

# A Service-Oriented Visualization of Medical Data

Ning Yang

MSc thesis

Supervisor: Dr. Elena Zudilova-Seinstra

Mentor: Dr. Alban Ponse

Professor: Prof. Chris Jesshope

submitted in partial fulfillment of the requirements for the degree of

MSc in computer science

the University of Amsterdam

August 2006



# Abstract

The purpose of scientific visualization is to enhance existing scientific methods by increasing scientist's abilities to gain the understanding and insight into data and computational results. Dataset sizes and the amount of details in these datasets are growing at explosive rates. Aside from developing efficient extraction techniques to act on these datasets, it is necessary to have available computational facilities to visualize the results of the extraction process.

Having this in mind we have developed a visualization framework under the concept of services in a distributed computing environment. The framework allows, on one hand, visualization experts to construct visualizations for end-users, and on the other hand, end-users to exploit the visualization functionalities irrespective to the available computational resources and geographical locations.

The test case of this research is the surface extraction and streamline-based visualizations applied to the medical data generated from medical imaging technology and results of the CFD simulation.

Parts of this thesis have been accepted for publication in and presentation at:

Zudilova-Seinstra E.V., Yang N. (2005): Towards Service-based Interactive Visualization, Proc. of the International Symposium on Ambient Intelligence and Life, July 21-22, 2005, San Sebastian, Spain, pp. 15-25.

# Acknowledgments

I would like to take this opportunity to express my gratitude to the people who supported me during the time of completion of this thesis project.

I thank Dr. Elena Zudilova-Seinstra, my supervisor, for proposing this project and giving me the opportunity to work with her. Visualization is science as well as art. Web services is an emerging standard for distributed environments. Challenging as it is, practicing visualization in the concept of service is certainly an interesting subject. Her kind supervision was reflected through out the whole time, even before the project was officially started. She helped me to get through the tough time of my study.

My mentor, Dr. Alban Ponse, who, as gentleman as he is, guided me through the time I studied at the University of Amsterdam. Your gentle smile makes me feel at home.

Adianto Wibisono and Dmitry Vasunin, who provided me precious kick-start advices and tremendous help along my work, who sitted next to me and helped me with smallest details. Without all these, I could not have performed my research.

The people in the Scientific Visualization and Virtual Reality Group. Especially Dr. Robert Belleman and Michael Scarpa, your guys' magic hands made my life easier.

At this point, I would like to raise my gratitude to a greater extent. This includes my parents back in China, for their patience and understanding. All my friends and people close to me in Amsterdam (Netherlands), who I spent the unforgettable great time with. This is indeed a lifetime memory for me.

Thank you all very much.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Scientific Visualization . . . . .	1
1.2	Medical Datasets . . . . .	2
1.3	Grid Computing . . . . .	3
1.4	Motivation and Goal . . . . .	4
1.5	Chapter Overview . . . . .	5
<b>2</b>	<b>Related Work</b>	<b>6</b>
2.1	Visualization Models and Systems . . . . .	6
2.1.1	Visualization Cycle Model . . . . .	7
2.1.2	Visualization Pipeline . . . . .	7
2.1.3	Layered Reference Model . . . . .	8
2.1.4	Selective Visualization Pipeline . . . . .	8
2.1.5	Visualization Systems . . . . .	10
2.2	Web/Grid Services . . . . .	10
2.2.1	Web Services . . . . .	10
2.2.2	Grid Services . . . . .	10
2.3	Related Projects . . . . .	11
2.3.1	Grid Visualization Kernel . . . . .	11
2.3.2	VIBRO . . . . .	12
2.3.3	G-Viz . . . . .	13
2.3.4	VizWiz . . . . .	13
2.3.5	E-Demand . . . . .	14
<b>3</b>	<b>Design and Implementation of the Visualization Framework</b>	<b>15</b>
3.1	Design Considerations . . . . .	15
3.1.1	Motivation . . . . .	15
3.1.2	A Service-Oriented Visualization Model . . . . .	16
3.1.3	Client-Server Architecture and Data Exchange . . . . .	17
3.1.4	Image-Based Medical Analysis . . . . .	18
3.2	Design Issues . . . . .	19

3.2.1	Service Design . . . . .	21
3.3	Implementation Issues . . . . .	22
3.3.1	Development Tools . . . . .	22
3.3.2	Development Steps . . . . .	23
<b>4</b>	<b>Test Case</b>	<b>25</b>
4.1	Medical Analysis . . . . .	25
4.1.1	Vascular Disorders . . . . .	25
4.1.2	Data Preparation . . . . .	25
4.2	Visualization of Medical Data . . . . .	26
4.2.1	Visualization of Experimental Data . . . . .	26
4.2.2	Flow Visualization . . . . .	27
4.2.3	Streamline-Based Visualization . . . . .	29
4.3	VTK Pipelines . . . . .	30
4.3.1	Visualizing an Artery . . . . .	30
4.3.2	Visualizing a Blood Flow . . . . .	30
4.4	Service-Oriented Pipelines . . . . .	31
4.5	Performance Measurements . . . . .	34
4.5.1	VTK Pipeline Execution . . . . .	34
4.5.2	Service Execution . . . . .	37
4.5.3	Breakdown of Round Trip Time . . . . .	39
<b>5</b>	<b>Interactive Composition of a Visualization Pipeline</b>	<b>41</b>
5.1	Scientific Workflow Systems . . . . .	41
5.2	VLAM-G . . . . .	42
5.2.1	Introduction . . . . .	42
5.2.2	From Services to Modules . . . . .	43
5.3	Graphical Editing . . . . .	45
<b>6</b>	<b>Conclusions</b>	<b>48</b>
6.1	Discussion . . . . .	50
6.2	Future Work . . . . .	51

# List of Figures

2.1	Visualization cycle model . . . . .	7
2.2	Haber-McNabb visualization pipeline . . . . .	8
2.3	Dataflow pipeline with computational steering . . . . .	8
2.4	Layered reference model . . . . .	9
2.5	Selective visualization pipeline . . . . .	9
3.1	Interactive service-oriented visualization framework . . . . .	16
3.2	Image-based medical analysis . . . . .	18
3.3	Architectural design of the framework . . . . .	19
3.4	Data flow control in the framework . . . . .	20
4.1	Critical points of the flow . . . . .	28
4.2	A surface extraction pipeline and the final image result . . . . .	30
4.3	A streamline extraction pipeline and the final image result . . . . .	31
4.4	Combined visualization of a selected zone of an artery and a blood flow calculated for this particular zone . . . . .	32
4.5	Main blocks of the surface extraction pipeline . . . . .	32
4.6	Main blocks of the streamline extraction pipeline . . . . .	33
4.7	Render time of a small dataset with different number of stream- lines . . . . .	35
4.8	Render time of a medium dataset with different number of streamlines . . . . .	36
4.9	Render time of a large dataset with different number of stream- lines . . . . .	36
4.10	Execution time of a reader service on different datasets . . . . .	37
4.11	Execution time of a filter service on different datasets . . . . .	38
4.12	Execution time of a render service on different datasets . . . . .	38
4.13	Components breakdown of round trip time . . . . .	39
5.1	Topology composition of a visualization pipeline . . . . .	46
5.2	Result of a visualization pipeline in VLAM-G . . . . .	47

# Chapter 1

## Introduction

### 1.1 Scientific Visualization

Modern technology provides access to large quantities of data in many application domains, such as medical imaging, fluid flow simulation and geographic information systems. Visualization is a powerful technique for interpreting data for many scientific problems. While computers excel at simulations, data filtering and data reduction, humans are experts at using their highly-developed pattern-recognition skills to look through the results for regions of interest, features and anomalies.

Visualization allows to[5]:

- explore data at various levels of detail;
- accomplish a great sense of engagement with data;
- provide users with a better understanding of data;
- discover details, relations and patterns in data through interactive exploration.

According to[42] visualization techniques have been categorized into two major areas:

- information visualization, which involves abstract, non-spatial data, and
- scientific visualization, which involves scientific data with an inherent physical component.

Scientific visualization is the process of the graphical representation of the scientific data as a means of gaining understanding and insight into the system that is studied. As a science, it is a study concerned with the interactive display and analysis of data, which comes largely from investigating complex dynamics and phenomena by means of experiments and/or simulations.

