Applegate's reference model [4]

Visualization systems have also been extended to support collaboration. Most of this work focuses on synchronous collaboration, while only a little is relevant to asynchronous collaboration. The GRASPARC project [11] proposed an approach for recording the progress of a visualization process, by storing information in a structure called History Tree. Asynchronous collaboration can be possible by extracting a History Tree and sharing it with other participants in the same collaborative session. A number of projects have been undertaken in the area of synchronous visualization which are introduced in section 2.3.3.

Distributed visualization utilizes the hardware and software resources at different individual machines, and collaborative visualization enables the utilization of scattered human expertise. Some visualization systems can be characterised as distributed and collaborative visualization systems, which enables the collaboration at both low level (hardware and software level) and high level (users' expertise). Again, these systems will be reviewed in the coming subsection.

2.3.3 Distributed and Collaborative Visualization Systems

Web based visualization can be recognized as an example of distributed visualization. Part of the visualization pipeline can be executed on a Web server, and the remainder of the pipeline will be run locally. Wood [82] developed a WWW-based visualization system built on IRIS Explorer. A Web page interface is provided for users to feed parameters into the visualization pipeline. The output from the Web server is a geometry representation (VRML description), which is then rendered by the user's VRML browser locally. Figure 2.5 shows how the visualization pipeline is distributed using a Client/Service pattern. The system does not require end-users to install any visualization software or special hardware on their local machines. As long as they have access to the Internet and install a Web browser with VRML plug-in, they can use the system from anywhere. However, the structure of the system determines that users have to largely rely on the centralized server and lack the ability to reconfigure the visualization pipeline deployed on the server.

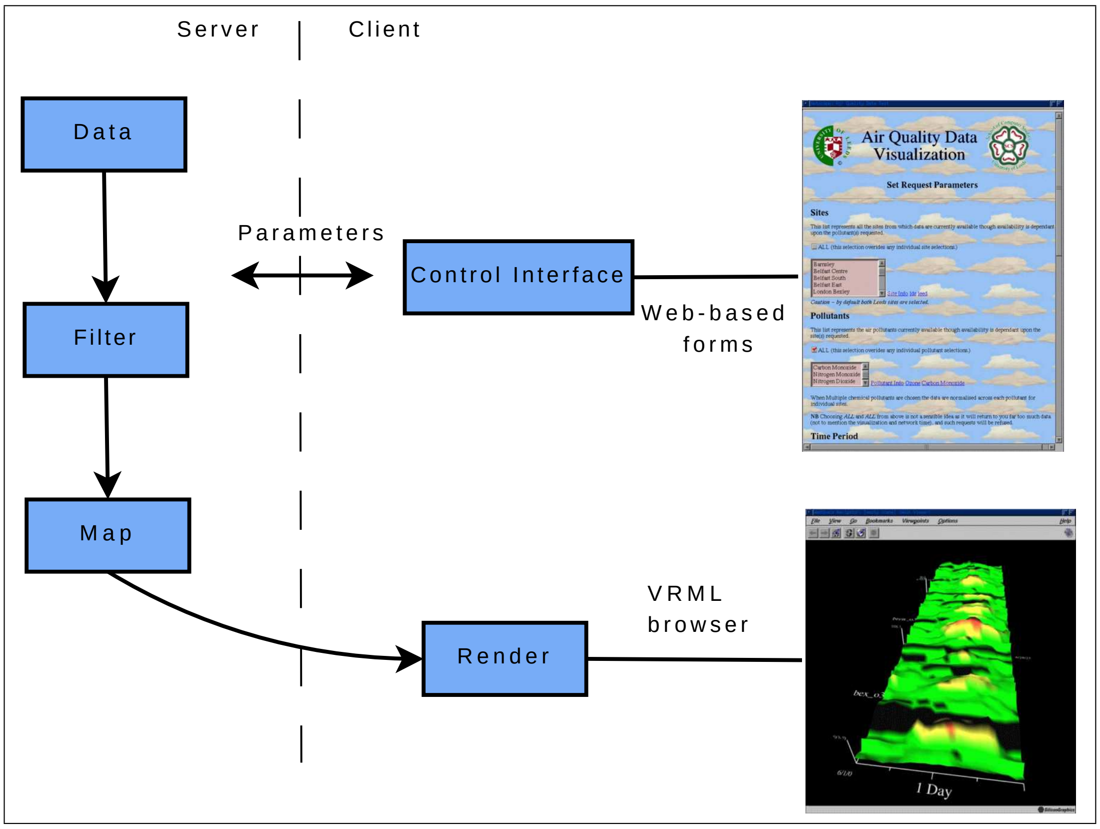
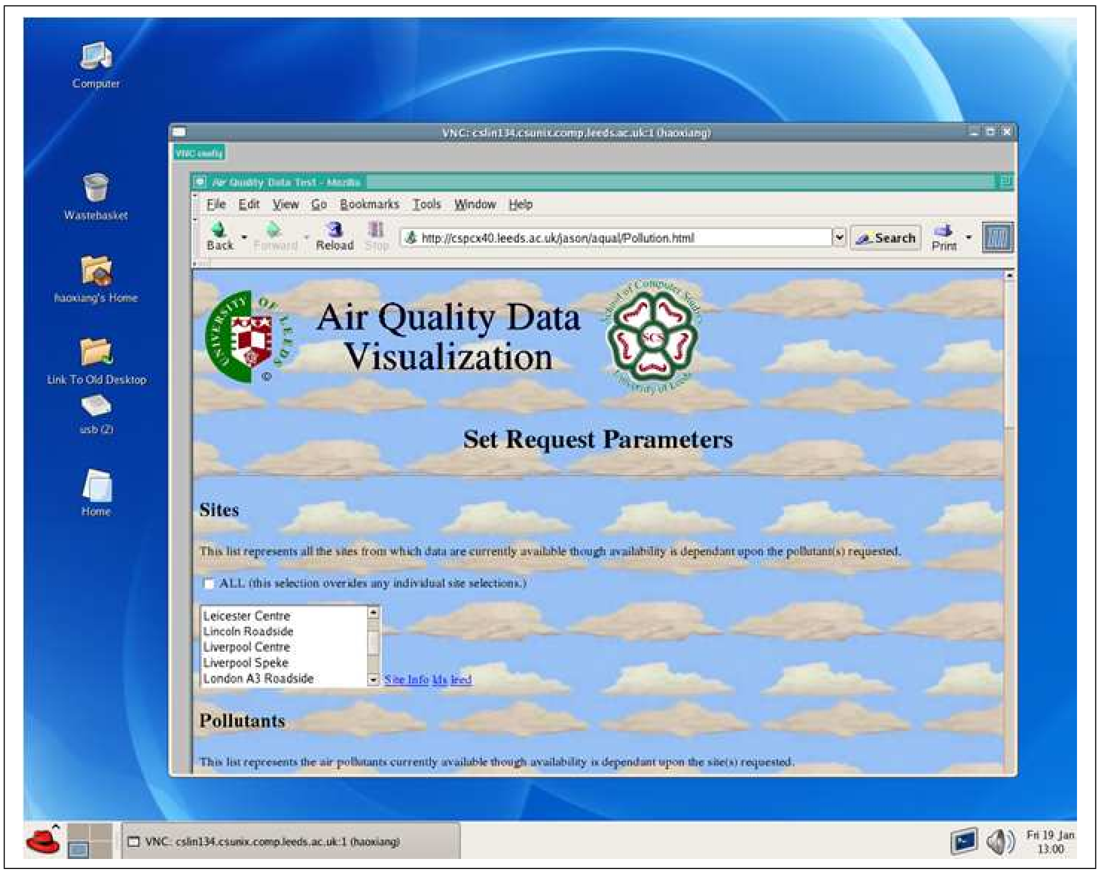


Figure 2.5: Web-based visualization project [82]

The distributed Web-based visualization systems do not allow different users to collaborate on the same visualization task at the same time. However, collaboration can be enabled by using some screen sharing software, such as **Virtual Network Computing** (**VNC**) [65]. VNC consists of a server part and a client part. Multiple clients may connect to the same VNC server at the same time. The server specifies what area on the screen is going to be shared and what control permission is to be offered to the connecting clients. VNC does not only share pixels on the screen amongst users, but also shares the control of mouse or keyboard which means clients can also manipulate the items on the shared screen. There are various flavours of VNC from different organisations, among which the widely used VNC systems are RealVNC [64] and TightVNC [73]. Figure 2.6 demonstrates the use of VNC to view Wood's Web-based visualization interface remotely.


Computer
Wastebasket
haoxiang's Home
Link To Old Desktop
usb (2)
Home
VNC: cslin134.csunix.comp.leeds.ac.uk:1 (haoxiang)
VNC config
Air Quality Data Test - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://cspcx40.leeds.ac.uk/jason/aqual/Pollution.html
Search
Print
Fri 19 Jan
13:00

Air Quality Data Visualization

Set Request Parameters

Sites

This list represents all the sites from which data are currently available though availability is dependant upon the pollutant(s) requested.

ALL (this selection overides any individual site selections.)

Leicester Centre
Lincoln Roadside
Liverpool Centre
Liverpool Speke
London A3 Roadside

Site Info kls leed

Pollutants

This list represents the air pollutants currently available though availability is dependant upon the site(s) requested.

Figure 2.6: Using VNC to enable collaboration on Web-based visualization by sharing screen

It is also notable that VNC can be used for the remote rendering. By running a VNC tool on a remote machine, the render part of a visualization pipeline run there can be controlled by users from their local machines through VNC clients.

Supporting collaboration through screen sharing (using tools such as VNC) is a very generic method which can be applied to any non-collaborative visualization system without re-implementing the original system, but the conflict over control of items on the screen can be a problem. If multiple users want to move the cursor in different directions at the same time, there is inevitable contention. Screen sharing amongst all the participants will lack the ability of keeping some individual work confidential during the collaboration, as they are only allowed to have one view. It can be that different users will want to view the visualization result from different view points, or one of the participants will want to keep the setting of parameters as confidential, and just share the other parts of the visualization.

To support more flexible collaboration, some visualization systems have been extended with collaborative features, by adding modules which support collaborative work into these systems. By adoption of this approach, AVS and NAG's IRIS Explorer were extended to be collaborative visualization systems.

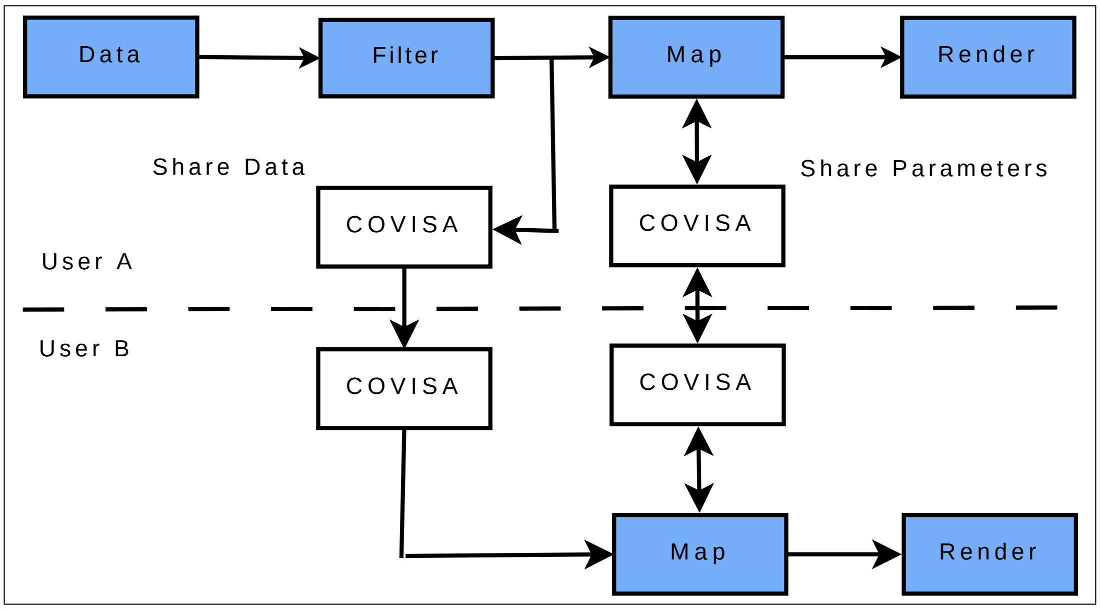


Figure 2.7: Collaborative visualization by using COVISA model

Wood et al introduced a collaborative visualization reference model in the **COVISA** [84] project, as an extension of the Haber and McNabb reference model. Two types of 'share' modules are involved in the collaborative reference model. One is used to allow data to flow from one participant's visualization pipeline to other users; and the other type of 'share' module is used to share the control of modules on each participant's pipeline. Figure 2.7 shows an example of using the COVISA reference model to share the data which flows out from User A's Filter module to User B's Map module and to share the control parameters of the Map module on both User A and User B's pipelines. By adding COVISA 'share' modules, data can be shared at any point on the pipeline and control parameters can be shared on any module in each participants' pipeline.

The COVISA collaborative model was implemented based on NAG's IRIS Explorer, allowing users the abilities of collaboration and distributing visualization pipelines. It has been integrated with IRIS Explorer's multiple platform installations, which means that collaboration can be achieved across heterogeneous machines by installing corresponding versions of IRIS Explorer on these machines.

Similar work has been done with AVS in the **Collaborative AVS (cAVS)** [16] project.

It provides a small number of additional modules that enable users to interactively share all major AVS data types, and the visualization parameter settings associated with other AVS modules. The **MANICORAL** project [24] also provided share modules for AVS by adopting a similar approach.

Compared with collaboration through VNC, systems such as COVISA, cAVS and MANICORAL have high flexibility. Users can run visualization pipelines individually and have the ability to share data or control at any point. Moreover, each user can have own interface, rather than sharing a unique interface as in VNC. Firstly, different interfaces allow users more confidentiality during the collaboration; secondly, it means different interfaces can be generated according to different experience and expertise of users. However, the flexibility also causes complexity at the same time. As users build and maintain their visualization pipeline individually, other users in the same collaborative session do not have the awareness of each other's actions during the collaboration.

Besides the systems mentioned above, which are extended from MVEs, some visualization systems are designed with collaboration in mind initially. The **COVISE** (**Collaborative Visualization and Simulation Environment**) system [21], which was initially developed by Wierse et al at the University of Stuttgart, allows users the abilities of distributing visualization processes on different hosts and collaborating on visualization. Like other MVEs, COVISE can link distributed processes on different hosts to produce a visualization. In COVISE, collaboration was achieved by replicating the user interface with a master/slave based floor control mechanism. The later releases of COVISE enhanced the performance by replicating the whole system on each cooperating host and synchronizing the interface by only passing the changed part in the interface.

Lovegrove [51] developed a collaborative visualization system called **SPIDER** which works in a similar manner to COVISE. The Visualization pipeline is replicated on collaborators' machines. Figure 2.8 displays how the system works. A session manager functions as the 'share' modules in COVISA or cAVS, and is responsible for the organisation and communication between different clients throughout the collaborative session.

Unlike collaborative systems such as COVISA, in the SPIDER system, conceptually there only exists one visualization pipeline across participants' machines, although it is replicated and run at each site. Therefore, it does not have the problem of awareness, as each user sees the same pipeline. The biggest challenge in SPIDER turns out to be the synchronization, as the replicated pipelines are run on different machines with different computational power, processing data at different speeds.

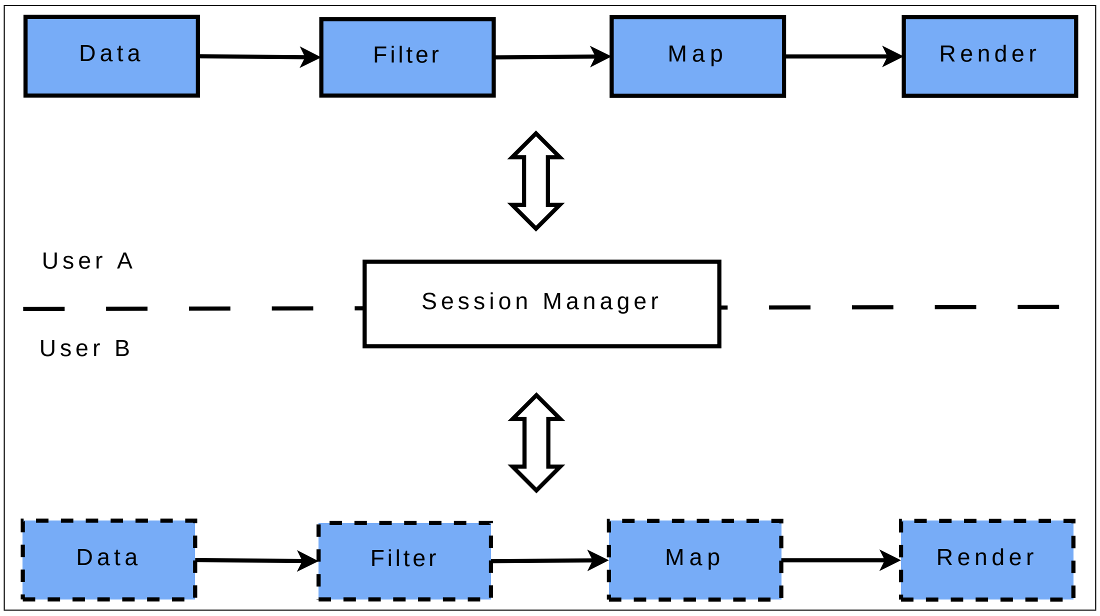


Figure 2.8: Collaborative visualization pipelines in the SPIDER system

2.4 Grid Computing

While visualization was increasing in importance in 1990s, a new research area emerged in computer science, called “Grid computing”. The Grid is recognized as the next generation of the ‘Web’. The term ‘Grid’ does not only consist of computer and network infrastructure, but also the software systems running on the Grid, specifications of the frameworks and the protocols for communication, etc. The ‘Grid’ aims to break through the boundaries between different organisations and institutions to share the resources on the network.

The ultimate vision of the Grid is to provide users with the ability to dynamically utilize resources to support large-scale, resource-intensive, and distributed applications [33]. By achieving that, the Grid will “provide the electronic underpinning for a global society in business, government, research, science and entertainment” [8].

This subsection discusses the Grid in depth and analyses the possibility of the usage of Grid technologies in the area of visualization.

2.4.1 The Emergence of Grid

The Grid originated from parallel computing, which attracted lots of research interest in the 1980s. Through the development of algorithms, software and architectures which support parallel computing, it became possible to build large-scale programs. However, even with parallel computing, the limited computational capability of a single machine was typically an obstacle for running very large programs. The Grid was initially born to create larger virtual supercomputers by spreading large-scale programs over well-coordinated distributed systems.

Meanwhile, research in many scientific areas needs large-scale computational infrastructure as a fundamental tool to process increasingly large data. Moreover, most key scientific research has become cross-disciplinary and involves geographically distributed human expertise and computational resources. Therefore, coordination and distribution have emerged as the main problems in conducting multidisciplinary and geographically dispersed scientific research projects. These are also recognized as two fundamental concepts in Grid Computing since 1990s [8].

The Information Wide-Area Year (I-WAY) project [23] is generally considered as the first modern Grid. Seventeen supercomputer centres dispersed across US were aggregated as a testbed for I-WAY and aimed to create an experimental environment for building large distributed applications and for exploring the issues of distributed data management and scheduling management. The I-WAY project revealed that research in Grid computing requires more emphasis on the integration, coordination and management of software, whereas distributed computing generally focuses on addressing the problems of the utilization of geographically dispersed computational resources.

Many Grid projects were initiated in order to investigate the full range of components, services and systems which make up Grid computing, following the pioneering work of I-WAY. One of the notable projects amongst these is the Globus project, which has evolved through several versions of toolkits to explore the development of a system-level infrastructure as a basic foundation the Grid computing. The Globus toolkits will be introduced in detail in section 2.4.2.

A widely accepted definition of the Grid was made by Foster and Kesselman in 1998, as “a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities” [33]. In addition to the original definition, Foster listed three features of the Grid [34]: 1) coordinating resources that are not subject to centralized control; 2) using standard, open, general-purpose protocols and interfaces; 3) delivering nontrivial qualities of service.

2.4.2 Grid Middleware

As the Grid aims to provide an infrastructure to provide easy access to widely spread heterogeneous resources, which is how the Grid was defined, a multi-layer structure and a software implementation based on this structure come to be very crucial to bridge the gap between the heterogeneous resources and easy access for end-users.

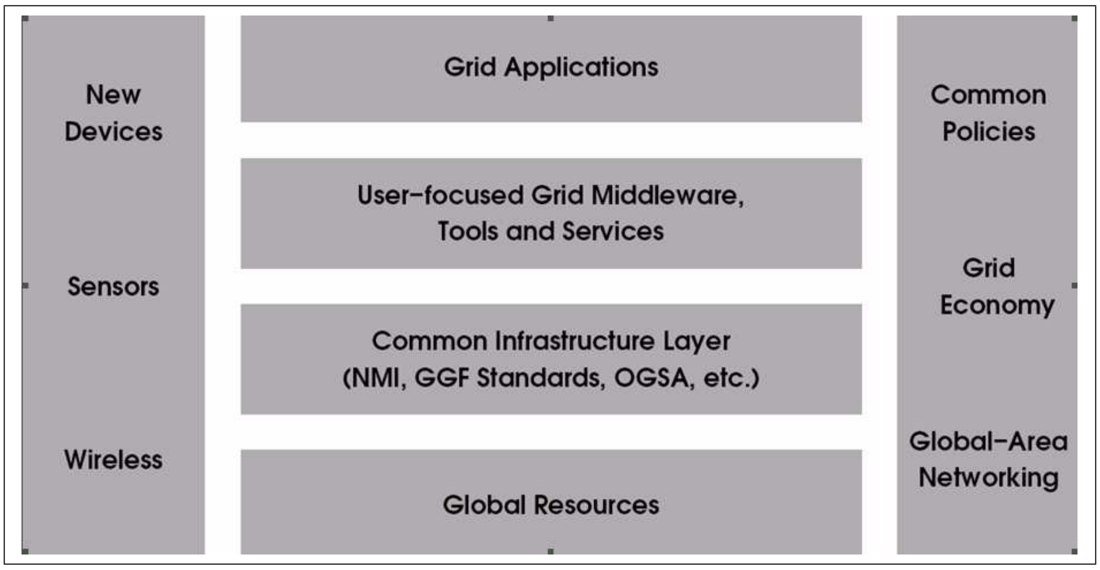

New Devices
Sensors
Wireless
Grid Applications
User-focused Grid Middleware, Tools and Services
Common Infrastructure Layer (NMI, GGF Standards, OGSA, etc.)
Global Resources
Common Policies
Grid Economy
Global-Area Networking

Figure 2.9: The Community Grid Model [8]

Berman et al abstracted a multi-layer model for the Grid, called the Community Grid Model [8]. As shown in Figure 2.9, the model consists of four horizontal layers and two vertical layers. The bottom horizontal layer represents heterogeneous resources across the Grid network which can be computers, network facilities, and even distributed data archives. As these resources are very disparate, a common infrastructure layer is added on top of the resources layer, consisting of a set of commonly agreed standards and specifications, such as the Open Grid Services Architecture (OGSA) [60] and the Web Service Resource Framework (WSRF) [88]. The next horizontal layer contains the Grid middleware, tools and services which are implemented on top of the specifications from the common infrastructure layer. The top layer represents the applications which use the Grid as a platform. The two vertical columns are the issues which affect the future development of the Grid. The column on the left represents the new devices and technologies, and the right column consists of the policies of sharing resources, the development of the Grid economy and the globalization of the Grid network.

To develop applications on the Grid platform, such as building visualization applications on the Grid, Grid middleware, that can provide relatively easy use of the heterogeneous Grid resources, is the layer we are most concerned with.

The Globus Toolkit [32, 37] is one such Grid middleware tool, which has been widely deployed and has evolved through different versions. The first version of the Globus Toolkit evolved from the I-WAY project and contains a set of basic services including job scheduling, security, resource monitoring and discovery, etc. Globus Toolkit 2 (GT2) continued to evolve to be a more mature Grid middleware with integration of a set of standard services/tools such as allocation manager, high performance data transfer (GridFTP) [38]. GT2 middleware has been successfully used for building the UK National Grid Service (NGS) [57], White Rose Grid (WRG) [85], etc.

The two early Globus Toolkit versions work in a batch processing manner. Although users can submit jobs to clusters through the GT tools, it still requires the application software to be deployed on the different machines across the network environment. In order to provide more user-friendly tools to the scientific community, rather than the batch processing manner, the third version of the Globus Toolkit (GT3) was released with the adoption of a Service-oriented architecture. GT3 can be considered as an implementation of the Open Grid Services Infrastructure (OGSI) [7, 34] which provides an infrastructure layer for the services defined by OGSA. The functionalities provided by GT2 were wrapped and exposed as Grid services in GT3.

GT3 is considered as an important development in Grid technologies, but it is incompatible with Web service standards. OGSI is considered as too dense, as it covers the full aspects of Grid service in a single specification. The Web-Service Resource Framework (WSRF) [88] replaced OGSI. Compared with the dense OGSI specification, WSRF consists of six specifications, covering different aspects of services. Another significant difference between WSRF and OGSI is that WSRF associates stateful resource with stateless Web service, rather than using stateful Grid service. Based upon WSRF, Globus Toolkit 4 (GT4) is built as a convergence of Grid service and Web service.

The UK Open Middleware Infrastructure Institute (OMII) [5, 62] also provides Grid middleware that aims to offer both academia and industry a sustained, well-engineered, interoperable, documented and supported Grid. OMII is compatible with Web services based on WSRF. Moreover, OMII provides a set of Grid-enabled tools such as OGSA-DAI for data access and integration on the Grid, GridSAM for job submission and monitoring, and Taverna for workflow management.

2.4.3 Web Service and Grid Service

Service-oriented architecture (SOA) defines a software system constructed from loosely coupled services. OASIS (the organisation for the Advancement of Structured Information Standards) [59] defines SOA as “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains”. Thus SOA can be regarded as a framework, and so it can be implemented with different kinds of technologies. In the early stages, most SOA systems were implemented with RPC (Remote Procedure Call) or CORBA. Now, Web service or Grid service technologies are widely adopted to build SOA systems. Compared with previous technologies (i.e. RPC or CORBA), the utilization of Web service or Grid service makes the components in SOA systems more independent, which implies the component services can be run on different platforms and written in different languages; and more standardized, in terms of the description of services and the communication between services.

The term 'Web service' refers to those services that use high level Internet communication protocols (i.e. HTTP or SMTP) to pass Simple Object Access Protocol (SOAP) [70] communication messages and have their interfaces described by the XML-based Web Service Description Language (WSDL) [86]. Web service is originally designed as a stateless technology. Each Web service only has one persistent instance which is invoked by all the clients. Clients are not allowed to keep stateful information on the service; otherwise it will affect other clients who use the same service.

While Web service keeps maturing through the effort from Web service communities, there is another strand of research from the Grid communities (i.e. Globus Alliance and Global Grid Forum) which can also be utilised to implement SOA systems. It is called Grid service which uses the concepts from Web service as a basis. OGSI defines how a Grid service can be implemented [7]. As a contrast to Web service, Grid service is stateful in nature. To use a Grid service, a service instance with lifetime needs to be created from the service factory. The service instance is integrated with Service Data Element (SDE) which is used to store stateful data for the client. An extension of WSDL (GWSDL) is used to describe Grid service interface.

Although the stateless problem of Web service is addressed by Grid service, there are still criticisms of Grid service and its OGSI specification [22], including the following aspects:

- OGSI does not separate the functions but puts them into one specification which can not support incremental adoption of extensions.
- It does not work well with the existing Web services and tools. For example, the OGSI introduced an extension of WSDL1.1, but some part of the extension is unsupported by existing services and tools.
- Grid service paradigm is too object-oriented. Grid service is based on Web service, but offers stateful features by encapsulating stateful resource with service. Some Web service purists insist that Web service should be stateless. In addition, some implementations do not accommodate dynamic service creation and destruction very well.

As a convergence of Grid service and Web service, **Web Services Resource Framework** (**WSRF**) is proposed by refactoring OGSI with standard Web service technologies. The criticisms of Grid service are addressed by building Web services based on WSRF:

- WSRF consists of a set of sub-specifications: WS- ResourceProperties, WS- ResourceLifetime, WS- RenewableReferences, WS- ServiceGroup, WS- BaseFault and WS- Notification family. The division of sub-specifications enhances the extensibility of WSRF.
- WSRF uses standard XML Schema mechanisms which are compatible with existing Web services and tools.
- Rather than integrating resource with service to build up stateful service, which is how Grid service is implemented, WSRF uses stateless Web service. However, stateful resource can be accessed through each Web service instance. Compared with Grid service, WSRF separates service and stateful resource, but still keeps the ability to store stateful information on service for each client who uses the service.

After WSRF was firstly proposed in 2004, organisations such as Web service communities and Globus Alliance have continued to work on the evolving of Web service standards. Globus Alliance released the WSRF based GT4 in 2005, through which Web service can be developed and deployed on the Grid instead of Grid service. It also means all the previous work that industry and academia have put into the Web service development can be easily migrated into the Grid environment.

2.4.4 Workflow Management in the Grid

Defined by the Workflow Management Coalition (WFMC) [80], workflow is the automation of a set of business processes according to a set of procedural rules. The emergence of SOA brought new characteristics to workflow management. When the SOA systems are implemented with Web services or Grid services, the business processes considered in workflow management become a number of services.

The Business Process Execution Language for Web Services (BPEL4WS) [10], released jointly by IBM, Microsoft and BEA systems, is a workflow language that describes the number of Web services that need to be executed, the order in which they are executed and the type of data they share. It grew from WSFL (Web Services Flow Language) (from IBM) and XLANG (from Microsoft) which are two WSDL-based descriptions of the composition of Web services, and has become the standard Web services Workflow description language.

A number of research projects have been carried out on the implementations of workflow management systems, especially in the scope of the Grid.

The Taverna project, as a component in the UK e-Science myGrid project, aims to provide tools to facilitate easy use of workflow and distributed computing technologies for scientific communities, especially for biology and bioinformatics [61, 71]. A Taverna workbench is provided as User GUI, which enables the construction and editing of workflows and supports loading and saving these workflows as an XML serialization (named as XScufl in the Taverna project). Taverna has been integrated as a standard tool in UK-OMII Grid middleware as a workflow management facility to easily compose distributed Web services and data resources to create a complex scientific research process.

The Triana project delivered an open source problem solving environment developed at Cardiff University. Triana combined an intuitive visual interface for users to construct complex workflows graphically and transparently [20, 53]. Therefore users can be relieved from the burden of the complexity of dealing with numerous and heterogeneous services. Users are allowed to save the workflows created using the Triana user interface in multiple formats, such as Directed Acyclic Graphs (DAG) and BPEL4WS. Users can replicate a previously created workflow by simply loading the saved description, or create a new workflow by altering previous work. Triana is similar to the Taverna system, but as a problem solving environment, it is more generic. Furthermore, the Taverna project focuses more on the Web service-based bioinformatics workflows, whereas Triana provides mechanisms for coupling workflows more tightly with Grid middleware tools.

The Kepler project has more emphasis on 'scientific workflow'. It considers less about control-oriented and task-oriented business workflow, but more about the dataflow passed between a set of processes for a particular large-scale and complex scientific task. It was also extended to support the composition of both Web services and Grid services. [52]

2.5 Service-oriented and Grid-enabled Visualization

Ever since the first emergence of Grid technologies and Service-oriented architectures, scientists have being working on evolving visualization by embedding these new technologies. This is a Service-oriented approach, we can consider implementing visualization functions as services. These services can be invoked by the client applications or other services to accomplish more complicated visualization tasks.

2.5.1 Potential Benefits from SOA and the Grid

By utilising SOA to develop visualization systems, we can gain a range of benefits:

- *Ability to interoperate* - Different individuals and companies can develop visualization services using different technologies and on different platforms and connect them together to generate the visualization result they require.
- *Ability to extend* - There are two aspects here: first, as long as the service interface remains the same, we can change the implementation or working environment of the service according to the requirements (e.g. for higher quality, higher performance, or the advantages of using a new algorithm), without affecting other parts of the system; second, when a new visualization function is required, we can simply build it as a new visualization service which will be easily linked into the visualization pipeline by using a SOAP message to invoke its interface.
- *Ability to collaborate* - Again there are two aspects to collaboration. Firstly, visualization specialists can collaborate to provide a global repository of visualization Web services, built and deployed by different specialists at geographically distributed locations. Secondly, visualization users can collaborate in a session, by connecting to the same services and sharing dataflow and parameter settings, in a similar way in principle to the collaboration-enabled extensions of MVE, such as COVISA, cAVS, etc.
- *Ability to easily migrate* - by implementing the visualization modules provided by MVEs as Web services or packaging several relevant modules as a Web service, the migration from MVEs to Service-Oriented visualization systems can be quite straightforward.

The Grid was born with the aim of providing easy access to heterogeneous and distributed resources, greater than can be provided by a single supercomputer. For visualization, the increasingly large data sets demand greater computational power than users typically have access to on the desktops. Meanwhile, as large projects usually involve scientists from different disciplines and institutions, it requires visualization to be decomposed into parts that can be distributed. For visualization the beneficial aspects from the Grid can be abstracted as:

- *Heterogeneous computational resources* - For visualization, particularly for large scale scientific visualization and real-time simulation, we can achieve higher performance from powerful Grid computational resources.

- *Standard, open, general-purpose protocols and interfaces* - Visualization systems/ components from different organisations can be linked together, if they follow the same standard.
- *Useful Grid middleware tools* - Visualization can harness existing Grid middleware tools. For example, GridFTP can be used for transferring distributed visualization data; the Access Grid can provide the capability for the interactions on the human level in collaborative visualization and for the sharing of geometric data stream between distributed visualization processes [48].

2.5.2 Service-oriented and Grid-enabled Visualization Systems

Building Service-oriented and Grid-enabled visualization systems is a very active research area in visualization. As there are many visualization systems which can be categorized in both the categories of Service-oriented and Grid-enabled visualization, we introduce them in the same subsection

GAPtk The Grid aware Applications Portal toolkit (GAPtk) [67] is an important and pioneering effort to utilize Web services in visualization. Compared with GAPtk, the previous application toolkits, such as Cactus and the Problem Solving Environment application (PSE) SCIRun, require a steep learning curve and also introduce a new development and execution environment for applications. It is usually intensive in terms of workload to customize an existing application, which users are familiar with, into the new environments. The GAPtk toolkit aims to develop a generic system consisting of a set of integrated services from which legacy applications can exploit Grid computing.

Figure 2.10 is the architecture diagram of the GAPtk application. GAPtk consists of two main parts: the client part and the server part. The GAPtk client part receives requests or callbacks from the PSEs and converts them into XML messages. This layer makes Grid resource and GAPtk services transparent to end-users, so that they only need to stay with familiar applications with their present knowledge. The GAPtk Web services on the Grid receive the XML messages about clients' requests and the implementation backend is responsible for actual processing of the requests. In the case that the process needs external resources (i.e. distributed data resources, computational power or third party services), the GAPtk server side forwards the request to other Grid resources.

The GAPtk implementation provides a generic toolkit that makes the Grid computing methodology available for existing visualization applications. A set of specific visualization services are provided, such as an isosurface service, but it lacks the ability for users to chain up these services and configure the visualization pipeline.

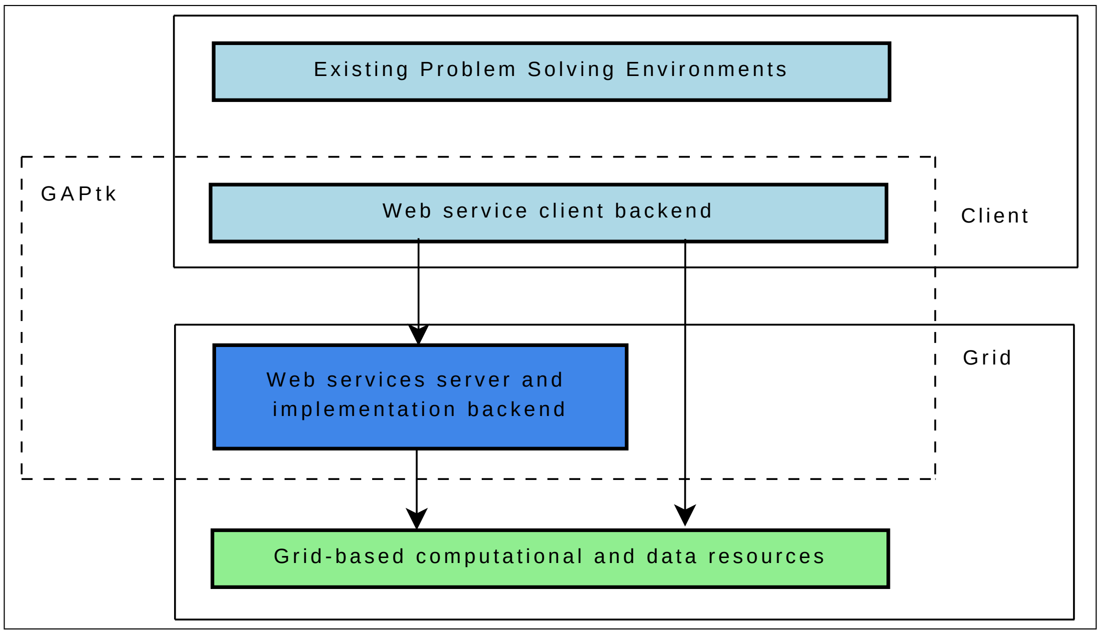

Existing Problem Solving Environments
GAPtk
Web service client backend
Client
Grid
Web services server and implementation backend
Grid-based computational and data resources

Figure 2.10: The basic architecture of GAPtk.

gViz Project

As one of the most popular MVEs, NAG's IRIS Explorer was extended to be Grid-enabled by the UK e-science gViz project [14, 40]. The gViz project incorporated Grid middleware within IRIS Explorer allowing it to be run securely in a distributed environment. The first strand of the project was to enable users to launch a visualization module on a remote machine from the IRIS Explorer run on their local machine; the second strand was to develop a gViz library which enables steering external components running on the Grid from the front-end visualization systems.

By using the gViz library, IRIS Explorer can run visualization pipelines partially on the Grid, but allows control of parameters and steering of simulations through the graphical interface on the user's desktop. Figure 2.11 and Figure 2.12 demonstrate two approaches to spread visualization pipeline across the Grid and the desktop.

In Figure 2.11, instead of running a simulation in the visualization pipeline, a simulation control module is wired into the pipeline created on the desktop visualization system, with the actual simulation running remotely on the Grid resource. The gViz library is used at both the desktop side and the Grid side. There are two types of communication displayed in the Figure 2.11: steering communication, which is the utilization of the gViz library to pass control parameters to the remote simulation; and the data transmission through gViz library.

An enhanced version is illustrated in Figure 2.12, using the ability of the Grid-enabled IRIS Explorer to run modules remotely. In this example not only the simulation is run

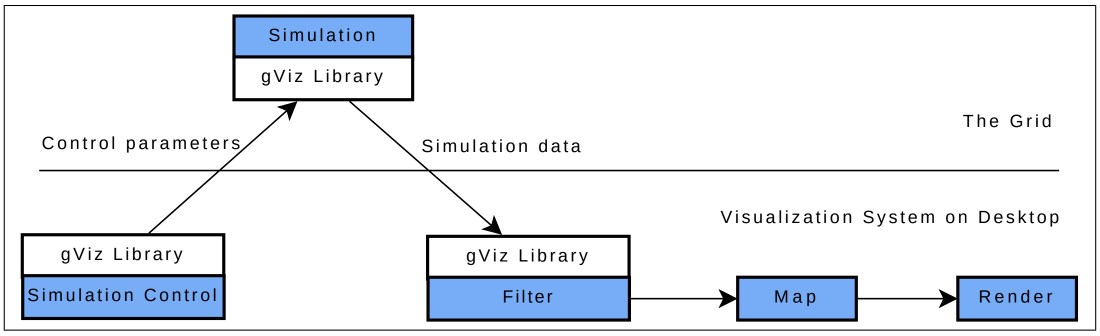


Figure 2.11: Run and steer simulation on the Grid by using gViz library.

remotely on the Grid, but also other modules in the pipeline are launched on the same machine/cluster where the simulation is. At the end of the pipeline running on the Grid, a rendered image is sent back to the desktop, rather than the large-scaled raw data generated from simulation. Compared with the first example explained in Figure 2.11, users can selectively run pipeline partially on the Grid with raw data, filtered data, geometrical data or images sent back to the desktop.

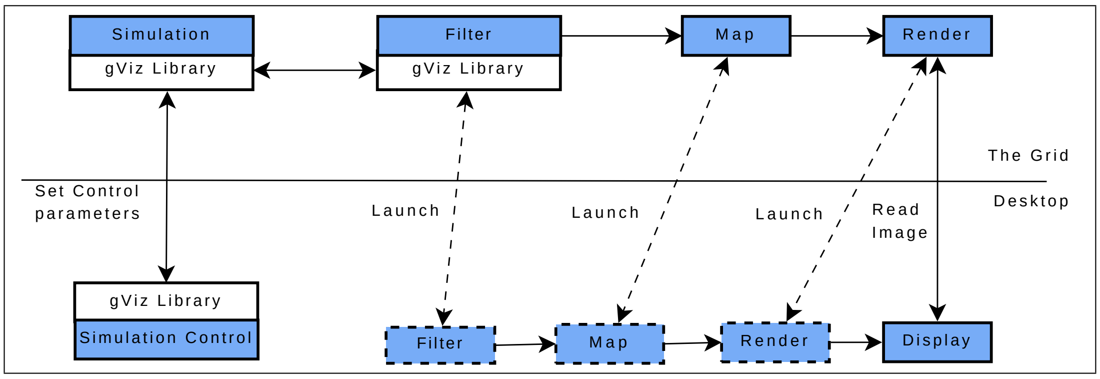


Figure 2.12: Distribute the visualization pipeline across the Grid and the desktop.

By using the COVISA extension in IRIS Explorer, collaborative visualization can be achieved in the environment of the Grid in the pattern which was introduced in Section 2.3.3.

As gViz project incorporates GT2 as the Grid middleware which works in a batch processing mode, the whole pipeline can not be recognized as a pure Service-oriented pipeline.

e-Viz project

Rather than moving traditional visualization systems onto the Grid or creating a specialized Grid-enabled visualization system, the e-Viz project [12, 66] aims at providing a generic Grid visualization system which can fulfil users' requests for any particular visualization techniques, and select the most appropriate hardware and software in order to implement it.