In general, there are two sources of data to be visualized[21]:

1. measured (physical data from experiments);
2. computed (theoretical data from numerical simulations)

Measured data, and data from stochastic simulation, will have associated uncertainty information; data from other types of simulation often do not. Apart from this, two sources of data can be treated as equivalent.

Data are typically sets of discrete samples, rather than continuous ones. In contrast, the underlying entity which has been measured or simulated is usually continuous (possibly with some well defined regions of discontinuity).

## 1.2 Medical Datasets

Medical datasets comprise a diverse range of measurements such as tissue densities, sensitivity of magnetization, blood flow velocity and material strain[48].

The past decades have seen the variety of medical imaging modalities[36]. Today's imaging technology even makes it possible to acquire digital samples of the subjects in 3D space. 3D medical data is usually acquired by means of such imaging techniques as Computed Tomography (CT), Magnetic Resonance Imaging (MRI) or Positron Emission Tomography (PET). The choice of the imaging technique is determined by the structure or anomaly that needs to be observed, given that some techniques are better suited for certain cases than others.

No longer only a field of experimental science, medicine now uses computer science and information technology extensively across its many research areas. For understanding and simulating organ functions and diseases, advanced computational methods are applied. Clinical data driven simulations have become very advanced and complicated.

The size and complexity of medical data available today makes it difficult to analyze and interpret them. Processing large medical datasets and dealing with the constantly growing amount of details in these datasets are the major cornerstones of today's healthcare.

Medical visualization applications and systems tackle this challenge by extracting the information available in medical datasets and creating an accurate presentation of such data through different visualization techniques. Medical datasets and their rendering and visualization are important elements of a growing number of medical diagnosis, treatment planning, training and educational activities.

Traditionally processing large medical datasets would require direct access to a powerful graphics supercomputer tightly coupled to local display systems ranging from the standard desktop to large scale immersive facilities such as CAVE[6]. However, medical datasets continue to grow faster than Moore's Law[33], so whilst local processing capacities are increasing, such datasets can quickly overwhelm standard local computational infrastructures especially when users demand interactive data visualization and navigation.

### 1.3 Grid Computing

Increases in network speed and connectivity are simultaneously allowing more cooperation between geographically distributed resources. Such development breaks some of previous dependencies on the availability of local resources and offer new ways of working.

Today data is produced at different scales, arrives from various sources (sensors, instruments, simulations, databases, direct user manipulations, etc.) and bares different structures. Besides the sheer sizes, large-scale multi-regional and multi-institutional collaborations result in the distribution of this data around the world. Nowadays bandwidth grows faster than processing power. Both our work and everyday life are becoming more and more dependent on distributed computing. We irretrievably turn from computer individualists into distributed nominees interacting with complex high-dimensional datasets and other people over the network.

The Grid technology was introduced recently to enlarge concepts of distributed processing, visualization and integration of information from various sources. The Grid is an emerging infrastructure that supports the discovery, access and use of distributed computational resources[22]. Its name comes by analogy with the electrical power grid, in that the intention is that computational resources (by analogy with power generators) should be able to be accessed on demand, with the location and ownership of the resources being orthogonal to their manner of use.

Based on service-oriented architectures (e.g., WSRF (Web Services Resource Framework), OGSA (Open Grid Services Architecture))[28], Grid computing permits users to remotely access heterogeneous resources with-

out considering their underlying implementations. The linkage to Web services provides fresh challenges and opportunities for data delivery, resource scheduling, authentication, information management and close coupling of simulation and visualization.

Web services allow a high level of abstractions that hide implementation details from applications. Using this technology, applications invoke other applications functions through well-defined, easy-to-use interfaces. Web services allow to solve many integration problems, since each organization can concentrate on its own competence and still leverage services that other research groups provide.

Although the Grid was originally devised principally to support scientific applications, the functionalities associated with middlewares, such as Globus[25] and Unicore[26], are potentially relevant to applications from many domains, in particular those with demanding, but unpredictable, computational requirements.

## 1.4 Motivation and Goal

Large-scale scientific computing activities will become soon Grid-based over high-capacity and high-speed networks. With the increasing sizes of datasets generated through large-scale simulations, visualization starts playing an essential role in gaining the insight into the data of interest. To support remote and even collaborative data analysis making use of geographically distributed high-performance computing and storage facilities, appropriate Grid-based visualization tools are also required.

Placing visualization routines into the Grid environment allows geographically distributed users to access visualization resources remotely having a feeling that they run locally. Through the integration with the Grid technology, visualization process embraces a new level of interactivity with high detailed process control and releases the users from the limited resources of local desktop systems.

The goal of this Master project is to develop an interactive Grid-based visualization framework to allow end-users to exploit visualization functionalities irrespective to their geographical location, available computational resources and display systems, while the visualization experts are empowered to construct visualization functionalities for end-users.

Traditional visualization models do not comply needs and requirements of Grid computing oriented towards service-oriented architectures. Having this in mind, we extended traditional visualization pipeline of Haber and McNabb[23], with the feature of separated service modules to ensure more

efficient resource allocation and to provide users with more intuitive control over the visualization process. The VLAM-G[16] workflow system has been used for the interactive composition of visualization pipelines from available Grid-enabled services.

Services in the framework were developed using Globus Toolkit version 4 (GT4)[25]. The Globus Toolkit is an open source software toolkit used for building Grids. GT4's Java WS Core[25] is an implementation of the Web Services Resource Framework (WSRF). It provides APIs and tools for developing Web services. The hosting environment of services is a standalone container included in the GT4 package. Globus GridFTP tools are also used to build the data repository in the framework.

To test and validate the proposed framework the image-based analysis of vascular disorders has been chosen as the test case of this research. In particular, we experimented with two different medical datasets:

1. experimental data of the patient's vascular condition, and
2. simulated blood flow data for the selected region of interest.

The actual visualization part was performed using Kitware's Visualization Toolkit (VTK)[27]. Surface extraction method was used for the visualization of arterial structures and streamline-based visualization was applied for presenting results of the blood flow simulation.

## 1.5 Chapter Overview

The thesis is divided into 6 chapters. In chapter 2, an overview of related work is presented. The design and implementation of the service-oriented visualization framework is described in the chapter 3. A test case of visualizing medical data in the proposed framework can be found in chapter 4, along with performance measurements. Chapter 5 describes the work on integrating the framework with a workflow system VLAM-G, followed by conclusions and future work in chapter 6.

# Chapter 2

## Related Work

This chapter presents an overview of scientific visualization models, the concept of Web/Grid services, as well as related visualization projects.

### 2.1 Visualization Models and Systems

Visualization transforms data into images that efficiently and accurately convey information of the data. Visualization models describe the methodology and techniques of the visualization system as an order of commands transforming the data into the visualized image.

To create visualization for the data, the functional model and object model need to be built. Also sometimes the functional model is referred to as the visualization pipeline. From an object-oriented point of view, the functional model consists of objects to represent data, objects to operate on data, and an indicated direction of data flow. Data objects represent information, while process objects operate on input data to generate output data. The input to a process object includes one or more data objects as well as local parameters to control its operation.

Process objects can be further categorized into source, filter and mapper objects. Source objects interface to external data sources or generate data from local parameters. Filter objects take one or more input data objects and generate one or more output data objects. Mapper objects represent the sink in the functional model, which takes one or more input data objects and terminates the visualization pipeline. Mapper objects usually convert data into graphical primitives, while other time they may write output data to file or other system devices.

Source, filter, and mapper objects are considered to be the elements that can be connected in a variety of ways to construct models. After the visual-

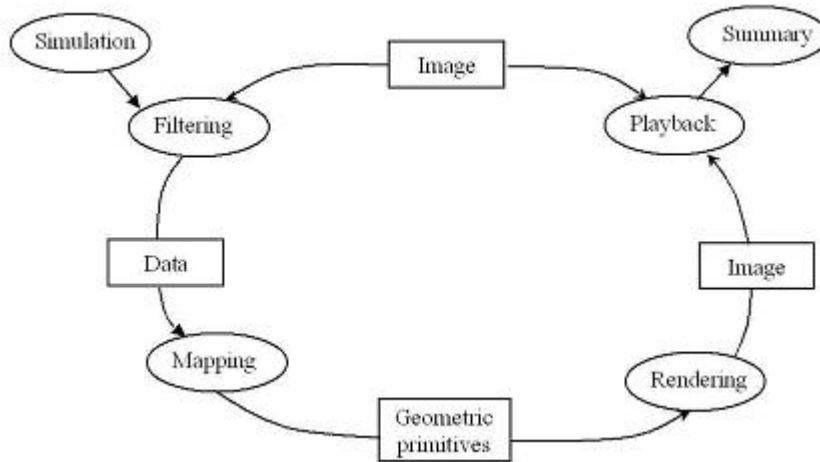


Figure 2.1: Visualization cycle model

ization model is properly constructed, it is executed to process data and to generate a desired result.

### 2.1.1 Visualization Cycle Model

The visualization process typically involves exploring, transforming and viewing data. First, raw data is acquired from some source. Next, the data is transformed by various methods, and then mapped to a form appropriate for the presentation to the user. Finally, the data is rendered or displayed by the graphics system.

A well-known model is visualization cycle model, introduced by Upson et al[13]. This visualization model is based on the notion of repeatedly performing simulations until a satisfactory set of results is produced, as shown in Figure 2.1.

### 2.1.2 Visualization Pipeline

Proposed by Haber and McNabb[23], the visualization pipeline, known as the traditional visualization model, is a model that describes visualization as a unidirectional pipe from a set of raw data to a visualization instance. Figure 2.2 illustrates the model.

This model takes the raw data into the visualization process and filters out (reduces) the unnecessary data so that to make the reduced data possible to map onto geometric representations. Finally, the geometry is rendered as images for final representation.

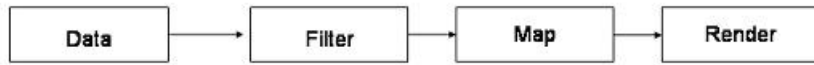


Figure 2.2: Haber-McNabb visualization pipeline

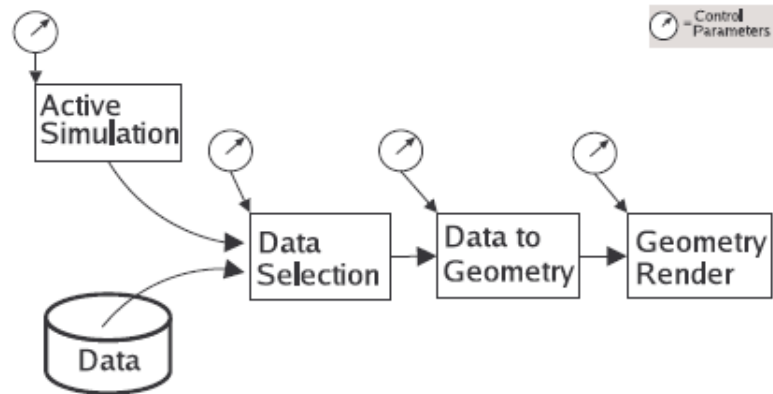


Figure 2.3: Dataflow pipeline with computational steering

### 2.1.3 Layered Reference Model

Brodie et al[15] proposed a layered reference model that based on the traditional pipeline concept as well. The model enables the computational steering by adding the element of active simulation to each stage, as shown in Figure 2.3.

The entire model has three layers, as Figure 2.4 illustrates: conceptual layer, where the dataflow pipeline resides; logical layer, where the binding of conceptual model to a particular configuration of software entities occurs; physical layer, which interprets the logical specification in terms of a particular Grid computing environment.

### 2.1.4 Selective Visualization Pipeline

Van Walsum[43] described the concept of selective visualization, a generic approach to feature extraction, the-state-of-the-art technique of flow field visualization. The selective visualization pipeline is shown in Figure 2.5.

The model takes the raw data into the pipeline to process. After the stage of attribute calculation, the features are extracted and original dataset is no longer needed. The achieved data reduction can be of a factor of 1000 or even more.

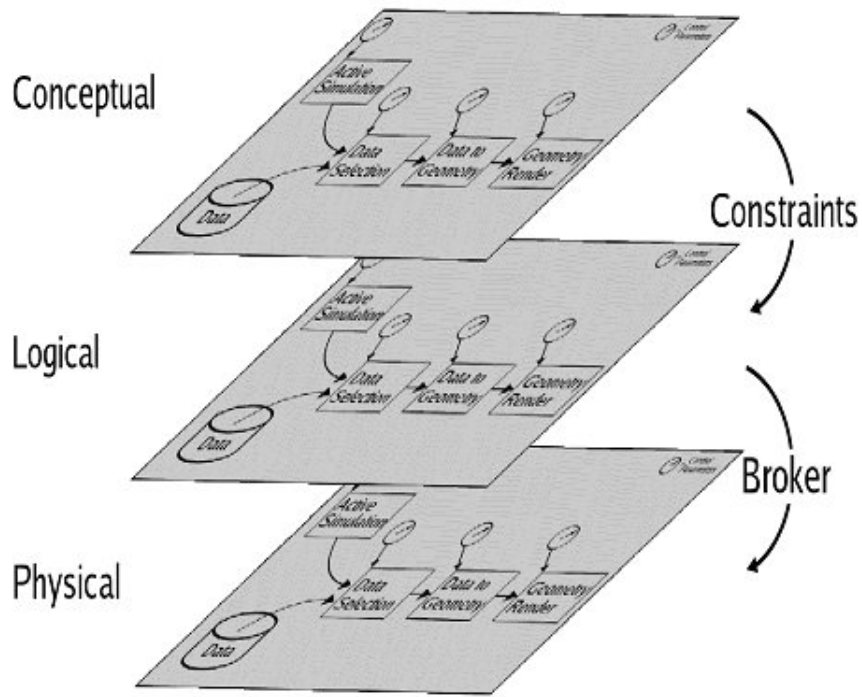


Figure 2.4: Layered reference model

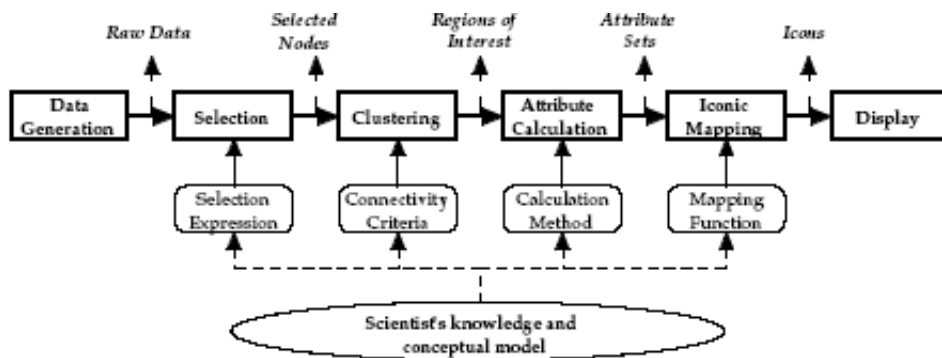


Figure 2.5: Selective visualization pipeline

## 2.1.5 Visualization Systems

Among aforementioned models, visualization cycle model and pipeline model are the most popular ones.

Examples of systems supporting the visualization cycle model are the NAG IRIS Explorer[18], AVS and AVS/Express[24]. These are so called modular visualization environments, which typically run on a single computer with a memory, a processor and graphics capabilities sufficient to handle the execution of each module.

Visualization systems based on the visualization pipeline concept are the Visualization Toolkit (VTK)[27] and the VIPAR package[44]. Their main advantage is that they allow building visualization routines that can be executed in parallel on several machines.

## 2.2 Web/Grid Services

### 2.2.1 Web Services

A service is a network-enabled entity that provides some capability in a distributed environment. Web services are software systems designed to support interoperable machine-to-machine interaction over a network. Web services incorporate a set of Extensible Markup Language (XML) based standards to integrate software applications over distributed environments. It has an interface described in Web Services Description Language(WSDL)[47]. The information is exchanged using messages that are enclosed in SOAP envelope. SOAP[39] stands for Simple Object Access Protocol, a network-neutral lightweight protocol for exchanging information in distributed environments. SOAP messages are conveyed using HTTP, a ubiquitous communication protocol used to transmit information over computer networks.

Using these open standards makes Web services language-neutral and platform-independent. This brings interoperability to distributed environments.

### 2.2.2 Grid Services

Grid computing is an emerging computing model that provides the ability to perform high-throughput computing by utilizing many networked computers to model a virtual architecture. Grids use the resources of many separate computers connected by a network to solve large-scale computational problems in distributed environments. The problems concerning the establishment of virtual organization are far more to be addressed by the current distributed

computing technologies such as Internet or virtual enterprise technologies like CORBA[1]. This gives the rise to Grid service technology.