The e-Viz project puts much emphasis on visualization end-users and categorizes end-users into four groups: Grid / visualization specialists - users with knowledge of both the Grid and visualization; e-Scientists who are the users with knowledge of the Grid technologies but not visualization; visualization specialists who are users with knowledge of visualization but not the Grid, as a contrast to e-Scientists; and application scientists who are users that do not have expertise either in the area of the Grid or visualization, but in other scientific areas. The e-Viz project targets mainly the last category of end-users, application scientists, by integrating the visualization system and the Grid resource and providing it as a black-box for end-users.

Figure 2.13 shows the architecture of the e-Viz system. The design of the e-Viz system comprises three parts: the client components; the broker components; and the server components.

The client components consist of a launcher and a generic user interface. The launcher is the user's entry point into the system. The user can specify the requirement of visualization through this wizard-based interface, such as the target data to feed into the visualization system and the desired visualization output from the system. The generic user interface is generated dynamically according to the visualization pipeline created by the system, with widgets that enable users to set parameters for the visualization. The generic user interface also includes a rendering library which can render multiple formats of possible visualization results (encoded using JPEG, PNG, Colour Cell Compression Delta and Run-Length Encoding algorithms) returned from the system.

The user's requirements are sent to the broker components in an XML description. The broker will find the most suitable solution to fulfil the user's requirement based on an integrated visualization knowledge base and the decision making module. The solution generated from the broker and described in XML format will be sent back to the client components, and then the client can start to communicate with the server components listed in the right column in Figure 2.13.

The Grid middleware is the only software which must be installed on the server, as it manages the allocation of the available Grid resources to the visualization tasks submitted from client. The visualization software can potentially be transferred onto the server at runtime. The visualization software does the actual visualization tasks and sends the result back to the client.

Although the e-Viz system offers application scientists great opportunity to harness Grid resources and does not have a steep learning curve for learning a particular visualization system, it lacks the freedom for advanced Grid or visualization experts, as the

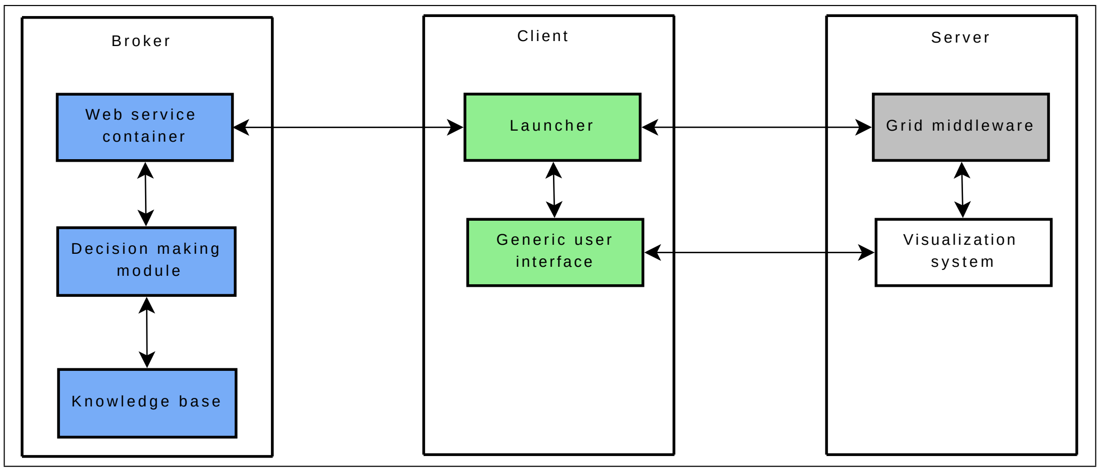


Figure 2.13: The architecture of e-Viz system.

whole system is viewed as a black box for end-users. Furthermore, the Grid middleware embedded in the system is still based on the batch processing GT2 rather than the later version of Globus toolkit which is compatible with standardized Web service specifications.

Grid Visualization Kernel project

The Grid Visualization Kernel project (GVK) [49] is designed as a generic system that allows running an arbitrary part of the visualization pipeline on the Grid. The implementation is based on IBM's OpenDX modular visualization system. The system also aims to provide transparent visualization for end-users. By feeding in requirements of the visualization into the system, GVK will automatically define a visualization pipeline and allocate suitable resources to run these visualization processes.

Figure 2.14 displays how a simulation client utilizes GVK to generate a visualization pipeline and gets the result. During the initialization stage, the simulation sends a request to the GVK portal, which describes the visualization requirement. The portal forwards the request to the GVK visualization planner which queries the Grid middleware to find out the available resources and allocate these resources to visualization processes through the Grid middleware tools, e.g. Globus Resource Allocation Manager (GRAM). At the same time, GVK sends back the input interface to the simulation, through which the simulation can supply latest generated data to the GVK system and keep updated with the latest visualization result from the GVK system.

Although GVK is generic in the sense that it can be implemented in different visualization systems, and transparent which means users do not need to consider the visualization pipeline running on the GVK system, it still requires users with visualization knowledge to build up the local part of the pipeline. Meanwhile, some Grid knowledge

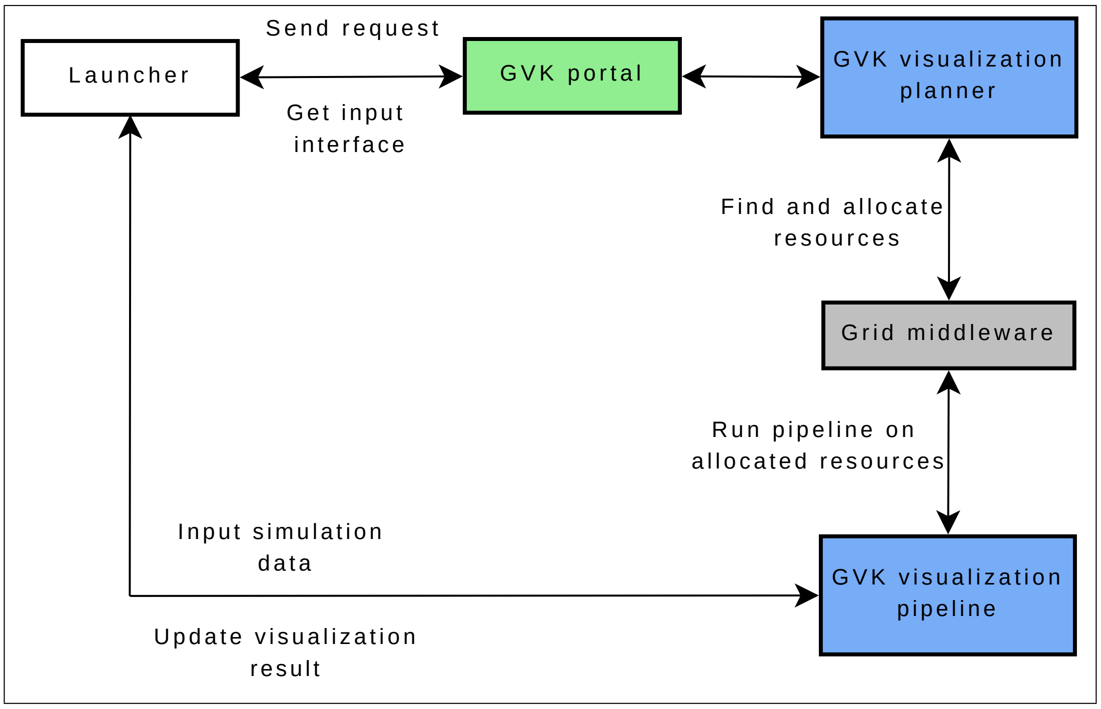

Launcher
Send request
Get input interface
GVK portal
GVK visualization planner
Find and allocate resources
Grid middleware
Run pipeline on allocated resources
GVK visualization pipeline
Input simulation data
Update visualization result

Figure 2.14: A simulation client sets up visualization pipeline through GVK system.

is also required for end-users to develop clients which can communicate with the GVK system.

Resource Aware Visualization Environment project

Different from the above projects, the Resource Aware Visualization Environment project (RAVE) [39] aims at enabling collaborative sharing of resources on the Grid and supplying a wide variety of rendering devices from PDAs to high performance systems. Collaboration is the one of the isssues the RAVE project has tried to support. The collaboration allows multiple users to render and view the same data source, but through different interfaces. For example, the user in the lab with high performance machines and a tiled-wall display screen can collaborate with a colleague on a trip with a PDA on hand which only displays the relatively small-scale images sent from the remote rendering service.

The basic RAVE architecture is shown in Figure 2.15. A central data service has been implemented in the RAVE system, which can either feed in data for rendering from local static data files or from other remote distributed data services. Two types of clients are provided by RAVE: an active render client integrated with a local rendering component and a thin client which is usually deployed on light-weight interface terminals (i.e. PDAs or mobile phones) utilizing the remote rendering service. A RAVE render service is offered for those thin clients. It connects to RAVE data services and requires a copy of the current dataset. After processing the data, the render service sends images to the

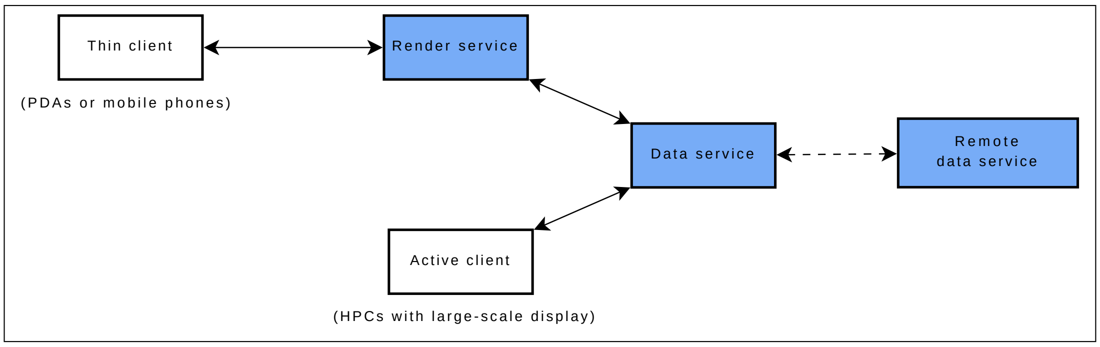


Figure 2.15: The diagram of the architecture of the RAVE system.

connecting clients.

The RAVE project enables relatively limited collaboration, as it only provides collaboration on the visualization rendering processes, not on the whole visualization pipeline. The ability of building up a visualization pipeline is not provided by the system either, as only data service and render service are implemented.

Java-implemented Grid visualization system

The Java-implemented Grid visualization system (jgViz) [29, 30] uses Grid middleware to enable transparent access to pipelines executed on parallel machines. The jgViz application also focuses on rendering large-scale graphics on network nodes and displaying the graphics on a tiled-wall screen.

Chromium [44], a general-purpose system for enabling cluster visualization, is used in the jgViz implementation for rendering large-scale graphics on distributed nodes and then displaying the graphics on a tiled screen. A Chromium rendering system is made up of two principal types of nodes: client nodes which produce graphics output and server nodes which handle these client data steams.

Accordingly the jgViz system is designed in the way shown in Figure 2.16. The jgViz client firstly finds all the available nodes on the Grid through Grid middleware. After this has been done, the Chromium configuration script and the application binary are sent to the Mothership node and the Application node. When the application is launched on the Grid, the Mothership node controls the render nodes on the network (with Chromium component on them) according to the Chromium configuration it received from the jgViz client, to render the graphics in parallel.

The jgViz project provides a generic and transparent way to utilize the Grid resources for visualization. However, as Chromium is used for cluster rendering, only OpenGL based applications can be compatible with the system. Moreover, as GT2 is chosen as the Grid middleware for the project, it is difficult to migrate a service-oriented visualization pipeline into the system.

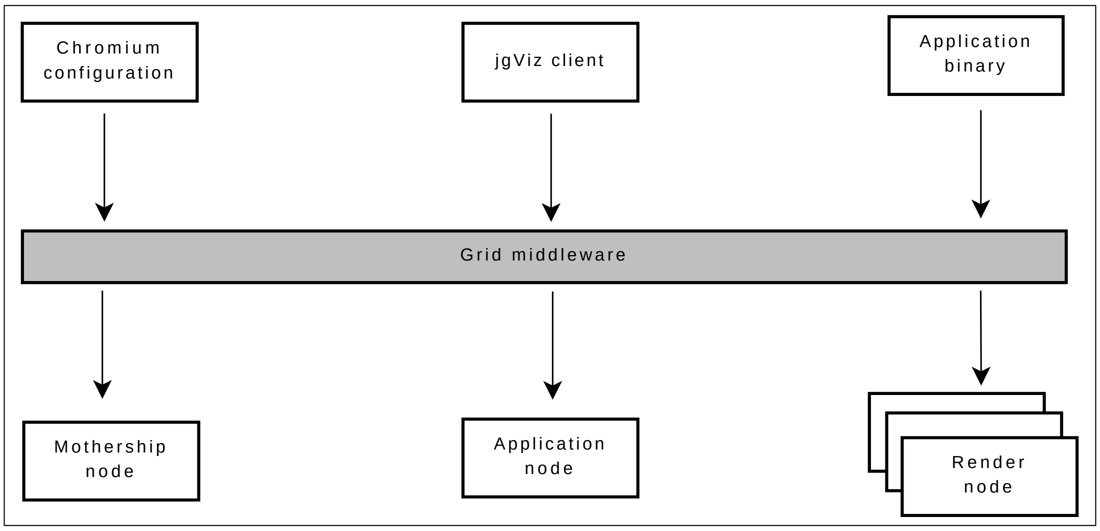


Figure 2.16: How jgViz application works.

Grid-enabled Service-oriented visualization

Charters invented a novel approach to building the Grid-enabled Service-oriented visualization in the e-Demand project [17, 18].

The traditional visualization pipeline offers the possibility of dividing a visualization process into a set of component parts and then deploying and running these parts on appropriate resources. The MVEs mentioned in section 2.2 implemented these component parts in the visualization pipeline as modules, whereas, in Charter's approach, these parts in the pipeline are implemented as services, and deployed on appropriate Grid resources.

Figure 2.17 is the distributed architecture for Charter's design. The visualization pipeline is implemented as a set of interconnected services, which are deployed across the Grid. These distributed visualization services can be steered through a centralized manager service. The two main challenges for this approach are: the coordination of data and the composition of the Service-oriented pipeline. The data coordination between services is achieved through the mechanism of the Grid level streaming; and the pipeline composition is supported by using the external pipeline composition tool.

Charter's system has undergone several iterative implementations. The Grid service versions based on GT3 and the Web service implementations all turned out successfully to prove the design concept. However, there still remain some issues. Charter's Grid-based visualization pipeline was designed with the feature of collaboration, but this was never implemented. Secondly, there is no mention of recording a previous visualization pipeline and replicating it for different users or at a different time. Thirdly, the invoking manner follows a client/service communication manner. The client gets response from the service only after it sends request to the service. However, in a visualization pipeline,

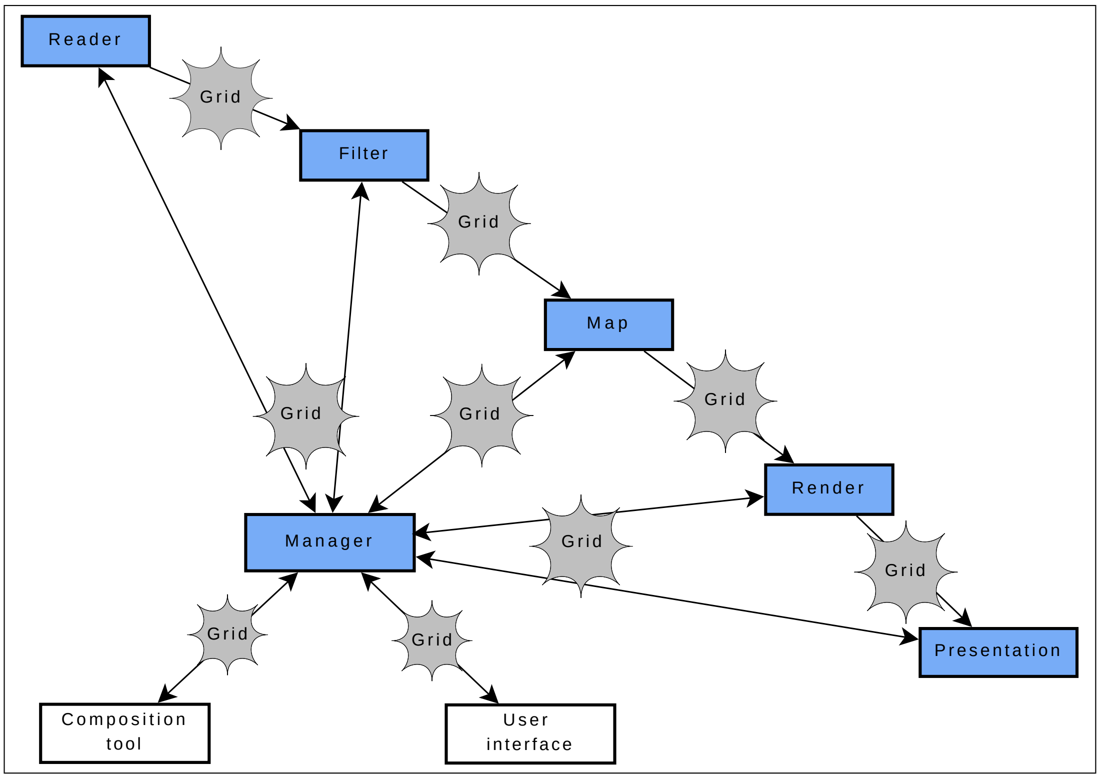


Figure 2.17: Charter's distributed visualization pipeline on the Grid [17]

the module downstream sometime needs to receive latest generated data from the module upstream automatically without keeping sending requests to it.

A critical analysis of the existing Grid-enabled or Service-oriented visualization systems is introduced in Chapter 3, which also derives the motivations of the research work explained in this thesis.

2.6 Formal Description of Visualization

While there is growing interest in the Grid community in workflow language and workflow management (which have been introduced in Section 2.4.4), the visualization community has also studied description languages to represent visualization pipelines and even to capture the provenance of visualization.

As a strand of the gViz project, Duce and Sagar [25] developed an XML application, named as **skML**, for describing the connection of modules in a pipeline for distributed and collaborative visualization.

The design of skML takes its inspiration from the Skm scripting language in IRIS Explorer. As Skm is not a standard language, standard XML was adopted to revise the Skm

```
<module id="MyRender" name="Render" style="Left:100;Top:100;">
        <param name="filename">pic.jpg</param>
</module>
```

Table 2.1: A snippet of skML describing a "Render" module.

```
<link id="MyLink">
        <module ref="MyData" out-port="DataOut">
        <module ref="MyFilter" in-port="DataIn">
</link>
```

Table 2.2: A snippet of skML describing a link between two modules.

language. The "module" and "link" elements are two fundamental elements in skML. The "module" element stands for a process stage in the visualization pipeline; and the "link" element describes how two modules are connected to each other.

The skML snippet listed in Table 2.1 describes a module called "MyRender" which is of type 'Render'. The style attribute indicates the position where the "MyRender" module appears on an MVE's visual editor window. The "MyRender" module is provided with a parameter called "filename" which has a string value of "pic.jpg".

A simple example of the "link" element is in Table 2.2. In this example, we name a link as "MyLink". It represents a connection between the out-port called "DataOut" of the "MyData" module to the in-port called "DataIn" of the "MyFilter" module.

A particular feature of skML is the use of Resource Description Framework (RDF) to associate annotations with skML elements: for example to describe the environment where a module is to be run or to give some supplementary explanation about a module.

skML has been used for the description of visualization pipeline for several visualization systems such as VTK, IRIS Explorer, AVS, OpenDX and so on. As a generic description language, skML describes connections of visualization processes, but it does not define the semantic definitions of these processes.

The **VisTrails** system from University of Utah [35] is an action based visualization description application, which aims to capture provenance information of the actions during the process of visualization and data exploration. The description is also recorded in XML format. But VisTrails has different purpose with the skML language, it provides an infrastructure to build up asynchronous visualization.

2.7 Summary

There are two strands of research which feed into this work.

The first strand is the research in the area of visualization. Drawn by the requirements from scientific communities, visualization is processing increasingly large datasets and handling more distributed computational resources and human expertise. These two aspects prompted modular visualization systems with distributed features embedded in the design, and collaborative visualization systems for geographically distributed researchers.

Secondly, the emergence of the Grid and the popularity of Service-oriented architecture gave visualization new opportunities to evolve. The Grid provides heterogeneous computation resources. By using Grid middleware, these heterogeneous resources can be easily and transparently accessed. The Grid concept also includes the standardization of protocols and specifications, which bring great advantages in developing distributed systems. There are also other Grid infrastructures offered to application development such as Grid workflow infrastructures and Grid data transfer infrastructures. The Service-oriented architecture is a revolutionary framework from the design to the implementation of applications. It defines the interactions within an application and considers an application as a set of loosely coupled components.

Hence, when these two strands are brought together, can visualization benefit dramatically from this convergence? Some pioneering works about developing distributed, collaborative and Grid-enabled visualization have been introduced in this chapter. In Chapter 3, we critically analyse these relevant works, from which the requirements for a new visualization system are derived.

Chapter 3

Motivations and Requirements

In this chapter, we summarise the unfulfilled requirements from the previously existing visualization projects, especially in the aspects of Service-oriented and Grid-enabled visualization and collaborative visualization. Based on these critical appraisals, we derive a set of requirements for our research, and hopefully, through this research, we can achieve a new vision for visualization.

3.1 Critical Appraisals on the Precursory Work

Based on the analysis of the previous relevant work mentioned in Chapter 2, the following issues can be recognized as unaddressed by most of the visualization projects that have studied Grid technologies for visualization.

3.1.1 Service-oriented Visualization Pipeline

Some of Grid-based visualization projects (such as GapTK, gViz, e-Viz and jgViz) were implemented with GT2 working in the pattern of batch processing. In these systems, visualization tasks or partial visualization tasks are submitted to the Grid middleware by transferring the binary code onto the resources which are allocated to run these tasks. Working in this pattern is incompatible with the Service-oriented paradigm, which involves passing messages between service and client rather than binary codes.

The Service-oriented concept appears in some of these projects. In the RAVE system, data service and render service are deployed for remote clients to use the results generated from these services. To some extent, the GVK system can also be regarded as an example of applying the concept of ‘service’. In the GVK system, visualization tasks processed using Grid resources are wrapped and can be accessed from a Grid portal, working similarly to a service deployed on the Grid. However, in these systems, only part of the visualization pipeline has been implemented in a Service-oriented architecture.

Charter’s work is an exception in that it was designed to implement the whole visualization pipeline as a set of connected services.

3.1.2 The Ability to Customize Visualization Pipeline

The ability offered to end-users to compose customized visualization pipelines of modules through a visual editor tool is a great advantage in traditional MVEs.

The projects extending traditional MVEs onto the Grid (such as the gViz project) inherited this feature, but some projects do not allow users to create their own pipelines. Although the e-Viz system and the GVK application can automatically create visualization pipelines based on users’ requirements and predefined visualization knowledge bases, these pipelines encapsulated as black-boxes are transparent for end-users. Users who are not visualization experts can benefit greatly from these systems, however, they lack the flexibility of creating user-designed pipelines which can be essential for visualization experts.

Again, Charter’s work is an exception to this criticism. An external editor tool is combined into his application for users to compose services into a visualization pipeline.

3.1.3 Supporting Collaborative Visualization

Projects such as gViz, e-Viz, GVK, etc. are not driven by the desire to provide collaborative working. But through applying external tools or collaborative extensions, it is still possible to support collaboration. By using the COVISA modules in IRIS Explorer (introduced in Chapter 2.3.3), the gViz system can enable users to work collaboratively with the utilization of the Grid resources in visualization. The VNC systems (also mentioned in Chapter 2.3.3), as generic tools, can also be applied to all these non-collaborative systems. However, by using these collaborative extensions or tools to gain collaboration, we inherit any of their original weakness or drawbacks. For example, the use of COVISA results in the loss of awareness of other users’ actions in the collaboration, and VNC tools cause inflexible collaboration.

RAVE considers collaboration as a key issue in the design, but collaboration does not take place throughout the whole pipeline, only at the rendering step.

For visualization systems built in a Service-oriented architecture, there is a clear opportunity to support collaboration, as clients can connect to the same service to share data or visualization configuration. Unfortunately, most of the existing work has not taken full advantage from this Service-oriented characteristic. Charter's Service-oriented visualization system was born with the collaborative feature in the initial design, but did not fully turn it into implementation.

3.1.4 Accommodating Users with Different Backgrounds

As represented in the e-Viz project paper [66], visualization end-users can be categorized according to whether they have expertise in the Grid technologies, visualization skills or other scientific knowledge. Most of the previous Grid-enabled visualization projects target on one or several user categories, rather than the whole range of users.

Users should be confident in both the Grid technologies and visualization in order to use GapTK and GVK systems, as users not only need to create whole or part of the visualization locally, but also need to know how to make use of the local client component (GapTK client component) or the system's remote interface (GVK portal). Users who do not have knowledge about either the Grid or visualization can be accommodated well with the e-Viz system, as a wizard interface is provided to adopt users' requirements, and visualization solutions are created by the system automatically. However, advanced users with expertise in visualization or the Grid may not like this system, as it does not provide much customizability. Charter's Service-oriented visualization system gives great flexibility to build up visualization pipeline or implement special visualization functions as a new service, but it involves a steep learning curve for scientists from other research areas such as biology or chemistry.

3.2 Motivations of the Research

The motivations of our research come from the following aspects:

For scientific research: More and more large-scale data and simulations in scientific research need to be processed, involving highly intensive computation. This prompts a requirement to take the best advantage of new computing technologies and infrastructures. Amongst these new emerging technologies, the Grid looks promising for visualization, as has been proved in the previous pioneering work. We aim to explore further the utilization of the Grid for visualization through our research.

For users: Users in visualization may come from multiple disciplines, and they are usually at different levels of visualization knowledge, computing skills and even Grid computing experience. Distinctive from other visualization systems, we look for a new framework that can offer a choice of an interface where the pipeline can be configured (for visualization experts) or an easy-to-use interface where the details of the pipeline are hidden (for application scientists). By this framework, different users with different knowledge background can collaboratively work with each other and make the best utilization of their expertise.

For Grid community: Different organisations and nations have invested massive funding in building up Grid infrastructures, however these Grid networks are separate. To make full use of these infrastructures, it might be worth to connect these separated networks and generate a globalized virtual Grid network. Our research plans to carry out some visualization experiments to test whether this idea is feasible and achievable.

3.3 Requirements

The requirements of our research are derived from the critical appraisals of the relevant work in section 3.1 and the motivations explained in section 3.2. These requirements are listed as below:

- Be able to utilize the Grid computational resources to visualize large-scale data sets by building a visualization infrastructure based on Grid middleware.
- Be compatible with the standard Web service specifications and implemented with Service-oriented Architecture, so that the visualization system can be extensible.
- Support collaboration, which includes collaboration in the providing of visualization functions/services, the building of visualization pipelines and the steering of these pipelines.
- Enable users to easily build up customized visualization pipelines through a visual editor window which supports drag-and-drop.
- Flexibly accommodate users with different background knowledge: visualization developers will be able to create customized visualization services and integrate these services into the system; visualization experts can build up pipelines using

their visualization expertise; and scientists can steer these pipelines by setting parameters according to their special scientific knowledge without caring about where the visualization is actually processed or how the visualization is produced.

- Be able to be widely distributed. Visualization services deployed at different locations on the Grid network, or even across different Grid networks can be linked together to produce visualizations through the system.

Besides the above requirements, there is another need to support asynchronous working, where the participation is spread over a period of time. Visualization information including how the pipeline is built and how the parameters are set is maintained over time by recording this information as a formal description. Users can connect to the system at a later time and pick up other users' previous visualizations by loading the description saved there by other collaborators.

Furthermore, security issues also need to be addressed. As the system is distributed and supports collaboration amongst multiple users, it is essential to define and restrict which resources can be accessed and what actions can be allowed for users.

3.4 Vision of the Research

The overall vision for our research is to produce a next generation visualization system - based on the well established Modular Visualization Environments which link basic visualization modules to build up a pipeline, but implemented upon service-oriented architecture and the Grid infrastructures, and involving collaboration between users and between visualization service providers, at a global scale.

In the vision, a worldwide repository of visualization services is jointly assembled by different developers, teams or institutions with the adoption of the standard Web service specifications and the utilization of globally distributed Grid resources. Visualizations can be collaboratively created by users with visualization expertise who can select suitable services from the repository and interconnect them appropriately. Formal descriptions are generated to capture the full information of these visualizations. Based on these descriptions, GUIs are dynamically created through which visualization users can collaboratively steer the visualizations. By saving the visualization descriptions and making them sharable for other participants, asynchronous collaboration is supported in both the stages of creating and steering visualizations.

The ideal visualization system in our final vision is embedded with security control mechanisms which can control the access to the system, to a particular collaborative session and even to an individual visualization service within a particular collaborative session.

3.5 Summary

Derived from the appraisals of previous relevant work and motivations arising from science itself, the Grid community and visualization users, we set out the requirements of our research as the following:

- Grid enabled
- Distributed and Service-oriented
- Support customizable visualization
- Flexible for different users
- Synchronous and asynchronous collaboration

Based on these requirements, a global visualization system is drawn as the final vision, in which there exists a visualization service repository consisting of services distributed at a global level, and whereby worldwide users can collaboratively provide and use these visualization services.

However, how can we find the path which can lead us to this final vision we drew? This is the question our research aims to address, hopefully, through which our 'visualization dream' can come true.

Chapter 4

The Initial Experiments

To achieve the vision of the next generation visualization system explained in Chapter 3, several experiments have been carried out, not only for gaining first-hand experience of using the Grid computing technologies in the implementation of our system, but also for the proving of the feasibility of the system requirements listed in Chapter 3. These early-stage experiments can also be seen as the initial steps of our research, based on which the detailed framework and the implementation of our Grid-enabled collaborative visualization system are derived.

These experiments started from exploring early Web service technologies with stateless services for visualization; then, evolutionally moving onto the Grid by using stateful Grid services; and finally shifting back to Web services following standard WSRF specifications and remaining on the Grid.

4.1 The First Step: Visualization Web Services

Although our first step is "one small step" based on the previous existing work, it is "a giant leap" for our research, as we started to investigate how to exploit Web service technologies for visualization by providing visualization functions as services with wide or even global access.

4.1.1 Objectives

The idea of building visualization Web services evolves from Web-based visualization. In terms of implementation, what needs to be done is simply to wrap the server part of a Web-based visualization system into a Web service, while the benefits from doing this can be significant.

The usual approach to access a Web-based visualization is through a Web page and there is no standard definition of the communication messages sent between servers and clients. It causes the difficulty of integrating the visualization result from a Web-based visualization system seamlessly into end users' own code or other tools. For Web services, the communication messages between services and clients are standard XML messages. Visualization Web services can be accessed from clients implemented in different programming languages, as long as they 'talk' in the same language with the services: messages in XML format. Therefore, the client can be customized code, existing tools, or even other services.

By this experiment, we aim to explore the advantages of building Web services for visualization. The objectives of the experiment include:

- Deliver some basic visualization functions as Web services. It is the main task of this experiment.
- Access these services through different kinds of clients. The aim is to investigate the usage of visualization services from the end-user's viewpoint.
- Provide ability for users to integrate visualization services from different providers and generate visualization according to their requirements.

4.1.2 Design and Implementation

As two of the basic methods of visualizing scalar 3D datasets, slice and isosurface are chosen as the two visualization functions to implement with Web services. Slice visualization makes a plane cut through the volume, at a fixed position along one of the coordinate axes with colours used to represent the value at each point. The isosurface visualization extracts a surface where the scalar value is estimated to be at a constant, threshold value.

The slice service and isosurface service are implemented in different programming languages and deployed on different machines, simulating the effect of being provided by different service providers. Clients with different programming languages are also developed to simulate the flexibility of accessing the visualization services. The last part of the experiment is to combine these two services, so that end-users can view both slice and isosurface in a single scene.

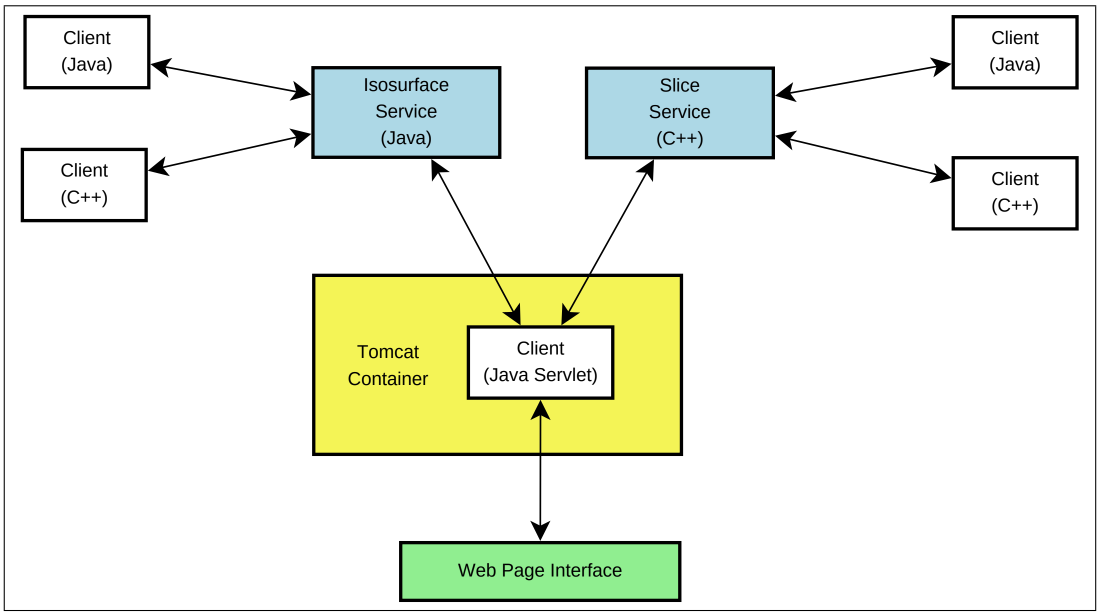


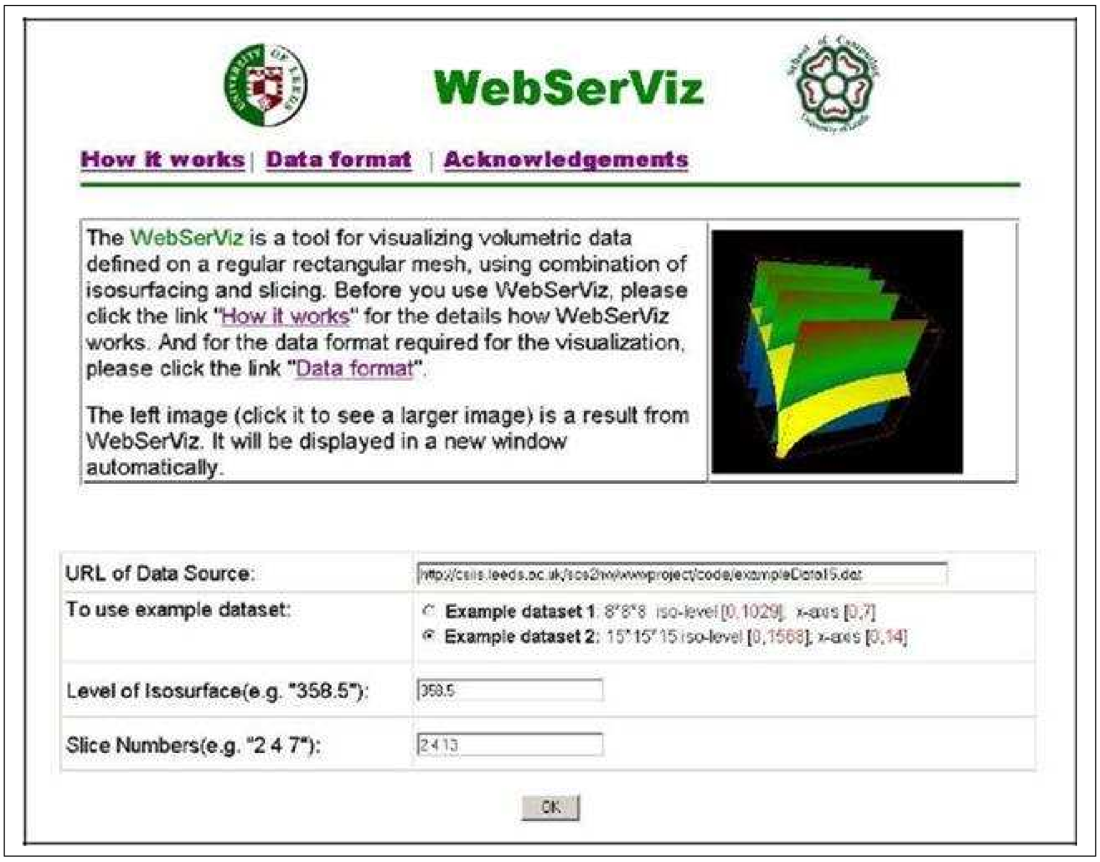
Figure 4.1: The design of visualization Web services experiment.

Figure 4.1 shows the design of this experiment. The isosurface service is written in Java, while the slice service is built in C++. Both Java and C++ versions of clients are implemented to invoke the service. A Java Servlet Client is deployed on an Apache Tomcat container, which provides Web-based access to both services. This experiment, named as WebSerViz, was demonstrated at Supercomputing 2003, Arizona, US, by running visualization services in UK and accessing them through the portal running in the exhibition hall in the US. Figure 4.2 and figure 4.3 are screenshots of the WebSerViz portal and show the result of integrating both slices and isosurface generated from the two services.

4.1.3 Assessment and Experience

As the experiment was built with Web service technologies, both advantages and disadvantages from Web services are inherited. The advantages of delivering visualization through Web services can be summarized as:

- Transparency: End-users can view visualization services as online APIs, which can be accessed from the end-users' code, as long as the WSDL descriptions are known. The details about how these services are implemented and on what environments they are hosted do not need concern end-users.



The WebSerViz is a tool for visualizing volumetric data defined on a regular rectangular mesh, using combination of isosurfacing and slicing. Before you use WebSerViz, please click the link "How it works" for the details how WebSerViz works. And for the data format required for the visualization, please click the link "Data format".

The left image (click it to see a larger image) is a result from WebSerViz. It will be displayed in a new window automatically.

URL of Data Source:	http://csiis.leeds.ac.uk/scs2w/wwwproject/code/exampleData15.dat
To use example dataset:	Example dataset 1: 8*8*8 iso-level [0,1029], x-axis [0,7] [✓ Example dataset 2: 15*15*15 iso-level [0,1568], x-axis [0,14]]
Level of Isosurface(e.g. "358.5"):	358.5
Slice Numbers(e.g. "2 4 7"):	2 4 13

Figure 4.2: Screenshot of WebSerViz Portal.

- Independence: Developers can build visualization services using their familiar programming languages and in their familiar environments, as these services are independent of their platforms and implementations.
- Interoperability: The communication between services and clients are standard XML messages, which offers the flexibility for end-users to build clients in different ways.
- Accessibility: These visualization Web services can be accessed worldwide, either through customized clients or through Web portals.

The disadvantages uncovered through this visualization Web service experiment include:

1. The first disadvantage is mainly caused by the use of XML messages in communication, as the transmission and processing of these messages influence the performance of the system. The XML messages are always bigger in size than the real information stored inside, so it costs more time to transmit these messages. The

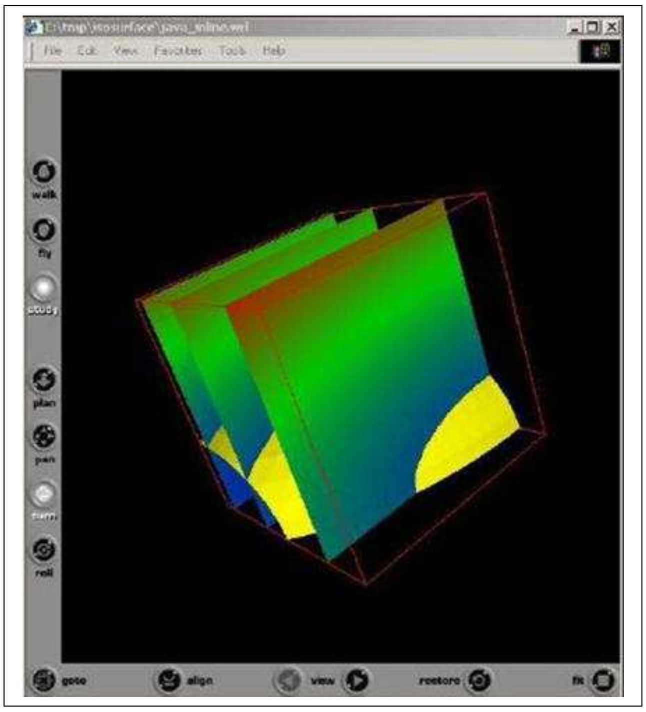


Figure 4.3: Screenshot of visualization result consisting of slices and an isosurface.

process of handling XML messages, which includes generating and parsing these messages on both service and client sides, takes extra time. However, as the extra transmission and processing time is almost fixed, when the visualization process itself becomes more complex and the XML messages involved are relatively small, those extra costs can be ignored by end-users.

2. Another disadvantage, also inherited from Web service characteristics, is the lack of the capability to re-use previous visualization results created by these services. If the result can be stored on the service, it can avoid repeating time-consuming computation when clients invoke the service with the same parameters. As the Web service itself is stateless, it is not responsible for keeping results within the service. Although external storing mechanisms can be exploited to address this issue, such as using an external database, it involves extra efforts or learning curve for developers.

The first-hand experience gained from this experiment includes how to implement and deploy a visualization process as a Web service; how to build clients to consume visualization services; and how to build a Web-based portal to provide easy access to visualization services.

4.2 To Step into the Grid: the gViz Portal

Following our first step of building visualization Web services, we step further to investigate the utilization of the Grid resources for visualization after GT3 was officially released in 2003. This gViz portal experiment provides an easy access interface for end-users to simulations on the Grid and a couple of Grid services which visualize the data produced by simulations.

4.2.1 Background

Since the emergence of the Grid and with the releases of Grid middleware tools GT2 and GT3, many efforts have been put into moving visualization into the Grid environment. This previous work has been mentioned in Chapter 2. Generally, there exist two ways to carry out visualization on the Grid: using Grid-enabled visualization systems (such as the extended IRIS Explorer described in Section 2.5.1), and developing special purpose distributed applications that communicate with the visualization on the Grid by SOAP messages. The former method requires visualization scientists to install these suitable visualization systems on their local machines, whereas the latter requires visualization scientists to be competent in computer programming within the Grid environment.

The WebSerViz portal discussed in section 4.1 inspired the idea of developing a similar Web-based portal to access a visualization system on the Grid. Different from the above two methods, the access through a portal requires the minimum computer skills for end-users; the users are only expected to be able to use a web browser. Furthermore, the portal access minimizes requirements of software resources for end-users, as only a standard web browser, rather than special visualization systems, needs to be installed. The 'Web-based' feature also offers the advantage of global access for end-users.

The differences between a portal to visualization Web services and a portal to the visualization Grid service include two aspects: firstly, instead of connecting to Web services, the Grid portal connects to Grid services; secondly, it becomes possible to build security access control on the portal by utilizing the Grid security infrastructure provided by Grid middleware tools.

The gViz portal is developed as a part of the UK gViz e-Science project [40]. One strand of the project is Grid-enabled computational steering, and a gViz computational steering library was developed. By using the gViz library, users can build applications as front-end interfaces to steer the simulations run on the Grid. The project implemented front-end interfaces based on IRIS Explorer [45], MATLAB [54], VTK [77], etc.. Users can control the simulations on the Grid through these visualization systems, and then visualize the data produced by the simulations with their visualization expertise on these systems.

The aim of building the gViz portal was to provide another front-end interface for end-users. The portal delivers visualization results to end-users based on the data generated by simulations on the Grid, so that no visualization expertise or knowledge of Grid computing are required for users. For the protection of the data, the portal only allows authorized users to access it.

4.2.2 Security in the Grid

Defined by Grid Security Infrastructure (GSI) [15], the security in the Grid mainly involves authentication, which concerns the identity of the entities in the Grid (hosts, services and users), and authorization, which concerns the permissions of who are allowed to perform what actions. Other security issues in the Grid include encrypting messages, managing user's credentials and maintaining group membership information. We are concerned more about the authentication and authorization for our applications, because based on authentication and authorization, access control can be performed on our Grid-based applications.

The authentication and authorization in the Grid are based on the concept of the certification. A certificate consists of a distinguished name of the entity, a public key belonging to the entity, the identity of a Certification Authority (CA) that has signed this certificate and the digital signature of this CA.

In order to set up mutual authentication between two users, they must first trust the CA that signed each other's certificate. The steps of a mutual authentication are shown in Figure 4.4. Initially, the first user (A) sends his certificate to the second user (B) to claim who he is. B can check whether the certificate is valid by checking the CA's signature included in the certificate. In order to verify A is who he claims to be, B sends a random message to A and asks A to encrypt it with his private key and send the encrypted message back to B. The encrypted message can be decrypted by B using A's public key which is included in A's certificate. If the decrypted message is the same as the original random message, A proves his identity to B. The same operation happens in reverse, in order to make A trust B's identity.

In the mutual authentication, the certificate is regarded as public, whereas the private key needs to be stored securely. Globus Toolkits store the user's private key as a file in user's local file system. The private key file is encrypted via a pass phrase and requires the pass phrase to decrypt it when user wants to use it. To avoid the frequent typing of

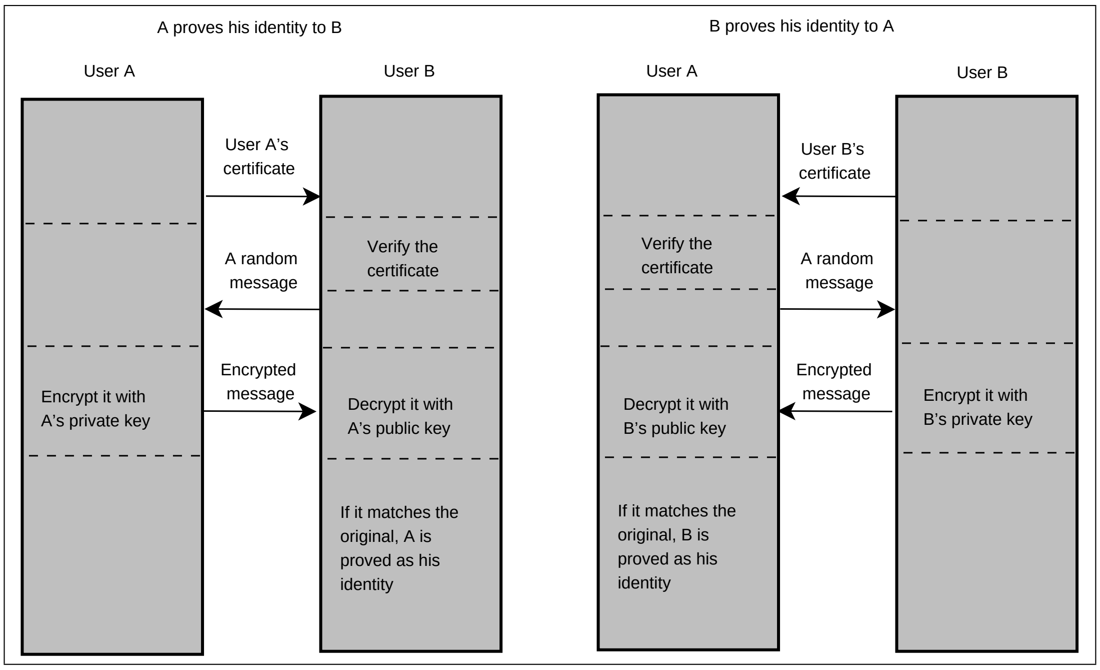

A proves his identity to B
User A
User B
User A's certificate
Verify the certificate
A random message
Encrypted message
Encrypt it with A's private key
Decrypt it with A's public key
If it matches the original, A is proved as his identity
B proves his identity to A
User A
User B
User B's certificate
Verify the certificate
A random message
Encrypted message
Decrypt it with B's public key
Encrypt it with B's private key
If it matches the original, B is proved as his identity

Figure 4.4: A mutual authentication between user A and user B.

the pass phrase, proxies can be created to act on the behalf of users. A proxy consists of a new certificate (with a new public key and the owner's identity) signed by the owner and a new private key. A proxy has limited lifetime, during which the user can use it without re-entering the pass phrase.

Both the user's private key and the proxy's private key are stored locally. If a user wants to be authenticated on different computers, he has to move his private key file onto these file systems and re-create proxies. Thus, it causes inconvenience for users to copy their private keys around, and increases the risk of being counterfeited.

MyProxy [58] offers the possibility of overcoming the inconvenience and reducing the risk. MyProxy is an online repository to store security credentials that include certificates and private keys. The usage of MyProxy is shown in Figure 4.5. Once users have stored their security credentials on a MyProxy server, they can retrieve proxy credentials from any other computers, or delegate these credentials to services acting on their behalf, as long as they are validated via MyProxy pass phrases. The credentials stored on a MyProxy server also have limited lifetimes; however, a MyProxy server is able to renew these credentials, so that long-running jobs do not fail because of expired credentials.

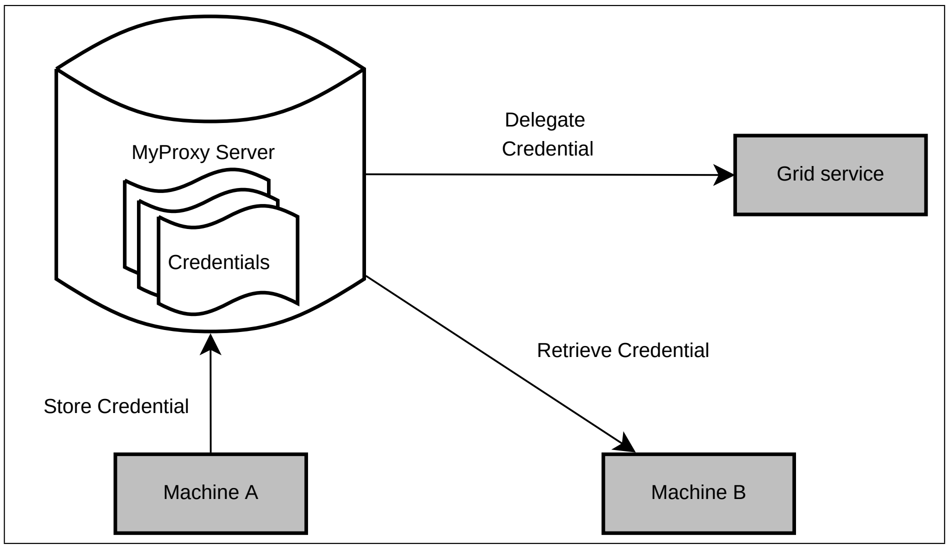


Figure 4.5: Use a MyProxy server to store security credentials.

4.2.3 Design of gViz Portal

The gViz portal system includes three layers: the interface layer, the portal layer and the Grid layer, as shown in Figure 4.6. The interface layer is the Web page interface for end-users. The portal layer sets up a connection with the Grid services on the users' behalf. It forwards the information submitted from the interface to the services. The Grid layer consists of Grid resources such as computational resources, data resources, and services. The portal layer lies between the Grid and interface layers, so it makes the complexity of the Grid transparent to end-users.

Within the Grid layer, the data to be visualized is generated from a set of simulations run on the Grid. Visualization is processed as Grid services, which obtain raw data from simulations and convert data into graphics. The simulations in the gViz project run as jobs submitted onto the Grid in the pattern of GT2's batch processing, while the visualization Grid services are deployed on GT3 containers which are based on a Service-oriented architecture and are significantly different to GT2. Fortunately, the gViz communication library is provided by the project to enable the communication between simulations and other applications. Figure 4.7 shows how a Grid service can obtain data from a gViz simulation. After the simulation has started and registered itself to a directory service called gVizDS, the service can contact gVizDS to find out the location of the simulation, and then try to connect directly with the simulation by using the gViz library. In the case that there is a firewall which blocks the communication between the service and the simulation, the communication can be set up via a proxy called gVizProxy, as long as both the simulation and the service can connect to the machine where gVizProxy is run.

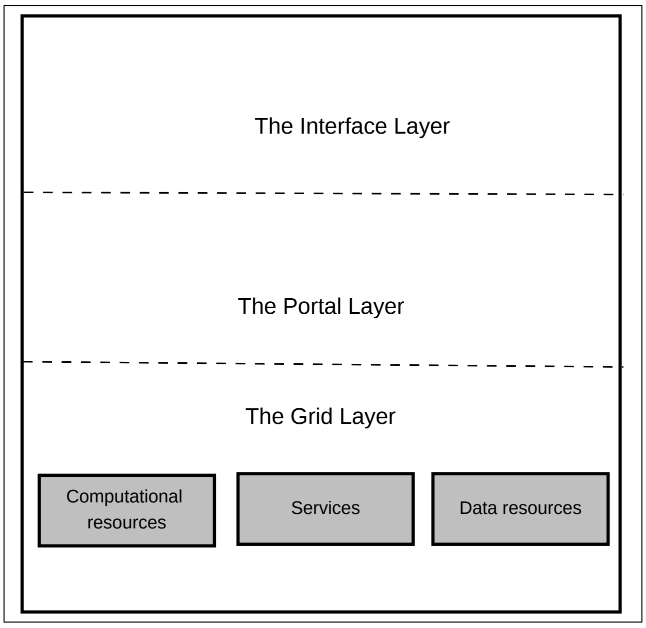


Figure 4.6: The three layers diagram of gViz portal system.

In order to build visualization Grid services as secure services, the management of certificates and private keys has to be addressed. Users are required to provide their certificates and carefully keep their private keys for a secure authentication. This is contrary to the purpose of using a portal, as the portal aims to enable users to access visualization Grid services easily from anywhere and without the overhead of storing secure credentials.

MyProxy, introduced in section 4.2.2, is a solution that can save us from this dilemma. By storing secure credentials temporally on a trusted online MyProxy server, both easy access and secure access can be achieved. Figure 4.8 demonstrates how a MyProxy server can be used to enable a Grid portal to access a secure Grid service. To use MyProxy with a Grid portal, firstly, users' Grid credentials should be stored on a MyProxy server that the portal has permission to use. After installing the MyProxy client, by running the command "*myproxy-init*", users can upload their credentials and set pass phrases to secure the credentials on the MyProxy server (step 1 in Figure 4.8). To enable the portal to retrieve a user's credential on MyProxy server, the user ID and pass phrase must be submitted to the portal (step 2 in Figure 4.8). The portal also has to obtain a portal certificate from the MyProxy server which can guarantee that only authorized portals can retrieve credentials from that MyProxy server. By providing the portal certificate, the user's ID and phrase (step 3 in Figure 4.8), the portal can get the user's credential back from the MyProxy server (step 4 in Figure 4.8) and connect to a secure Grid service using delegated credential on that user's behalf (step 5 in Figure 4.8).

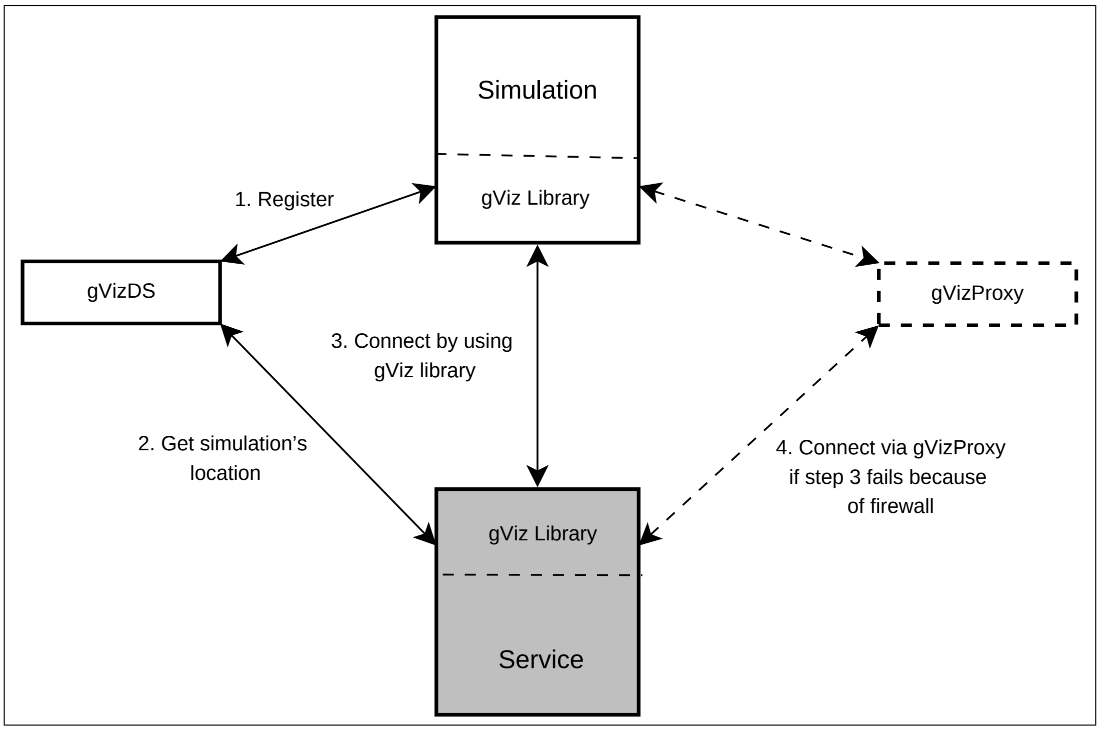

Simulation
gViz Library
1. Register
gVizDS
gVizProxy
3. Connect by using gViz library
2. Get simulation's location
4. Connect via gVizProxy if step 3 fails because of firewall
gViz Library
Service

Figure 4.7: The communication between a service and a simulation.

Thus, the three-layer diagram shown in Figure 4.6 can be refined as in Figure 4.9 with a MyProxy server for secure access and gVizDS and gViz Proxy for the communications between visualization Grid services and simulations.

4.2.4 The Heart Modelling Demonstration

The gViz project developed a heart modelling simulation to help scientists to investigate re-entrant arrhythmia and irregular heartbeat by simulating the electric behaviour of cardiac virtual tissue. The simulation runs on the Grid, thus it requires scientists to have Grid computing knowledge, so they can retrieve the data produced by the simulation. To visualize the retrieved raw data, scientists also need to be capable of creating visualization. The gViz portal aims to reduce the technical requirements in both Grid computing and visualization, by providing easy access and delivering graphical visualization result to users.

As the heart modelling data can be regarded as confidential, the gViz portal also aims to put secure access control on the confidential data, only allowing authorized users to use this portal.

Based on the design discussed in section 4.2.3, the heart modelling demonstration has three layers: the interface layer, the portal layer and the Grid layer.

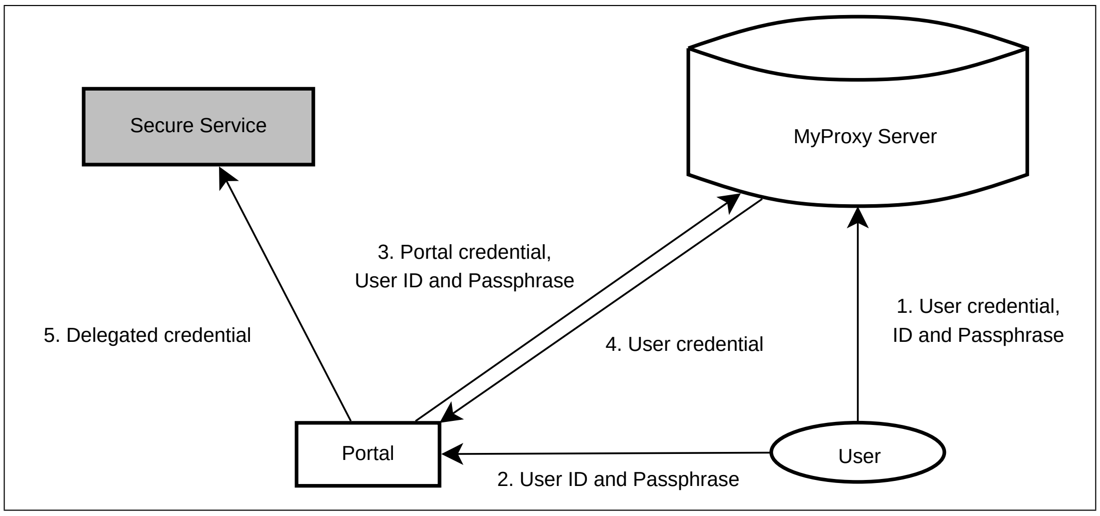

Secure Service
MyProxy Server
3. Portal credential, User ID and Passphrase
1. User credential, ID and Passphrase
5. Delegated credential
4. User credential
Portal
User
2. User ID and Passphrase

Figure 4.8: Use a MyProxy server to access a secure Grid service from a Grid portal.

The interface layer: The web page interface (the screenshot is shown in Figure 4.10) to the gViz portal has 4 parameters for the user to fill in. The first two parameters specify the location (machine name and port number) where the gViz Directory Service (gVizDS) is run. gVizDS holds the detailed information of gViz heart modelling simulations registered, such as how these simulations can be accessed or how their proxies can be accessed. The user ID and pass phrase fields are required to be submitted to the portal in order to retrieve the user's credentials stored on MyProxy Server.

The portal layer: As shown in figure 4.11, the portal consisting of JSP files, JavaBeans and JavaServlets, acts as a Grid Service client. The JSP files capture the information submitted through the Web page interface and enable the user to trigger the JavaServlets from a standard web browser. The JavaBeans are used for holding parameters and transferring these parameters to the JavaServlets. JavaServlets do not only retrieve the user's credentials from MyProxy, but also use the credentials to connect to the secure Grid service on the user's behalf. Some of the JavaServlets also take charge of the control of simulations, such as pausing, killing or re-starting simulations.

The JavaServlets and JavaBeans are hosted on an Apache Tomcat Server container which enables HTTP or HTTPS connection from a web browser.

The Grid layer: The Grid layer includes gViz heart modelling simulations, and two visualization Grid services: ImageMaker service and HTMLmaker service. The implementation of the Grid layer is shown in Figure 4.12.

In this demonstration, **gViz simulations** are run on different sub-nodes of the White Rose Grid [85] with a firewall preventing direct connection from outside. The gVizDS and gVizProxy are located on a head node of the White Rose Grid which is accessible

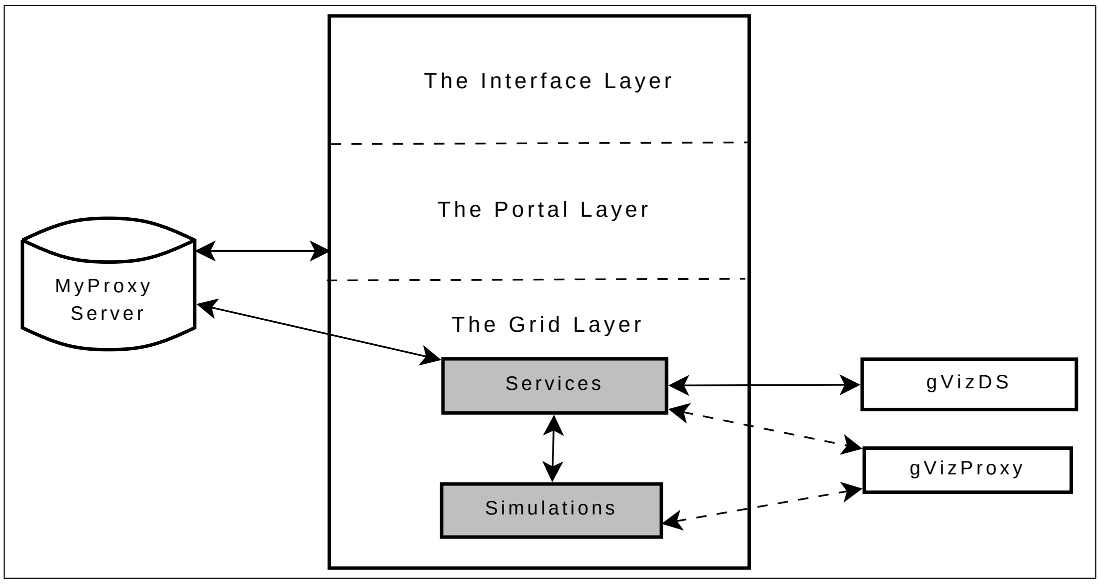


Figure 4.9: The refined three layer diagram of the gViz portal system.

from outside.

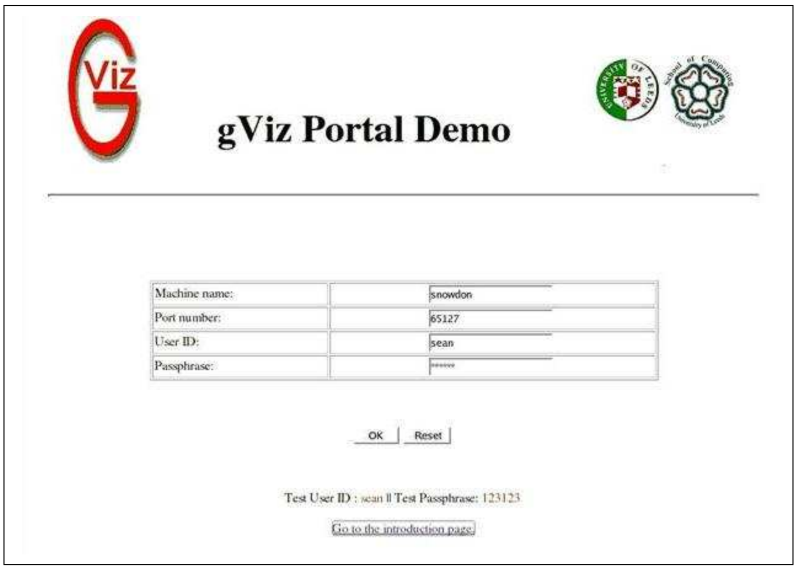
The **ImageMaker** aims to transform binary datasets into JPEG format images. After the binary data has been read as float values, each float value will be mapped to a colour using a colour map. By using Sun's Java JIMI library [47], a Java program can generate JPEG images. The generated images are saved under a Tomcat Server container directory, and the URLs of the images will be sent back to the service requester.

The **HTMLmaker** is a "grid-map" secure Grid service, which means not only the connection should be made with an appropriate Grid credential (authentication) but also the user ID has to be listed in a file called "grid-map" on the service side (authorization). By adding IDs to or removing IDs from the "grid-map" file, the service can easily control users' access permits.

HTMLmaker is a Grid service, but meanwhile it can also be regarded as a client of both gVizDS and the ImageMaker service. With the information about the location of gVizDS submitted from the portal, HTMLmaker can obtain the details about each simulation registered on the gVizDS. According to the simulation details, HTMLmaker always tries to talk to the simulations directly. As in our scenario, HTMLmaker and ImageMaker are both deployed on test Grid network which is outside the White Rose Grid, HTML-maker will use the gVizProxy on the head node to retrieve the binary simulation datasets.

When getting the URLs of the images of all the latest simulation results, HTMLmaker creates a web page with all the images and sends the URL of that web page back to the requester.

Having described the layers, we now explain how the gViz portal system works. Be-



Machine name:	snowdon
Port number:	65127
User ID:	sean
Passphrase:	******

Test User ID : sean || Test Passphrase: 123123

Figure 4.10: The screenshot of gViz portal interface.

fore anything happens, the users must store their Grid credentials on MyProxy server in advance (task 'a' in Figure 4.13). After getting the parameters from the web page interface (task 1), the portal tries to retrieve the user's credential on the MyProxy server (task 2) and then using the credential, the portal acts on the user's behalf to connect with the HTMLmaker Grid Service (task 3). When the connection has been authorised, the HTMLmaker talks to gVizDS in order to get the details (location) of the gViz simulations (task 4). The binary datasets of simulations are transferred directly or via gVizProxy(task 5). In the demonstration, as simulations are behind a firewall, HTMLmaker retrieves simulation datasets via gVizProxy. The ImageMaker Grid Service does the job of transforming binary datasets into JPEG images and returns the URL of the images back to the HTMLmaker (task 6). After all the images of simulation results have been generated, the HTMLmaker creates a web page with all the images displayed and passes the URL of the web page back to the gViz portal (task 7). The end-user can view the images from the latest results of simulations according to the URL returned from portal (task 8).

The images visualized from the heart modelling simulations are returned to end-users as a Web page with images embedded, shown in Figure 4.14. Interactive functions are provided through the result Web page for end users to pause, restart and kill remote simulations, or view simulation parameters.

4.2.5 Conclusions

The following aspects have been achieved in the gViz portal system:

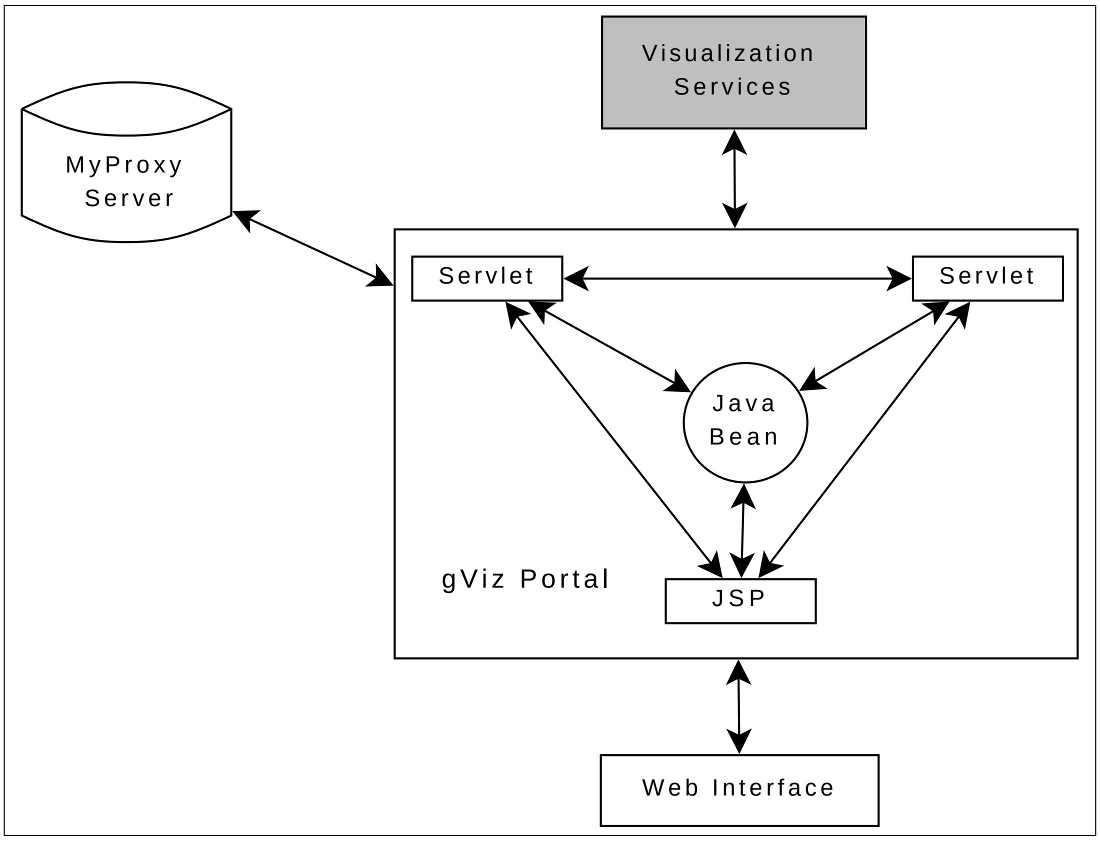


Figure 4.11: The components in the gViz portal.

- Visualization is processed on the Grid. The raw datasets are generated by gViz simulations on the Grid, and the data sets are visualized by two visualization Grid services which are also deployed on the Grid.

- Access control is provided on the gViz portal, for security reasons and for data protection. The visualization Grid services authenticate users based on their Grid secure credentials. Only authorized users, listed in the 'grid-map' file, can use the portal to visualize simulation datasets.

- The requirements for end-users are minimized. End-users do not need to have any Grid computing or visualization skills. Although users are asked to provide Grid credentials to access the visualization services, the management of credentials is transparent for end-users. Users only need to submit their IDs and pass phrases through the portal, and the portal will obtain their credentials from the MyProxy server.

- As now the visualization Grid services are 'stateful', it offers the possibility of storing previous visualization results or client information (e.g. parameter settings)

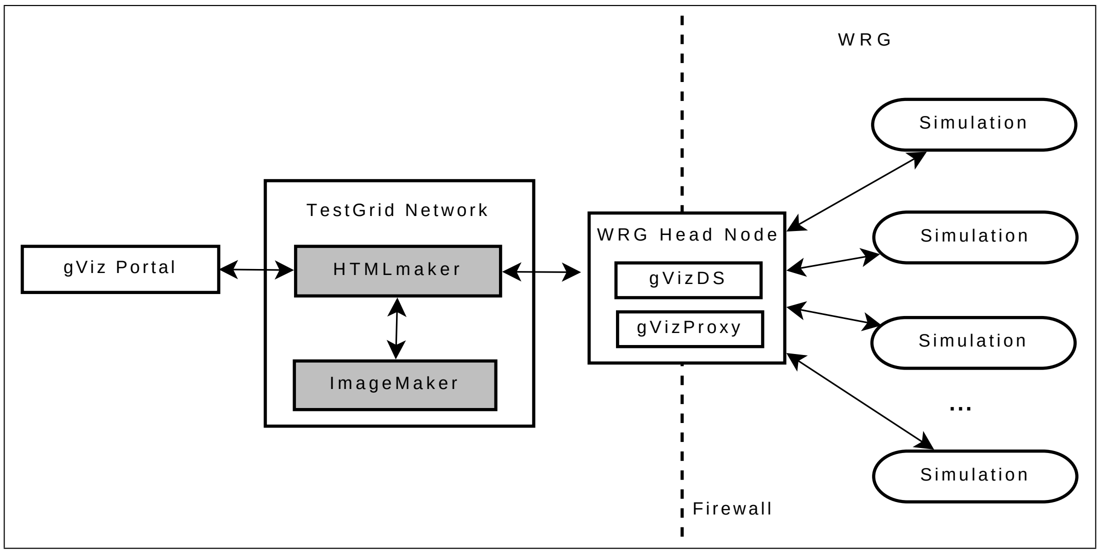


Figure 4.12: The implementation of the Grid layer.

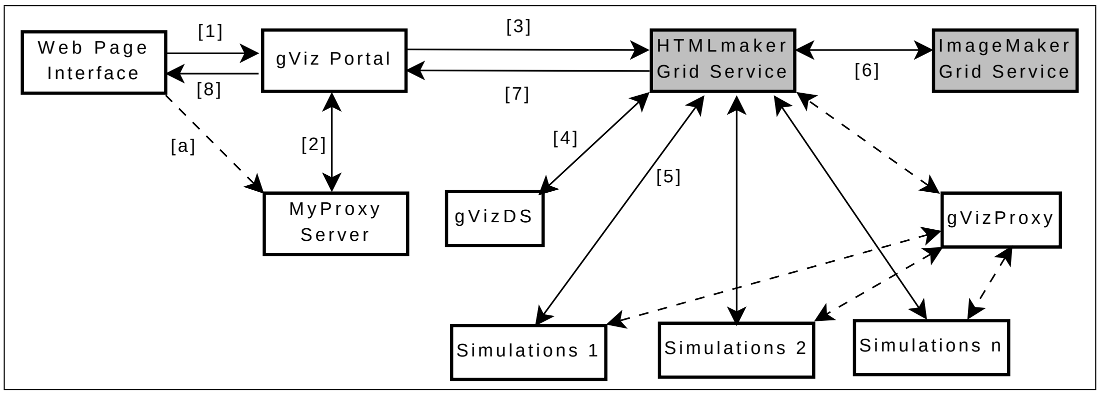


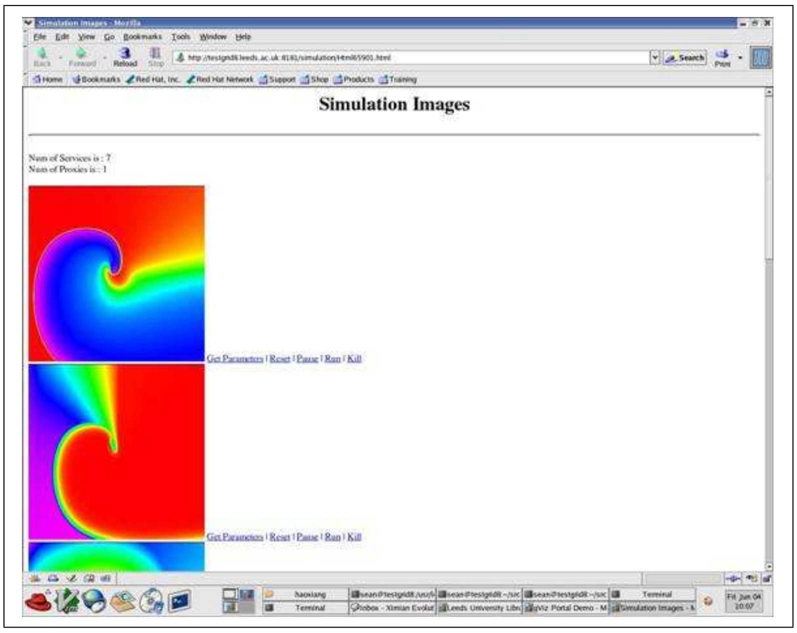
Figure 4.13: How the gViz portal works.

on the service. This feature can also be utilized to support asynchronous work or visualization provenance.

Although the gViz portal offers the above advantages for end-users, there also exist criticisms:

- In this experiment, visualization functions are delivered as Grid services, however, there is no easy approach provided for end-users to build up customized visualization pipelines by linking these services together.

- The interactions between users and visualization on the Grid are limited. In the gViz portal, only simple interactions with simulations are enabled, whereas the interactions between end-users and visualization services (i.e. change the colour map used to generate images) have not been implemented yet. The Web page interface


Simulation Images - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training
Search Print
Get Parameters | Reset | Pause | Run | Kill
Get Parameters | Reset | Pause | Run | Kill

Simulation Images

Num of Services is : 7
Num of Proxies is : 1

Figure 4.14: The visualization result returned from the gViz portal.

also makes it difficult to build a complicated interactive interface. Interactive widgets, such as slider, dial, colour map creator, need extra effort to be added onto a Web page.

- End-users have to 'pull' the visualization result out from the system by manually submitting a request through the portal interface. Simulations keep generating datasets at every time step, and so users may miss some important time steps, as they are not automatically updated with the latest visualization result.

The gViz portal was demonstrated as a part of the gViz demonstration at the UK e-Science All Hands Meeting in September 2004. The experience learned from this experiment includes implementing a visualization pipeline as a set of linked Grid services and developing an easily accessible portal to visualization Grid services.

4.3 To Cross the Continent: the CROWN gViz Portal

4.3.1 Background and Aims

CROWN, standing for **C**hina **R**esearch and Development Environment **O**ver **W**ide-area **N**etwork, is a Grid testbed in China. It consists of distributed computational resources, the CROWN Grid middleware and a number of typical applications deployed on CROWN to support e-Science research in China.

CoLaB, short for the Collaboration of Leeds and Beihang, is a joint laboratory involving research teams from University of Leeds, UK and Beihang University, China. One of the CoLaB research themes is concerned with building visualization applications across the White Rose Grid in UK and the CROWN Grid in China. The CROWN gViz portal was developed as the first CoLaB visualization experiment.

The issues aimed to be explored by this CROWN gViz portal include:

- The utilization of different Grid networks. We aim to distribute the visualization pipeline across the White Rose Grid and the CROWN Grid.
- The move from OGSA based Grid services (gViz portal described in Section 4.2) to WSRF based Web services.
- The interoperability between two Grid middleware tools. Visualization services will be deployed on two different Grid middleware tools: GT4 and the CROWN middleware. These services need to be connected together to produce a visualization result.
- The global visualization. Visualization can be widely distributed in this experiment, so that we can test the feasibility of exploiting visualization on a global-scale.

4.3.2 The CROWN gViz Portal

The scenario of the experiment is as follows: a group of doctors are at different locations around the world. They are not skilled in visualization or Grid computing, but want to collaboratively investigate some heart illness cases. Different doctors own different simulations, but want to collaborate by sharing the visualization results, and moreover by jointly controlling the parameters of the simulations. During the collaborative session, all the participating doctors use their Web browsers to access the Web portal of a trusted third party visualization system which will retrieve raw data from simulations and generate a graphic representation for each set of simulation data. Through the Web portal, the doctors can investigate the images of simulations and control the running of simulations, without being concerned about the details of generating the visualization.

The CROWN gViz portal is developed based on the previous gViz portal. As we now put emphasis on testing visualization applications across Grid networks at a global scale, the criticisms listed in section 4.2.5 have not been addressed in this experiment. However, the significant differences between these two experiments include: visualization services now are implemented as WSRF defined Web services rather than OGSA defined Grid services; and visualizations are distributed more widely across the world.

```
<gwsdl:portType name="ImageServicePortType" extends="ogsi:GridService">
        <operation name="makeImage">
                <input message="tns:MakeImageInputMessage"/>
                <output message="tns:MakeImageOutputMessage"/>
                <fault name="Fault" message="ogsi:FaultMessage"/>
        </operation>
</gwsdl:portType>
```

Table 4.1: An example of GWSDL snippet.

```
<portType name="ImageServicePortType"
        wsdlpp:extends="wsrpw:GetResourceProperty"
        wsrp:ResourceProperties="tns:MakeImageResourceProperties">
        <operation name="MakeImage">
                <input message="tns: MakeImageInputMessage"/>
                <output message="tns: MakeImageOutputMessage"/>
                <fault message="tns: MakeImage_ERROR"/>
        </operation>
</portType>
```

Table 4.2: An example of WSDL snippet.

Migration onto GT4. The migration from Grid services to GT4 Web services involves a number of aspects. A WSDL file replaces a GWSDL file to describe the service. A GT4 Web service follows WSRF specifications which use pure WSDL 1.1 to describe a service, while a GT3 Grid service is described in GWSDL which has some features not supported by WSDL 1.1. Table 4.1 shows a GWSDL snippet describing a PortType for an Image service, which has an operation called "makeImage". The presentation of the same PortType in WSDL is listed in Table 4.2.

Another major difference between GT3 Grid services and GT4 Web services concerns the deployment of services. A WSDD (Web Service Deployment Descriptor) file is required for both services to be deployed. However, a GT4 Web service requires an extra JNDI (Java Naming and Directory Interface) configuration, as GT4 manages service resources by a Resource Home, whereas a GT3 Grid service attaches service data directly to the service. The JNDI configuration tells GT4 what kinds of service resource properties need to be generated. Table 4.3 illustrates a JNDI snippet which sets up a resource home for the Image service. This resource home can generate a customized Image service resource identified by a unique key.

Migration onto the CROWN middleware. The CROWN middleware also supports WSRF specifications, which means standard Web services can be deployed on the CROWN middleware. Hence, the migration of the visualization Web services from GT4

```
<service name="ImageService">
    <resource name="home"
        type="viz.services.factory.impl.ImageServiceResourceHome">
    <resourceParams>
    <parameter>
        <name>resourceClass</name>
        <value>viz.services.factory.impl.ImageServiceResource</value>
    </parameter>
    <parameter>
        <name>resourceKeyType</name>
        <value>java.lang.Integer</value>
    </parameter>
    <parameter>
        <name>resourceKeyName</name>
        <value>{http://www.vizservice.org/namespaces/
            viz/services/ImageService_instance ImageServiceResourceKey }
            </value>
    </parameter>
    <parameter>
        <name>factory</name>
        <value>org.globus.wsrf.jndi.BeanFactory</value>
    </parameter>
    </resourceParams>
    </resource>
</service>
```

Table 4.3: A JNDI configuration of a GT4 service resource home.

to the CROWN middleware is straightforward.

Distribution of the system. In order to reflect the aims of the experiment, namely to build a widely distributed visualization application, the CROWN gViz portal system is distributed on the White Rose Grid and the CROWN Grid, as shown in Figure 4.15. Both Grid networks host a number of simulations, which register to the same directory service. The HTMLmaker service and ImageMaker service are deployed respectively on these two Grid networks with GT4 and the CROWN middleware. For the convenience of deployment and maintenance, the portal is deployed on a Leeds experimental Grid network, called TestGrid.