Grid technology builds upon and extends the Web service technology, Service-oriented nature is possessed by both Web services and Grid services. To present the distributed components as services, two problems need to be addressed: the definition of service interface and the identification of protocol for invocation. While Web services focus on platform-independent description, discovery and invocation of components as services, Grid services focus on the dynamic discovery and efficient use of distributed computational resources. Web services cannot be created dynamically and are stateless to the service requester while Grid services support the dynamically created instances of service simultaneously accessible by multiple requesters.

Open Grid Services Architecture (OGSA) realizes the possibility of extending the Web service to a Grid service, making the functionality of Grid services available through Web services interfaces. To build the architecture, Globus toolkit and Web service standard are the technology of choice.

In this thesis, we will use term "services" to represent both "Web services" and "Grid services" for the generalization purpose.

## 2.3 Related Projects

When visualizing the medical information and computational fluid dynamics (CFD), large amounts of datasets generated from image techniques or simulation are to be dealt with. Even though techniques on reducing the size of datasets are applied to the field, the whole procedure is still considered computationally and resource intensive.

With the emergence of the Grid technology, computing resources can be utilized as required by scientific problems. Rather than running the visualization routine locally on a desktop, the challenge is to run it using distributed computational resources on the Grid, disregarding their physical locations. Aiming at this goal, researchers have been focusing on the integration of visualization systems within the Grid environment and the implementation of the visualization frameworks that allow this. Several projects aimed to Grid-enable visualization systems are discussed below.

### 2.3.1 Grid Visualization Kernel

Tirado-Ramos et al[11] presented the FlowFish package integrated with the Grid Visualization Kernel (GVK) to visualize the flow simulation data in a Virtual Environment like CAVE or DRIVE[3]. GVK is a middleware that

enables the use of visualization service within the Grid. It encapsulates the visualization pipeline into modules and distributes them among the Grid.

FlowFish is developed to visualize the blood flow of patient before actual medical operation. It is built as a collection of flowfield visualization classes and tools, which is written on top of VTK, for the use in immersive virtual environment.

GVK integrates the visualization system with Grid and is implemented using Globus services, providing Grid visualization services via dedicated interfaces. Integrating GVK functionality within the VTK pipeline of FlowFish enables the flowfield visualization service on the Grid and currently GVK classes for the glyph based visualization are implemented. GVK functionalities can be accessed via direct connection or dedicated portal service, including Migrating Desktop[7], a Grid portal service application that allows users to handle the Grid and local resources.

By replacing the VTK classes in the visualization stages with GVK-enabled classes, the visualization pipeline can be placed remotely on the Grid. This fulfills the goal of integrating visualization system with the Grid. However, currently only one visualization method has been implemented. This implies the lack of diversity.

### **2.3.2 VIBRO**

Knight and Munro[30] presented a combination of a visualization framework and a brokering system, known as VIBRO, which is built on the service mechanism and runs on the Grid. The implementation work is based on Jini, software developed by the Sun Microsystem, that supports the distributed computing over a network where the service mechanism is applied.

The visualization framework is essentially a Jini[29] client that is able to locate and utilize available services over the network. Services that are self-contained include data service, filter service, and visualization service. The client interface is implemented in Java using the standard Swing API, with composition window, lookup services, basic visualization service and console log that fulfills the interaction with users. Communication scheme is implemented using XML, an industry standard for data representation and communication among distributed systems.

The driving force of this work is the software visualization for program comprehension, which has a very little relevance to the scientific visualization of medical data. Also the issue of scalability would be one of the biggest concerns.

### 2.3.3 G-Viz

Brodie et al[15] presented, as a part of the gViz project, a similar scheme of visualization on the Grid. They based their approach on the layered reference model introduced in section 2.1.3.

The visualization system of choice is IRIS Explorer, upon which the extension to support remote visualization and computational steering in Grid is implemented. To describe the visualization at logical level, an XML application, skML, is developed. The SVG Map Editor is built for the creation of a visualization pipeline and is modeled in IRIS Explorer as a user interface. A nice thing about IRIS Explorer is that it allows distributed computing scheme, which gives the possibility for Grid middleware to be incorporated within the internals of IRIS Explorer, allowing the integration of a visualization application in a distributed computing environment.

### 2.3.4 VizWiz

Kolb[31] proposed an approach to visualize data over the Internet. This approach is also built on the traditional visualization pipeline and presents three scenarios of applying this pipeline over the Internet based on the client-server model.

The first scenario is basic, putting all the stages in a pipeline on the WWW server side and using HTML as the communication scheme. This is actually a process of accessing a Web site and downloading the result images as required, with which little amount of interactivity can be achieved.

The second scenario extends the visualization result from a 2D image to a 3D model and places the rendering stage on the client side using VRML[45] such that it gives more interactivity in the study of presentation of visualization results.

The third scenario is visualization software publishing and places all the pipeline stages on the client side. It defines the server as the only software publisher and the client has to provide all computational resources. One example of such work is Java applet VizWiz, a visualization tool that is platform independent and applicable over WWW. By using VizWiz, a tradeoff of performance must be made for the platform independence because of the nature of Java language. The method presented here provides an easy-to-use, platform-independent visualization system over the Internet, suitable for the basic use of visualization functionality. However, while placing all the computational tasks on the client side, it may not be the best candidate for the computationally intensive visualizations.

### **2.3.5 E-Demand**

Presented by Charters et al[4], e-Demand project focused on the use of service architectures for stereoscopic visualizations in distributed environments. Peer-to-Peer computing scheme was introduced in the project, which allows each node in the network to act as both a server and a client. The developed framework allows decoupling of the visualization design performed by end-users.

# Chapter 3

## Design and Implementation of the Visualization Framework

Web service is of a promising industry standard for distributed environments. Dividing the visualization pipeline down to service entities and their distribution over the network will lead to higher flexibility and better resource utilization. The service-oriented architecture of the visualization framework developed in this project is discussed in this chapter.

In order to validate the concept of a service-oriented visualization, the framework prototype has been developed. This chapter describes the design and implementation of the prototype.

### 3.1 Design Considerations

#### 3.1.1 Motivation

The visualization framework is aimed at assisting users in building visualization pipelines and their execution on the Grid. The goal of the framework is to allow not only visualization experts but also end-users to visualize and analyze large datasets irrespective to their geographical locations and available computational resources.

This research concern can be reflected in the following user scenario. An end-user (a radiologist or a surgeon) has the need to visualize a large dataset generated by medical imaging techniques like CT or MRI. He/She may not have prior knowledge in advanced visualization algorithms and/or sufficient amount of computational resources available at hand to carry the task. Thus, he/she decides to use our prototype.

The prototype provides users with a Graphical User Interface (GUI).

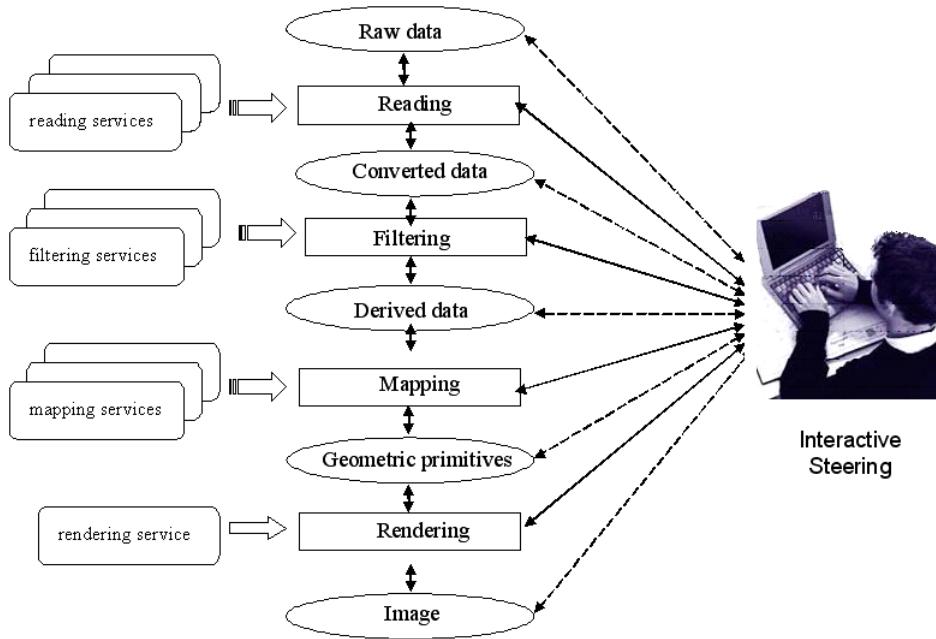


Figure 3.1: Interactive service-oriented visualization framework

Based on the nature of visualization tasks to be performed, the user selects appropriate services listed in the GUI window. He/She is able to construct his/her own visualization pipeline easily in a drag-and-drop fashion. Details of the visualization task can as well be defined at this stage. After the customized visualization task is validated, user submits a dataset to be visualized and starts the process. User can step through each service individually to inspect the intermediate results, or execute the whole visualization pipeline and receive the final visualization result only.