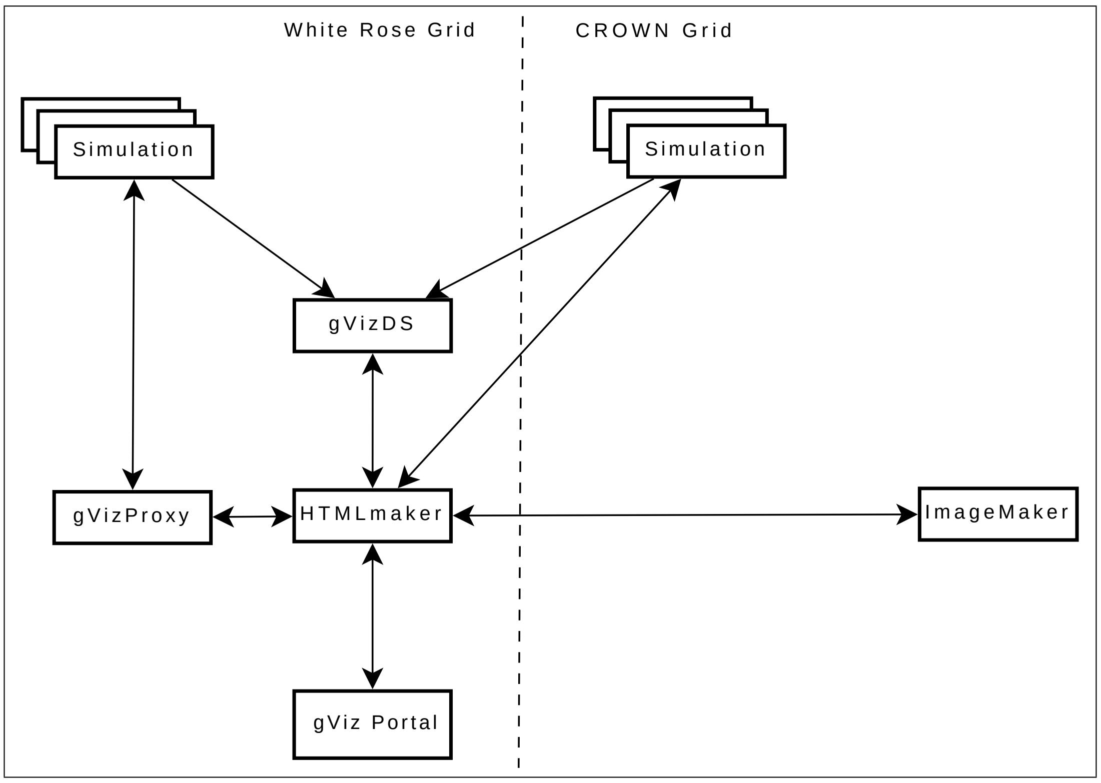


Figure 4.15: Distribution of the CROWN gViz portal.

This allocation involves relatively heavy data transfer across the two Grid networks. Firstly, the raw data generated from the simulations run on the CROWN Grid need to be passed to the service deployed on the White Rose Grid. Secondly, when the HTML-maker invokes the ImageMaker service to create images, data needs to be sent back to the CROWN Grid. The heavy data transfer causes the performance of the portal to become unacceptable. End-users have to wait for long time to retrieve the images generated from the portal.

To achieve better performance and reduce the data transmission across the Continents, the portal components can be re-allocated. Meanwhile we still want to retain the distributed feature of the system. The CROWN gViz portal was demonstrated in the UK e-Science All Hands Meeting 2005. In the demonstration, the HTMLservice was shifted from the White Rose Grid onto the CROWN Grid (shown as Figure 4.16), therefore, the major data transferred between the two Grid networks is only the data generated from the simulations run on the White Rose Grid. This change reduced the time cost by the portal system greatly, and it showed acceptable performance in the demonstration. This demonstration proofed the feasibility that, through the portal, the visualization processed on the Grid networks across the Continents can be accessed by end users worldwide.

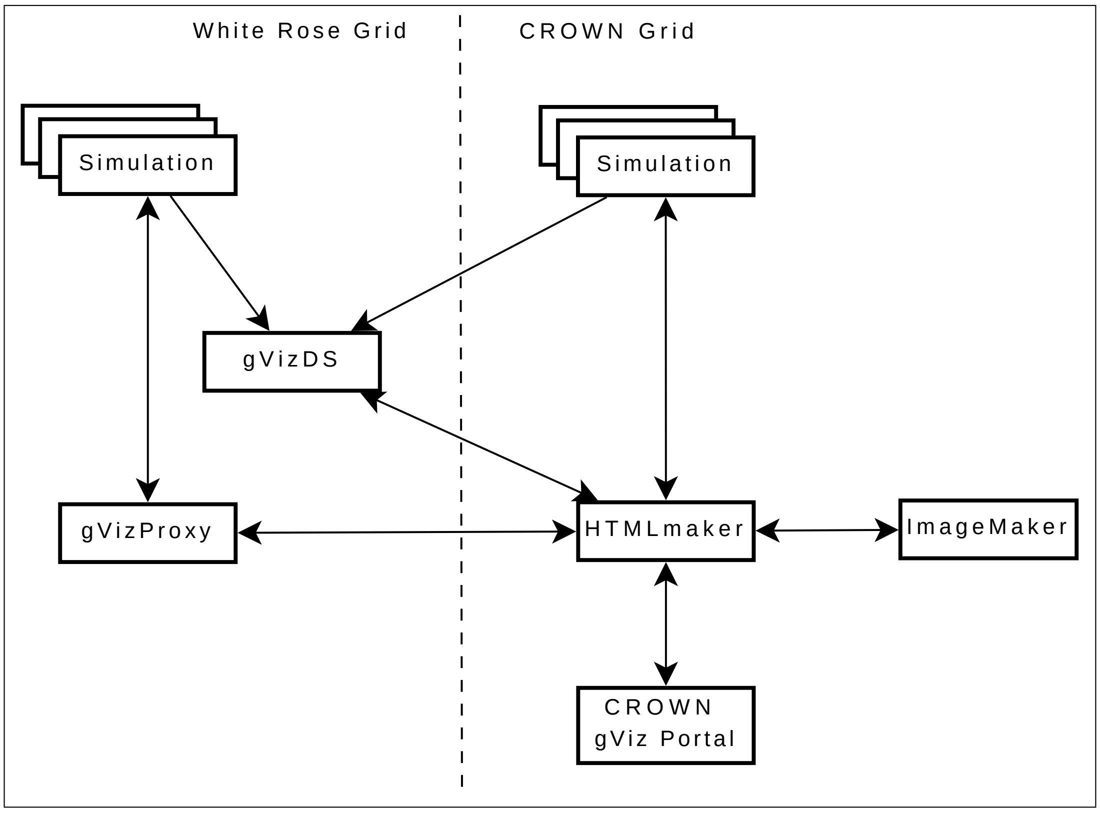


Figure 4.16: The re-located CROWN gViz portal.

4.3.3 Conclusions

The components of the CROWN gViz portal system are located across different Grid networks and deployed on different kinds of Grid middleware containers. It proves the possibility of building global-scale visualization applications utilizing heterogeneous Grid resources across the boundaries of organisations, institutes and countries.

In this experiment, we also tried the approach of delivering visualization as standard

Web services in the Grid environment. It offers the opportunity of migrating previous work on visualization Web services onto the Grid. As Web services follow standard WSRF specifications, visualization Web services from different developers can be easily integrated as a distributed system. It enables the re-use of previous work to reduce the workload of developing new visualization applications.

The portal is provided with the aim of easy and global access. The visualization running behind the portal is transparent for end-users, so that end-users can avoid the complexity of dealing the Grid-enabled visualization which is distributed across the Continents.

However, the wide distribution of the system also causes several issues which need to be addressed.

The first issue is the network condition. As visualization usually involves large-scale datasets, the transfer of large datasets across such a large distance can take more time than the visualization process itself. To enhance the performance of widely distributed visualization systems, the better network condition is desired, however it requires large a mount of investment on network facilities.

Secondly, it is notable that secure control is not implemented in the CROWN gViz portal. Security is one of the desired features for distributed systems. As the system is now across two Grid networks, a third part security server, trusted by both domains, needs to be introduced into the system and map identities and secure credentials from one domain to the other. The security between different domains is an active research area,but is out of the scope of the research discussed in this thesis.

4.4 Summary

In this chapter, three previous experiments are introduced. These experiments verify some aspects of building a collaborative visualization in the Grid environment. They include:

- Delivering visualization functions as standard Web services following WSRF specifications.
- Implementing a visualization pipeline as a set of linked visualization services.
- Allowing visualization services to be jointly provided by different organisations and run in different environments.
- Distributing visualization in a global scale, although the performance of such a widely distributed system is significantly affected by the network condition.

- Restricting the access to visualization systems by using Grid secure mechanisms, although it becomes more complicated when the systems involve multiple organisations or Grid network domains.
- Providing easy access and usage for end-users to the Grid-enabled visualization systems. As Grid computing requires computing skill and Grid knowledge which involves a steep learning curve for the scientists in other disciplines, the portal can bridge the gap between Grid computing and these scientists and enable them to access heterogeneous Grid resources and services easily.

There is also the first hand technical experience learned from these experiments. The experience includes: building visualization Grid/Web services and deploying them on different Grid middleware containers (GT3, GT4 and CROWN); linking these visualization services to generate visualization pipelines; and using a 'grid-map' file to control the access to the visualization running in the Grid environment.

However, the limitations reflected from these experiments also drive the design of the collaborative visualization system which is the main contribution of our research.

- The connection between services are relatively fixed, compared with the pipelines created in MVEs. Pipelines can be easily modified by users through visual editor windows in MVEs. In these experiments, the modification of pipelines requires Grid computing developers to modify the visualization services. It will be a useful feature if we can integrate a visual editor for users in our system for easy composition of visualization services.
- The interaction between end-users and visualization is limited. A real visualization application usually involves a set of parameters which can be set by users to steer the visualization. In MVEs, users can set parameters to each module in a pipeline. This feature can also be incorporated in our system to enable users to set parameters for each visualization service.

These three experiments reflect the technical evolution: from the non-Grid stateless Web service to the stateful Grid service, and then to the standard Web service with stateful feature on the Grid. These experiments also lead us step by step to the design and the implementation of our Grid-enabled collaborative visualization system which is introduced in detail in the following chapters.

Chapter 5

The Design of NoCoV Framework

Based on the motivations and requirements mentioned in Chapter 3 and the early experiments described in Chapter 4, in this chapter, we introduce our novel collaborative visualization system - NoCoV. The design of the NoCoV framework consists of three aspects, including: the creation of Service-oriented visualization pipelines; the management of Service-oriented visualization pipelines; and the interaction from clients. In this chapter, we start with an overview of the NoCoV framework, and then explain the above three aspects in detail.

5.1 Framework Overview

NoCoV stands for **No**tification-service-based **Co**llaborative **V**isualization. It provides a framework implemented using notification Web services for collaborative visualization. As is the case with most collaborative versions of MVEs, the NoCoV framework only expects a small number of collaborators in a collaborative session. (It is simply very difficult at the human level to sensibly involve more than 3 or 4 people in an active collaborative visualization session.) NoCoV is an evolution from Modular Visualization Environments (MVEs), moving to Service-oriented architecture. The visualization modules in MVEs are replaced with visualization services, and the links between modules in MVEs are implemented as subscriptions and notifications between these visualization services.

The NoCoV framework can be divided into three layers: service layer, management

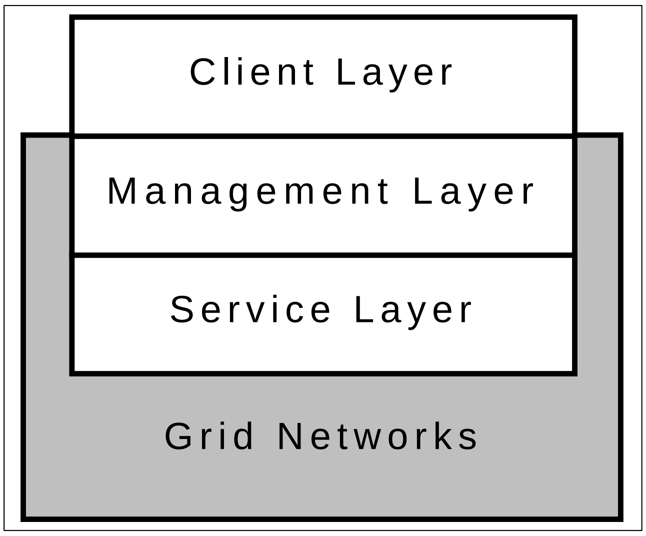


Figure 5.1: Three layers in the NoCoV framework.

layer, and client layer (shown as figure 5.1). The clients are geographically distributed, each client associated with a user in the collaboration. The management and service layers are dispersed across different Grid networks.

The Service Layer. The service layer consists of a set of notification Web services which can be provided by different institutes and deployed on different Grid networks. These services offer different visualization functionalities, similar to modules in MVEs. By composing these services, users can build customized visualization pipelines. The connections between visualization services are implemented as subscriptions and notifications.

The Management Layer. The management layer lies between services and clients. It does not only pass communication messages between visualization services and clients, but also manages visualization services and clients. The management of services involves:

- The registration and discovery of visualization services;
- The operations on services on a user's behalf, such as launching service instances from service factories, destroying service instances, connecting and disconnecting service instances, etc.;
- The storage and parsing of visualization information, including the description of pipelines and parameter settings;
- The access control to the visualization service instances.

The management of clients includes the following aspects:

- The management of collaborative sessions. There can be multiple collaborative sessions in parallel. The management layer controls which users have permission to join a particular collaborative session.
- The management of the access to the collaborative visualization system. Only authorized users are allowed to use the system.
- The support of synchronous collaborative work. The management layer keeps visualization information for each collaborative session, and sends this information to every participant in the session.
- The support of asynchronous work. By storing visualization information in the management layer, users can pick up previous work at a later time.

The Client Layer. The client layer accommodates two kinds of users: visualization experts who know how to create visualization pipelines by connecting suitable services, and application scientists who know how to set appropriate parameters to steer the visualization, but do not have visualization expertise to create their own visualization pipelines. Therefore, a visual pipeline editor GUI is required for visualization experts to create a visualization pipeline, and a steering GUI is needed for application scientists to set parameters for the visualization.

5.2 Service Layer of NoCoV Framework

In this subsection, we focus on the service layer of the NoCoV Framework. As notification Web services are the fundamental concept in the design of the framework, we start with an introduction to notification Web services and the advantages of exploiting them to build up visualization pipelines. It is also illustrated, in this subsection, how to create pipelines by subscribing to notification services.

5.2.1 Notification Web Services

WS-Notifications (WSN) [87] is a family of specifications that standardize the process of creating an event-based notification system in a Web services environment. They were developed in conjunction with the Web Services Resource Framework (WSRF) [88], which provides a way to use state in a stateless environment.

Notification is a pattern that allows clients to be notified when interesting events happen in a service or interesting information changes in a service. In the common service/client communication pattern, if the client wants to be updated with the latest events or changes which have happened in a service, the client has to use a polling approach. As shown in Figure 5.2, the client has to keep sending requests to the service asking whether the change has happened. In the case of the example in Figure 5.2, the interesting change in the service can be whether the resource is available.

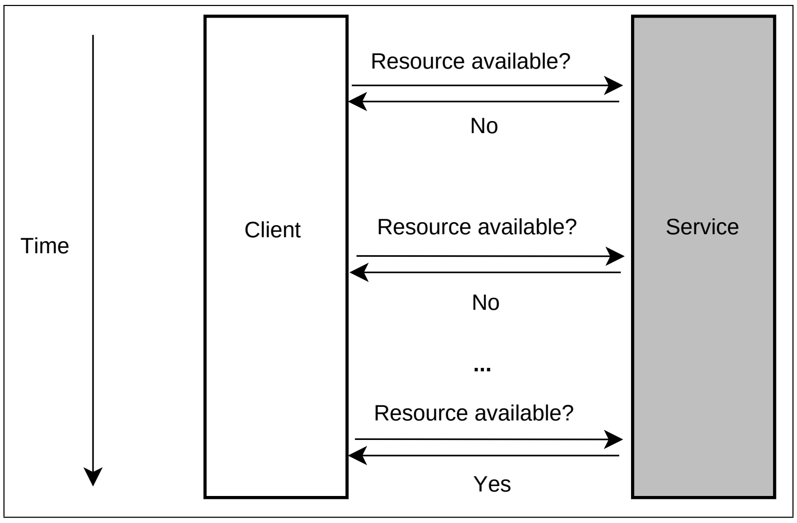


Figure 5.2: The polling approach to update changes happened in a service.

The polling approach is not efficient in at least three aspects:

- More workload for client and also more network traffic is caused, as the client keeps generating requests every time step, especially when the time between two calls is very small.
- The client can miss important information. As the client sends requests to the service every time step, even if the time between two requests is very small, there still exists the possibility of missing some information which lasts for a short life time.
- The server can be saturated, if there are many clients using the polling approach to invoke the same service at the same time.

Instead of periodically asking the service if there are any changes, the client just needs to make an initial call asking the service to notify the client whenever a certain event occurs. As shown in Figure 5.3, after the client sends a subscription message to the service, there will be no more communication between the service and client, until the resource becomes available.

In the contrast with the polling approach, the advantages of the notification approach include:

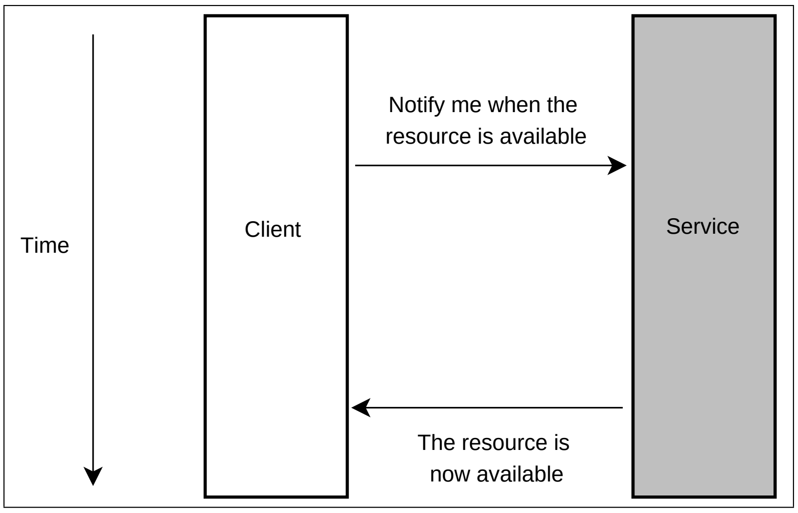

Time
Client
Notify me when the resource is available
Service
The resource is now available

Figure 5.3: The notification approach to update changes happened in a service.

- Less workload for the client, and less network traffic. Each client only invokes the service once with a subscription message, and then waits for the notification.
- No change will be missed, even if the change only exists for a very short period. As long as the change happens, the service will automatically notify the client with the change.
- It is easier for the service to deal with multiple users. As each client only invokes the service once, there is no burden for the service to repeat answering clients' calls periodically, even if nothing has happened in the service.

There are two main uses of notification service: a service whose outputs keep changing but need to be tracked during the whole run time; and a service with information which needs to be shared by multiple users.

5.2.2 Notification Service in GT4

The concept of notification service was introduced into the Globus Toolkit at the third version. However, the notification service in GT4 is significantly different to the notification service in GT3, as GT4 introduces the new concept of service resource.

Instead of using Service Data Element (SDE) to store stateful information in the GT3 service, a GT4 service itself is stateless. However, a Resource Home is used to generate and manage service resources for each service instance created. Figure 5.4 demonstrates how a GT4 service manages stateful information by using a Resource Home. After receiving a request from the client, the Resource Home creates a resource with identity Y.

Although different clients can connect to the same factory service and instance service, they have separate resources in which to store their stateful information. Different clients can also share the same resource, as long as they all know the resource identity, which is represented as an Endpoint Reference (EPR).

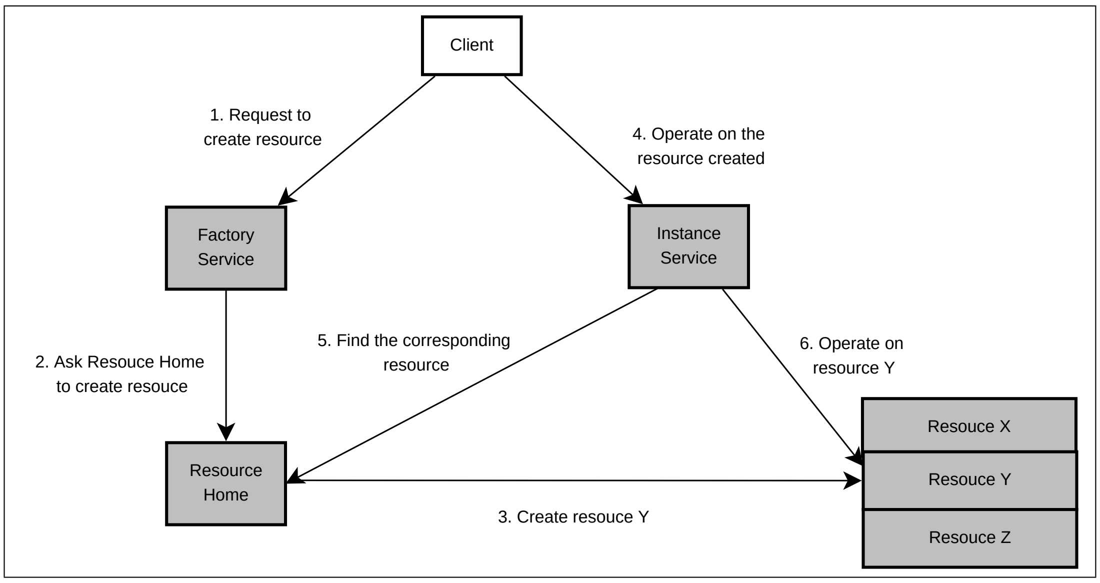

Client
1. Request to create resource
4. Operate on the resource created
Factory Service
Instance Service
2. Ask Resouce Home to create resouce
5. Find the corresponding resource
6. Operate on resource Y
Resouce X
Resource Home
Resouce Y
Resouce Z
3. Create resouce Y

Figure 5.4: How a Resource Home works in a GT4 service.

The notification in GT4 is also based on the concept of service resource. GT4 allows resources to be published as notification topics which can be subscribed to by users. Whenever any changes happen on the published resources, the subscribing users will be notified with the changes. The notification pattern in GT4 can be drawn as in figure 5.5. One client initiates a service resource which will be published as a notification topic. By sharing the EPR of the resource, different clients can subscribe to the same notification topic. This offers the possibilities of sharing data and collaborative working among these clients.

5.2.3 Reasons for Adopting Notification Services for Visualization

The characteristics of notification services mentioned in Section 5.2.1 are well suited to distributed visualization, especially distributed visualization supporting collaboration, where there is a need to share data between processes and between users.

- For each visualization process (e.g. modules in MVEs, and services in Service-oriented visualization systems), the outputs keep changing when the parameter settings are modified or different datasets are fed into the visualization process.

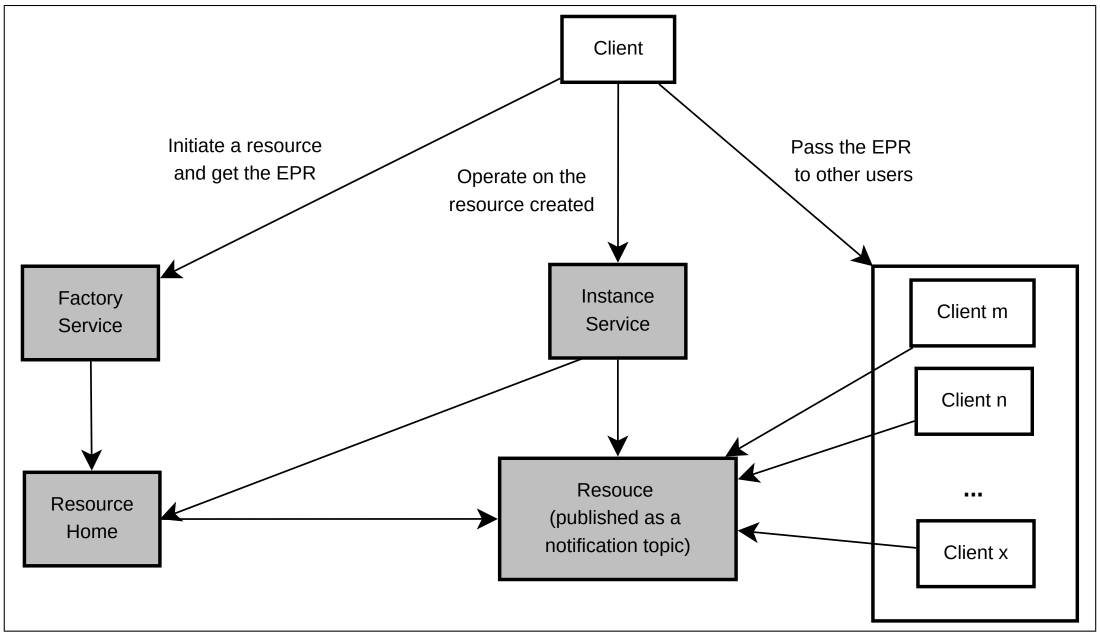

Client
Initiate a resource and get the EPR
Operate on the resource created
Pass the EPR to other users
Factory Service
Instance Service
Client m
Client n
...
Client x
Resource Home
Resouce (published as a notification topic)

Figure 5.5: The notification pattern in GT4.

- When a simulation is involved in a visualization process, the simulation typically generates data every time step, and needs to send the latest data to a visualization process.
- In collaborative visualization, multiple users usually want to share the same visualization result, visualization pipeline and visualization parameter setting.

The alternative to a notification approach is socket communication. In the pattern of socket communication, the service delivers information onto a certain socket port which the client keeps listening to. However, the implementation of notification service has two main advantages over the socket communication implementation:

- The communication details, such as port numbers and machine names, need to be specified in a socket communication. These details are covered and transparent for users by using notification services, as notification services are accessed by Uniform Resource Identifiers (URIs) and notification topics are identified by Endpoint References (EPRs).
- The passing of customized or complex data types can cause overhead in socket communication, as the data passed between service and client is converted into byte arrays in the service, and the client has to convert the byte data back to the original message according to different data types. In a notification service, the

data delivered to clients can be published as notification topics and formally described in a WSDL file. Users do not need to be concerned with the conversion of communication messages.

Besides these benefits of implementing visualization systems with notification services, the advantages of distributed systems and Service-oriented architectures (mentioned in Chapter 3 and Chapter 4) are also inherited. We will explain in the following subsection how a visualization pipeline can be composed with subscriptions and notifications between visualization services.

5.2.4 Building Visualization Pipeline with Notification Services

The visualization pipeline, as introduced in Chapter 2, is a set of processes which transform data into graphical representations. These processes are implemented as modules in MVEs. Each module can have multiple input and output ports, and several control widgets. The modules are composed as a visualization pipeline with dataflow streaming from one module's output port to another module's input port.

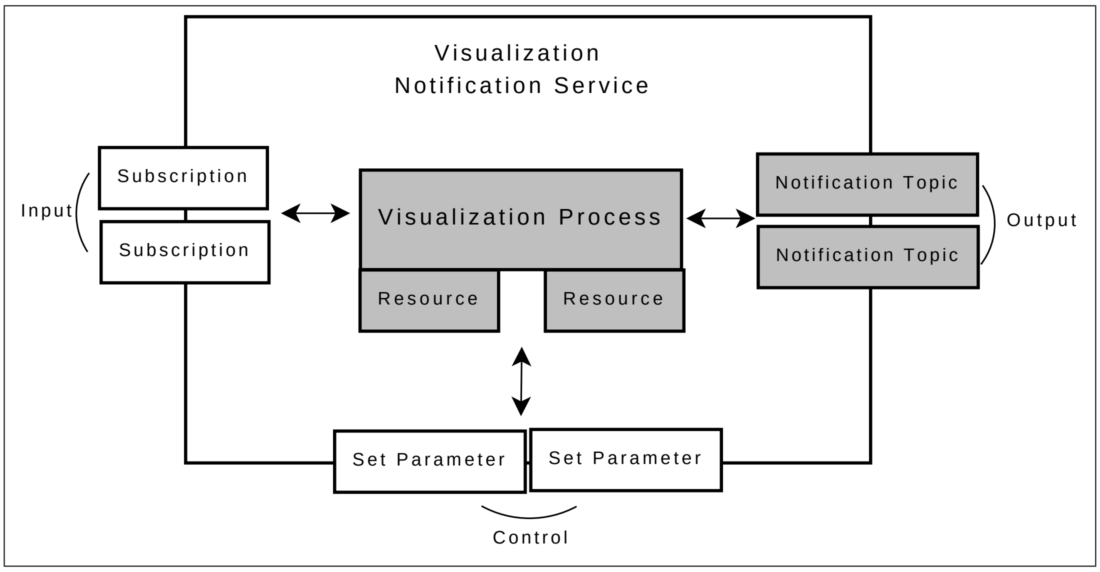


Figure 5.6: Implement a visualization module as a notification service.

When visualization modules are implemented as visualization notification services, the output ports become notification topics, and the input ports become subscription operations (shown in Figure 5.6). Steering parameters in these services are stored as service resources, which can be set through the method provided by each service.

Figure 5.7 shows a simple example of how visualization notification services can be linked as a visualization pipeline. The service (Service B) 'downstream' of the pipeline subscribes to a notification topic on an ‘upstream’ service (Service A). Every time Service A generates new data, the data will be delivered to Service B as a notification.

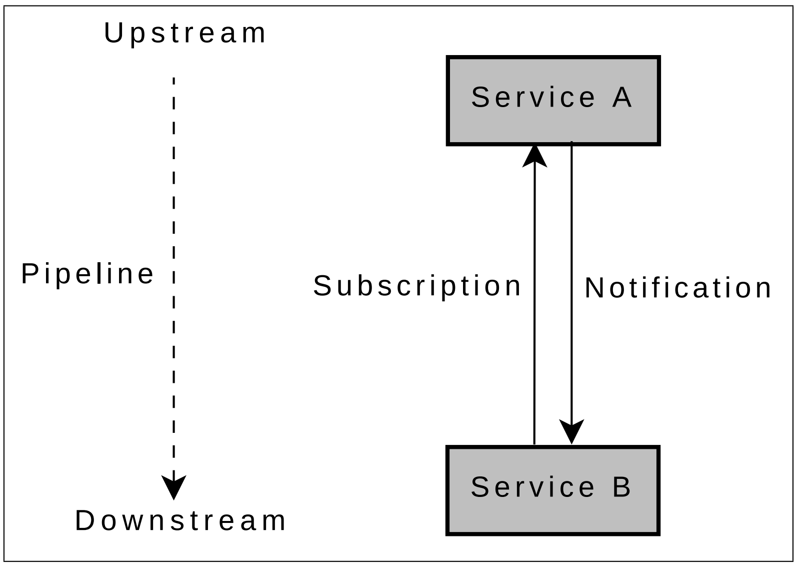


Figure 5.7: A simple pipeline composed of two notification services.

Figure 5.8 shows the abstract pipeline model and different implementations in MVEs and the NoCoV framework. A typical visualization pipeline consists of a Data step, a Filter step, a Mapping step and a Rendering step. In MVEs, these steps are implemented as visualization modules, whereas in the NoCoV framework, these visualization modules become notification services providing various visualization functions. These notification services are linked by using the notification/subscription mechanism to pass communication messages between them, but it is possible to use other mechanism (e.g. HTTP or FTP) to transfer data between them.

5.3 Management Layer of NoCoV Framework

The visualization services in the service layer can be heterogeneous in functionality, interface style, and location. Therefore, there arises the requirement for a management layer which sits between visualization services and end users. The management layer aims to provide easy access to visualization services and make the complexity of Grid computing transparent for end users.

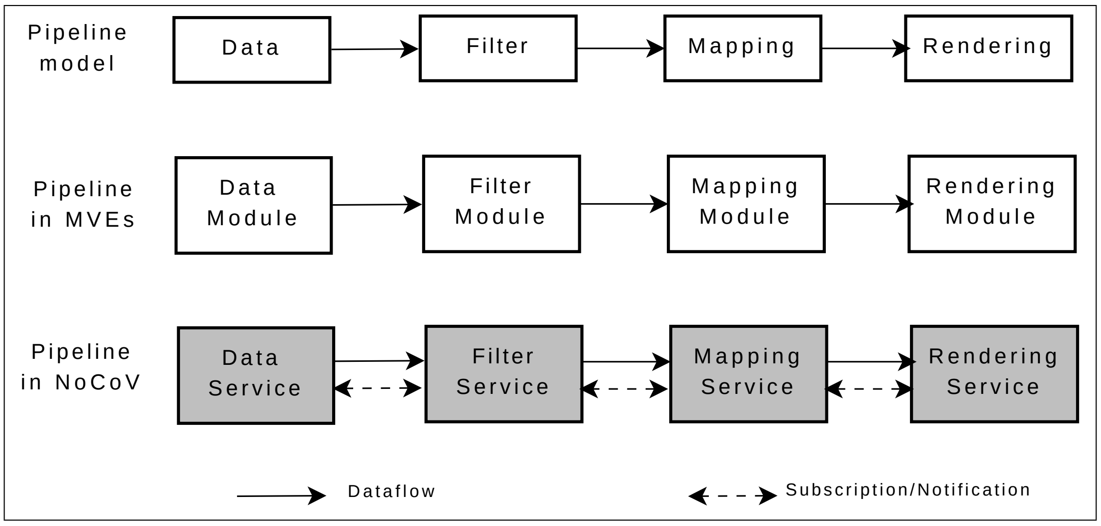


Figure 5.8: Implementations of pipeline model in MVEs and NoCoV framework.

5.3.1 Service Discovery

In experimental scenarios, we can assume the clients know exactly where the services are allocated and how the clients can invoke these services. However in realistic applications, clients usually do not have any information about services before the first time they invoke these services.

A UDDI registry can be used to bridge the gap between services and clients. UDDI [74], short for Universal Description, Discovery and Integration, is a standard interoperable platform that enables services across the boundary of different organisations and companies to be found and used easily. Figure 5.9 shows how a UDDI server can be used for service registration and discovery. After the service has been deployed, it can register to a UDDI server with the description of itself and the Uniform Resource Identifier (URI) where it is located. Clients can then discover the service from the UDDI service, and decide whether to connect to it based on the service description.

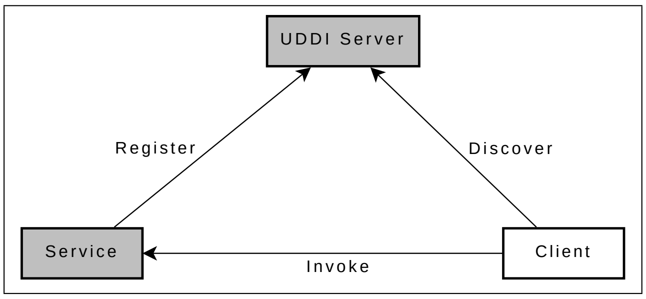


Figure 5.9: Service registry and discovery through a UDDI server.

By using a UDDI registry, information about widely distributed and heterogeneous visualization services can be gathered at a centralized server, which allow users to discover these visualization services easily. For a business service, the registry information usually consists of three aspects: the address to access the service; service description based on standard taxonomies; and the interface through which the service can be accessed. For visualization services, the addresses are the URIs, and the interfaces describe how to communicate with the service. As the research of visualization taxonomies still keeps maturing, we can expect, in the near future, there will be standard visualization taxonomies which can be used to describe visualization services.

In the NoCoV framework, the UDDI service is located in the management layer (in Figure 5.10). When a new visualization service is introduced into the system, it registers itself to the UDDI registry. The control services can access it programmatically and generate stubs automatically from the visualization service's WSDL file, so that the control services can communicate with the services. On the other hand, clients can connect to the UDDI registry to obtain a list of available visualization services with their descriptions. Choosing from this list, users can visually link these services on client GUIs to create customized visualization pipelines from their desktop, whereas these services are physically distributed on different Grid networks and managed by control services.

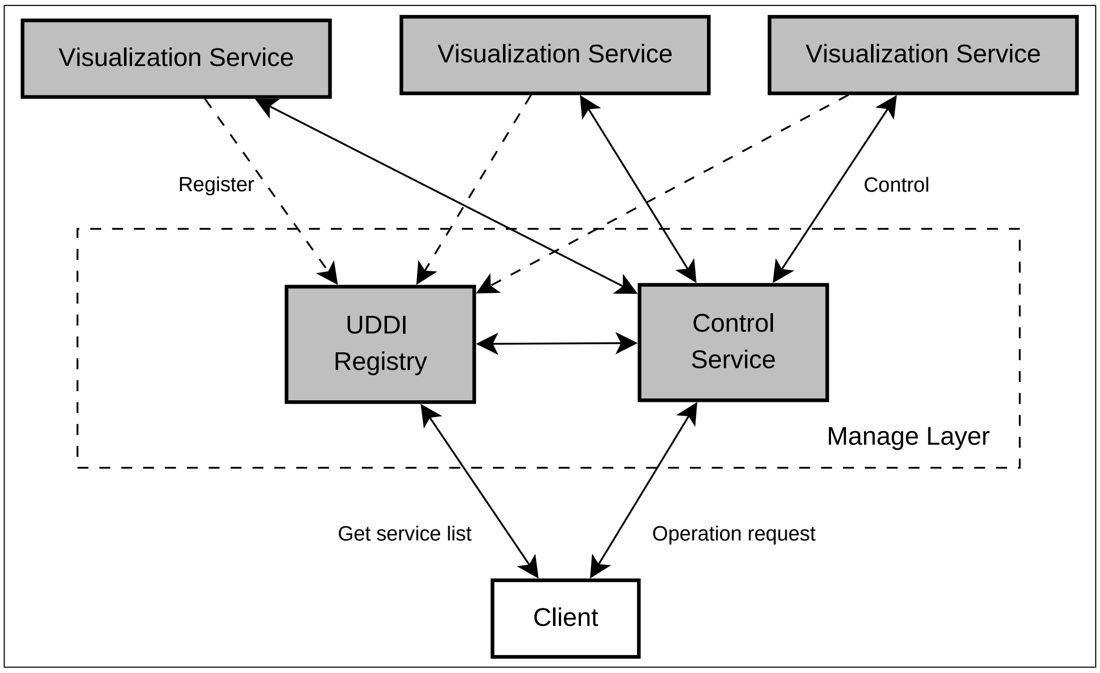


Figure 5.10: The usage of UDDI registry in NoCoV framework.

5.3.2 Visualization Description

There are two reasons why NoCoV requires descriptions for the visualization process.

- For synchronous collaboration. In a collaborative session, participants need to share the same visualization pipeline and steering settings. This visualization information needs to be captured and recorded in a well specified format.

- For asynchronous collaboration. The visualization descriptions can be stored as service resources. Users can work on the same visualization but at different times, as they can retrieve their previous visualization's description from the service when they initiate the connection to the control services in the management layer.

Although there are no standard visualization taxonomies which we can follow to describe visualization, there are dataflow description languages and workflow description languages (mentioned in Section 2.4.4) which can be used to describe visualization pipelines. As explained in Section 2.6, skML is an XML-based visualization description language. It aims to capture information of a visualization pipeline at an abstract level, which means that a skML description places no restriction on the module's implementation. Therefore, a visualization pipeline described in skML can be implemented either in an MVE, or in a Service-oriented architecture.

Although the skML description is independent of the implementation, it lacks certain features to describe characteristics of Service-Oriented collaborative visualization pipelines. Rather than create a different description language, we have chosen simply to modify and extend skML to reflect the Service-oriented and collaborative features of NoCoV.

One of the significant differences is that the NoCoV extension aims to present richer information about the visualization pipeline. As a skML description is not concerned the meaning of the visualization pipelines, there is no restriction on how detailed the information should be. However in NoCoV, as the visualization is shared and manipulated collaboratively by users, the description has to capture every details of the visualization. For example, all the output and input ports for each service are recorded in the pipeline description. These ports are then made visible at the user interface, to allow users the ability to change the connections to/from that service. In contrast with the original skML, all control parameters for a service instance must be explicit in the description, again so that they can be presented in an automatically generated user interface.

Another difference is the adding of new properties such as 'owner' and 'share', which identify who owns this visualization service instance and who are allowed to access it.

```
<map>
    <instance>
        <id>MyIso</id>
        <factory>iso</factory>
        <owner>UserA</owner>
        <share>sharable</share>
        <xPosition>127</xPosition>
        <yPosition>42</yPosition>
        <output>VizFileRPIso</output>
        <input>ValuesRPIso</input>
        <input>DimRPIso</input>
        <param>
            <name>isoValue</name>
            <value>0.78</value>
            <type>double</type>
            <range>
                <max>1.0</max>
                <min>0.0</min>
            </range>
        </param>
    </instance>
</map>
```

Table 5.1: A snippet of the visualization description adopted in NoCoV.

These new properties make it possible to add security control in NoCoV. A user will be able to keep some private service instances during the collaboration, if he sets his service instance as ‘non-sharable’. Or, a user can just share with other participants the output of the service instance he created, but not the setting of control parameters on that instance, if the service instance is set as ‘readable’.

There are two basic tags in skML: ‘module’ and ‘link’. The ‘module’ tag is replaced by ‘instance’, as now in NoCoV, the visualization pipeline is composed of a set of service instances which have service resources associated, instead of modules. The ‘link’ element in skML is kept, but now it is used to represent the subscriptions and notifications.

The snippet listed in Table 5.1 describes an instance named as ‘MyIso’ described in the NoCoV extension of skML. It is created by ‘UserA’ from a service factory called ‘iso’, whose function is to generate an isosurface for the provided dataset. The instance is set as ‘sharable’ for all the participants in the collaborative session. The ‘xPosition’ and ‘yPosition’ specify where a symbol for the instance will appear in the user’s GUI. According to the description, the instance has two input ports and one output port through which users can wire it into a visualization pipeline. There is one parameter called ‘isoValue’

```
<link>
    <instance ref=MyData, output=ValuesRPData/>
    <instance ref=MyIso, input=ValuesRPIso/>
</link>
```

Table 5.2: Description of a link between two instances.

where the user can set the threshold of the isosurface to be generated. The value of the current setting, the type of the parameter and the range of values are all recorded in the description.

Table 5.2 describes a subscription/notification between the two instances: 'MyData' and 'MyIso'. 'MyIso' instance subscribes to the notification topic on 'MyData' called 'ValuesRPData', and the notification delivered from 'MyData' will be received at the 'ValuesRPIso' input port on 'MyIso'.

5.3.3 Access Control

The requirement for access control in the NoCoV framework arises from two of its features:

- **The distributed feature of the system**. As the NoCoV system consists of loosely-coupled services, NoCoV can be distributed across different Grid networks. The wider distribution implies the higher possibility of being attacked by unexpected users, therefore access control can make sure only authorized users can access the system.
- **The collaborative feature of the system**. Collaboration implies the NoCoV system accommodates multiple users either at different times or at the same time. Access control is needed for each collaborative session, so that only one particular group of users is allowed to participate in a particular session. Moreover, inside a collaborative session, each service instance involved can have access restricted, because different users in the same session may play different roles in the collaboration.

Therefore, the access control involved in the NoCoV framework can be applied at three levels: system level, session level, and visualization instance level.