### 3.1.2 A Service-Oriented Visualization Model

We developed a service-oriented visualization model, which is shown in Figure 3.1. This model originates from the traditional visualization pipeline and is based on a Web service scheme in distributed environments[28]. Visualization pipeline model consists of objects to represent data (data objects), objects to operate on data (process objects), and a direction of data flow[38]. In the pipeline, process objects such as reader, filter, mapper and renderer objects are fundamental as they operate on input data and produce output data. To convert raw data into final presentation, certain intermediate objects have to be generated.

Our service-oriented visualization model identifies data and process ob-

jects in the pipeline, groups objects into major blocks, and then encapsulates object(s) into service entities such that pipeline can be constructed in terms of services. As services can be deployed at different geographical locations in a distributed environment and executed on demand, this allows the interactive steering of the pipeline execution and intermediate result to be explicitly produced for the desired examination and manipulation.

To allow configuring and controlling the pipeline as a whole, the forward flow has been replaced by a bi-directional one. Thanks to this, each service can be scheduled taking into account the amount of computation needs to be performed.

Interactive steering plays the central role in the proposed architecture. By steering we refer to changes that can be applied to the visualization pipeline through the adjustment of service parameters eventually affecting its output.

The benefit of the prototype lies in that it is well suited for both the work of visualization experts and end-users. Using the interactive visualization framework experts can compose or build visualization routines for end-users, who may later exploit them irrespective to their geographical locations and available computational resources. Thanks to steering interfaces, it becomes possible for experts to control every stage of the visualization pipeline, its input and output, and therefore to significantly facilitate the process of finding the optimal solution to the visualization problem. Moreover, resource sharing provides opportunities for the collaborative work, including the collaboration between experts and end-users.

### **3.1.3 Client-Server Architecture and Data Exchange**

Each entity in the model that represents certain stage in the visualization pipeline is encapsulated as a service. Services take data in certain format as input and produce the desired output. Each service behaves independently without the knowledge of the existence of other possible services. Once services are constructed and deployed at some service host (i.e., container), they are ready to be invoked. The framework is capable of performing the service invocation in a client-server fashion.

In a typical communication between a client and a service in the framework, a client is provided with the prior knowledge of services, i.e. services' URI's and operations that services carry. The actual invocation is done by SOAP messaging via HTTP protocol. More specifically, client sends a SOAP request message asking a service to perform the operation it carries. The service will generate a SOAP response message indicating the results.

During communication, the transfer of large amount of data will certainly take place. In principle, data can be serialized in the SOAP messages (or as

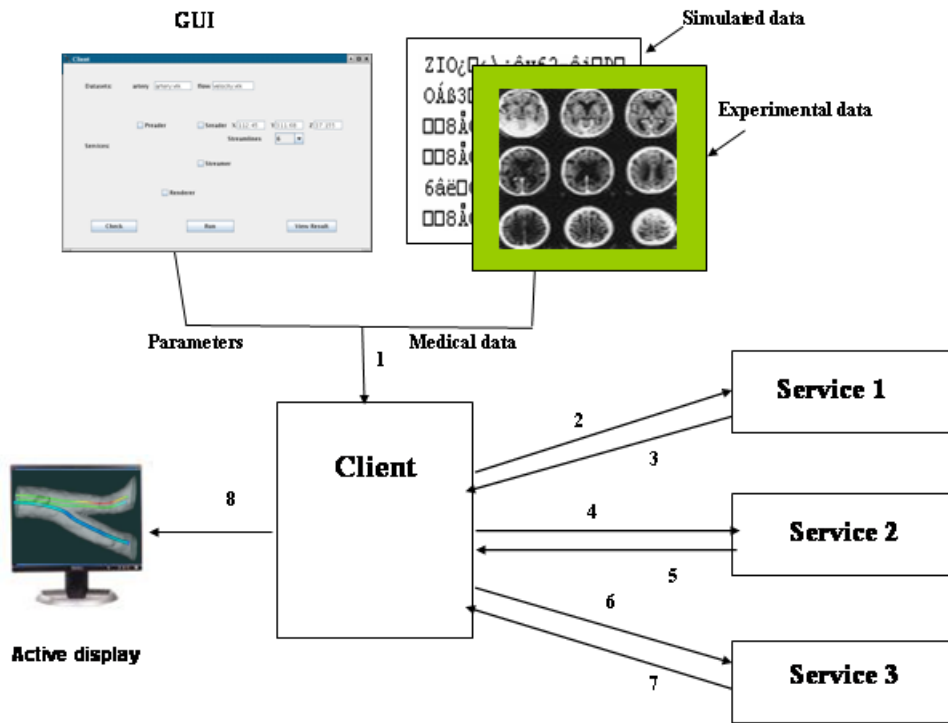


Figure 3.2: Image-based medical analysis

attachments) and sent over the network via HTTP protocol. However, when the size of data reaches a certain level, this is no longer an option due to the poor performance. For this reason GridFTP protocol was chosen for data transfer because GridFTP protocol has obvious advantage in the efficiency over other protocols and mechanisms.

### 3.1.4 Image-Based Medical Analysis

A possible scenario of an image-based medical analysis is shown in Figure 3.2. The numbered lines are steps of the analysis path. A user selects the medical data and visualization parameters (1). Client then in turn invokes the desired services 1-3 (2 to 7). Finally, the result from the last service is passed to the viewer (8) on an active display. During the analysis, user is able to examine the intermediate results generated by each service, steer the visualization towards the desired result.

Till now we have roughly described the design considerations of the framework. Details can be found in the following sections.

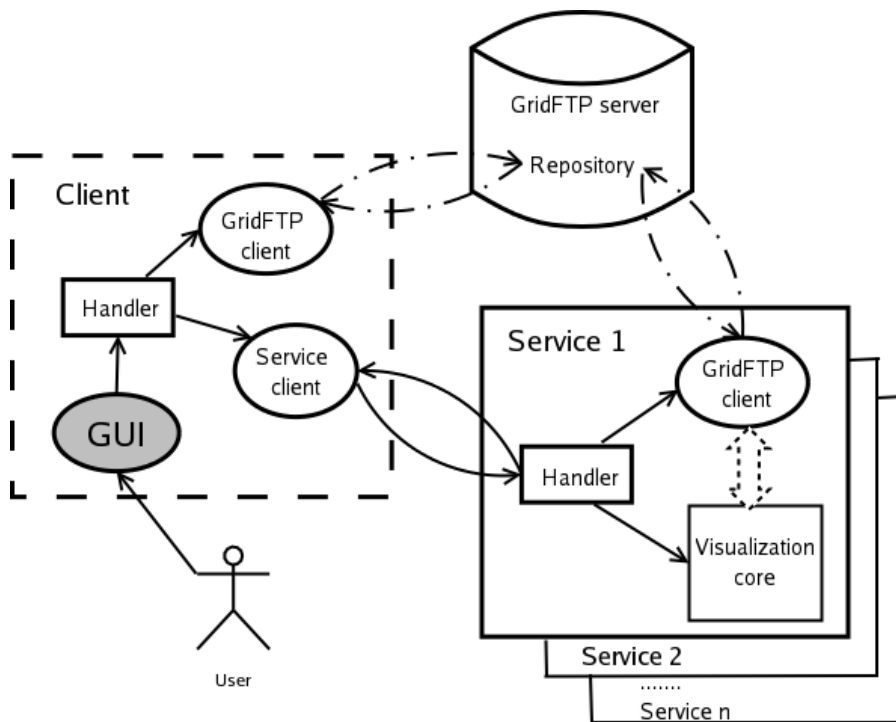


Figure 3.3: Architectural design of the framework

## 3.2 Design Issues

Design activities can be described from several perspectives.