System level. At the system level, users are divided into two groups: master users and common users. A common user can join existing collaborative sessions with permission offered from the creator/owner of the session. A master user can not only join existing sessions, but also create and own new collaborative sessions. As the owner of a collaborative session, a master user can then specify who are allowed to join the session.

"/O=Grid/O=White Rose/OU=leeds.ac.uk/CN=Haoxiang Wang" sean
"/O=Grid/O=White Rose/OU=leeds.ac.uk/CN=Test User" test

Table 5.3: An example of a grid-map file.

A list of valid users is stored as a grid-map file which can be used for authorization on the control service. Each authorized user is represented in the grid-map file as a Distinguished Name (DN). The grid-map file example listed in Table 5.3 contains two valid users: 'sean' with DN '/O=Grid /O=White Rose /OU=leeds.ac.uk /CN=Haoxiang Wang' and 'test' with DN '/O=Grid /O=White Rose /OU=leeds.ac.uk /CN=Test User'. When a service uses grid-map authorization, only the users listed in the grid-map file are allowed to invoke the service.

Displayed as Figure 5.11, the Control Service Factory is used to create resources for each collaborative session. By associating the grid-map file of master users with the factory service, only master users are allowed to create new sessions. Both master users and common users can access the Control Service Instance, through which they can manipulate corresponding service resources.

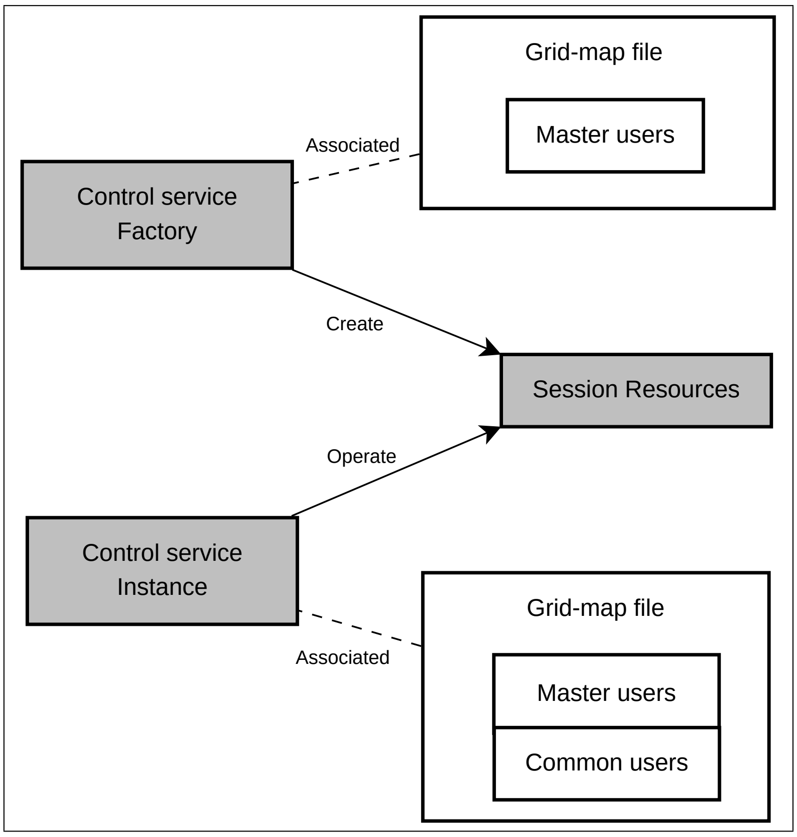


Figure 5.11: Access control at the system level in NoCoV framework.

Session level. In Figure 5.11, the session resource holds information about a particular collaborative session. As there can be multiple collaborative sessions running in parallel, there exist multiple session resources with the control service. Only the creator/owner of the session and users authorized by him can join the session, in other words, access the session resource. Therefore, access control is required for each session resource (figure 5.12).

Different from the grid-map files used for the access control at the system level, for each session, the grid-map file associated is temporary and dynamically updated. When a collaborative session finishes, the corresponding grid-map file will be discarded. Also, every time the owner authorizes a valid user or removes access permission from a user, the associated grid-map file will be updated.

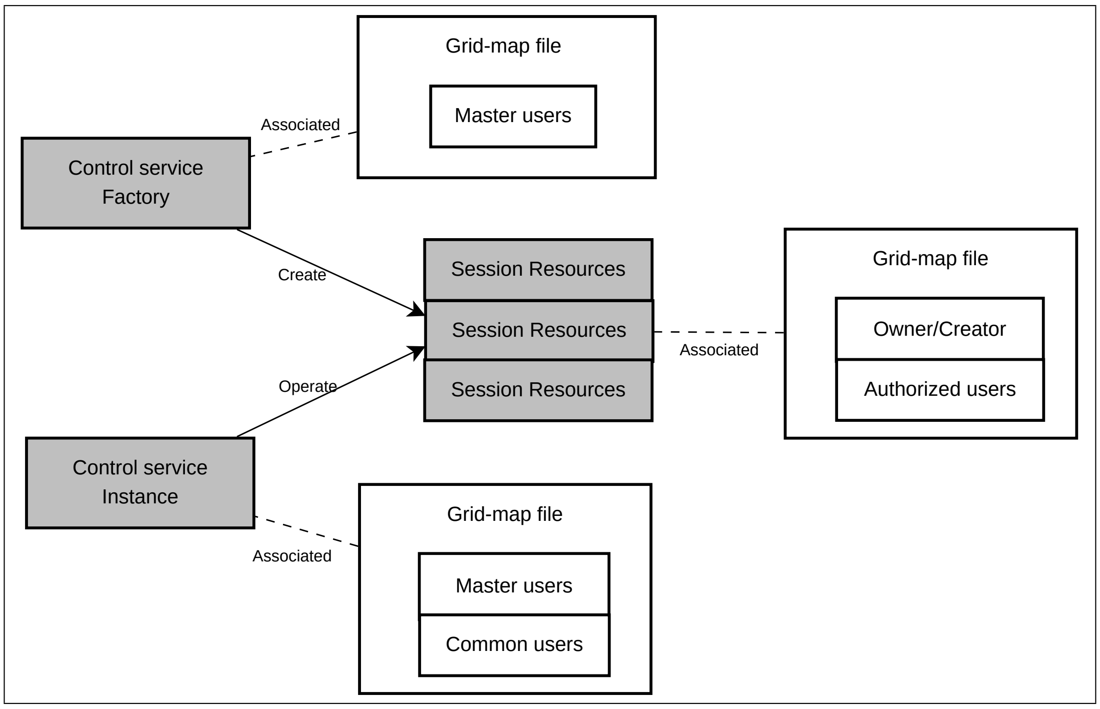


Figure 5.12: The grid-map authorization with each session.

Visualization Instance level. Within a collaborative session, a visualization pipeline can be generated jointly by different participants. The visualization pipeline consists of visualization service instances which might be initiated by different collaborators. The access control at the visualization instance level is concerned with how users can share visualization service instances created by different participants.

As mentioned in section 5.3.2, there are two tags that control the access to service instances in the visualization description used by NoCoV: 'owner' and 'share'. The owner has full control of the instance. The 'share' tag has three values to choose: 'sharable', 'readable' and 'non-sharable'. As indicated by the names, 'sharable' offers the other participants full control of the instance; 'readable' only allows other users to use the output data from the instance, but without the setting of its control parameters; and 'non-sharable' labels that instance as a private instance which can only be accessed by the owner.

5.3.4 Pipeline Control

One of the main functions provided by the management layer is to forward operational requests from the client layer to the distributed visualization services. The operations sent from the client layer include:

- Creating a visualization service resource. The message sent from the management layer to the corresponding service factory is to initiate a new service resource and return its EPR.
- Destroying an existing visualization service resource. The management layer sends a request to the corresponding service factory to destroy a service resource.
- Connecting an output port from service instance A to an input port of service instance B. The management layer generates a request to ask service B to subscribe to a notification topic published by service A.
- Discarding the connection between two service instances. The message received by the corresponding visualization service will be 'stop the subscription'.
- Setting control parameters for a visualization service instance. The management layer invokes the corresponding service's interface for setting parameters.
- Specifying the endpoint of a visualization pipeline. The endpoint of a pipeline is where the visualization result is finally generated. By specifying the endpoint, the management layer can know which service's output is of interest to end-users and needs to be sent back to the client layer.

The requests are described in XML representations. Table 5.4 is an example of the XML description of setting a control parameter for a visualization service instance. In the snippet, the target service instance is called 'MyIso', and the target parameter is named as 'isoValue'. The request is to set 'isoValue' to 0.78.

```
<set>
    <instance>MyIso</instance>
    <param>
        <name>isoValue</name>
        <value>0.78</value>
    </param>
</set>
```

Table 5.4: An XML description of a request to set a parameter.

These requests are parsed at the management layer, and then delivered to the visualization services involved (shown in Figure 5.13). With the pipeline control functions provided by the management layer, the heterogeneous visualization services become transparent for the client layer. As there is no direct connection between the client layer and the service layer, clients do not need to be concerned about either the creating of stubs to communicate with visualization services, or the management of the URIs and EPRs to these services and resources. As long as clients are aware of the different functionalities provided by services and the meanings of different control parameters, these visualization services distributed across Grid networks can be used and managed from GUIs on users' desktops. The client GUIs will be introduced in section 5.4.

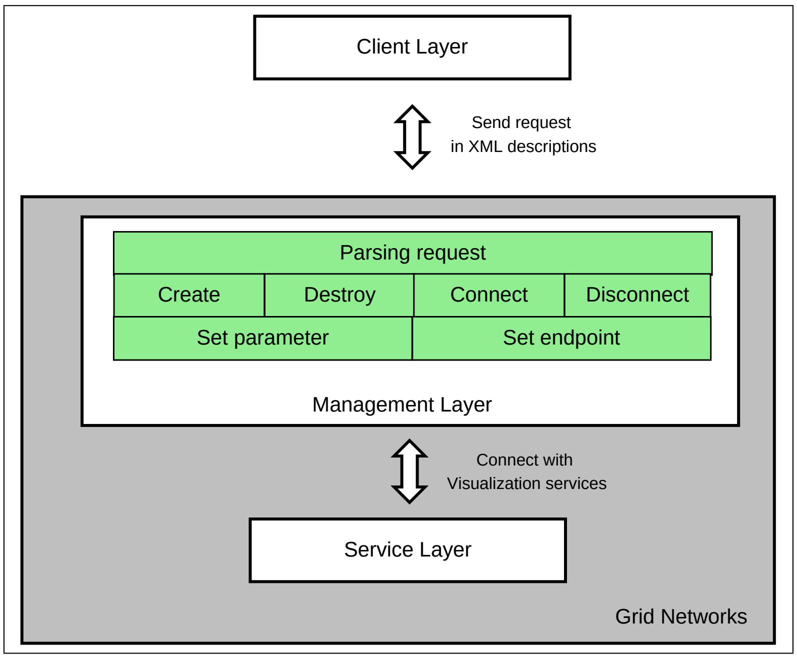


Figure 5.13: The pipeline control functionalities provided by the management layer.

The transparent pipeline control is not the only functionality provided by the management layer. Other features, such as synchronous and asynchronous collaboration, are introduced in the following sections.

5.3.5 Synchronous Collaboration

The NoCoV framework supports both synchronous and asynchronous collaboration for end-users. We focus on synchronous collaboration in this subsection.

Synchronous collaboration in the NoCoV framework allows multiple users to work on the same visualization at the same time. This involves the following aspects:

- **The sharing of visualization pipeline**. The NoCoV framework allows users to jointly create a visualization pipeline and share the pipeline with other participants in the same collaborative session.
- **The sharing of visualization steering**. Different users can collaboratively set control parameters for the visualization services wired in a pipeline to steer the visualization.
- **The collaboration between visualization experts and application scientists**. The visualization experts can firstly generate a pipeline with suitable services chosen from visualization service registry, and then provide an interface for application scientists to set suitable parameters to steer the pipeline and produce the visualization result they require.

The sharing of visualization information is based on the visualization description mentioned in Section 5.3.2. At the management layer, a visualization description is stored as a service resource. When the resource is published as a notification topic, multiple users can share the description by subscribing to the same notification topic. Any changes in the pipeline description caused by any participants will be delivered to all the clients subscribing to it.

Figure 5.14 shows how the synchronous collaboration works. The notification topics published at the management layer include:

- **Visualization pipeline description**. It holds the detailed information about the current pipeline. It includes how the services are connected, how the control parameters are set and the properties of each visualization service involved.
- **Service operation description**. It records the last operation on the visualization service in XML format. The operation can be initiating, destroying, subscribing, etc.

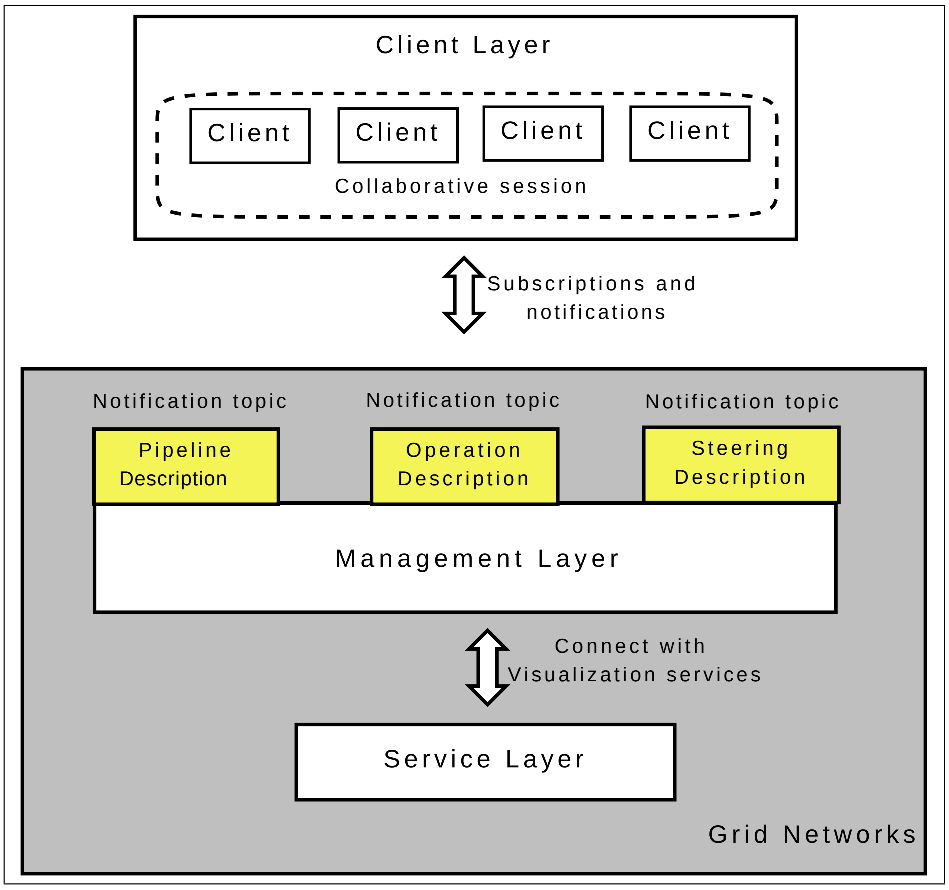


Figure 5.14: Synchronous collaboration by subscribing to notification topics.

- **Visualization steering description**. It stores the last steering operation from participants in a collaborative session. The steering information is about the settings of visualization services' control parameters.

When a client receives the visualization pipeline description from the notification topic, it will be converted into a visual representation on the client's GUI. As the participants in a collaborative session subscribe to the same pipeline description topic, all the participants will view the same pipeline on their GUIs. The operation description and steering description give users the information about other participants' actions in the collaboration.

5.3.6 Asynchronous Collaboration

In contrast to synchronous collaboration, asynchronous collaboration allows different users to work on the same visualization at different times.

The NoCoV system captures the detailed information of the visualization pipeline and the visualization services involved in a description using XML format. Each collaborative session has a corresponding service resource to store its visualization description. This description resource will be kept at the management layer for any client to retrieve until the collaborative session is ended by the session owner.

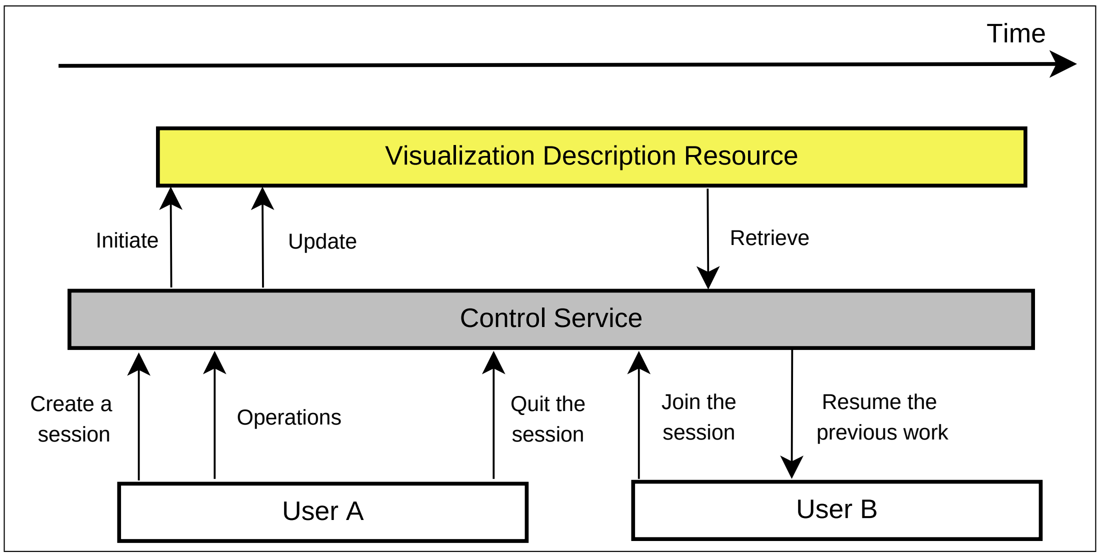


Figure 5.15: Asynchronous collaboration in the NoCoV framework.

Figure 5.15 illustrates how asynchronous collaboration is supported in the NoCoV framework by using a simple example with two different users involved. When user A creates the collaborative session, a service resource is initiated to keep this session's visualization information. Every time the pipeline or the control parameters are changed, the resource is updated with the latest visualization description. When user A quits the session, he leaves the session alive, so that user B can join the session at a later time and retrieve the visualization description previously stored by user A. By using this description, user B can continue user A's work asynchronously.

Asynchronous collaboration can be useful for worldwide collaboration. As participants can come from different time zones, it will be difficult for all of them to work at the same time. The capability of storing the visualization description at the management layer does not only support collaboration for multiple users, but also provides a storage for keeping the visualization pipeline and control settings for a single user who may want to work across several sessions.

In GT4, a service resource is able to store historical information by recording previous information in a resource stack. It offers the possibility to support visualization provenance, so that when a user returns to a session, he can choose a certain point of the historical record at which to resume the previous work.

5.3.7 Dynamic and Transparent Visualization Migration

Another functionality provided by the management layer is the dynamical and transparent visualization migration. After a visualization pipeline is created by end-users, the NoCoV system can dynamically migrate the visualization tasks to different locations to optimize the performance. As there is no direct connection between the client layer and the service layer, as long as the service provides the same interface and visualization functionality with the one to be replaced, the migration of visualization tasks will also be transparent to end-users.

The migration of visualization tasks can be used in the following scenarios:

- When a visualization service is overloaded, we can achieve better performance if the visualization task run on that service can be migrated to another service with a light workload.

- In a widely distributed system, the network connection to a remote service can be relatively poor. By migrating the visualization task to another service with better network connection (e.g. physically near to the data source), the time cost for transferring data will be reduced.

- When the service host experiences some fatal error and has to be switched off, if the visualization task carried by the service can be moved to a substitute service, the whole visualization pipeline can be more robust.

The replacement of visualization services is managed at the management layer, without affecting the client layer. As shown in Figure 5.16, the migration of a visualization task is transparent to the client layer. The management layer firstly finds out the service to be replaced and its parameter settings from the visualization description, and then, finds the substitute service which has the same visualization function as the original service. By setting the substitute service with the same parameters with the original , the visualization task can be migrated to the substitute service seamlessly. After the substitute service has started and been linked into the existing pipeline in the same way as the original service, the original one will be disconnected and stopped.

A further idea of visualization migration is the dynamic deployment of visualization services. The distributed visualization services can store their Grid Archives (gars) at the management layer. When clients start to build their visualization pipelines, they will have the ability to choose where to deploy these services. Therefore, these visualization services can be hosted in the Grid environment which is familiar, robust or trusted by end-users.

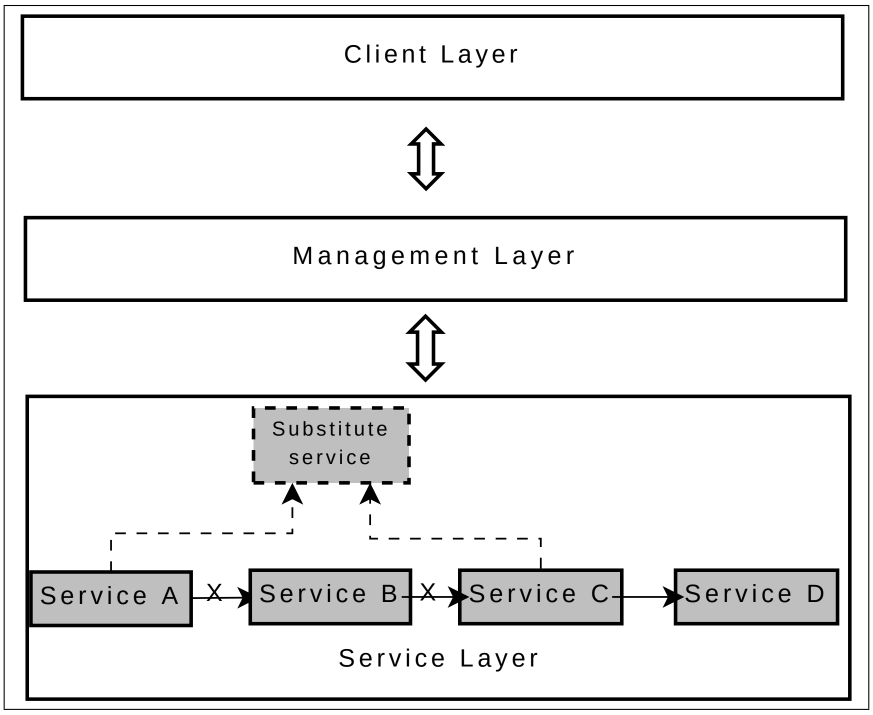


Figure 5.16: The transparent migration of a visualization task from Service B to its substitute service.

5.4 Client Layer of NoCoV Framework

At the service layer, visualization service providers/developers are enabled to collaboratively produce a visualization registry for the NoCoV system. However, it requires expertise in both Grid computing and visualization to combine these services in an appropriate way to create a visualization. For the visualization experts who are not capable of programming in the Grid environment, or application scientists who have neither Grid computing skills nor visualization expertise, user-friendly GUIs are required to create or manage the Grid-based visualization pipeline easily from their desktops.

At the client layer of NoCoV framework, both visualization experts and application scientists are accommodated by using two kinds of clients: a visualization pipeline editor client and a parameter control client.

5.4.1 Visualization Pipeline Editor Client

The visualization pipeline editor client is the GUI which allows end-users to build up visualization pipelines simply by drag-and-drop operations. Collaboration is also supported. By running their own pipeline editor clients, the participants in the same collaborative session can jointly generate a visualization pipeline.

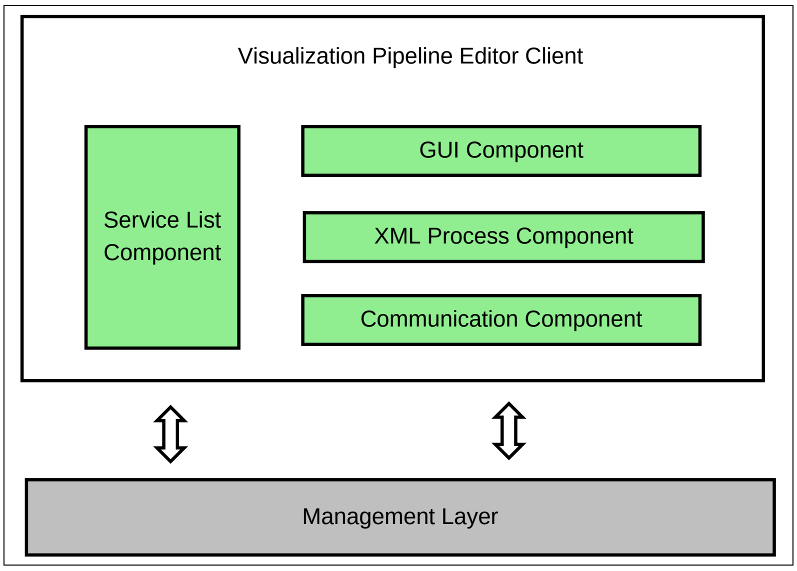


Figure 5.17: The components inside the visualization pipeline editor client.

The components in the visualization pipeline editor client are shown in Figure 5.17. The service list component obtains information about all the available visualization services registered on the UDDI server at the management layer. The services listed in the pipeline editor GUI have supplementary descriptions, through which end-users can know what visualization functionalities are provided by these services and what control parameters can be set to steer visualization.

The GUI component manages the interactions between end-users and the pipeline editor client. It allows end-users to initiate/destroy a visualization service instance, connect/disconnect two service instances, and set an endpoint to a visualization pipeline.

The XML process component acts as a translator. It parses the pipeline description retrieved from the management layer, so that the description can be visually represented in the client GUI. Every time an end-user makes a change to the visualization pipeline, the XML process component converts the operation into an XML representation and updates the pipeline description.

The communication between pipeline editor client and the management layer is managed by the communication component. The communication includes: the notifications delivered from the management layer and the requests of visualization service operations sent from the pipeline editor client.

The participants in the same collaborative session can subscribe to the corresponding notification topics published at the management layer. The visualization pipeline description is delivered to all the participants as a notification so that the end-users can share the visualization pipeline. The notification topic holding the latest operation from the participants is also broadcast as a notification to all the end-users, so that they can share the awareness of each other's actions during the collaboration. Therefore, with the pipeline editor client, multiple visualization experts can collaboratively build up a visualization pipeline.

5.4.2 Parameter Control Client

To create a visualization pipeline from a list of available services requires visualization expertise from end-users. However, application scientists might not be capable to build up visualization pipelines. Hence, there emerges the requirement that visualization experts jointly produce a visualization pipeline and provide a control GUI for application scientists to steer the pipeline.

A parameter control client is dynamically created from a visualization pipeline description. Each service instance has corresponding control widgets presented on the parameter control client GUI, through which application scientists can set parameters to control these services.

The components in the parameter control client include: a communication handler, a description handler, and a GUI handler (as displayed in Figure 5.18).

The communication handler subscribes to the notification topic at the management layer to receive the visualization pipeline description generated by the visualization pipeline editor client. When end-users change parameter settings on the parameter control client, the changes are firstly sent by the communication handler to the management layer, and then forwarded to the corresponding visualization services.

The description handler parses the visualization description received from the management layer, so that the GUI handler can generate widgets for each service used in the visualization pipeline. When end-users manipulate the control widgets on the GUI, the operations are converted into XML representations by the description handler, and then sent to the management layer.

After the GUI is initially generated by the GUI handler, it keeps updating the control widgets in the GUI according to the changes in the pipeline description. When a service instance is wired into the pipeline, its control widgets will automatically pop up in the GUI. When a service instance is destroyed, its control widgets will be removed from the GUI.

With multiple parameter control clients connecting to the same collaborative session, application scientists can collaboratively steer the visualization pipeline which can be jointly created by visualization experts from the pipeline editor clients.

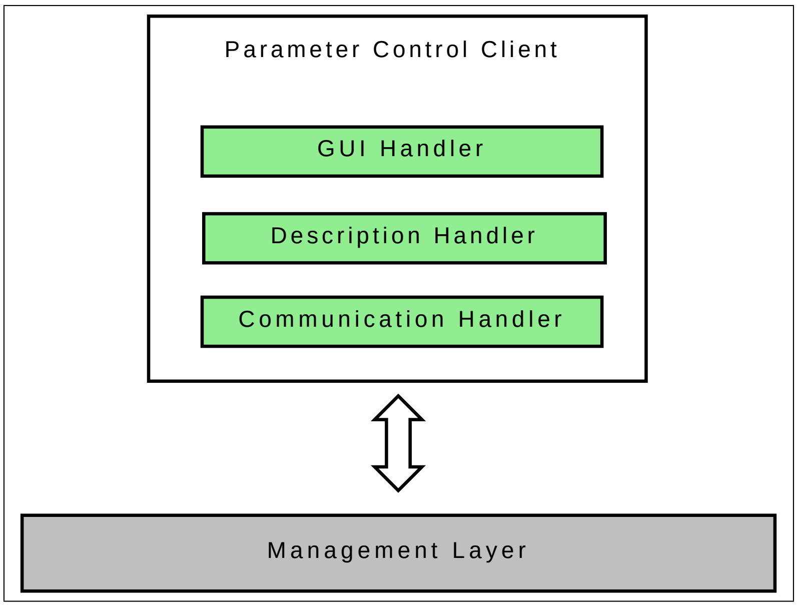


Figure 5.18: The components composing the parameter control client.

5.5 Summary

5.5.1 Framework Design

The NoCoV framework is a notification-service-based visualization system supporting collaborative working. In summary, the design of NoCoV incorporates the following key features:

Distributed notification-service-based visualization pipeline. Visualization pipelines in NoCoV are composed of distributed notification services which provide various visualization functionalities. Each downstream service subscribes to an upstream service, so that the output from the upstream service will be delivered to the downstream service as a notification. The visualization services in NoCoV can be collaboratively provided by different developers and deployed on different Grid networks.

Pipeline management. In order to simplify the use of heterogeneous visualization services for end-users, the NoCoV framework has a management layer which hides the complexity of Grid computing and makes visualization services transparent for end-users.

A UDDI registry is involved in the management layer for the discovery of visualization services. End-users manipulate these services through the management layer. Visualization pipelines created and operations are recorded as XML-based descriptions. By publishing a visualization description as a notification topic, a visualization pipeline can be shared within a collaborative session, so that the participants can jointly create, modify and steer the visualization.

Widely distributed visualization services are transparent for end-users. The visualization tasks can be migrated between services deployed at different locations but with the same function and interface, so as to enhance the performance and robustness of the system.

Access control is integrated with the management layer at three levels. At the system level, NoCoV can only be accessed by authorized users, among whom only the master users are able to create their own collaborative sessions. At the session level, the session owner/creator can dynamically grant or remove access permissions to or from participants, so that only authorized users can be allowed to join a particular collaborative session. At the visualization service instance level, within a collaborative session, options are provided for users to set their instances as sharable, readable or non-sharable to other collaborators.

NoCoV stores visualization descriptions at the management layer as service resources with limited lifetime, therefore it is possible to support asynchronous working. When an end-user quits a session, as long as the session is still alive, he or other participants can login to the session later and continue the previous work.

Collaboration. In addition to the collaborative provision of visualization services and the asynchronous work just mentioned, synchronous collaboration is supported by the NoCoV framework for the creation of visualization pipelines and the steering of visualization. Collaboration between visualization experts and application scientists is also allowed by the framework. Visualization experts can build up pipelines through pipeline editor client, whereas the application scientists can steer the visualization through parameter control client which dynamically creates control widgets from the pipeline description created by visualization experts.

5.5.2 Framework Structure for Implementation

The three layer structure of the NoCoV framework is shown in Figure 5.19. Both service layer and management layer are composed of distributed services deployed on Grid networks, whereas the client layer provides access to the heterogonous Grid resources from end-users' desktops.

The service layer has different visualization services, which can be linked together as a pipeline. The management layer directly connects to the distributed visualization services. It also provides the support for both asynchronous and synchronous collaboration. The client layer offers two kinds of GUI client for end-users to build up or steer a visualization pipeline. In the following chapter, we will describe a prototype system which is

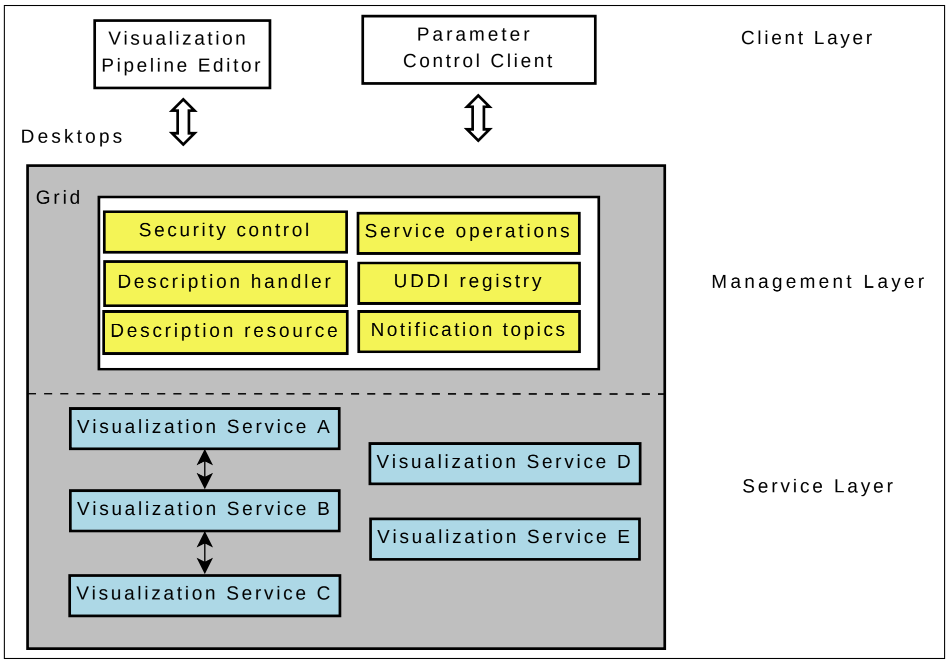


Figure 5.19: The structure of the NoCoV framework.

implemented using the three layer structure illustrated in Figure 5.19.

Chapter 6

The NoCoV Prototype Implementation

Based on the NoCoV framework design described in Chapter 5, in this chapter, we describe how we implemented a prototype to act as proof of concept.

The NoCoV prototype consists of a set of sample visualization services, a control service, a pipeline editor client and a parameter control client. The prototype aims to reflect the main features of the NoCoV framework: notification-service-based visualization pipeline, remote control of Grid-based pipeline and collaborative working.

6.1 Aims of the Prototype

The NoCoV prototype is intended to examine a number of concepts put forward in the design framework. It mainly focuses on the following aspects:

- **Notification services for visualization**. The prototype aims to verify the composition of a visualization pipeline in terms of a set of notification services which provide various visualization functionalities. The services are linked by subscriptions and notifications implementing the output and input ports between two connected services. This means the data produced from an output port can be delivered to other subscribing services as notification messages.

- **Transparent control of remote visualization pipeline**. NoCoV visualization pipelines are distributed on different Grid networks with Grid computing technologies. Therefore, for end-users who do not have Grid computing skills, there is the requirement of easy creation and steering of Grid-based visualization pipelines. The prototype will use a control service which passes communication messages between visualization services and end-users, so that the complexity of Grid computing and communications with various services can be transparent for end-users.

- **Collaborative creating and steering of visualization**. Collaboration is one of the main design features of the NoCoV framework. By saving the visualization description as a published notification topic, different end-users can share the same visualization description which includes information about the visualization pipeline and parameter settings in each service involved. The prototype will develop client GUIs, which end-users can run on their local desktop. Through the GUIs, they can collaboratively create and steer visualization pipelines.

Certain features, such as security control, service discovery, etc., are not implemented in the prototype. The first reason is the time restriction; and the second reason is the lack of mature standard technologies. For example, the discovery of visualization services requires standard taxonomies to describe various services and these do not exist yet. For the security control, as the system can be distributed across different Grid networks, the authentication and authorization across multiple organisations also needs a standard mechanism to implement.

6.2 NoCoV Prototype Overview

Based on the NoCoV framework described in Chapter 5, the prototype implementation can be divided into three layers: visualization service layer, control service layer, and client layer.

Visualization service layer. Four simple visualization services are developed as notification services at the service layer. These services are:

- A data service. It allows users to feed raw data into a visualization pipeline for processing. The data source can be either a data file stored on the server or an online data source accessible by its URL.

- A slice service. The slice service can generate a cross-section in a 3D space representing the values at each point on the cross section with different colours by using a colour map. A method is provided by the slice service for users to specify the position of the slice to be created.

- An isosurface service. It can produce an isosurface according to the threshold value set by users. An isosurface contains all the points which have the same value in a 3D space.
- An inline service. The inline service can allow users to combine different visualization results in the same scene. By using it, users can easily create a visualization with multiple slices and isosurface.

Control service layer. A pipeline control service is developed at the control service layer. The pipeline control does not only communicate with the visualization services on users' behalf, but also allows end-users to share a visualization and collaboratively work on it. A visualization description handler component, which can parse and generate XML-based visualization descriptions, is integrated with the pipeline control service, so that visualization information can be recorded in a service resource and published as a notification topic.

Client layer. As designed in the framework, the client layer provides different client GUIs for different types of end-users. The NoCoV prototype implements a pipeline editor client for visualization experts to jointly create a visualization and a parameter control client for application scientists to collaboratively steer a visualization.

6.3 Notification Services for Visualization

The visualization services involved in the NoCoV system are Web services which support notification features. In the NoCoV prototype, these notification visualization services are implemented with GT4.

For these visualization services, the use of the notification mechanism is mainly concerned with the implementation of dataflow between two connected services in a visualization pipeline. The input port and the output port of a service can be regarded as a subscription and a notification respectively, therefore the output data generated from a service (service A) can be delivered to the next service (which subscribes to the service A) in the pipeline as a notification message.

There is an alternative way of implementing the dataflow between two services. Instead of passing the data values as a notification message, we can simply wrap the reference (URL or address) to the data in the notification message. The receiving service then needs to fetch the data values by using HTTP, FTP or other mechanisms.

These two approaches actually work following the same principle - using notification/subscription mechanism to pass communication messages between services. The difference is in the contents wrapped in the notification messages: either the dataset itself, which will vary in size according to the data and can be quite large; or the reference to the dataset, which is a short string. A comparison between these two approaches is given in the evaluation of the NoCoV system (Chapter 7). Here, for the ease of description, we only describe the approach which passes data values as notification messages in this chapter.

In the prototype, four example visualization services are developed, so that it is possible for end-users to build up simple visualization pipelines to verify the design of the NoCoV framework. The implementation of these visualization services is discussed in detail in the following subsections.

6.3.1 Data Service

Raw data can be loaded into a visualization pipeline by using a data service, in which users need to specify the raw data's address (this can be a URL address). In the prototype, the data service does not require input from other services, however it requires a string parameter which represents the address of the raw data. The output ports are implemented as three notification topics: a ValueArray which contains the values of each point in a three-dimensional space; a DimArray which holds the size of each of the three dimensions; and a MetaData which is additional information about the raw data (e.g. the range of the raw data).

The data service involves two WSDL files: one is the description of the service itself (Data.wsdl), the other is the description of the factory service (Factory.wsdl).

Table 6.1 is a snippet of the Data.wsdl file. It describes a method called 'setFile' provided by the data service, labelled as (1) in Table 6.1. The snippet describes what input parameter is required by the method (a string value to specify the file address) and what output can be expected from the service (a string value to indicate whether the invoking of the method is successful). It also declares three service resource properties to store data, dimension of the data and meta data, labelled (2) in Table 6.1.

The Factory.wsdl file describes how users can initiate the service resource described in Data.wsdl through the interface provided by the factory service. As labelled by (1) in Table 6.2, there is only one method exposed in the factory service called 'createResource'. No parameter is required to invoke the method to create resource properties, labelled by (2) in Table 6.2. After the successful creation of the resources, the corresponding message returned from the factory service will be an Endpoint Reference (EPR) of the resource created, labelled by (3) in Table 6.2.

```
        ...
<xsd:element name="setFile" type="xsd:string"/>                          (1)
<xsd:element name="setFileResponse" type="xsd:string"/>
        ...
<xsd:element name="dimRP">                                                (2)
        ...
</xsd:element>
<xsd:element name="valuesRP">                                             (2)
        ...
</xsd:element>
<xsd:element name="metaDataRP">                                           (2)
        ...
</xsd:element>
        ...
<message name="SetFileInputMessage">                                      (1)
        <part name="parameters" element="tns:setFile"/>
</message>
<message name="SetFileOutputMessage">
        <part name="parameters" element="tns:setFileResponse"/>
</message>
        ...
<portType name="DataPortType"
        wsdlpp:extends="wsrpw:GetResourceProperty
        wsntw:NotificationProducer">
        <operation name="setFile">                                        (1)
                <input message="tns:SetFileInputMessage"/>
                <output message="tns:SetFileOutputMessage"/>
        </operation>
</portType>
...
```

Table 6.1: The snippet from file Data.wsdl.

```
      ...
<xsd:element name="createResource">                                    (2)
      <xsd:complexType/>
</xsd:element>
<xsd:element name="createResourceResponse">                            (3)
      <xsd:complexType>
      <xsd:sequence>
            <xsd:element ref="wsa:EndpointReference"/>
      </xsd:sequence>
      </xsd:complexType>
</xsd:element>
      ...
<message name="CreateResourceRequest">
      <part name="request" element="tns:createResource"/>
</message>
<message name="CreateResourceResponse">
      <part name="response" element="tns:createResourceResponse"/>
</message>
      ...
<portType name="FactoryPortType">                                      (1)
      <operation name="createResource">
      <input message="tns:CreateResourceRequest"/>
      <output message="tns:CreateResourceResponse"/>
      </operation>
</portType>
      ...
```

Table 6.2: The snippet from file Factory.wsdl.

```
    ...
private ResourceProperty valuesResourceRP;                                    (1)
    ...
valuesResourceRP = new SimpleResourceProperty(DataQNames.RP_VALUESRP);
    ...
this.topicList = new SimpleTopicList(this);                                   (2)
    ...
valuesResourceRP = new ResourcePropertyTopic(valuesResourceRP);               (3)
((ResourcePropertyTopic) valuesResourceRP).setSendOldValue(false);
    ...
this.topicList.addTopic((Topic) valuesResourceRP);                            (4)
    ...
```

Table 6.3: The snippet from DataResource.java.

The implementation of the data service involves the following Java classes:

- DataService. It reads in raw data using the address provided by users, and stores data values and meta description in the associated service resources.
- DataFactoryService. It asks DataResourceHome to create service resources for different users and returns the EPR generated by using the key identity returned from DataResourceHome.
- DataResourceHome. It implements a 'create' method, which initiates service resources according to the definition set by the DataResource class, and returns a unique key to identify the resource created.
- DataResource. It defines the service resource associated with the data service. Get/Set methods are provided by the DataResource class to access the resource. DataResource also publishes these service resource properties as notification topics, so that subscribers can automatically be updated with the changes of the raw data.
- DataQNames. It maps the names used in the Java code to the names used in the XML representation (WSDL files). It also specifies the namespace of the names used in the service.

Table 6.3 is a snippet from DataResource.java. It illustrates how the service resource property holding data values is published as a notification topic. The snippet listed in Table 6.3 is explained as follows:

1. Initiate a ResourceProperty value to store the resource containing data values.
2. In order to set up the notification mechanism, the DataResource class implements TopicListAccessor class. At this step, the topicList inherited from TopicListAccessor class needs to be initiated with an empty list.
3. Convert the Resource Property into a Resource Property Topic, which can be subscribed to by clients.
4. Add the topic representing data values into the topic list. After this step, the resource holding data values has been published as a notification topic.

With the adoption of the notification mechanism, when users change the data file to be visualized during run time, the subscribers, which can be either other services or customized clients, will immediately receive notification messages about the change.

6.3.2 Slice Service

The slice service is deployed on a GT4 container and produces slices visualization using the data values received at the input port. The slice generated by the service is a restriction of the 3D dataset to a plane orthogonal to one of the three axes. If the 3D data is provided on an axis-aligned mesh,

$$\begin{aligned} F_{ijk} = F(x_i, y_j, z_k), \quad where \quad & i = 1, ...m \\ & j = 1, ...n \\ & k = 1, ...p \end{aligned}$$

then a slice orthogonal to the x-axis, say, will display the data:

$$\begin{aligned} F_{jk} = F(x^*, y_j, z_k), \quad where \quad & j = 1, ...n \\ & k = 1, ...j \\ & x^* = x_i \end{aligned}$$

Here x^* is some value between 1 and m, specifying the position of the intersecting point of the x-axis and the slice. By setting the value of x^*, users can configure the position of the slice to be generated.

The slice service also involves two WSDL description files, one for the service itself, and the other for the factory service. The factory service WSDL file is similar to the data service factory's description. It describes how service resources can be created for different users. The slice service's WSDL file describes the methods provided and associated service resource properties, including:

- **Method: "setSlice"**. This method allows users to locate the slice to be created. The input parameters include two int values which are for the axis and position correspondingly. The relevant WSDL description is listed in Table 6.4.
- **Method: "link"**: This method is to link the input port of the slice service with another service's output port. As visualization services are implemented as notification services, the invoking of the 'link' method will actually request the slice data to subscribe to another service which publishes the data values to be visualized as a notification topic. As shown in Table 6.5, the 'link' method requires three input parameters: the name of target service; the output port of target service; and the name of input port.
- **Resource properties**. The resource described in the WSDL file has two properties: a resource property holding the axis which the slice is orthogonal to and the position where the intersecting point is, and a resource property containing the slice produced.

The generated slice is represented by using the 'IndexedFaceSet' node in VRML (Virtual Reality Modeling Language) [76]. An *'IndexedFaceSet'* node (with its fields listed in Table 6.6) can be rendered as a 3D shape formed by constructing a number of faces (polygons) from the vertices listed in the node. The *coord* field contains the 3D vertices referenced by the *coordIndex* field. *IndexedFaceSet* uses the indices in its *coordIndex* field to specify the polygonal faces by indexing into the vertices in the *coord* field. By setting the *colorPerVertex* field as *FALSE*, colours specified in the *color* field can be applied to each face generated. Therefore, by associating different colours with different faces, a slice can be represented in a geometric format and can be rendered as a 3D graphic in a VRML viewer.

The service firstly saves the geometric representation as a VRML file, and then stores the reference (address or URL) of the file as the resource property. The reference to the visualization result will be delivered to subscribers, every time a new slice is produced. There are two reasons to save a VRML slice representation as a physical file and publish its reference as a notification topic, rather than publishing its content as a text string in the notification topic:

```
        ...
<xsd:element name="setSlice">
        <xsd:complexType>
        <xsd:sequence>
                        <xsd:element name="position" type="xsd:int"/>
                        <xsd:element name="axis" type="xsd:int"/>
        </xsd:sequence>
        </xsd:complexType>
</xsd:element>
<xsd:element name="setSliceResponse">
        <xsd:complexType/>
</xsd:element>
        ...
<message name="SetSliceInputMessage">
        <part name="parameters" element="tns:setSlice"/>
</message>
<message name="SetSliceOutputMessage">
        <part name="parameters" element="tns:setSliceResponse"/>
</message>
        ...
<operation name="setSlice">
        <input message="tns:SetSliceInputMessage"/>
        <output message="tns:SetSliceOutputMessage"/>
</operation>
        ...
```

Table 6.4: The description of ‘setSlice’ method in Slice.wsdl file.

```
    ...
<xsd:element name="subscribe">
    <xsd:complexType>
    <xsd:sequence>
        <xsd:element name="target" type="xsd:string"/>
        <xsd:element name="port" type="xsd:string"/>
        <xsd:element name="localPort" type="xsd:string"/>
    </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="subscribeResponse">
    <xsd:complexType/>
</xsd:element>
<message name="SubscribeInputMessage">
    <part name="parameters" element="tns:subscribe"/>
</message>
<message name="SubscribeOutputMessage">
    <part name="parameters" element="tns:subscribeResponse"/>
</message>
    ...
<operation name="link">
    <input message="tns:SubscribeInputMessage"/>
    <output message="tns:SubscribeOutputMessage"/>
</operation>
    ...
```

Table 6.5: The description of 'link' method in Slice.wsdl file.

```
IndexedFaceSet {
    coord               ♯ contains all the vertices
    coordIndex          ♯ index into the list of vertices in cood
    colorPerVertex      ♯ set it as FALSE to colour each face
    color               ♯ contains the colours for each face
 }
```

Table 6.6: The fields in the VRML node - *'IndexedFaceSet'*.

- By saving a slice as a file, it offers users the flexibility to retrieve it through other mechanisms, such as HTTPS, FTP, or GridFTP, which can be more secure or more efficient.
- The 'inline' tag provided by VRML only needs the URLs of different VRML files to combine them into a single VRML file, so that clients can easily integrate slices with the VRML results generated from other visualization services in a single VRML world.

The implementation of the slice includes three components: slicing component; subscriber component; and service component.

- **Slicing component**. This component includes two Java classes: cxLattice.java and Slicesurface2.java. These two classes actually carry out the visualization process to generate slices on users' requests.
- **Subscriber component**. The subscriber component is in charge of the subscription to other services so that the slice service can obtain input data for visualization. The Java classes involved with this component are: ValuesRPListener.java which subscribes to the notification topic holding data values; and DimRPListener.java which subscribes to the notification topic containing dimension information.
- **Service component**. The service component exposes the visualization function provided by the slicing component as a notification Web service. It provides a method through which users can configure the position of the slice. The address of the generated slice is also published by this component as a notification topic, so that the slice service can be consumed by other services or clients.

6.3.3 Isosurface Service

Similar to the slice service described in Section 6.3.2, the isosurface service provides isosurface visualization functionality as a notification Web service deployed on a GT4 container.

An isosurface consists of all the points with the same value in a 3D space. Supposing the data

$$\begin{aligned} F_{ijk} = F(x_i, y_j, z_k), \quad where \quad & i = 1, ...m \\ & j = 1, ...n \\ & k = 1, ...p \end{aligned}$$

represents the samples taken on a regular mesh of some underlying function $F(x,y,z)$, then an isosurface represents the subset of points (x, y, z), for some threshold C, where

$$F(x,y,z) = C$$

Therefore, the service has a control parameter through which users can set different threshold values for generating different isosurfaces. The input required by the service is the data values to be visualized, and the output is the isosurface produced.

Two WSDL files are involved to describe the isosurface service and the service factory respectively. Iso.wsdl file describes the methods and service resource properties implemented in the isosurface service, which include:

- **"setIso" method**. It requires a real valued parameter to control the threshold of the isosurface to be produced.
- **"link" method**. This method is used for subscribing the isosurface service to a data provider service.
- **Service resource**. The generated isosurface is kept as a resource property and published as a notification service, which can be regarded as an output of the isosurface service.

The factory description file - Factory.wsdl, specifies the method to create service resources for different users. No parameter is required to invoke this method. After successful creation of a service resource, it responds with an EPR which can be used to access the resource created.

The implementation of the isosurface service also includes three components:

- Isosurface visualization component: The component produces an isosurface by using the Marching Cubes algorithm [50]. The code was written based on Bourke's [9] C version source code and Teong's MSc project [72].
- Subscriber component: It subscribes to a data provider service, so that the isosurface can receive data values to be visualized.
- Service component: The service component wraps the visualization as a notification Web service and provides the isosurface generated as a notification topic. Here, for the reasons explained for the slice service in Section 6.3.2, the isosurface created is stored as a VRML file and the notification message delivered to subscribers actually contains the address (URL) to access the VRML file.

```
♯VRML V2.0 utf8
        ...
         inline {
                url http://testgrid9.leeds.ac.uk:9090/vrml/slice967.wrl
         }
         inline {
                url http://testgrid9.leeds.ac.uk:9090/vrml/isosurface370.wrl
         }
        ...
```

Table 6.7: The snippet of the usage of ‘inline’ in a VRML file.

6.3.4 Inline Service

The inline service can integrate two VRML scenes into a single VRML world. The service has two data input ports which subscribe to two notification topics holding two VRML files correspondingly. By integrating these two VRML files, the inline service generates a new VRML file, and publishes the URL to the file as a notification topic for clients to retrieve.

The combination of two VRML files uses the ‘inline’ element in VRML. The VRML snippet listed in Table 6.7 illustrates the use of the ‘inline’ element. The snippet retrieves two VRML files by using their URLs. The retrieved VRML files, in the sample shown in Table 6.7, include a slice VRML representation and an isosurface VRML representation.

The implementation of the inline service also involves two WSDL files. The inline service description specifies a ‘link’ method for users to link it with other services which have VRML files as outputs. In the NoCoV prototype, the inline service can be connected to both the slice service and the isosurface service. By adding multiple inline services into a pipeline (shown as Figure 6.1), users can produce a complex visualization result combining multiple slices and multiple isosurfaces.

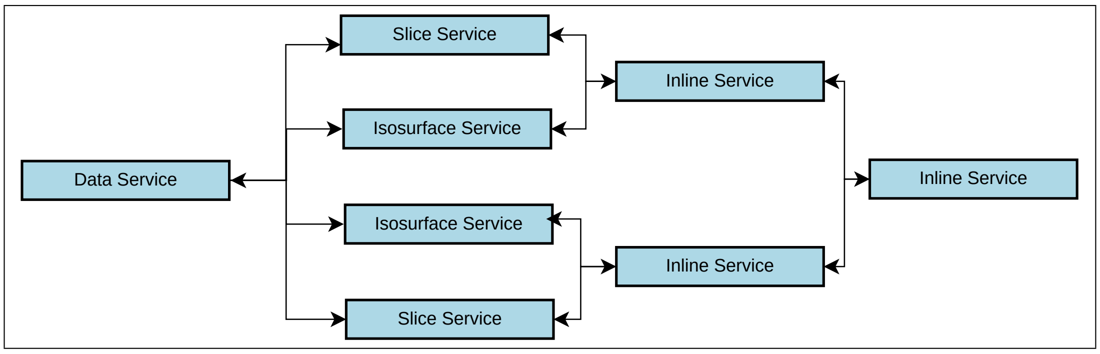


Figure 6.1: Using the inline service to produce multiple slices and isosurfaces.

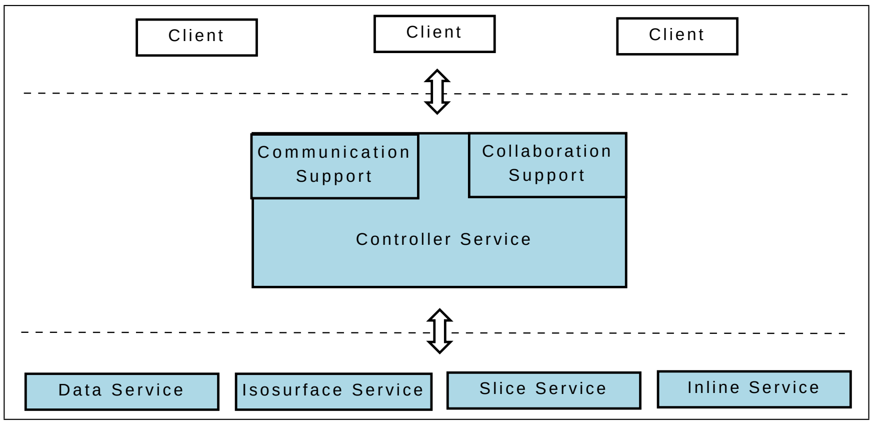


Figure 6.2: The support provided by the controller service for clients and visualization services.

6.4 Controller Service

The Controller service is placed between the end-users' clients and the visualization notification services described in Section 6.3, as shown in Figure 6.2. The implementation of the controller service satisfies the following two aspects:

- **Communication support**. The controller service acts as a proxy for the communication between clients and visualization services. The requests to manipulate visualization pipelines or visualization services are firstly sent to the controller service, and then, they are forwarded to the corresponding visualization services. It makes these distributed visualization services transparent for end-users.

- **Collaboration support**. The controller service can be regarded as a workspace for collaboration. It publishes visualization descriptions as notification topics. By subscribing to these notification topics, end-users can share visualization information and collaboratively create/steer visualization pipelines.

6.4.1 Communication Support

There is no direct connection between end-users' clients and the visualization services deployed on the Grid networks. The controller service manages the communication with these services by using the stubs created from these services' WSDL files.

In the design of the framework, the services at the management layer retrieve a list of the available visualization services and their WSDL files from a UDDI registry service. Based on these description files, the controller service can generate stubs to communicate with the visualization services. To simplify the implementation of the prototype, we assume the controller service already knows the available visualization services, and we manually create communication stubs from these services' WSDL files for the controller service.

The main methods involved with the communication support are: createInstance; link; setParam; and setEndPoint.

The 'createInstance' method. By invoking this method, the controller service can request a service factory to create a service resource for a particular service instance. As described by the snippet listed in Table 6.8, the 'createInstance' method requires two string value parameters. The 'action' string holds an XML description (listed in Table 6.9) of the user's action which, in this case, is to create a service instance. The 'action' string specifies which service factory is to be invoked and gives a name to identify the service instance. The 'xml' string contains the description of the current visualization pipeline. The description string is generated on the client, and then passed to the control service to broadcast to all the participants in the collaborative session. It can reduce the workload of XML parsing on the controller service, as the service only needs to update the notification topic with the description.

The 'link' method. This method can ask a particular visualization service to subscribe to another service's notification topic. The 'link' method also has an 'action' parameter and an 'xml' parameter. In contrast to the 'createInstance' method, the 'action' parameter specifies the output port and input port on the two services to be connected (with an example listed in Table 6.10).

The 'setParam' method. Clients can set steering parameters on visualization services by invoking the 'setParam' method on the controller service. The 'action' parameter (with an example listed in Table 6.11) required by the method specifies the visualization service to be controlled, the parameter that needs to be changed and the value to be set.

The 'setEndPoint' method. This method allows users to specify which visualization service is the endpoint of a visualization pipeline. The endpoint service's output needs to be subscribed to by the controller service, so that the visualization result can be delivered to end-users. The parameter required is a string value which specifies the name of the endpoint service (with an example listed in Table 6.12).

Further methods (e.g. 'unlink', 'cancelEndpoint') have not been implemented within the prototype, but would be straightforward extensions.

```
<xsd:element name="createInstance" >
    <xsd:complexType>
    <xsd:sequence>
        <xsd:element name="action" type="xsd:string"/>
        <xsd:element name="xml" type="xsd:string"/>
    </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="createInstanceResponse">
    <xsd:complexType/>
</xsd:element>
    ...
<message name="CreateInstanceInputMessage">
    <part name="parameters" element="tns:createInstance"/>
</message>
<message name="CreateInstanceOutputMessage">
    <part name="parameters" element="tns:createInstanceResponse"/>
</message>
    ...
<operation name="createInstance">
    <input message="tns:CreateInstanceInputMessage"/>
    <output message="tns:CreateInstanceOutputMessage"/>
</operation>
```

Table 6.8: The WSDL description of the ‘createInstance’ method.

```
<action type = "createInstance">
    <instanceID>MyIso</instanceID>
    <factoryID>IsoFactory</factoryID>
</action>
```

Table 6.9: The XML description for the action of creating an instance called ‘MyIso’ from the ‘IsoFactory’ factory.

```
<action type = "link">
    <target>MyData</target>
    <output>valuesOut</output>
    <client>MyIso</client>
    <input>valuesIn</input>
</action>
```

Table 6.10: The XML description of a ‘link’ request to connect the input port ‘valuesIn’ on ‘MyIso’ service with the output port ‘valuesOut’ on ‘MyData’ service.

```
<action type = 'setParam'>
<instance>MyIso</instance>
<param>
<name>Threshold</name>
<value>0.75</value>
</param>
</action>
```

Table 6.11: The XML description of a 'setParam' request to set the 'Threshold' values on 'MyIso' service as 0.75.

```
<action type = 'setEndpoint'>
      <endpoint>MyIso</endpoint>
</action>
```

Table 6.12: An example of the XML description of the action to set an endpoint of an existing pipeline.

6.4.2 Collaboration Support

The support of collaboration is achieved by the use of notification. The visualization information to be shared among different collaborators is published as notification topics. After subscribing to these notification topics, each collaborator will be synchronized with the latest visualization information, every time the visualization changes.

The notification topics published by the controller service include:

- **XMLResource**: This topic contains a full description of a visualization pipeline. The description includes which services are involved, how these services are connected, and how their control parameters are set. With this full description, on the one hand, the visualization pipeline can be visually represented on client GUIs; on the other hand, the widgets, through which users can set parameters in visualization services, can be generated on client GUIs. By subscribing to this XMLResource notification topic, the visualization pipeline and its parameter settings displayed at all the client GUIs can be synchronized.

- **ChangedParamResource**: The latest change in a service parameter is published as a 'ChangedParamResource' topic. By subscribing to this notification topic, when a participant modifies the parameter values, the corresponding control widgets in all the participants' GUIs can be updated.

- **ChangedPipelineResource**: The latest change of visualization pipeline is published as a 'ChangedPipelineResource'. These changes include: the change of connections between service and the adding of services. When a service is added into a pipeline, its control widgets can be created on subscribers' GUIs automatically. When the change of connections happens, the pipeline visual representation on subscribers' GUIs will be updated as well.

- **ResultResource**: The 'ResultResource' holds the final visualization result generated from the visualization pipeline. By subscribing to the 'ResultResource' topic, the participants can be updated with the latest visualization result generated. It also helps them to modify the pipeline connections or steering settings to achieve enhanced visualization.

6.4.3 Summary of the Controller Service

The controller service aims to provide transparent access to visualization services distributed on the Grid networks. It also enables collaboration in creating and steering a visualization pipeline amongst different users.

The overview of the implementation of the controller service can be represented as Figure 6.3. As the requests from end-users are described in XML representations, a set of XML parsers are used to interpret these messages. After the requests are parsed, the communication methods will be invoked to communicate with corresponding visualization services by using their communication stubs. When the requests are processed successfully, the controller service updates relevant notification topics, so that all the subscribing clients can be notified with the changes.

6.5 NoCoV Clients

As designed in the NoCoV framework, the system aims to provide two types of client GUI for end-users who have different background knowledge:

- A client GUI (named as pipeline editor client) for visualization experts, who have the knowledge and skill to create visualization pipelines.
- A client GUI (named as parameter control client) for application scientists, who are more concerned with using existing pre-defined pipelines, and steering these pipelines in order to explore their data.

Moreover, the two GUIs can be used together, for a user who wishes to both create and modify pipelines, and execute these pipelines with different parameter settings.

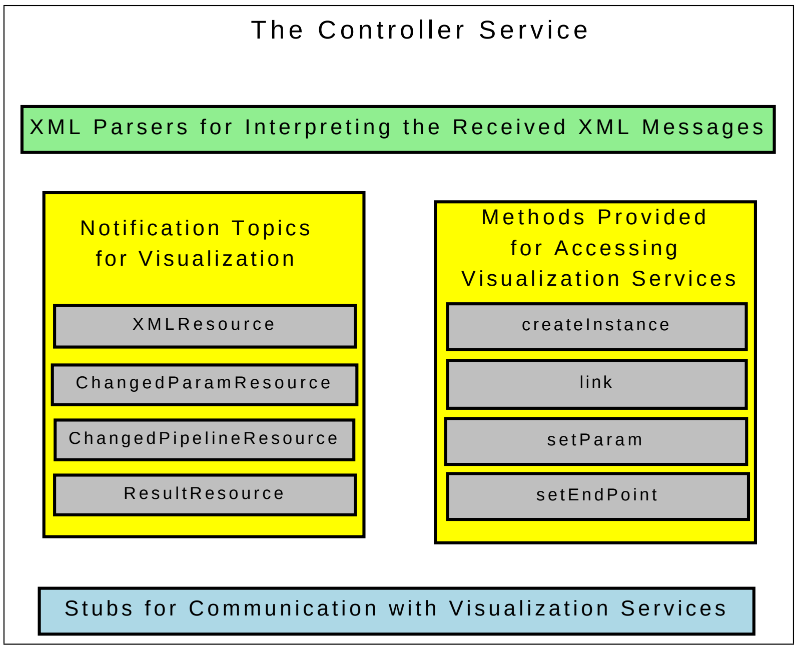

The Controller Service
XML Parsers for Interpreting the Received XML Messages
Notification Topics for Visualization
XMLResource
ChangedParamResource
ChangedPipelineResource
ResultResource
Methods Provided for Accessing Visualization Services
createInstance
link
setParam
setEndPoint
Stubs for Communication with Visualization Services

Figure 6.3: The overview of the controller service's implementation.

End-users can collaboratively create or steer a visualization by running different clients and joining the same collaborative session. The collaboration is also supported across these two types of clients. It allows visualization experts to jointly create pipelines and provides the parameter control GUI for application scientists to jointly control the steering parameters to achieve desired visualization result.

6.5.1 Pipeline Editor Client

The pipeline editor client (PEC) provides a GUI for end-users, so that they can visually launch visualization services and connect them as a visualization pipeline.

As shown in Figure 6.4, the implementation of the PEC involves the following components:

- The GUI manager. It initializes a GUI which includes a list of available services, a visual pipeline editor window and a VRML viewer. The GUI component also displays a visualization pipeline as a visual representation, and supports interactions between end-users and the pipeline.

- The XML parser. Firstly, it converts a visualization pipeline from its XML description into a visual display. Secondly, the XML parser also generates XML descriptions of end-users' interactions with the visualization pipeline, such as adding services, connecting services, etc..

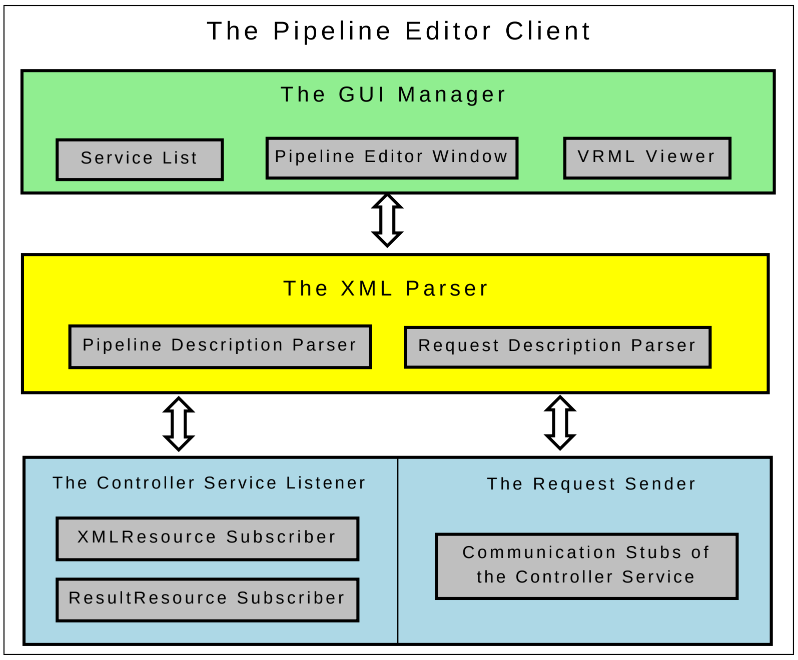

The Pipeline Editor Client
The GUI Manager
Service List
Pipeline Editor Window
VRML Viewer
The XML Parser
Pipeline Description Parser
Request Description Parser
The Controller Service Listener
XMLResource Subscriber
ResultResource Subscriber
The Request Sender
Communication Stubs of the Controller Service

Figure 6.4: The implementation of the PEC.

- The request sender. Using the XML descriptions created by the XML parser, the request sender component sends end-users' requests to the controller service using the stubs generated from its WSDL file.

- The controller service listener. It subscribes to the notification topics (XMLResource and ResultResource) published on the controller service. The XMLResource delivers the latest visualization pipeline description, when collaborators make any changes to it. The ResultResource can send the PEC the latest visualization result generated from the endpoint of the visualization pipeline.

The screenshot of the PEC is displayed as Figure 6.5. The service list displayed on the left hand side is initialized from an XML format service list file (snippet listed in Table 6.13), which makes it possible to retrieve a list of available visualization services from a UDDI registry in a future implementation. By clicking on the services listed, a corresponding service instance will be launched and displayed as a box in the editor window (at the top-right area on the GUI). By right clicking on the instance displayed in the editor window, users can specify the output or input to that instance on the popped up window (shown as Figure 6.6), to connect different services together to create a visualization pipeline such as the one displayed in Figure 6.7. In this prototype, we simply assume end-users know the notification topics published by each service, but in our future work, each visualization service will provide a method which returns a description of notifica-

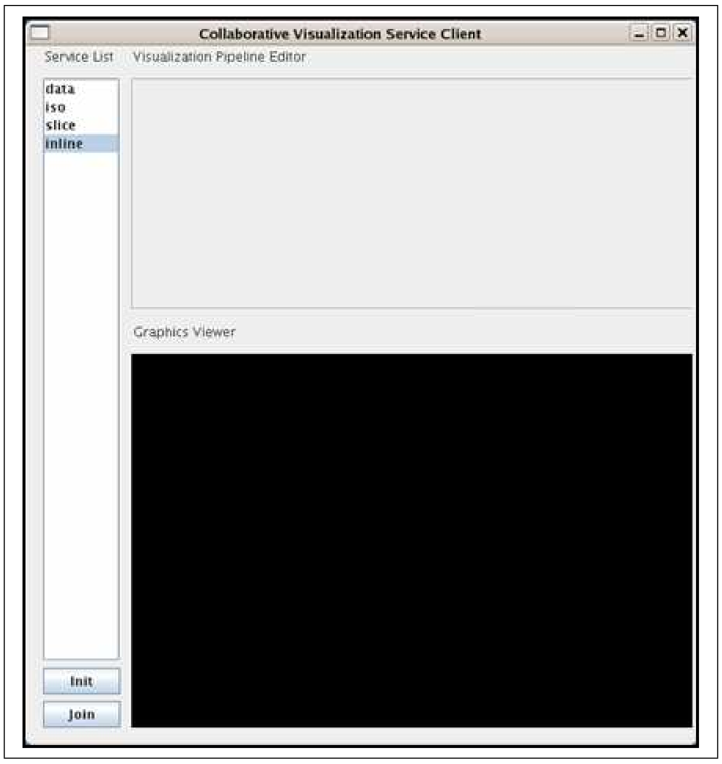
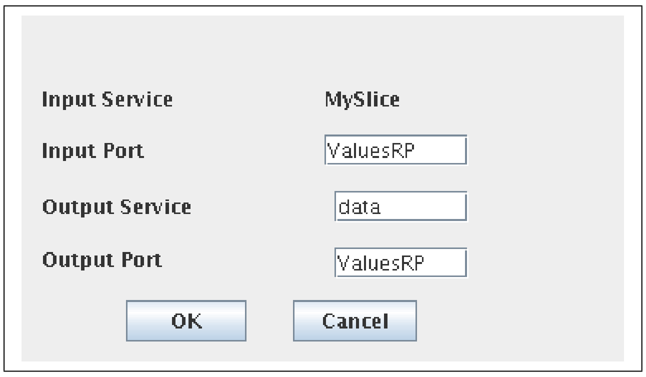

Collaborative Visualization Service Client
Service List
Visualization Pipeline Editor
data
iso
slice
inline
Graphics Viewer
Init
Join

Figure 6.5: Screenshot of the PEC GUI.

tion topics published for clients to subscribe to. Therefore, it will be possible, in the future development, for the text fields of input and output ports in Figure 6.6 to be replaced with a drop-down list with all the available notification topics.

All the participants in the collaboration will see the same pipeline displayed on their editor windows, as they subscribe to the same notification topic on the control service - the XML format description of the current pipeline.



Input Service	MySlice
Input Port	ValuesRP
Output Service	data
Output Port	ValuesRP

Figure 6.6: The GUI to set input and output ports for connecting different services.

Figure 6.7 displays a visualization pipeline composed of a data service providing raw data, a slice service generating a slice, an isosurface creating an isosurface and an in-

```
<factories>
      ...
      <factory>
            <id>iso</id>
            <address>http://129.11.144.73:9090/wsrf/
                  services/coviz/iso/IsoFactoryServic</address>
            <output>VizFileRP</output>
            <input>ValuesRP</input>
            <input>DimRP</input>
            <param>
                  <name>isoValue</name>
                  <value>0.78</value>
                  <type>double</type>
                  <range>
                        <max>1.0</max>
                        <min>0.0</min>
                  </range>
            </param>
      </factory>
      ...
</factories>
```

Table 6.13: The XML description of a service named as 'iso'.

line service combining both slice and isosurface. The generated visualization result is rendered in the VRML viewer at the bottom-right area on the client GUI.

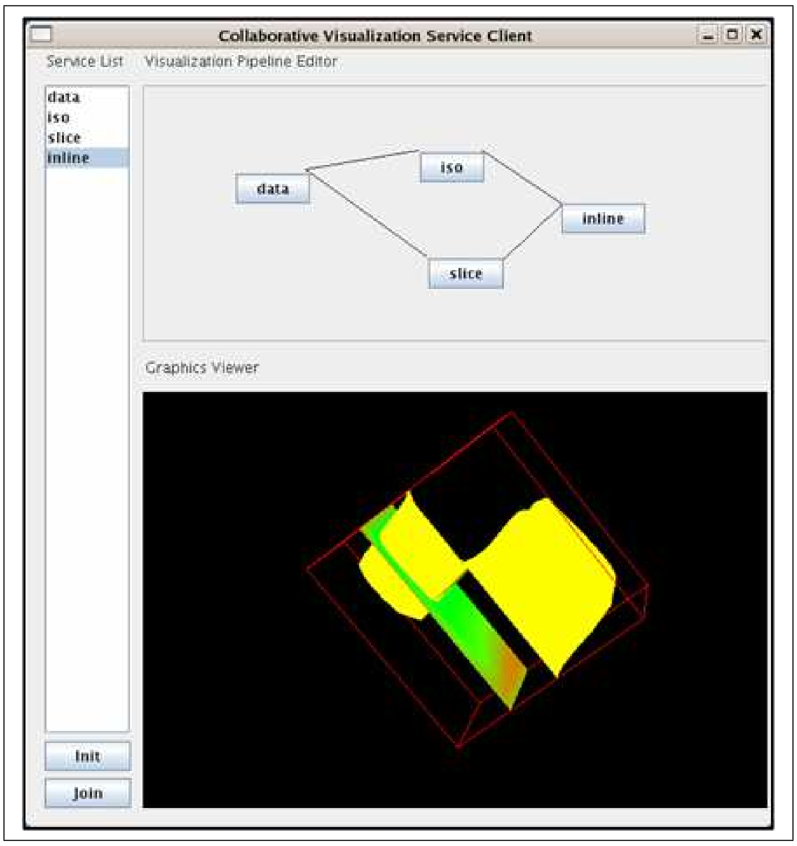

Collaborative Visualization Service Client
Service List
Visualization Pipeline Editor
data
iso
slice
inline
Graphics Viewer
Init
Join

Figure 6.7: The screenshot of an example pipeline created from the client GUI, with the generated visualization result displayed on the GUI.

6.5.2 Parameter control client

The parameter control client (PCC) provides a GUI for setting control parameters on the services wired in a pipeline to steer the visualization. The client GUI consists of a set of widgets initialized from the description about the services used to build the pipeline. Through these widgets, users can change the parameters on the visualization services to steer the pipeline.

Figure 6.8 illustrates the overview of the PCC's implementation, which includes following parts:

- **e-Viz library**. The implementation of the PCC used a Java library produced by the e-Viz project [12,66]. The e-Viz library can take an XML description of a visualization pipeline and generate a client GUI for each component based on its description.

By using an XML description of parameter changes, the e-Viz library can update the existing widgets with the new values. Vice versa, the changes made by users from the GUI are received by the e-Viz library and turned into XML descriptions.

- **XML interpreter**. As the XML description required by the e-Viz library is different from the NoCoV descriptions (different in the name of tags and the format of representation), an XML interpreter component is added to convert these two types of descriptions.
- **Connection component**. There are two kinds of connections between the PCC and the controller service: 1) request/response communication which is used to get the existing pipeline description for initializing the GUI and to send the changes of parameters made through widgets to the controller service; and 2) subscription/notification communication which is used for sharing the update of visualization information amongst all the participants.

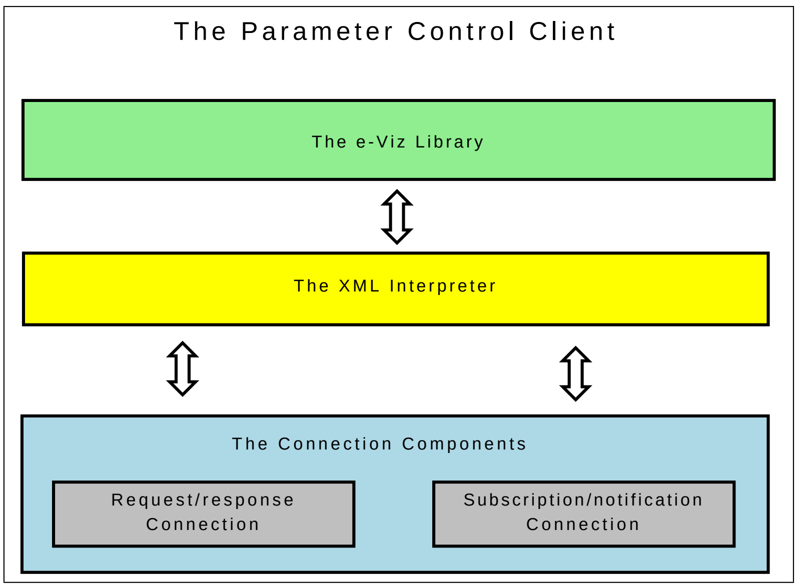

The Parameter Control Client
The e-Viz Library
The XML Interpreter
The Connection Components
Request/response Connection
Subscription/notification Connection

Figure 6.8: The overview of the implementation of the PCC.

Figure 6.9 shows the tabs generated on the PCC GUI based on the visualization pipeline displayed in Figure 6.7. Each tab corresponds to a service used in the visualization pipeline. By subscribing to the same 'ChangedParamResource' notification topics published by the controller service, the parameter values displayed on every PCC GUI can be synchronized. By subscribing to the same 'ChangedPipelineResource' notification topic, the collaboration can be supported between the PEC and the PCC. In a collaborative session, when a visualization expert adds a service into the existing visualization pipeline through the PEC GUI, there will be a corresponding tab created on other participants' PCC GUI with the new service's control widgets.

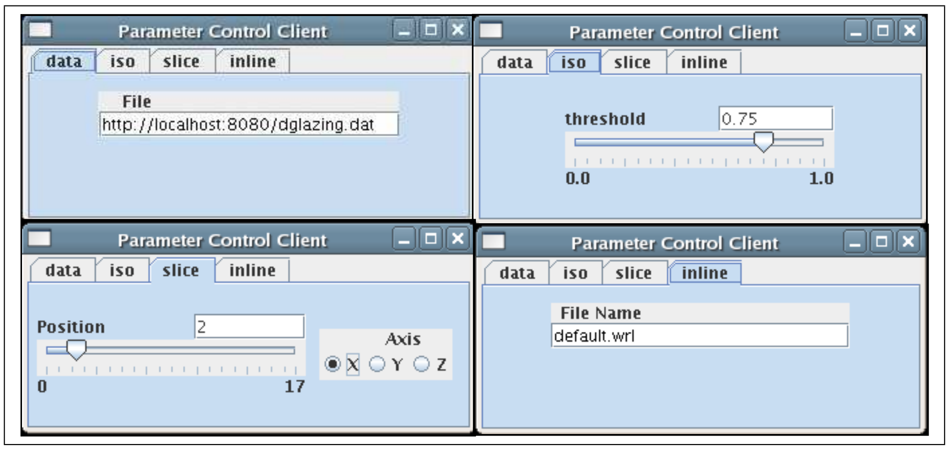


Figure 6.9: The tabs generated on the PCC GUI corresponding to the visualization pipeline displayed in Figure 6.7.

There is a stand-alone VRML viewer developed by using Xj3D library [89] for application scientists who use the PCC (shown as Figure 6.10). The stand-alone VRML viewer subscribes to the 'ResultResource' notification topic on the controller service, so that application scientists can view the latest visualization result, while they change the control parameters through the PCC GUI.

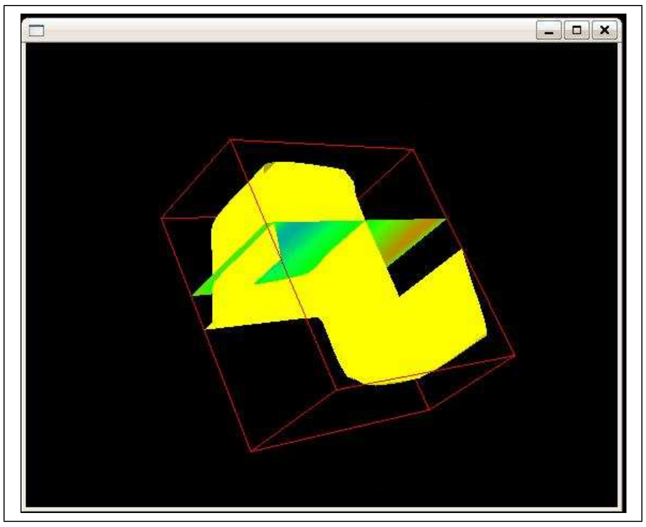


Figure 6.10: The stand-alone VRML Viewer for PCC users.

6.6 A Simple Illustration of the NoCoV Prototype

In this subsection, we illustrate the use of the NoCoV prototype with a very simple example. The dataset to be visualized is called 'double glazing' data (dglazing.dat) which records the distribution of temperatures in a 3D space between two parallel glazing panels which are hot (the maximum temperature in the 3D space) and cold (the minimum temperature in the 3D space) respectively.

There are four users involved in this specially created scenario: two visualization experts (V1 and V2) who run PECs, and two application scientists (A1 and A2) who run PCCs.

Figure 6.11 illustrates the steps that happen in the collaboration, with the shaded boxes representing the actions of the system, and the white boxes representing the actions from users. During the first stage, after the collaborative session has been initiated, by using a PEC, a visualization expert (V1) builds a pipeline of two services to visualize the volumetric dataset (*data* - to read in the raw data, and *slice* to display a cross-section), shown as Figure 6.12.

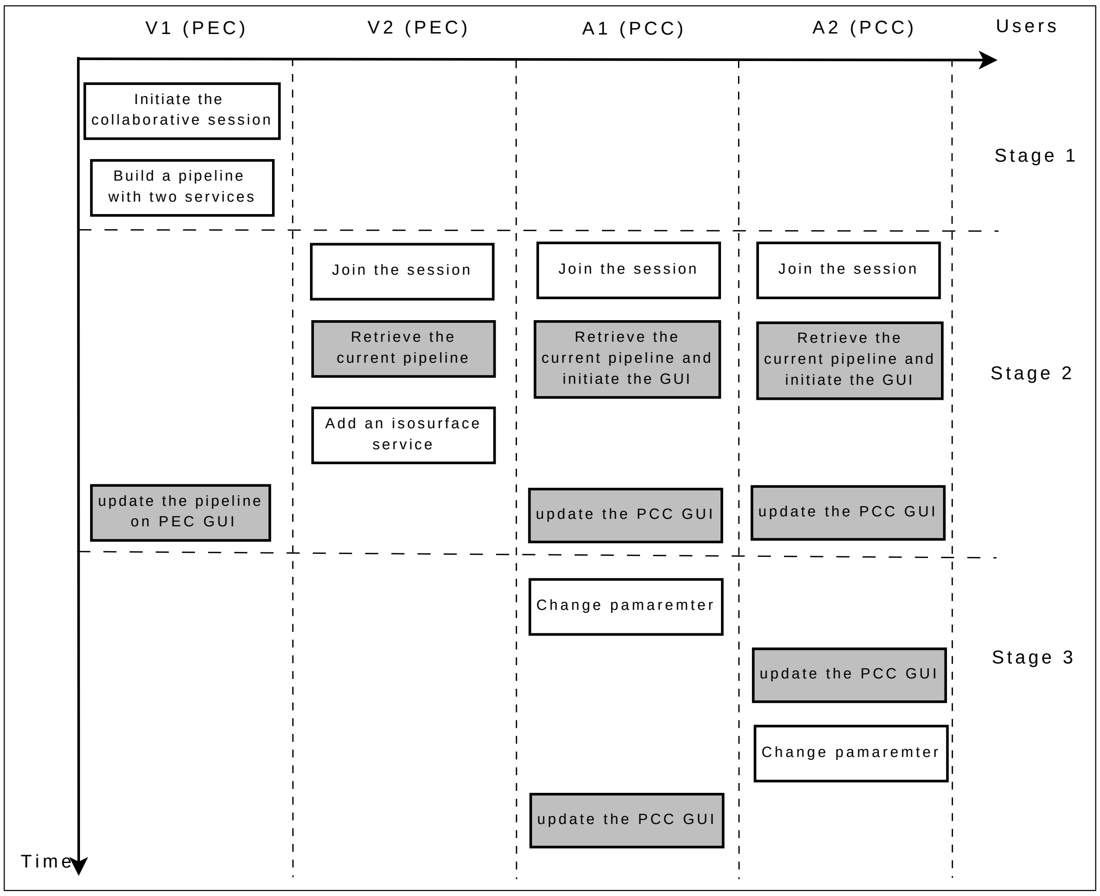

V1 (PEC)
V2 (PEC)
A1 (PCC)
A2 (PCC)
Users
Initiate the collaborative session
Build a pipeline with two services
Stage 1
Join the session
Join the session
Join the session
Retrieve the current pipeline
Retrieve the current pipeline and initiate the GUI
Retrieve the current pipeline and initiate the GUI
Stage 2
Add an isosurface service
update the pipeline on PEC GUI
update the PCC GUI
update the PCC GUI
Change pamaremter
Stage 3
update the PCC GUI
Change pamaremter
update the PCC GUI
Time

Figure 6.11: The outline of the collaborative scenario for the NoCoV prototype.