In the architectural design shown in figure 3.3, components are a client, services and a GridFTP server.

- The client part includes a GUI to interact with users, a service client to invoke services and a GridFTP client to communicate with GridFTP server.
- Each service includes a visualization core and a GridFTP client, which provides visualization functionality and communicates with GridFTP server, respectively.
- GridFTP server acts as a centralized data repository for the raw data and results of services.

As mentioned earlier, GridFTP protocol was chosen to carry out the data transfer. To optimize the design and maximize the performance, a centralized data repository (a GridFTP server) has been of a design choice.

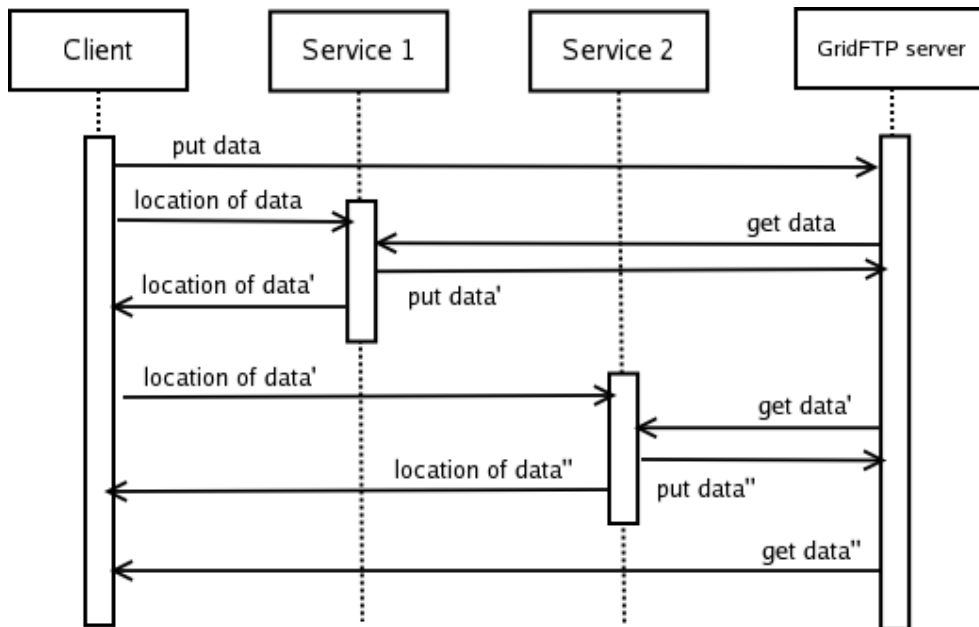


Figure 3.4: Data flow control in the framework

This repository is responsible for hosting the data from the client as well as data generated by each service. With this design consideration comes better data management and synchronization comparing with multiple repositories for the client and each service.

During the service invocation, instead of sending data directly to a service, the client stores data in the repository and sends SOAP message specifying the location of data to the service being invoked. Service will then in turn get the indicated data from repository to process. After data is processed, service will transfer the processed data back to repository and send a SOAP message specifying data location to the client or other services.

Since services bare no knowledge of the existence of others, the control and manipulation of services in the pipeline takes place on client side. To illustrate the control flow and data flow in the framework, an example is shown in the Figure 3.4.

The customized visualization pipeline consists of two services, which client will invoke in a batch mode. Before the actual service invocation, client transfers data to the GridFTP server. The host of the GridFTP server is a third party host other than those of services. However, in principle, the GridFTP server can be hosted on any of the service hosts, or even on the client host. After initiating the data transfer, client will invoke the first service by sending a SOAP message indicating the location of the data on the GridFTP server.

Once receiving the message, service 1 will communicate the GridFTP server to copy the data and store it locally. Next, service 1 will invoke the visualization core to process data. After the procession, output data will be saved in a new file and sent back to the GridFTP server. If completed successfully, service 1 will return client a SOAP message indicating the location of output data on the GridFTP server. Based on this returned SOAP message, client will then invoke the service 2 by repeating the above scheme. After the successful execution of the service 2, the pipeline processing is finished and the client will get the final result as an image file from the GridFTP server.

### 3.2.1 Service Design

As aforementioned, a service entity consists of a visualization core and a GridFTP client. On top of this, there is a service handler which is in charge for responding and processing the client invocation.

Visualization core objects are essential for the service design as they are responsible for representing the visualization functionalities in the pipeline. Simply put, the goal of service design is to divide the whole pipeline into stand-alone entities that represent certain data or process objects in the pipeline, which are eligible for client-server computing scheme in distributed environments.

While constructing a pipeline to solve the visualization problem, certain objects can be observed. The raw data may be read, filtered, mapped and finally rendered into an image presentation. Along the pipeline, intermediate data objects are implicitly outputted by process objects and inputted to the next process object further down the pipeline. Thus, we first need to identify process objects and possible implicit data objects in the pipeline.

The identified objects are potential candidates of service entities. A service entity may consist of one or more process objects depending on the visualization problem and the issue of resource utilization. The rules of thumbs are as follows. Objects that bare immediate relations should be grouped together, and computationally intensive objects can be set apart to leverage the use of resources. Another concern is data objects. In a visualization pipeline, data objects are implicitly passed by process objects as input/output ports. For each service entity, it needs to deal with its input and/or output explicitly and independently (i.e. read from a file and write to another one). For this reason, process objects in the service are wrapped with certain reader and writer objects to take their input and generate desired output.

After the objects in the services have been defined, the preparation of the visualization core is finished. Objects' methods and attributes (i.e. specified parameters) can be declared as operations and messages in the services'

interfaces.

## 3.3 Implementation Issues

### 3.3.1 Development Tools

For the development of visualization services, we used Globus Toolkit version 4 (GT4)[25] and Java programming language. The visualization system of choice was Visualization Toolkit (VTK)[27].

#### **Globus Toolkit**

The open source Globus Toolkit is a fundamental enabling technology for the "Grid," letting people share computing power, databases, and other tools securely on-line across corporate, institutional, and geographic boundaries without sacrificing local autonomy.

The Globus Toolkit has been developed to support the development of service-oriented distributed computing applications and infrastructures. Version 4 of the Globus Toolkit, GT4, represents a significant advance relative to the GT3 implementation of Web services functionality.

GT4 is a set of software components that implement Web services mechanisms for building distributed systems and can be used to build both service-oriented applications and service-oriented infrastructures. It includes a Java WS core component that provides Web/Grid services container as a hosting environment for GT4 standard services and customized services of users. Our visualization services have been developed using GT4 and deployed in GT4 Web service container.

GridFTP[25] tools are the data transfer components in the Globus Toolkit. GridFTP is a protocol defined by Global Grid Forum that provides secure and efficient transfer of data over the Grid. The implementation of the GridFTP client and server along with a set of development libraries are included in the Globus Toolkit. We used GridFTP tools to build the data transfer functionality in our framework.

#### **VTK and Java**

The Visualization ToolKit[38] is an open source, freely available software system for 3D computer graphics, image processing, and visualization. VTK consists of a C++ class library, and several interpreted interface layers including Tcl/Tk, Java, and Python. The design and implementation of the library has been strongly influenced by object-oriented principles.

VTK supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods; and advanced modeling techniques such as implicit modelling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. In addition, dozens of imaging algorithms have been directly integrated to allow the user to mix 2D imaging / 3D graphics algorithms and data.

To work with services using GT4 package, VTK application has been chosen to be built in Java. Though C++ is supposed to be more efficient, it is not the first choice in the service realm as it could be sometimes troublesome.

### **Mesa 3D Graphics Library**

For services that involve rendering process (i.e., transformation from geometric primitives to final presentation), off-screen rendering can be applied. Off-screen rendering provides users with the final presentation of the visualization process as static imagery. Off-screen rendering functionality is provided by Mesa 3D graphics library[32], an open source implementation of the OpenGL specification, Mesa library allows VTK to generate 3D imagery without having to open a window on the display, enables the development of off-line, batch-oriented renderer in VTK.