The other three collaborators (V2, A1 and A2) then join the session, in the second stage. They firstly retrieve the description of the current pipeline created by V1. The description will be visually displayed on V2's PEC GUI. For the application scientists, a PCC GUI will be created for each of them using the retrieved pipeline description. After investigating the existing pipeline, V2 thinks that the addition of an isosurface would enhance the understanding of the dataset. Through his PEC GUI, V2 alters the pipeline by adding an isosurface and an inline service. Figure 6.13 and Figure 6.14 shows the participants' GUIs at the end of the stage 2 in the collaboration.

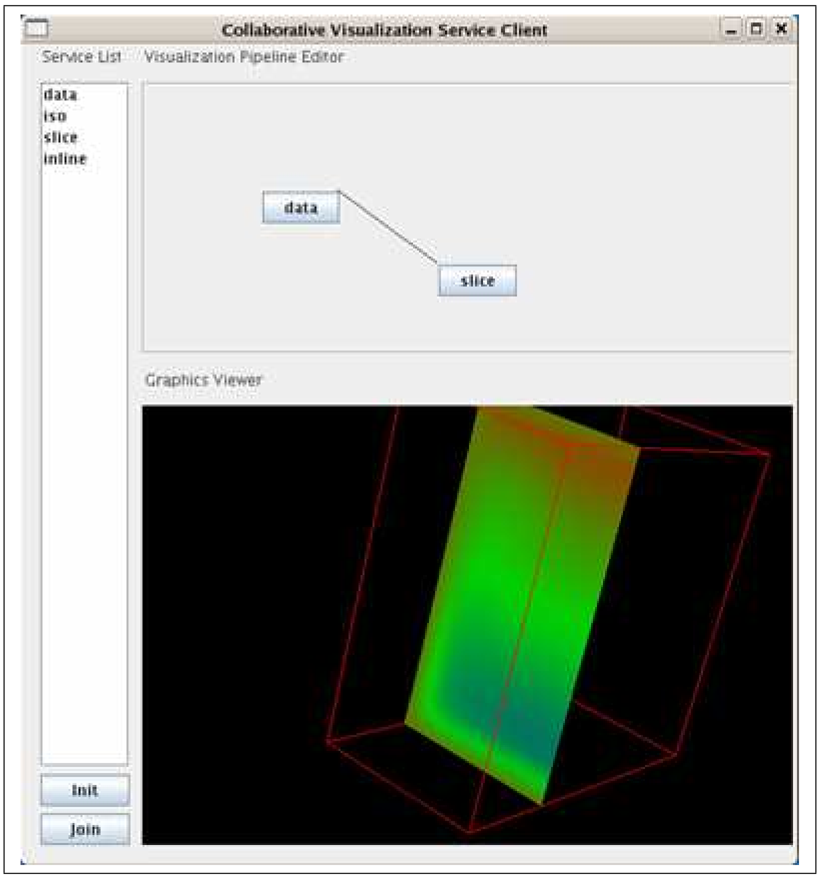

Collaborative Visualization Service Client
Service List
Visualization Pipeline Editor
data
iso
slice
inline
data
slice
Graphics Viewer
Init
Join

Figure 6.12: The PEC GUI at the end of the stage 1 in the collaboration.

Until this stage, the visualization services used in the pipeline only use the default parameter values. As application scientists, A1 knows what value the isosurface threshold needs to be set to improve the visualization, and A2 knows which cross-section needs to be investigated. In the third stage, A1 and A2 jointly change the control parameters for the isosurface service and the slice service to generate the visualization result they want. Figure 6.15 shows how the application scientists change the control parameters through PCC GUIs. Figure 6.16 shows a PEC GUI at the end of the third stage in Figure 6.11.

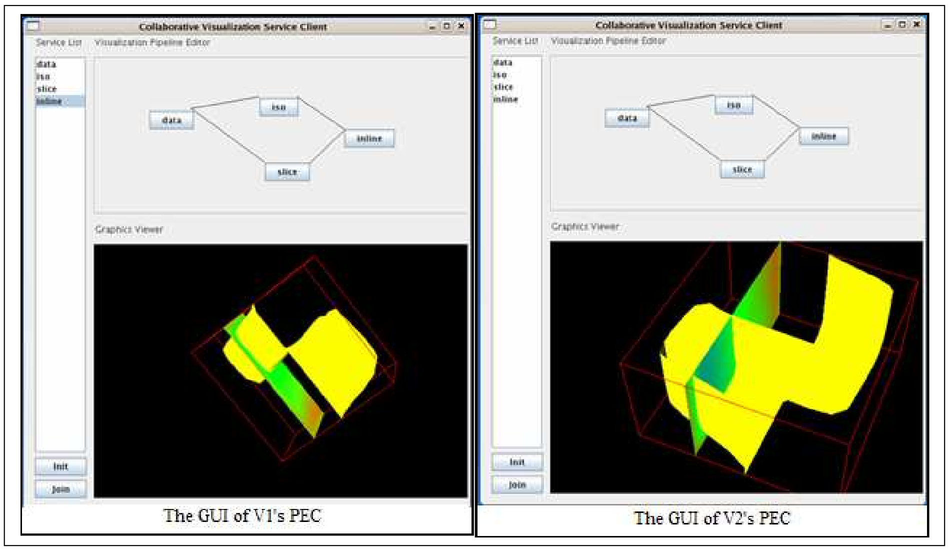


Figure 6.13: The two visualization experts' PEC GUIs at the end of the stage 2 in the collaboration.

The pipeline displayed is the same as the pipeline created in stage 2, but the visualization result is different due to the parameter changes.

To avoid situations where more than one collaborator wishes to interact at the same time, the NoCoV prototype uses a 'social' floor control policy in conjunction with video/audio conference applications for synchronous collaboration. In the future work, a 'token-based' floor control policy to support smooth interactions between participants could be added.

Through the above three stages, these four participants jointly create a visualization by using different clients and taking different roles in the collaboration. The collaboration that occurs in the scenario consists of three aspects:

- The collaboration between the two visualization experts in creating the visualization pipeline through their PEC GUIs.
- The collaboration between visualization experts and application scientists. The application scientists' PCC GUIs are initialized based on the pipeline created by the visualization experts.
- The collaboration between application scientists in steering the pipeline to produce the desired visualization result.

The characteristics of asynchronous collaboration can be reflected, if we change the above scenario slightly. As shown in Figure 6.17, the significant difference to the collaborative scenario displayed in Figure 6.11 is that there is only a single user working on

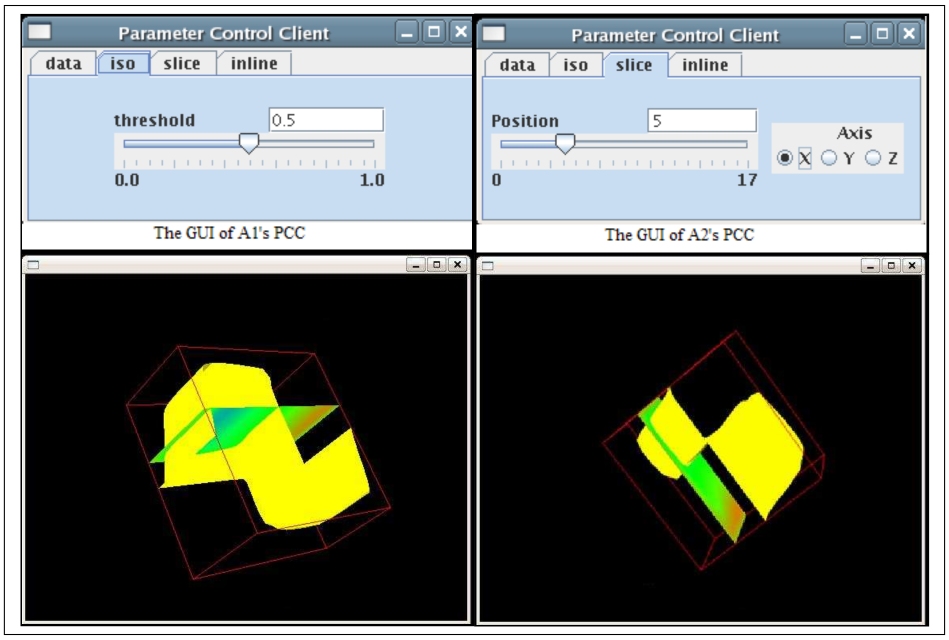


Figure 6.14: The two application scientists' PCC GUIs and their stand-alone VRML viewers at the end of the stage 2 in the collaboration.

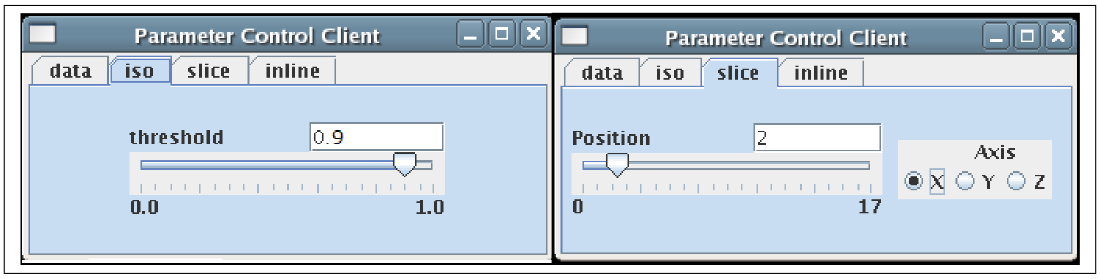


Figure 6.15: The setting of control parameters through PCC GUIs in stage 3.

the visualization at each stage, however the two scenarios produce the same visualization result.

Asynchronous collaboration offers participants freedom to work at different times. The NoCoV prototype currently only supports users to continue from the latest visualization generated and stored in the controller service. It would be more sophisticated if the system allowed users to explore the historical provenance of the visualization and choose from which point they want to continue their work. Some relevant work in this area has been done in the GRASPARC project [11] and VisTrails [35]. We also plan to add the feature of visualization provenance in NoCoV in the future development.

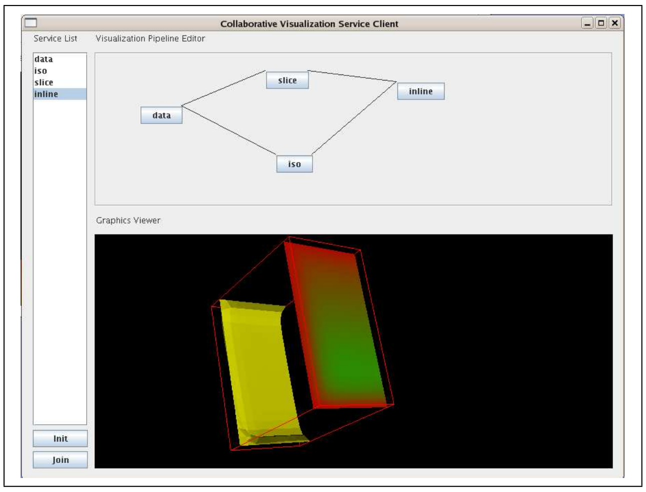

Collaborative Visualization Service Client
Service List
Visualization Pipeline Editor
data
iso
slice
inline
Graphics Viewer
Init
Join

Figure 6.16: The PEC GUI with the updated visualization result at the end of stage 3.

6.7 Summary

In this chapter, a NoCoV prototype is implemented following the framework described in Chapter 5.

The prototype consists of a set of notification Web services built with GT4 and providing different visualization functionalities, a controller service which is also a notification Web service and two kinds of client for editing a pipeline and steering a pipeline respectively. By deploying the visualization services in the Grid environment, the prototype allows end-users to collaboratively create and control Grid-enabled visualization pipelines remotely from their desktops.

The following features of the NoCoV framework have been demonstrated by this prototype:

- Notification-service-based visualization pipeline, in which the services are linked by the subscription/notification communication mode;
- Composing visualization services distributed in Grid networks as a pipeline through a visual editor client;
- Steering distributed Grid-based pipelines from the desktop by setting control parameters used by these services;
- Enabling collaboration in both creating a pipeline and steering the visualization;
- Accommodating users with different expertise and skills;

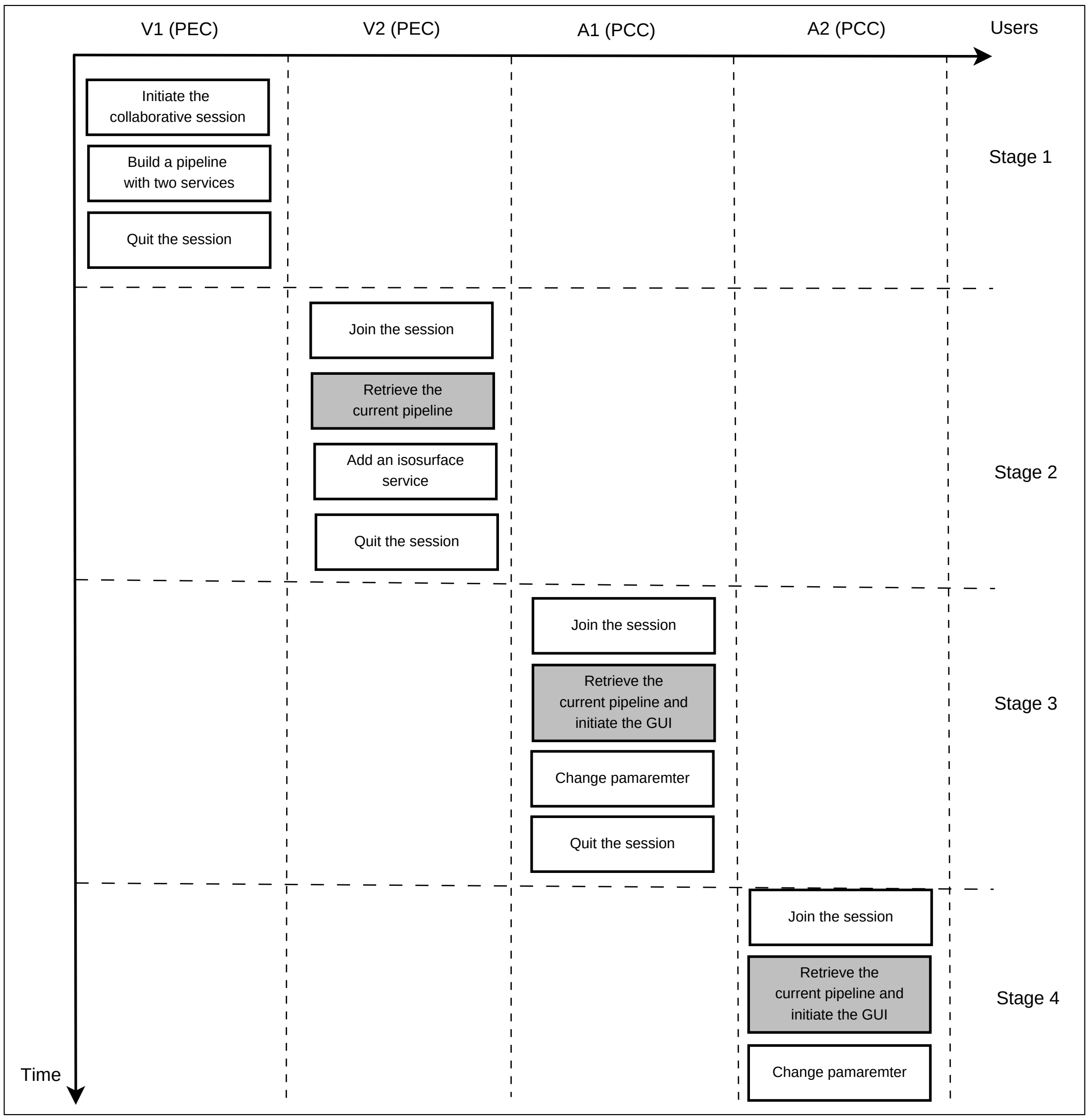

V1 (PEC)
V2 (PEC)
A1 (PCC)
A2 (PCC)
Users
Initiate the collaborative session
Build a pipeline with two services
Quit the session
Stage 1
Join the session
Retrieve the current pipeline
Add an isosurface service
Quit the session
Stage 2
Join the session
Retrieve the current pipeline and initiate the GUI
Change pamaremter
Quit the session
Stage 3
Join the session
Retrieve the current pipeline and initiate the GUI
Change pamaremter
Stage 4
Time

Figure 6.17: The outline of the asynchronous collaborative scenario.

- Recording visualization pipeline information in a formal description for sharing amongst participants.

Chapter 7

Technical Evaluation

This chapter evaluates the NoCoV system by using different sizes of datasets and distributing the system in different ways. The comparison of the features supported by the NoCoV system and other collaborative systems is also laid out at the end of this chapter.

7.1 Performance of Visualization Services

In this section, we evaluate the performance of the NoCoV system on a Grid machine. The computing environment of that Grid machine is listed in Table 7.1. We evaluate both versions of the prototype mentioned in Section 6.3. The first wraps the data values within the notification message when linking two services; the second simply places a reference to the data values within the message, and uses an alternative mechanism such as HTTP to transmit the data.

	balrog.vipl.mcc.ac.uk
Location	Manchester
CPU	4 x (AMD Athlon 64 2.6GHz)
Memory	4.0 GB
OS	Ubuntu Linux 7.04
Grid Middleware	Globus Toolkit 4.0

Table 7.1: The system information of the machine balrog.vipl.mcc.ac.uk.

Dataset	Resolution	Points	Size(bytes)
dglazing1.dat	18x18x10	3240	48,609
dglazing2.dat	35x35x19	23275	286,939
dglazing3.dat	52x52x28	75712	1,307,942
dglazing4.dat	69x69x37	176157	2,560,291

Table 7.2: The ‘double glazing’ datasets used for the evaluation.

7.1.1 Design of the Test

The volumetric data used for the evaluation is called ‘double glazing’ data. The data has been described in Section 6.6. Table 7.2 lists the four ‘double glazing’ datasets used for this evaluation.

We used the data service to feed these four datasets into the visualization pipelines and visualized them by using the isosurface service and the slice service respectively. The performance data we recorded was the time cost for the communications between services and the time for producing visualization geometries. The experiments have been repeated several times, and the averages are recorded in the tables and charts listed in this chapter for evaluation analysis. It is notable that the standard deviations were very small.

7.1.2 Passing Data Values in Notification Message

A visualization pipeline is built, shown as Figure 7.1, to test the performance of the isosurface service. Here, the data service reads in a ‘double glazing’ dataset and sends the values as a double array in the notification message to the isosurface service.

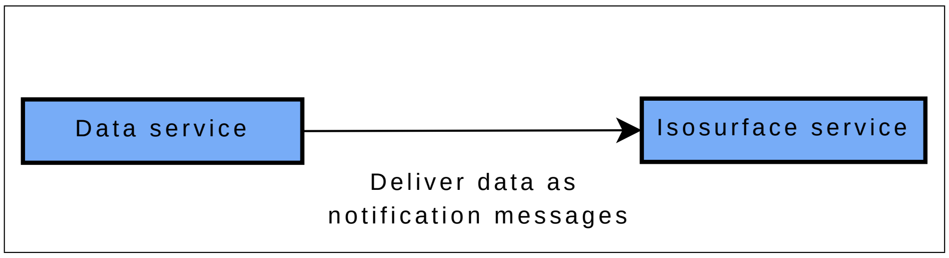


Figure 7.1: Pipeline for isosurface visualization with passing data in notification message.

We recorded the time cost for processing the notification message communication between the data service and the isosurface service (indicated as ‘Time D→I’ in Table 7.3), and the time cost by the isosurface service to generate the visualization result (indicated as ‘Time I’ in Table 7.3).

When the performance figures are drawn as line charts, we find as expected the time cost for notification message communications increases approximately linearly with the

Dataset	Resolution	Points	Time D→I (ms)	Time I (ms)
dglazing1.dat	18x18x10	3240	113	726
dglazing2.dat	35x35x19	23275	897	7553
dglazing3.dat	52x52x28	75712	2455	52,817
dglazing4.dat	69x69x37	176157	4516	298,039

Table 7.3: The performance of the isosurface service.

increase of the dataset size (shown in Figure 7.2). However the time cost of the isosurface service increases exponentially with the increase of the dataset size (shown in Figure 7.3).

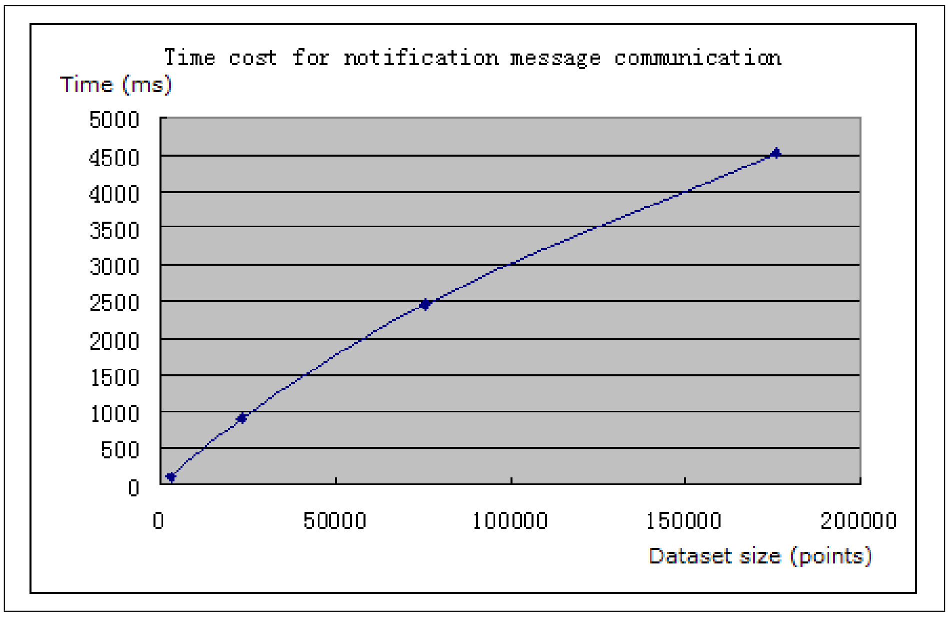


Figure 7.2: Line chart of the time cost of notification message communication between the data service and the isosurface service, when data delivered as notification messages.

To test the performance of the slice service, the isosurface in Figure 7.1 was replaced by a slice service (displayed in Figure 7.4). We also recorded the time cost for communications between two services (indicated as 'Time D→S' in Table 7.3) and the time cost for generating slice (indicated as 'Time S' in Table 7.4). We again aimed to find out how the performance would change when the size of the dataset changed.

Similar to what we found from the evaluation of the isosurface service, the change of the time cost for communications between the two services remains linear, and the performance of the slice service drops significantly when the dataset increases(see Figure 7.5 and Figure 7.6).

The linear increase in the time for transferring notification messages is expected, as the size of notification messages increases linearly. However, the exponential increase of

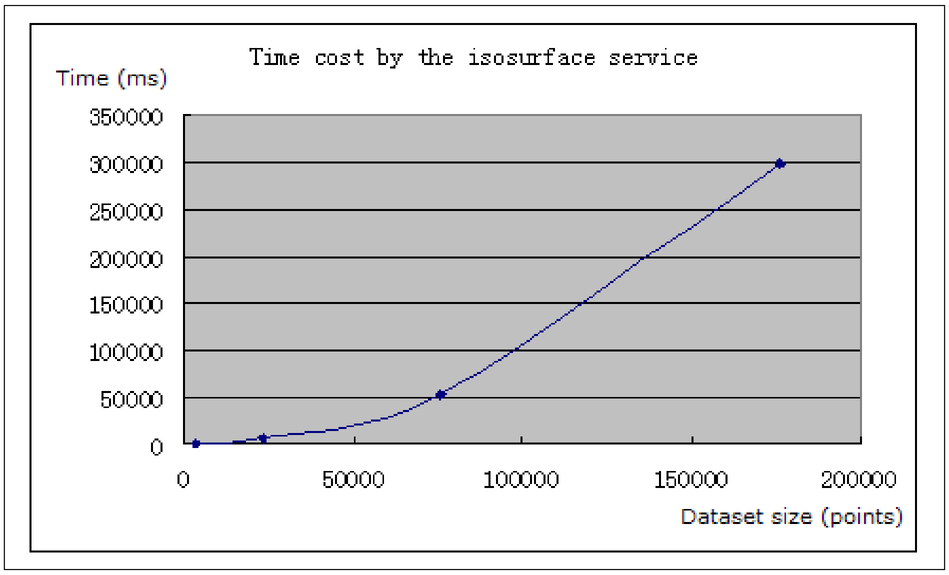


Figure 7.3: Line chart of the time cost by the isosurface service, when data delivered as notification messages.

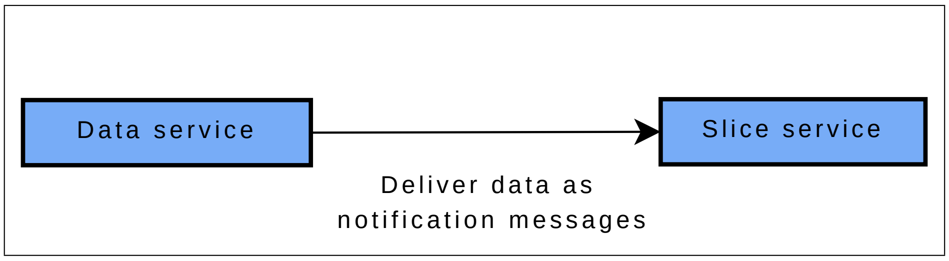


Figure 7.4: Pipeline for slice visualization with passing data in notification message.

the time cost by the slice service and the isosurface services causes the significant drop of the performance of the NoCoV system when it handles larger datasets.

In theory, the rate of increase in the cost of generating the slice or isosurface geometries should be linear (or even sub-linear in the case of the 2D slice). So, we suspect extra time cost can be due to the parsing of the notification messages received at the slice service and the isosurface service. When the large datasets are visualized, as the data values are wrapped in the notification messages, the visualization services have to parse these

Dataset	Resolution	Points	Time D→S (ms)	Time S (ms)
dglazing1.dat	18x18x10	3240	121	312
dglazing2.dat	35x35x19	23275	873	4762
dglazing3.dat	52x52x28	75712	2412	39,143
dglazing4.dat	69x69x37	176157	4498	247,219

Table 7.4: The performance of the slice service.

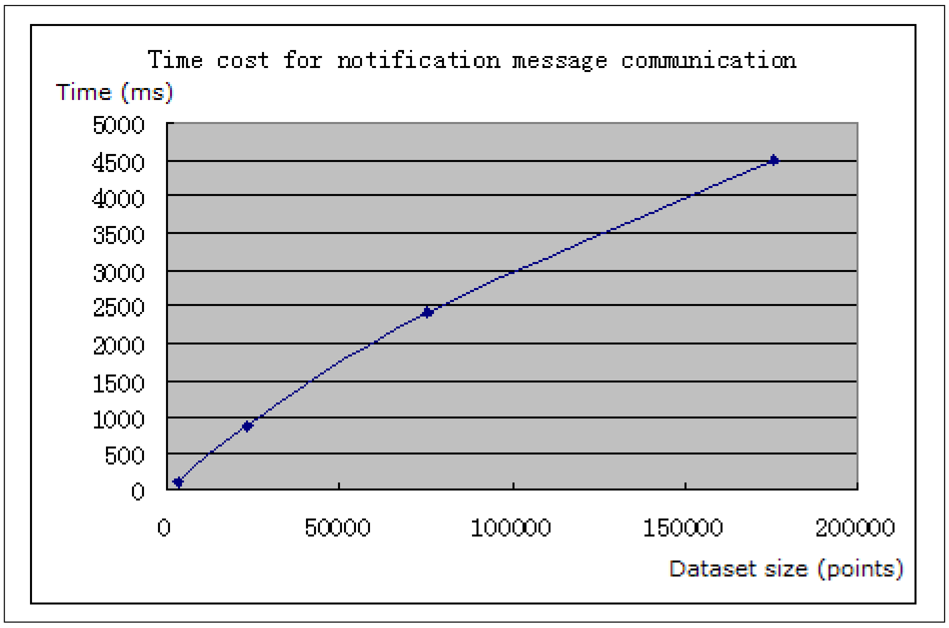


Figure 7.5: Line chart of the time cost for notification message communications between the data service and the slice service, when data delivered as notification messages.

large XML messages, which can be extremely time consuming.

To enhance the performance of the NoCoV system, another NoCoV prototype was implemented. In the second prototype, instead of passing all data values as notification messages, services pass the reference to the data in notification messages.

7.1.3 Passing Data Reference in Notification Message

In this section, we evaluate the NoCoV prototype which passes a data reference in notification messages, and compare its performance against the version which passes data values.

When the visualization services receive the notification message with a reference to the data input, the parsing can be much faster than when dealing with a large XML message. Now the extra time caused is the time for retrieving the input data across the network. Assuming there is good network connection available, this extra time can be relatively small.

We used the pipelines displayed in Figure 7.7 and Figure 7.8 to evaluate the performance of the revised isosurface service and slice service. The evaluation figures are listed in Table 7.5 and Table 7.6. The time cost for notification message communication (D→I and D→S) is constant and small (around 33ms), as the reference to the data is the same length and these notification messages are quite small in size. There is also a dramatical

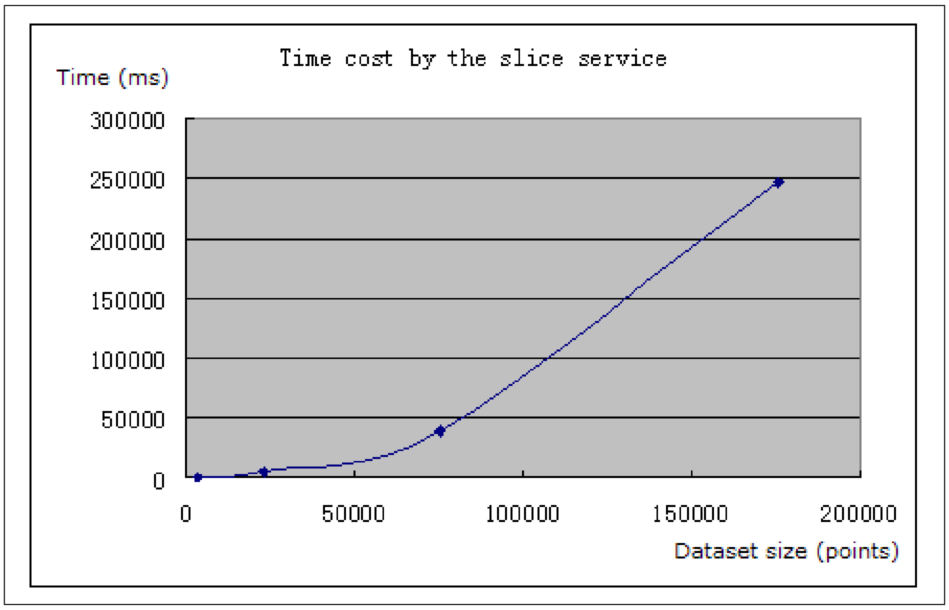


Figure 7.6: Line chart of the time cost by the slice service, when data delivered as notification messages.

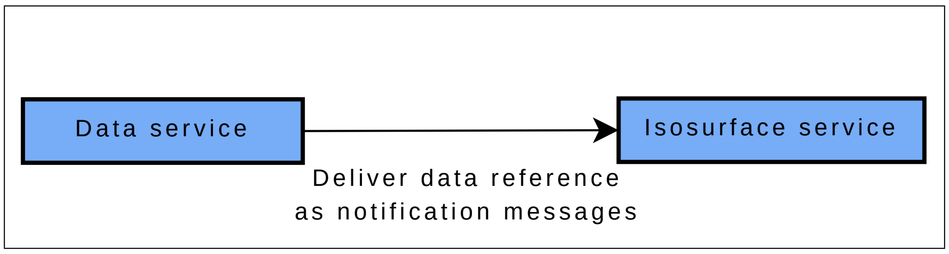


Figure 7.7: Pipeline for isosurface visualization with passing data reference in notification message.

drop in the time cost of the visualization services even allowing for the retrieval of the data. The comparison of the time cost in two versions of the NoCoV systems is shown in Figure 7.9 and Figure 7.10. Indicated by these two line charts, the changes in the performance of the two visualization services are approximately linear in the version where the reference is passed, but exponential where the data is passed.

7.1.4 Summary

Through the evaluation of the visualization services, we find the transfer and parsing of large notification messages are very time consuming. The visualization services passing a reference in the notification message show better performance.

This prompts the idea of creating a Data Center for the NoCoV system in the future development. The data used by visualization services can be stored at a centralized Data

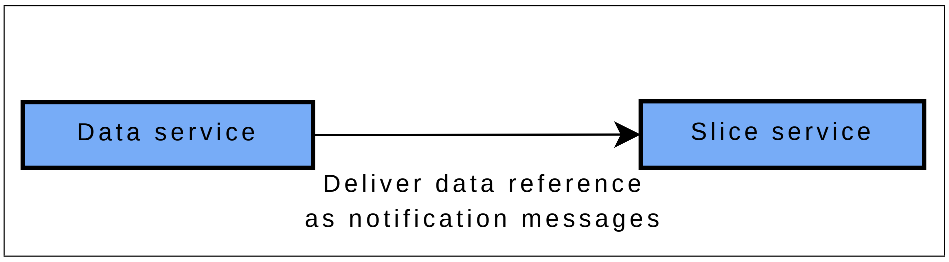


Figure 7.8: Pipeline for slice visualization with passing data reference in notification message.

Dataset	Resolution	Points	Time D→I (ms)	Time I (ms)
dglazing1.dat	18x18x10	3240	33	488
dglazing2.dat	35x35x19	23275	31	4540
dglazing3.dat	52x52x28	75712	33	16,121
dglazing4.dat	69x69x37	176157	34	46,052

Table 7.5: The performance of the isosurface service.

Center. The communication between these services only uses the reference to the data to avoid the overhead of processing large XML messages. As visualization services can be widely distributed at different locations, system administrators need to make the machine hosting Data Center accessible for these services through different firewalls and with high connection speed.

7.2 Evaluation Across Different Grid Networks

To demonstrate that the NoCoV system allows the composition of a visualization pipeline with widely distributed visualization services, we built a visualization pipeline with services deployed on a Grid network at Leeds and a Grid network at Manchester.

7.2.1 Test Environment

The evaluation involves two Grid machines:

Dataset	Resolution	Points	Time D→S (ms)	Time S (ms)
dglazing1.dat	18x18x10	3240	31	151
dglazing2.dat	35x35x19	23275	32	1151
dglazing3.dat	52x52x28	75712	35	5,800
dglazing4.dat	69x69x37	176157	32	25,044

Table 7.6: The performance of the slice service.

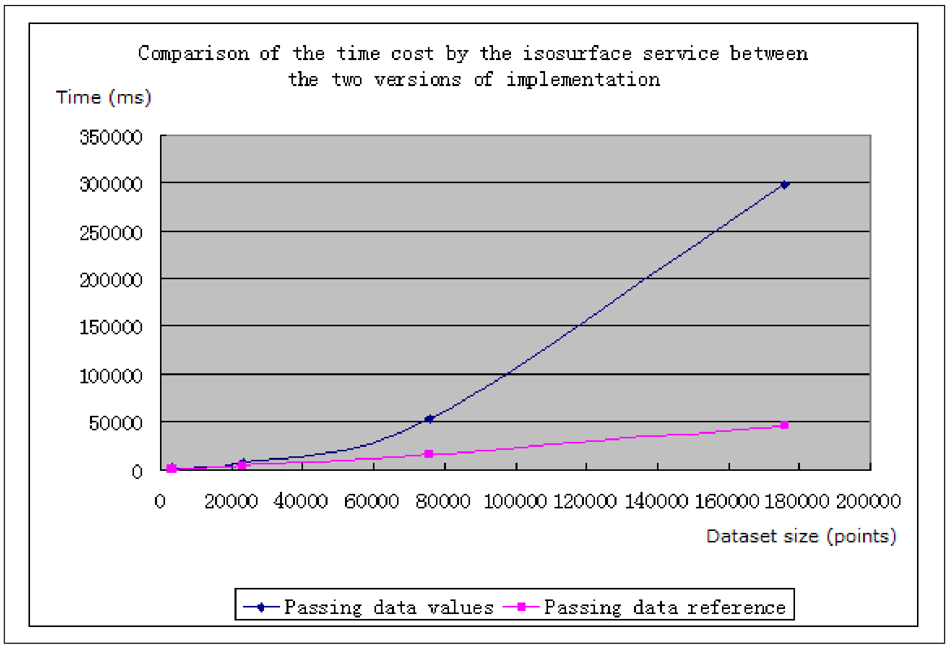


Figure 7.9: Comparison of the time cost by the isosurface service between two versions of implementation.

	testgrid9.leeds.ac.uk	balrog.vipl.mcc.ac.uk
Location	Leeds	Manchester
CPU	2 x (Intel Pentium4 3.00GHz)	4 x (AMD Athlon 64 2.6GHz)
Memory	1.0 GB	4.0 GB
OS	Fedora Core 3	Ubuntu Linux 7.04
Grid Middleware	Globus Toolkit 4.0	Globus Toolkit 4.0

Table 7.7: The system information of the two Grid machines used in the evaluation

- *testgrid9.leeds.ac.uk at Leeds*
- *balrog.vipl.mcc.ac.uk at Manchester*

The system information of these two machines is listed in Table 7.7. The two Grid machines are linked through the UK JANET network [46], which provides up to 10Gbit/s network connection between the two Grid networks.

No firewall is set up on the Manchester Grid network, but the Leeds Grid network is located behind a firewall and the range of port numbers, which are accessible for the Manchester machine, is from 9090 to 9100. There is high speed network connection between these two Grid machines.

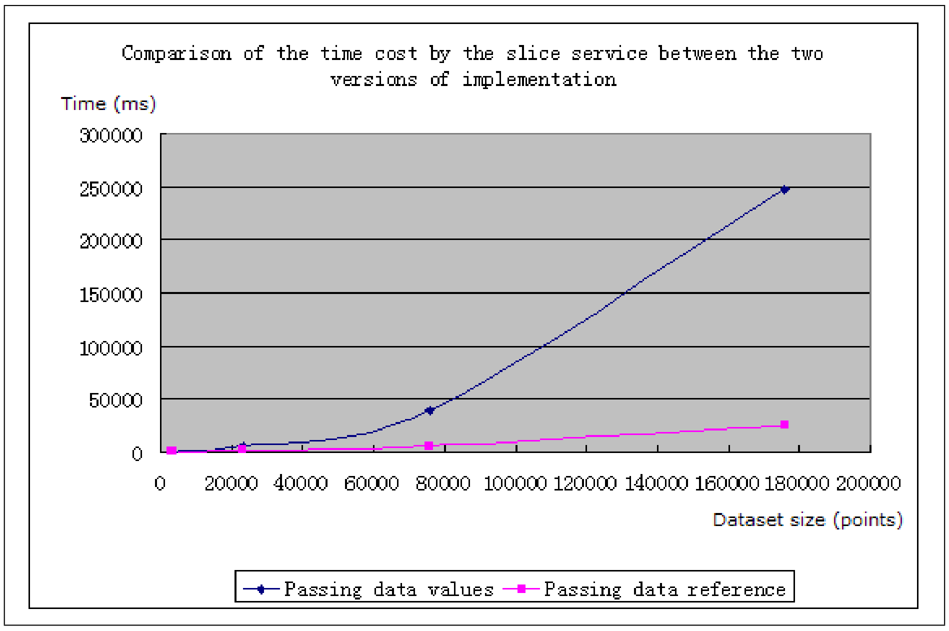


Figure 7.10: Comparison of the time cost by the slices service between two versions of implementation.

7.2.2 Scenario

The aim of the test is to prove the feasibility of distributing the NoCoV system widely across different Grid networks. We can envisage a typical scenario in which users wish to build a visualization pipeline using services deployed on different Grid network.

In the scenario, the pipeline controller service is deployed on the Manchester machine. A system developer at the University of Manchester deploys three services (data service, isosurface service and inline service) on that machine. End-users can build up simple visualization pipelines by connecting these services through the pipeline controller service. However, as slicing is one of the basic visualization techniques, there is a requirement from end-users to have a slice service available in the system.

Meanwhile, another developer at the University of Leeds has implemented a slice visualization service and has deployed it on a Leeds Grid machine. Instead of re-deploying the slice service on the Manchester machine, the developers want to make the slice service accessible for the services deployed at Manchester, so that end-users can create a pipeline to produce both slice and isosurface through the pipeline controller service.

By distributing the NoCoV system across the Manchester Grid network and the Leeds Grid network, the following advantages can be achieved:

- Enabling visualization service developers to implement services in their own Grid environments and jointly provide these services for end-users.

- Reducing the workload of re-deploying the service on the Manchester Grid network.
- Utilizing the remote computational resources on the Leeds Grid network.
- Ease of maintenance of the system. If a visualization service needs to be upgraded or re-implemented, the changes will not affect other parts of the system, as long as the interface of that visualization service remains the same.
- Different centers with different algorithm expertise can take responsibility for provision of particular services.

In the following section, we distributed the NoCoV prototype across two Grid networks to reflect this scenario. We used the version which passes data references as notification messages. The performance of the system is compared against the version which is deployed on the same Grid network (i.e. the Manchester Grid network), so that we can find out how much extra cost is taken to achieve the above advantages of a distributed NoCoV system and we can consider whether it is worthy to distribute the NoCoV system across different Grid networks instead of deploying the whole system on a single Grid network or even on a single machine.

7.2.3 Implementation and Performance

According to the scenario, the NoCoV system is distributed as shown in Figure 7.11. The four visualization services are distributed across two Grid networks. The pipeline controller service has information about where these services are deployed and how these services can be invoked; therefore, through the pipeline controller service, end-users can build visualization pipelines by connecting these visualization services and can set steering parameters on these services.

In order to restrict the communication between the two Grid machines to only uses the opened port numbers (from 9090 to 9100), a 'GLOBUS_TCP_PORT_RANGE' system property on the both machines is set as '9090,9100' using the command:

export GLOBUS_TCP_PORT_RANGE=9090,9100

A visualization pipeline shown as Figure 7.12 was created to test the performance of the NoCoV system distributed across the two Grid networks. A data service feeds in a dataset to an isosurface service and a slice service. The generated isosurface and slice are combined into a single VRML world by using an inline service.