## **3.3.2 Development Steps**

### **Building Services with GT4**

To build services with GT4, certain development steps have been followed[40]:

1. The service's interface has been defined in WSDL. WSDL file describes a Web service's interface (also known as Port type) including declarations of available operations, messages and other parts. For the service client, WSDL file provides all necessary information on how to invoke the service. For the encapsulated objects in a service, its methods and attributes are mapped to the operations and messages that service carries.
2. Implementation of the service. The second step was to implement the service in Java.
3. The deployment parameters have been defined in the Web Service Deployment Descriptor (WSDD). By defining the deployment parameters, WSDD makes a service package available in a Web service container. It addresses the question of how services should be deployed, i.e. what is the name of a service, what is the encoding style, etc.

4. Compilation of the source. With all the pieces ready, the source was compiled into a Grid Archive (GAR file), which includes the stub classes generated by GT4.
5. Deployment of the service. The last step was to deploy the service GAR file into GT4's service container. When the container is up and running, service is ready to be invoked.

### **Client**

Service client is developed along with the service. Necessary classes are imported by both service and client such that in the later deployment stage, service and client can be distributed apart from each other on the Grid. GUI is developed using Java Swing package.

# Chapter 4

## Test Case

This chapter describes a test case chosen for the validation of the service-oriented visualization framework, the flow visualization in a simulated vascular reconstruction environment.

Surface rendering is used for the visualization of arteries in the patient's body. Streamline extraction technique is used for the blood flow visualization.

### 4.1 Medical Analysis

#### 4.1.1 Vascular Disorders

Vascular diseases affect arteries and veins[35]. Arteries and veins are flexible tubes that transport blood throughout the human body. Vascular disorders can take place in arteries as well as in veins. Aneurysms and stenosis are two categories of vascular disorders in general. An aneurysmal disease is a balloon like swelling in arteries while stenosis is a narrowing or blockage in arteries. The treatment of vascular reconstruction is to redirect and increase blood flow or repair arteries if necessary.

#### 4.1.2 Data Preparation

In the treatments of vascular diseases, there are situations in which the possible course of action is not clear to the surgeon. Treatments are with risks, which must be avoided as much as possible. An environment in which a real-life situation can be simulated and several treatments can be tested, will provide useful clues to the surgeon. By applying visualization to a patient's vascular condition, surgeons are able to examine the patient's blood flow condition prior to the real operation.

Medical data visualization starts with the acquisition of data by means of medical imaging. Among popular imaging techniques applied in medical treatment, Computed Tomography (CT)[34] and Magnetic Resonance Imaging (MRI)[10] are most widely used. They produce so-called experimental data of the patient's vascular condition.

The 3D data generated by CT or MRI is sent to a simulator, which operates on this 3D data and defines its own formats for input (geometry to calculate flow) and output (velocity, pressure or shear stress values of a blood flow). The simulator, developed at the SCS group of the University of Amsterdam, is based on the Lattice-Boltzmann method(LBM)[37], a mesoscopic approach for simulating fluid flow based on the kinetic Boltzmann equation. The approach is fast and accurate such that it is able to assist surgeons to make quick and correct diagnosis.

To convert the medical scans into LBM-grids, the raw data is first segmented so that only the arterial structures of interest remain in the dataset. The segmented dataset is then converted into a mesh that can be used in LBM and a simulator generates patient's blood flow. Since, in this project, the visualization system of choice is VTK, simulation data has been converted into VTK's file formats, which are readable by the VTK visualization algorithms.

## 4.2 Visualization of Medical Data

### 4.2.1 Visualization of Experimental Data

The medical imaging data to be visualized is of 3D spacing (x, y and z dimensions). The collection of the values representing three dimensions is referred as a scalar field. Scalars are single data values associated with each point and/or cell of a dataset. To visualize the volumetric data represented by scalar fields, many visualization techniques can be applied. Among these techniques, surface rendering and volume rendering are commonly used.

Constant scalar value in a scalar field can be contoured into regions with boundaries. These boundaries correspond to contour lines in 2D space or surfaces in 3D space. Three-dimensional contours are also called isosurfaces, which can be constructed by many polygonal primitives. Marching cubes[8] is one of the commonly used algorithms to generate isosurface for a 3D dataset. The algorithm assumes a finite number of ways of an isosurface passing through a cell. Based on this assumption, algorithm will march through data domain cell by cell and determine the relations between surface and each cell. This procedure will construct independent geometric

primitives in each cell. Isosurfacing is a simple yet effective method to visualize scalar data and one of the best candidates for visualizing medical data representing body tissues such as skin, bones and internal organs.

Volume rendering, in a broad definition, also represent methods that visualize volumetric data. However, here we refer to it as direct volume rendering, which is a process that operates directly on the dataset to produce an image without generating an intermediate geometric representation[38], as opposed to geometric primitive rendering techniques such as aforementioned isosurfacing. The efficiency and compactness issue of direct volume rendering are of importance due to its high complexity comparing with surface rendering.

For efficiency advantage of surface rendering over volume rendering, the former one has been chosen to be used for the visualization of the patient's vascular condition.

## 4.2.2 Flow Visualization

Flow visualization is an attractive subfield of visualization. Data generated during flow visualization can be extremely large. Applications of flow visualization are enormous, ranging from aerodynamics to weather simulation including an important field: Computational Fluid Dynamics (CFD). The wide range of applications results in a variety of techniques applicable to the fields, spanning multiple dimensions such as 2D and 3D techniques, and techniques for steady and time-dependent data.

CFD is an analysis of systems involving fluid flow by means of computer simulation. Visualizing the CFD simulation data has to address several challenges, namely the large size of data, geometric versatility, interaction, etc. In this setting, when visualizing the CFD data, users are interested in interactive control of flow visualization results. This trend has given the rise to the term “interactive visualization”, which enables technical users to explore their data and computations through visualization, and allows the exploration, analysis and refinement of visualization results towards the individual needs.

Most of flows are time-dependent. Changes are recorded by taking several snapshots of the flow separated by a constant period of time. It is the job of visualization to reveal the general attributes and the properties of area of interest to users by intuitive visual presentations. Searching for certain phenomena in time and space is an important task of flow visualization.

To study the flow, phenomena like vortices, singularities and boundary layers are of the interesting features. Vector field topology is a feature extraction algorithm for analyzing the property of the flow. The algorithm determines the critical points (where velocity is equal to zero)[35] in the

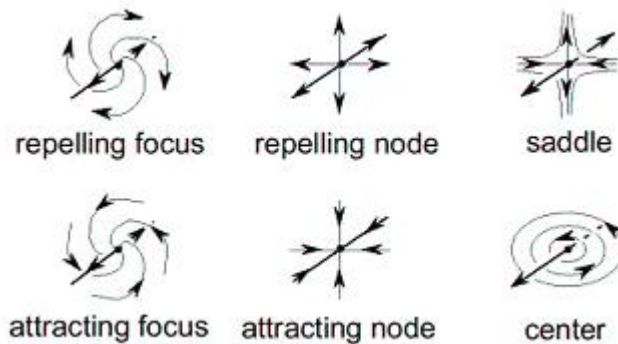


Figure 4.1: Critical points of the flow

flow, which are the starting points of streamlines generated for visualizing the flow. Figure 4.1 shows the scheme of critical points.

In a broad term, flow visualization techniques can be categorized into four groups that are listed below:

- Direct flow visualization
- Texture-based flow visualization
- Geometric flow visualization
- Feature-based visualization

Feature-based visualization techniques seek the way of automatically deducing the special features of flow without human intervention. That is why they are also called automatic feature extraction techniques.

In the category of geometric flow visualization, various techniques can be applied. The simplest is to display the velocity field data itself, either as displacement vectors using such things as arrow glyphs or magnitude scalar values using image, surface, or volume visualization techniques (mapping values to color, size, or position).

The next most common technique is the generation of streamlines based on a static velocity field. User selects seed locations (often along a line or at 2D grid locations), and computes a path for each seed point through the field, maintaining a continuous tangent to the flow field.

Besides lines, we can use ribbons and tubes to indicate the streams as well. Other attributes of the field such as magnitude or vorticity, can then be mapped onto attributes of the streamribbons or streamtubes, such as size or twist.

Streaklines are often represented as a continuous stream of particles emanating from a discrete set of points and flowing through the field. Individual points in a given trace (all particles coming from a particular location belong to the same trace) may be identified by color-coding to help distinguish related points which get separated when entering areas of high velocity.