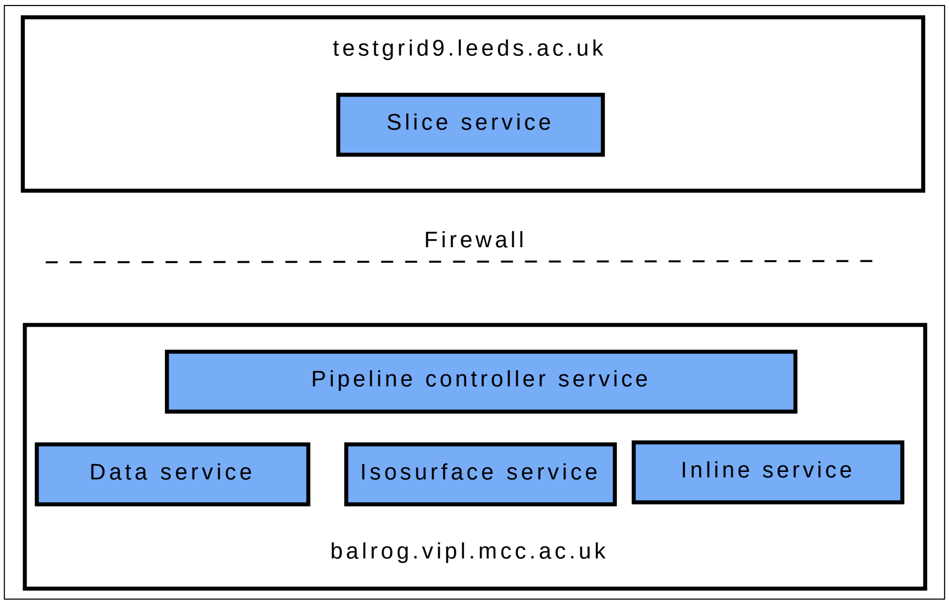


Figure 7.11: The distribution of the NoCoV system across two Grid networks.

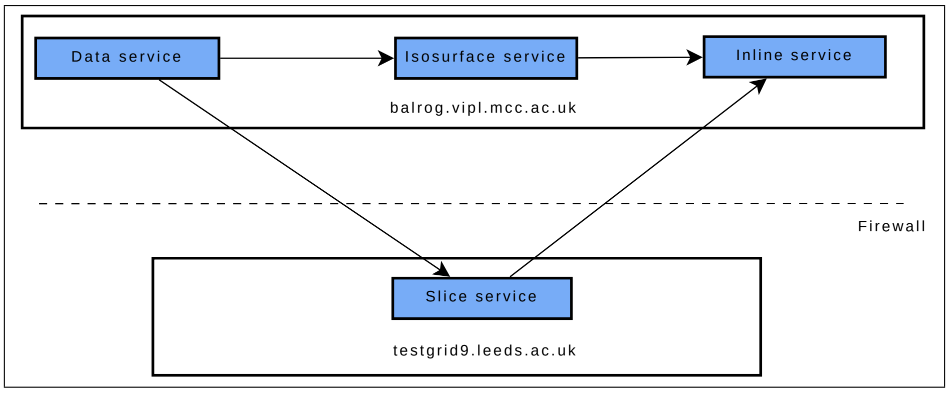


Figure 7.12: A visualization pipeline composed by the services deployed on two Grid networks.

As the process of creating a slice and the process of creating an isosurface are run in parallel, the time cost of the distributed version (t_1) can be represented as:

$$t_1 = t_d + max(t_i, t_{s'}) + t_n + t_{o1},$$

where t_d, t_i and t_n are the time cost by the data service, the isosurface service and the inline service deployed at Manchester respectively; $t_{s'}$ is the time cost by the slice service at Leeds and t_{o1} is the time cost for transferring communication messages between all these services which are on two Grid networks. t_i and $t_{s'}$ include the cost of fetching the data, so if the data is at Manchester, then $t_{s'}$ includes the overhead of moving the data to Leeds.

	Distributed	Local
T	4902ms (t_1)	4711ms (t_2)
T_i	4535ms (t_i)	4535ms (t_i)
T_s	1682ms ($t_{s'}$)	1162ms (t_s)

Table 7.8: The camparison of the performance of the distributed version NoCoV and local version NoCoV.

When all these services are deployed on the Manchester Grid machine, the time cost (t_2) can be represented as:

$$t_2 = t_d + max(t_i, t_s) + t_n + t_{o2},$$

where t_s is the time cost by the slice service at Manchester and t_{o2} is the time cost for transferring communication messages between the services locally.

Therefore, the extra time cost will be:

$$t_1 - t_2 = max(t_i, t_{s'}) + t_{o1} - max(t_i, t_s) - t_{o2}$$

If t_i is larger than $t_{s'}$ and t_s (as the isosurface algorithm is much more complex than the slice algorithm, this condition is typically satisfied in this case), the extra time of the distributed version cost ($t_1 - t_2$) will be $t_{o1} - t_{o2}$, which is the extra time cost in transferring communication messages across the two Grid networks.

We used the 35x35x19 ‘double glazing’ dataset to try this test. The performance figures are listed in Table 7.8. Here, we are interested in how much time is taken by the whole pipeline (labelled as T in Table 7.8), the time cost by the isosurface service (labelled as T_i in Table 7.8) and the time taken by the slice service (labelled as T_s in Table 7.8).

As the figures listed in Table 7.8 satisfy the condition that the time cost by the isosurface service is larger than the time cost by the slice service in both the distributed version and the local version, the time cost for transferring notification messages across the networks can be calculated as:

$$t_{o1} - t_{o2} = t_1 - t_2 = 4902ms - 4711ms = 191ms$$

Compared with the overall time cost, the extra time cost for transferring notification messages across the two Grid networks only reduces the performance of the whole by 4%. Although there is slight drop in the performance, we achieve the advantages of distributing the NoCoV system across different Grid networks.

7.3 Collaborative Visualization

Section 7.2 proves the feasibility of distributing the NoCoV system across different Grid networks. In this section, we will use this distributed version of NoCoV to demonstrate the support for collaboration involving geographically distributed end-users.

The scenarios illustrated in Section 6.6 have demonstrated the support for both synchronous and asynchronous collaboration in the NoCoV system. Figure 7.13 shows how the services and clients involved are distributed. The pipeline controller service, the data service, the isosurface service and the inline service are hosted on the Manchester Grid machine, and the slice service is deployed on the Leeds Grid machine. The collaborators run clients connecting to the pipeline controller service from Manchester and Leeds.

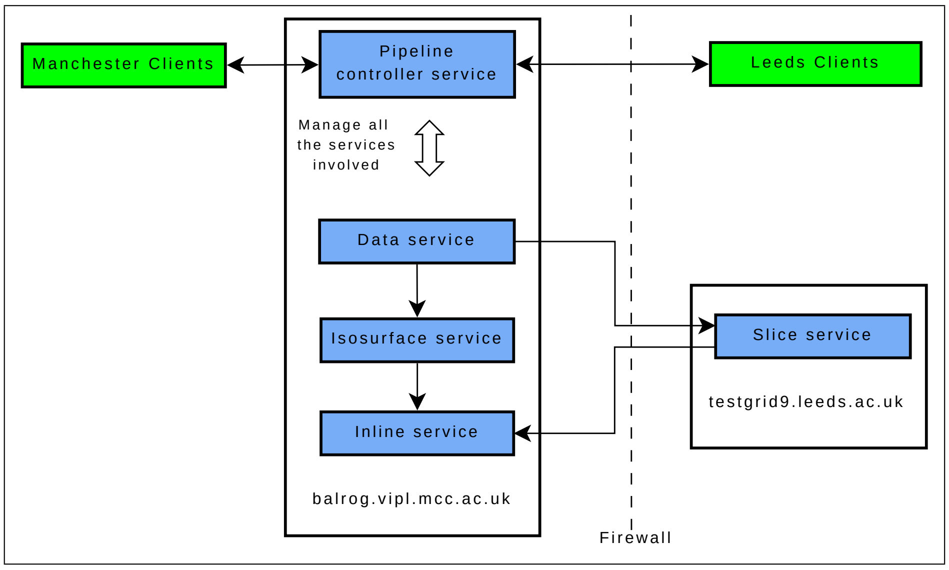


Figure 7.13: The distribution of the NoCoV services and clients

The following collaborative features in the design requirements of the NoCoV system (listed in Chapter 3) have been satisfied by this collaborative experiment:

- *Extensibility of the NoCoV system*
  Visualization services developed by different organisations or researchers are jointly provided for the NoCoV system.

- *Support of collaborative visualization*
  Different end-users can collaboratively create and steer visualization pipelines synchronously or asynchronously.

- *Accommodation of different end-users*
  Two client GUIs are provided: one (Pipeline Editor Client) for visualization experts

to create pipeline and other one (Parameter Control Client) for application scientists to set steering parameters.

- *Capability of wide distribution*
  The experiment has proved that both the services and clients can be distributed across different Grid networks.

7.4 Comparison with Other Systems

In this section, we evaluate NoCoV against a number of other approaches to collaborative visualization.

Screen sharing. Screen sharing is a generic approach which can be applied to any application. All the collaborators view the same display screen and interact with the same interface.

VNC [64, 73] is one example of this solution. By using VNC to share the display screen of a visualization application, users will see the same pipeline and visualization results. This is very effective when one person is driving the visualization from their desktop, and wishes others to see what is going on – ie they are passive collaborators. It becomes very hard to use when the whole team wish to be active collaborators as they can not interact differently with the pipeline at the same time.

Sharing dataflow and parameters. Each collaborator can specify at which point of the pipeline the data or parameters can be shared with other participants.

COVISA [84] uses this concept to enable collaboration. Each collaborator works independently with their own copy of IRIS Explorer [78]. By using COVISA modules, users have the ability to send data from any point on the pipeline to each collaborator – who can then feed this data into their own pipelines. Overall the concept is a set of individual pipelines, with cross-linking between them. This is an extremely versatile approach, offering the team the maximum flexibility to program. However, the collaboration can be rather difficult to work: each person is only aware of their own pipeline, and has no real way of building even a mental image of the full set of interlinked pipelines.

Sharing visualization process code. As visualization process code is sharable amongst participants, each collaborator can obtain the code on their local machines, and run these visualization processes locally.

The SPIDER system [51] can be classified in this category. SPIDER is distinctive from COVISA in that there is a single pipeline shared by all (so there is a common model of the visualization), and distinctive from VNC in that each person runs their own copy

	VNC	COVISA	SPIDER	NoCoV
Generic application	Yes	No	No	No
Awareness in collaboration	Yes	No	Yes	Yes
Avoid synchronization issue	Yes	Yes	No	Yes
Different user interface	No	Yes	Yes	Yes
Participation from all members	No	Yes	Yes	Yes

Table 7.9: Summary of the comparison between VNC, COVISA, SPIDER and NoCoV.

of the pipeline. This feature of each individual running their own copy of the pipeline was intended to minimize network traffic, but in practice it proved hard to synchronise the activity amongst the group of users.

Sharing visualization pipeline and control. Distinctive from the above three solutions, the NoCoV system runs the visualization pipeline as a set of services and allows collaborators to jointly create and control a pipeline.

The comparison between NoCoV and the other three systems includes:

- Compared with VNC, the service-based NoCoV pipeline allows users to connect from different user interfaces, adapting the interface to the experience and expertise of the user. As each user has their own user interface, the NoCoV collaboration also supports active participation from all the collaborators, rather than just only one active user in the collaboration.

- As the users in a NoCoV collaboration session share the same pipeline, it overcomes the problem of awareness in a COVISA collaboration session, enabling all the collaborators to have an overview of the jointly created pipeline.

- Distinctive from the SPIDER system, in the NoCoV system, there is only one execution of the pipeline run on a set of linked remote services. Therefore, the issue of synchronization can be avoided.

- Furthermore, the NoCoV system exploits a service-oriented architecture, in which the user interface is cleanly separated from the engine of the application, and this engine runs as a set of communicating services. The service-oriented architecture also enables the collaborative provision of visualization services from different developers.

A summary of the comparison is listed in Table 7.9.

7.5 Summary

This chapter has evaluated the performance of the NoCoV system. We found that passing a data reference to communicate between services has better performance than passing data values as notification message.

We also tested the feasibility of distributing the NoCoV system across different Grid networks. Based on this widely distributed NoCoV implementation, we proved the system has the capability of supporting both synchronous and asynchronous collaboration and has satisfied the requirements for this research.

At the end of this chapter, a comparison has been given between the NoCoV system and other collaborative visualization systems.

Chapter 8

Conclusions and Future Work

This chapter draws together the features of the NoCoV framework, the proof of concept NoCoV prototype, and the evaluation of the framework to present the conclusions of the notification-service-based collaborative visualization system. Based on the framework designed and the prototype implemented, the future work is also described in this chapter.

8.1 Conclusions

Visualization is a process that can enhance data exploration and data presentation by using computer graphics technologies. Collaborative visualization allows the creation of visualization or the control over parameters to be shared amongst different users. A computational Grid is described as "a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities" [31]. Some work has been done in different projects to investigate the utilization of powerful Grid computational resources for visualization. In this thesis, a novel approach has been introduced to support collaborative visualization in the Grid environment by using notification Web services to build a Service-oriented visualization system.

Chapter 2 of this thesis consists of two strands: visualization and Grid computing. An overview of visualization and widely used visualization systems is presented at the beginning of this chapter. These visualization systems are called Modular Visualization Environments (MVEs) which are based on the pipeline reference model. This model decomposes visualization into a set of processes which are implemented as modules in MVEs. The decomposable visualization model also makes it natural to distribute a visualization at different locations. Meanwhile, as there is a trend that research projects involve multidisciplinary and geographically distributed teams of scientists, there arises the requirement for visualization systems which support collaborative working. The emergence of Grid computing has offered a new opportunity for visualization to utilize powerful computational resources. Furthermore Grid computing provides standard specifications for Web services and Grid services which can be used straightforwardly to build Service-oriented visualization systems. In Chapter 2, after the introduction to some previous projects about distributed and collaborative visualization, I explored some pioneering projects on the utilization of the Grid computing technologies for visualization. Through these previous projects, I gained inspiration and ideas for building the visualization system introduced in this thesis.

Based on the analysis of the earlier visualization projects, especially in the aspects of Service-oriented, Grid-enabled, and collaborative visualization, the motivations and the requirements of our research are explained in Chapter 3. I aim that the visualization system should have the following features:

- Supporting of collaboration;
- Service-oriented;
- Capability for users to customize visualization pipelines;
- Support for users with different background knowledge;
- Grid-enabled;
- Capability to be distributed across different Grid networks.

By fulfilling these requirements, I also drew an overall vision of our research at the end of this chapter. Hopefully this research can help to achieve an ideal visualization system which has a worldwide visualization service repository jointly provided by different organisations/institutes, and supports collaboration on both the creation and the steering of visualization amongst different users.

Chapter 4 described the early experiments which are the initial steps in our research to achieve the vision of the next generation visualization system. I started the experiments by building a couple of visualization services: a slice service and an isosurface service. These visualization services were implemented as 'Pre- Web services' which are distinguished from the Web services defined in WSRF. With the emergence of the OGSA and WSRF specification, the experiments were moved onto GT3 and GT4 middleware deployed in the Grid environment. a secure Grid portal was built to access the g-Viz heart modelling simulations and the visualization Grid services deployed on a GT3 container in the Grid environment. In our third experiment, the visualization Grid services were evolved as WSRF Web services deployed on the GT4 containers and distributed across two different Grid networks: the UK White Rose Grid and the CROWN Grid in China.

Through these early stage experiments, I did not only learn the first hand technical experience, but also verified some aspects for building a collaborative visualization system in the Grid environment:

- The implementation of a visualization pipeline as a set of linked visualization Web services following standard WSRF specifications;
- The easy access for end-users to the Grid-based visualization system;
- The capability of distributing the system across different Grid networks with loose-coupled visualization services;
- The restriction of the access to the visualization system by using Grid secure mechanisms.

I explained the framework of our Notification-service-based Collaborative Visualization (NoCoV) System in Chapter 5. The novel framework can be divided into three layers: the service layer; the management layer; and the client layer.

- The service layer consists of a number of different notification visualization services which can be jointly provided by different organisations or developers and deployed on different Grid networks. By connecting suitable services together, end-users can create customized visualization pipelines.
- The management layer lies between the service layer and the client layer. With all the visualization services registering on it, the management layer can help the clients to discover the services users want. It also manipulates these services on behalf of a user and records visualization information in XML-based representations. Furthermore, for end-users the management layer acts as a platform for collaboration amongst different participants. Asynchronous work is supported by storing visualization information at the management layer and being fetched by different users at different times.

- The client layer has two types of client for the end-users with different expertise. A visual pipeline editor GUI is provided for visualization experts to easily create pipelines from the heterogeneous visualization Web services. For application scientists, the NoCoV system offers a steering GUI which is generated from the pipeline created by visualization experts and has corresponding widgets on it for a user to set steering parameters.

Chapter 6 introduced a NoCoV prototype which was implemented as proof of concept. A set of notification Web services were implemented with various visualization functions, so that users can create pipelines by connecting different visualization services. At the management layer, a pipeline controller service was developed to support the communication with visualization services and the collaboration amongst end-users. An extension of the skML visualization description is used to record the visualization information in the NoCoV system. By storing descriptions on the controller service, different users can asynchronously work on the same visualization. With these descriptions published as notification topics on the controller service, end-users can share visualization information and collaborate synchronously. A pipeline editor client (PEC) is implemented for visualization experts to create pipelines, and a parameter control client (PCC) is created for visualization steering.

In this chapter, it is illustrated through two simple examples how the NoCoV prototype supports both synchronous and asynchronous collaboration. The collaboration includes the joint creation and steering of visualization pipelines amongst different visualization experts and application scientists.

In Chapter 7, two different implementations of the NoCoV system (passing data values as notification messages and passing references as notification messages) have been evaluated and compared by using different sizes of datasets. The feasibility of distributing the NoCoV system widely and the capability of the support of collaboration have been tested by experiments which deployed the system across a Leeds Grid network and a Manchester Grid network, and involved clients from both Leeds and Manchester. At the end of Chapter 7, I have laid out a comparison between the NoCoV system and other collaborative visualization systems.

In this thesis, I have proposed a notification-service-base framework for collaborative visualization in the Grid environment. We can also see great prospects for improvement and future work, as described in the next section.

8.2 Criteria for Success

The success of the research discussed in this thesis can be measured by how the objectives of the research mentioned in Chapter 1 have been achieved.

- *Utilizing powerful computational resources for visualization.*
  In the NoCoV system, visualization functions are implemented as visualization services and deployed on the Grid machines. Users can untilise Grid computational resources, by running NoCoV clients on users' desktops and connecting to these services from their desktops.

- *Allowing collaboration.*
  Visualization pipeline information, including the connection between services and setting of control parameters, is recorded as XML descriptions and stored as service resource properties. When users participate in the collaboration at different times, they can retrieve the visualization previously stored on the service, so that they can collaborate asynchronously. By publishing visualization descriptions as notification topics, any changes occurred on these descriptions can be delivered to all participants at the same time, so that they can synchronously collaborate on the same visualization.

- *Accommodating users with different background knowledge.*
  The NoCoV system provides two kinds of client for end-users. Visualization experts can create visualization pipelines through the Pipeline Editor Client, whereas scientists can use these pipelines and steer them through the Parameter Control Client. For advanced users who have programming skills in the Grid environment, they can develop customized visualization services and integrate them into the NoCoV system.

- *Enabling extensibility and distribution.*
  The NoCoV system is based on the Service-Oriented Architecture. A visualization pipeline is composed of a set of loosely-coupled visualization services, therefore it is straightforward to distributed these services on suitable locations. When new visualization techniques or algorithms emerge, the NoCoV system can also be easily extended by adding new visualization services, which implement these new techniques or algorithms, into the system.

8.3 Future Work

In order to achieve our final vision of the next generation of visualization system, there are many areas of further work that need to be completed. This future work can be categorised into further implementation, further collaborative support, the development of standards and a visualization taxonomy, the research into advanced services and Grid deployment, and visualization provenance.

8.3.1 Further Implementation

Partly because of the time restriction of this PhD thesis, and partly because the supporting technologies are still evolving, some features designed in the NoCoV framework have not been implemented in the prototype. This further implementation work is:

- **Service Development**. In the prototype, only four simple visualization services were implemented. For a real application system, extra visualization services are clearly needed to be developed, so that the NoCoV system can deal with more diverse visualization problems.
- **Access control**. For collaboration and distributed systems, security is one of the important issues. The feasibility of adding an access control to visualization services in the Grid environment has been proved in our early experiments by using the grid-map mechanism. However, when the system is distributed across different organisations' networks, the identities or credentials used in one organisation might not be recognized in other organisations. The solution can be that either all organisations use the certificates authorized by the same trusted Certificate Authority (CA), or a mapping mechanism can be used to map certificates across different organisations. The security across the boundaries of different organisations is outside the scope of the research explained in this thesis.
- **Dynamical visualization task shifting**. To enhance the reliability and the efficiency of the system, we can deploy the same visualization service at different machines. When users connect a visualization service into their pipelines, if the machine where the service is deployed is overloaded or has a hardware failure, the visualization task can be carried on by replacement services deployed on other healthy machines. This design aims to shift the visualization task running within a service seamlessly to its replacement service. The implementation needs to consider the disconnection of the service from the existing pipeline and the connection

of the replacement into the pipeline. As pipelines have been recorded as XML descriptions, the migration of visualization task between two services also requires the modification of the visualization pipeline. Note that in the e-Viz project, information about where the visualization task is actually processed is included in the XML description, and this could be incorporated into out work.

- **Lifetime management of asynchronous collaborative sessions**. A house-keeping mechanism can be added into NoCoV by using the lifetime management mechanism from GT4. Therefore, it can be avoided that inactive asynchronous collaborative sessions occupy resources for very long time.
- **Registration and discovery of visualization services**. A UDDI registry can be implemented at the management layer of the framework. The registry will hold descriptions (WSDL files) of heterogeneous visualization services deployed at different locations. With these WSDL files, the controller service can automatically generate the communication stubs to invoke these services. By implementing the UDDI registry, the extensibility of the NoCoV system can be improved, as there will be no manual modification of the system required, when new visualization services are introduced into the system. However, a standard visualization taxonomy to describe heterogeneous visualization services is lacking.
- **Retrieval of service list at the client**. In the current implementation of the prototype, a PEC loads the list from an XML file which holds information of available visualization services. With a UDDI registry implemented at the management layer, a PEC can retrieve the list dynamically from the UDDI registry instead of a static XML file.

8.3.2 Further Collaboration Support

The collaboration features in the prototype are simply implemented as proof of concept. In the prototype, the pipeline or control parameters displayed on clients' GUIs will be updated, if any participants make a change. However, for collaboration in a realistic scenario, more collaborative features are required in the aspects of the awareness in the collaboration and the communication amongst participants.

- **Awareness in the collaboration**. The current prototype supports the sharing of awareness when users work in the same collaborative session. Participants view the same pipeline and all the steering parameter settings. However, features giving extra awareness could be supported in a future implementation. A message describing the changes made by different participants can be published as a notification topic within the controller service. By subscribing to this notification topic, each participant in the collaborative session can receive a detailed description of the last change made about either pipeline or control parameters, such as who makes the change, when the change is made and what are the previous status and the current status.Wright and Wood have done some research [83] to support awareness in collaborative visualization and present visualization and parameters consistently.

- **Communication amongst participants**. In a scenario with multiple users working collaboratively, users usually require some communication methods to discuss the work, such as video or audio communication tools, text chatting tools, telephone, e-mail, etc. In a Grid environment, systems like the Access Grid (AG) system [2] can provide our collaborative visualization system with a powerful conference system. The research of integrating Access Grid with a collaborative visualization system has been done in the ICENI project [48].

8.3.3 Visualization Taxonomy and Data Format Standard

As the final vision of the NoCoV framework is to develop a worldwide collaborative visualization system, the visualization taxonomy and the data format standard are important to achieve this final goal.

- **Visualization taxonomy and ontology**. For both humans and machines, a widely accepted visualization taxonomy and ontology can help the quick discovery of different visualization techniques, which are implemented as different visualization services in the NoCoV framework, as these techniques or services can be well categorized. Standard visualization taxonomy and ontology are also important for collaboration, as collaborators can describe visualizations precisely without causing any ambiguity in their discussion and communication. Some research has been done to create visualization taxonomies and ontology, such as [13,19,26,69]. However, further research is needed to produce uniform visualization taxonomy and ontology.

- **Data format standard**. In order for visualization services to interwork, there needs to be a well-defined format for the transfer of data. In NoCoV, I have defined my own format, as indeed do current visualization systems. For example, IRIS Explorer uses lattice data, while AVS defines its data structures as AVS/Express Field data type. The Hierarchical Data Format (HDF) [43] is developed by NCSA (National

Center for Supercomputing Applications) as a self-contained data format which is designed to be easily used by visualization software. As the latest version of HDF - HDF5, is custom designed for the handling of huge scientific datasets for visualization, HDF5 could be a good choice for adoption as the standard numerical data format in the future development of the NoCoV system. For geometric data, as there are wide range of 2D and 3D formats, such as JPEG, FIT, VRML, X3D, etc., it is difficult to define a single standard within a worldwide distributed system. However, the NoCoV system can specify a set of these formats which will be compatible with different visualization services.

8.3.4 Advanced Service and Grid Development

As a widespread research topic, Grid computing involves many strands looked promising in the area of visualization. We can identify the following aspects which can possibly be applied in the NoCoV system.

- **Data transport in the Grid environment**. Within the NoCoV system, there will be mass datasets that need to be transported between different visualization services. A reliable and efficient data transport mechanism will be vital for the NoCoV system. As NoCoV visualization services deliver addresses (URIs or URLs) of data as notification messages to its subscribers, for these subscribers, they can adopt data transport systems such as GridFTP [38] to retrieve the data.

- **Dynamic service deployment**. The dynamic service deployment can offer greater flexibility than the dynamic visualization task migration mentioned in Section 8.2.1, as it allows users to choose the location on which to run visualization services. The performance of a widely distributed system largely relies on the network connection. By using a dynamic service deployment mechanism, distributed visualization services can be re-deployed on a local Grid network, so the time cost in data transports can be greatly reduced. The Dynasoar project [28] shows promise in the area of dynamic service deployment. An initial prototype [79] has been produced and successfully tested.

8.3.5 Visualization Provenance

In most visualization scenarios, the visualization experiments need to be done repeatedly with different parameters. Therefore, there arises the requirement to record a visualization in its entirety, including both the visualization result and the visualization provenance.

An example of early research about historical visualization is the GRASPARC project [11]. It used a structure called a History Tree to record the historical information about how a visualization is produced with each node in the tree representing each historical stage. Work on visualization provenance has been investigated by the myGrid project [90] and the VisTrails project [35]. In these projects, a visualization pipeline is considered as a workflow and the provenance information is recorded in XML/RDF files for users to repeat a visualization from certain point in the history.

8.4 Summary

The conclusions of the research that have been undertaken and the directions of the future work have been laid out in this chapter.

The research proposes a novel collaborative visualization framework called NoCoV in the Grid environment with an initial prototype as proof of concept. The NoCoV framework is based on notification service technologies to link distributed visualization services and support collaboration amongst multiple users. The framework accommodates users with or without Grid computing skills and visualization expertise. It allows end-users to access heterogeneous computational resources across different Grid networks and steer Grid-based visualization pipelines from their desktops.

Although there is still much future work to be completed to reach the final vision of our design, the NoCoV framework shows great promise for building visualization systems. This thesis has successfully demonstrated the potential of NoCoV for collaborative visualization in the Grid environment on a UK national scale. The next step is to explore whether NoCoV can be used on an even wider scale (i.e. a global scale) - our long-term vision.

Bibliography

[1] G Abram and L Treinish. An extended data-flow architecture for data analysis and visualization. *Computer Graphics*, 29(2):17–21, 1995.

[2] AccessGrid. AccessGrid website.
http://www.accessgrid.org. last accessed 03 September, 2007.

[3] Amira. Amira Homepage: 3D Data Visualization.
http://www.amiravis.com/. last accessed 03 September, 2007.

[4] L Applegate. Technology Support for Cooperative Work: A framework for Studying Introduction and Assimilation in Organization. *Journal Organizational Computing*, pages 11–39, 1991.

[5] M Atkinson, D DeRoure, A Dunlop, G Fox, P Henderson, T Hey, N Paton, S Newhouse, S Parastatidis, A Trefethen, P Watson, and J Webber. Web Service Grids: an evolutionary approach: Research Articles. *Concurrency and Computation: Practice & Experience*, 17(2-4):377–389, 2005.

[6] AVS. Advanced Visual Systems (AVS) website.
http://www.avs.com/. last accessed 03 September, 2007.

[7] T Banks, A Djaoui, S Parastatids, A Mani, S Tuecke, K Czajkowski, I Foster, J Frey, S Graham, C Kesselman, T Maguire, T Sandholm, D Snelling, and P Vanderbilt. Open Grid Service Infrastructure Primer. *Technical Report GFD.31, Global Grid Forum*, 2004.

[8] F Berman, G Fox, and A Hey. The Grid: Past, Present and Future. *Grid Computing: Making the Global Infrastructure a Reality, Geoffrey C Fox, and Anthony J G Hey, Eds. Wiley.*, 2003.

[9] P Bourke. Polygonising a Scalar Field. http://local.wasp.uwa.edu.au/∼pbourke/geometry/polygonise/, (last accessed 03 September, 2007).

[10] BPEL4WS. Business Process Execution Language for Web Services Specification Version 1.1. ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf. last accessed 03 September, 2007.

[11] K Brodlie, L Brankin, A Poon, G Banecki, H Wright, and A Gay. GRASPARC: A Problem Solving Environment Integrating Computing and Visualization. *IEEE Visualization , Los Alamitos, CA. IEEE Computer Society Press.*, pages 102–109, 1993.

[12] K Brodlie, J Brooke, M Chen, D Chisnall, C Hughes, N John, M Jones, M Riding, N Roard, M Turner, and J Wood. Adaptive Infrastructure for Visual Computing. *Theory and Practice of Computer Graphics 2007, Bangor. ISBN 978-3-905673-63-0*, pages 147–156, 2004.

[13] K Brodlie, L Carpenter, R Earnshaw, J Gallop, R Hubbold, A Mumford, C Osland, and P Quarendon. Scientific Visualization - Techniques and Applications. *Springer-Verlag*, 1992.

[14] K Brodlie, J Wood, D Duce, and M Sagar. gViz: Visualization and Computational Steering on the Grid. *The 3rd UK e-Science All Hands Meeting, Nottingham, UK, ISBN 1-904425-21-6*, pages 54–60, 2004.

[15] R Butler, D Engert, I Foster, C Kesselman andS Tuecke, J Volmer, and V Welch. A National-Scale Authentication Infrastructure. *IEEE Computer*, 33:60–66, 2000.

[16] cAVS. The Texas Advanced Computer Center cAVS website. http://www.tacc.utexas.edu/cavs/. last accessed 03 September, 2007.

[17] S Charters. Virtualising Visualisation: A distributed service based approach to visualisation on the Grid. *PhD thesis, University of Durham*, 2006.

[18] S Charters, N Holliman, and M Munro. Visualisation on the Grid: A Web Service Approach. *The 3rd UK e-Science All Hands Meeting*, pages 139–146, 2003.

[19] E Chi. A Taxonomy of Visualization Techniques using the Data State Reference Model. *The Symposium on Information Visualization (InfoVis '00), IEEE Press*, pages 69–75, 2000.

[20] D Churches, G Gombas, A Harrison, J Maassen, C Robinson, M Shields, I Taylor, and I Wang. Programming Scientific and Distributed Workflow with Triana Services. *Concurrency and Computation: Practice and Experience (Special Issue: Workflow in Grid Systems)*, 18:1021–1037, 2006.

[21] COVISE. COVISE: "Collaborative Visualization and Simulation Environment" website.
http://www.hlrs.de/organization/vis/covise/. last accessed 03 September, 2007.

[22] K Czajkowski, D Ferguson, I Foster, J Frey, S Graham, T Maguire, D Snelling, and S Tuecke. From Open Grid Services Infrastructure to WSResource Framework: Refactoring & Evolution. *Whitepaper report*, 2004.

[23] T DeFanti, I Foster, M Papka, R Stevens, and T Kuhfuss. Overview of the i-way: Wide-area Visual Supercomputing. *International Journal of Supercomputer Applications and High Performance Computing*, 10(2):123–130, 1996.

[24] D Duce, J Gallop, I Johnson, K Robinson, C Seelig, and C Cooper. Distributed Cooperative Visualization - The MANICORAL Approach. *Eurographics UK Conference, ISBN 0-952 1097-7-8*, pages 69–85, 1998.

[25] D Duce and M Sagar. skML a Markup Language for Distributed Collaborative Visualization. *EG UK Theory and Practice of Computer Graphics*, pages 171–178, 2005.

[26] D Duke, K Brodlie, D Duce, and I Herman. Do you see what i mean? *IEEE Comput. Graph. Appl.*, 25(3):6–9, 2005.

[27] DX. OpenDX website.
http://www.opendx.org/. last accessed 03 September, 2007.

[28] Dynasoar. Dynasoar Project website.
http://www.neresc.ac.uk/projects/dynasoar/. last accessed 22 June, 2007.

[29] A Fewings. Real-Time and Interactive Computer Graphics in Grid Environments. *PhD thesis, University of Wales, Bangor*, 2006.

[30] A Fewings and N John. Distributed Graphics Pipelines on the Grid. *IEEE Distributed Systems Online*, 2007.

[31] I Foster. What is the Grid? A Three Point Checklist. *GRIDToday*, 2002.

[32] I Foster and C Kesselman. Globus: A Meta-Computing Infrastructure Toolkit. *International Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[33] I Foster and C Kesselman. The Grid: Blueprint for a New Computing Infrastructure. *San Francisco, CA: Morgan Kaufmann*, 1999.

[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 22*, 2002.

[35] J Freire, C Silva, S Callahan, E Santos, C Scheidegger, and H Vo. Managing Rapidly-Evolving Scientific Workflows. *International Provenance and Annotation Workshop (IPAW)*, 2006.

[36] M Friendly. Gallery of Data Visualization. 2001. http://www.math.yorku.ca/SCS/Gallery/ electronic document, (last accessed 03 September, 2007).

[37] Globus. Globus Alliance website. http://www.globus.org/. last accessed 03 September, 2007.

[38] GridFTP. Globus Alliance website GridFTP. http://www.globus.org/grid_software/data/gridftp.php. last accessed 03 September, 2007.

[39] I Grimstead, D Walker, and N Avis. Collaborative Visualization: A Review and Taxonomy. *Distributed Simulation and Real-Time Applications, 2005. DS-RT 2005 Proceedings. Ninth IEEE International Symposium*, 2005.

[40] gViz. UK e-Science gViz Project website. http://www.comp.leeds.ac.uk/vvr/gViz/. last accessed 03 September, 2007.

[41] R Haber and D McNabb. Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. *Visualization in Scientific Computing, IEEE Computer Society Press*, pages 74–93, 1990.

[42] R Hamming. Numerical Methods for Scientists and Engineers. *McGraw-Hill New York*, 1962.

[43] HDF. NCSA HDF home page. http://hdf.ncsa.uiuc.edu/HDF5. last accessed 22 June, 2007.

[44] G Humphreys, M Houston, R Ng, R Frank, S Ahern, P Kirchner, and J Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph.*, 21(3):693–702, 2002.

[45] IRIS. NAG IRIS Explorer website. http://www.nag.co.uk/welcome_iec.asp. last accessed 03 September, 2007.

[46] JANET. UK's education and research network website. http://www.ja.net. last accessed 22 June, 2007.

[47] JIMI. Sun Developer Network JIMI website. http://java.sun.com/products/jimi/. last accessed 03 September, 2007.

[48] G Kong, J Stanton, S Newhouse, and J Darlington. Collaborative Visualisation over the Access Grid using the ICENI Grid Middleware. *The 2nd UK e-Science All Hands Meeting, Nottingham, UK, ISBN 1-904425-11-9*, pages 393–396, 2003.

[49] D Kranzlmuller, P Heinzlreiter, H Rosmanith, and J Volkert. Grid-Enabled Visualization with GVK. *European Across Grids Conference*, pages 139–146, 2003.

[50] W Lorensen and H Cline. Marching Cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 21:163–169, 1987.

[51] S Lovegrove. Distributed Co-operative Visualization. *PhD thesis, University of Leeds*, 2003.

[52] B Ludascher, I Altintas, C Berkley, D Higgins, E Jaeger-Frank, M Jones, E Lee, J Tao, and Y Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows*, 2005.

[53] S Majithia, I Taylor, M Shields, and I Wang. Triana as a Graphical Web Services Composition Toolkit. *The 2nd UK e-Science All Hands Meeting, EPSRC*, pages 494–500, 2003.

[54] MATLAB. Mathworks MATLAB website. http://www.mathworks.com/products/matlab/. last accessed 03 September, 2007.

[55] B McCormick, T DeFanti, and M Brown. Visualization in Scientific Computing. *ACM SIGRAPH, Computer Graphics*, 21November(6), November 1987.

[56] MeVisLab. MeVisLab Medical Image Processing and Visualization website. http://www.mevislab.de/. last accessed 03 September, 2007.

[57] NGS. National Grid Service website. http://www.grid-support.ac.uk/. last accessed 03 September, 2007.

[58] J Novotny, S Tuecke, and V Welch. An Online Credential Repository for the Grid: MyProxy. *The Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*, pages 104–111, 2001.

[59] OASIS. The Organization for the Advancement of Structured Information Standards (OASIS) website. http://www.oasis-open.org/home/index.php. last accessed 03 September, 2007.

[60] OGSA. Globus Alliance OGSA website. http://www.globus.org/ogsa. last accessed 03 September, 2007.

[61] T Oinn, M Greenwood, M Addis, M Alpdemir, J Ferris, K Glover, C Goble, A Goderis, D Hull, D Marvin, P Li, P Lord, M Pocock, M Senger, R Stevens, A Wipat, and C Wroe. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. *Concurrency and Computation: Practice and Experience, Grid Workflow Special Issue*, 19:1067–1100, 2005.

[62] OMII. Open Middleware Infrastructure Institute (OMII) UK website. http://www.omii.ac.uk/. last accessed 03 September, 2007.

[63] J Rasku and M Ward. Visualizing Changes in 2-D and 3-D Shapes. *Proceedings of DICTA-93 Digital Image Computing: Techniques and Applications*, 1993.

[64] RealVNC. RealVNC website. http://www.realvnc.com/index.html. last accessed 03 September, 2007.

[65] T Richardson, Q Stafford-Fraser, K Wood, and A Hopper. Virtual Network Computing. *IEEE Internet Computing*, pages 33–38, 1998.

[66] M Riding, J Wood, K Brodlie, J Brooke, M Chen, D Chisnall, C Hughes, N John, N Jones, and N Roard. e-Viz: Towards an Integrated Framework for High Performance Visualization. *The 4th UK e-Science All Hands Meeting, EPSRC, ISBN 1-904425-53-4*, pages 1026–1032, 2005.

[67] M Sastry and M Craig. Scalable application visualization services toolkit for problem solving environments. *The 2nd UK e-Science All Hands Meeting, Nottingham, UK, ISBN 1-904425-11-9*, pages 520–525, 2003.

[68] W Schroeder, K Martin, and W Lorensen. The Visualization Toolkit. *Kitware, Inc. ISBN 1-930934-07-6*, 2002.

[69] G Shu, N Avis, and O Rana. Investigating visualization ontologies. *The 6rd UK e-Science All Hands Meeting*, pages 249–256, 2006.

[70] SOAP. W3C SOAP Version 1.2 Primer.
http://www.w3.org/TR/soap12-part0/. last accessed 03 September, 2007.

[71] Taverna. UK myGrid, Taverna Project website.
http://taverna.sourceforge.net. last accessed 03 September, 2007.

[72] E Teong. Data Visualization of World Wide Web. *Master thesis, University of Leeds*, 1997.

[73] TightVNC. TightVNC website.
http://www.tightvnc.com/. last accessed 03 September, 2007.

[74] UDDI. OASIS UDDI website.
http://www.uddi.org/. last accessed 03 September, 2007.

[75] C Upson, T Faulhaber, D Kamins, D Laidlaw, D Schlegel, J Vroom, R Gurwitz, and A van Dam. The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics , Applications*, pages 30–41, 1989.

[76] VRML. Virtual Reality Modeling Language specificationd.
http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/. last accessed 03 September, 2007.

[77] VTK. The Visualization Toolkit (VTK) website.
http://www.vtk.org/. last accessed 03 September, 2007.

[78] J Walton. NAG's IRIS Explorer. *Visualization Handbook. C Johnson and C. Hansen (eds), Academic Press, New York.*, pages 633–654, 2004.

[79] P Watson, C Fowler, C Kubicek, A Mukherjee, J Colquhoun, M Hewitt, and S Parastatidis. Dynamically deploying web services on a grid using dynasoar. *The Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2006)*, pages 151–158, 2001.

[80] WFMC. The Workflow Management Coalition website. http://www.wfmc.org/. last accessed 03 September, 2007.

[81] J Wood. Collaborative Visualization. *PhD thesis, University of Leeds*, 1998.

[82] J Wood, K Brodlie, and H Wright. Visualization over the World Wide Web and its application to environmental data. *IEEE Visualization 1996 Conference, edited by R.Yagel and G.M. Nielson, ACM Press. ISBN 0-89791-864-9.*, pages 81–86, 1996.

[83] J Wood and H Wright. Steering via the image in local, distributed and collaborative settings. *The 5rd UK e-Science All Hands Meeting*, 2005.

[84] J Wood, H Wright, and K Brodlie. Collaborative Visualization. *IEEE Visualization 1997 Conference, edited by R. Yagel and H.Hagen, ACM Press. ISBN 1-58113-011-2*, pages 253–260, 1997.

[85] WRG. White Rose Grid website. http://www.wrgrid.org.uk/. last accessed 03 September, 2007.

[86] WSDL. Web Services Description Language (WSDL) Version 1.1. http://www.w3.org/TR/wsdl. last accessed 03 September, 2007.

[87] WSN. Web Services Base Notification Specification version 1.3. http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.htm. last accessed 03 September, 2007.

[88] WSRF. Globus Alliance WSRF website. http://www.globus.org/wsrf/. last accessed 03 September, 2007.

[89] Xj3D. Xj3D website. http://www.web3d.org/x3d/xj3d. last accessed 03 September, 2007.

[90] J Zhao, C Goble, M Greenwood, C Wroe, and R Stevens. Annotating, linking and browsing provenance logs for e-science. *The 2nd International Semantic Web Conference (ISWC2003) Workshop on Retrieval of Scientific Data, Florida*, October 2003.