The streamline extraction has been chosen for the visualization of the simulated blood flow data. The reason is that it is a CFD visualization technique where interactivity plays an important role.

### 4.2.3 Streamline-Based Visualization

The results generated by flow simulation contain both vectors that represent the velocity of blood flow, and scalars that represent the magnitude of the velocity field in three dimensions. To visualize the vector field, streamline extraction technique and some of its enhancements can be applied. In general, term streamline extraction is referred to the method of tracing trajectories in a vector field. Its result is a numerical integration of a particle trace represented as a line.

Streamlines are always computed based on numerical integration methods [41], where the integration step together with the speed of flow determines the detail of the particle trace. The smaller the step is, a more detailed streamline is generated. However, the achieved accuracy may cost additional computations and be very time consuming. On the other hand, when a big integration step is applied, it may lead to neglecting interesting areas.

The service-oriented architecture provides a nice solution to this problem. The use of high-performance resources available on the Grid allows computing of several parameterization schemas simultaneously in order to find an optimal solution. Thanks to interactive steering a user may decide to stop, pause or continue the execution of visualization in order to concentrate only on those ones, which are of the most interest.

Streamline visualization helps a researcher to investigate how the flow curves and whether it diverges or converges at specific points. To assist in the analysis of the flow behavior, a tube with rectangular shape can be wrapped around the path to make a streamline better visible. By tuning the service parameter users will be able to interactively change the number of streamlines, their location and radius and apply different color palettes to the ongoing visualizations.



Figure 4.2: A surface extraction pipeline and the final image result

## 4.3 VTK Pipelines

To visualize a patient’s artery and a blood flow using VTK, two basic pipelines have been constructed.

### 4.3.1 Visualizing an Artery

Patient’s artery is visualized by applying surface rendering. Visualization pipeline is constructed as shown in Figure 4.2, left. The dataset that represents an artery is of structured points file format. Therefore, a reader object called `vtkStructuredPointsReader` is applied. The purpose of `vtkStructuredPointsReader` object is to ingest data, create a data object, and then pass the object down. `vtkContourFilter` is applied next. It is a filter object that generates isosurfaces or isolines on the input dataset. Further down the pipeline contains the `vtkPolyDataMapper`, a mapper object that maps polygonal data (in this case passed by `vtkContourFilter`) to graphics primitives for graphics hardware or rendering software. The last one in the pipeline is `vtkRenderer`, an object that renders geometric primitives into a final image representation (see Figure 4.2, right).

### 4.3.2 Visualizing a Blood Flow

Simulated blood flow is visualized by applying streamline extraction technique. In the pipeline, a `vtkStructuredPointsReader` object reads the dataset and then passes a data object to a `vtkStreamLine` object. `vtkStreamLine` is a filter object that generates streamlines for an arbitrary dataset. It has an important variable `StepLength`, which defines the time increment used to generate individual points along the streamlines. Smaller `StepLength` results in smoother streamlines but with more line primitives generated. It is one

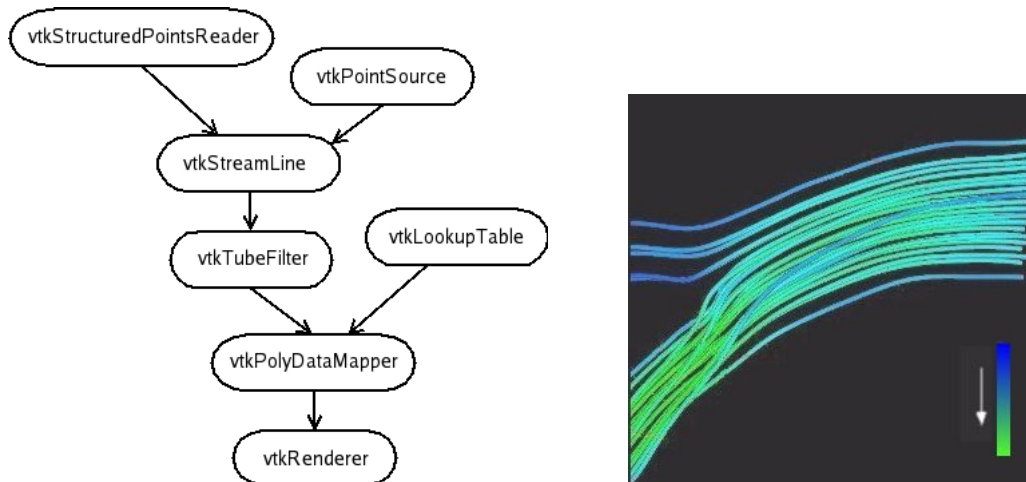


Figure 4.3: A streamline extraction pipeline and the final image result

of the decisive aspects of the performance of a streamline visualization application. In order to yield an interesting result, positions of the streamlines in the dataset need to be specified. A `vtkPointSource` object is passed to `vtkStreamLine` to serve this purpose. `vtkPointSource` is a source object that generates random points in a sphere space with the specified number and radius. The position of the point cloud varies depending on the subject of interest. For a better image presentation, `vtkTubeFilter` is applied next. It is a filter that generates a tube wrapped around each streamline. By combining the `vtkStreamLine` with `vtkTubeFilter`, streamtubes can be generated for the dataset. Further down the pipeline contains `vtkPolyDataMapper` and `vtkRenderer`, serving similar purposes as in the pipeline of surface extraction. A `vtkLookupTable` is passed to `vtkPolyDataMapper` to apply the color to the streamtubes. In the project, we experimented with the velocity of the blood flow. Red color corresponds to the highest velocity value and blue corresponds to the lowest. The complete pipeline and its execution result are shown in Figure 4.3.

Aforementioned two pipelines can be combined together to visualize the artery and the blood flow running inside, where streamlines (tubes) are aligned with the geometrical representation of the artery (See Figure 4.4).

## 4.4 Service-Oriented Pipelines

On the conceptual level, pipelines shown in Figures 4.2 and 4.3 can be divided into major blocks, each of which is then encapsulated as a visualization core

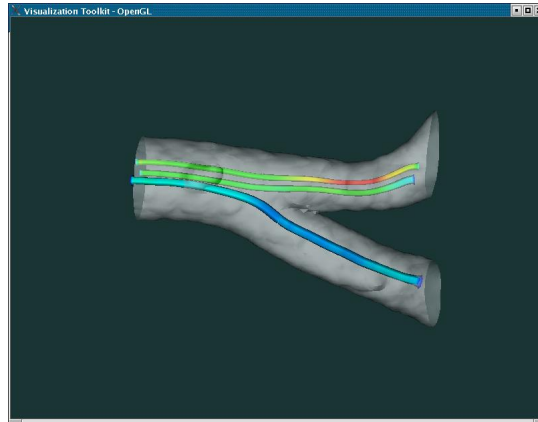


Figure 4.4: Combined visualization of a selected zone of an artery and a blood flow calculated for this particular zone

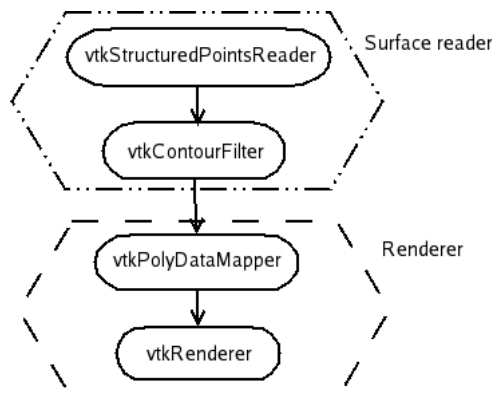


Figure 4.5: Main blocks of the surface extraction pipeline

of the service. Blocks are then given conceptual names to represent services. Figures 4.5 and 4.6 illustrate the division of pipelines.

The first pipeline consists of three processes: reading of the raw dataset, extraction of the surface from this dataset and mapping and rendering geometric primitives into final image presentation. Therefore, this pipeline is divided into two major blocks: a surface reader and a renderer (See Figure 4.5).

The first block consists of `vtkStructuredPointsReader` and `vtkContourFilter`. On top of that, it is wrapped with a writer object `vtkPolyDataWriter` to write polydata into an intermediate dataset. The second block consists of `vtkPolyDataMapper` and `vtkRenderer`. It is enriched with a reader object `vtkPolyDataReader` to read polydata from a dataset (e.g. dataset generated in the first block). In such way, two blocks can be executed separately (run

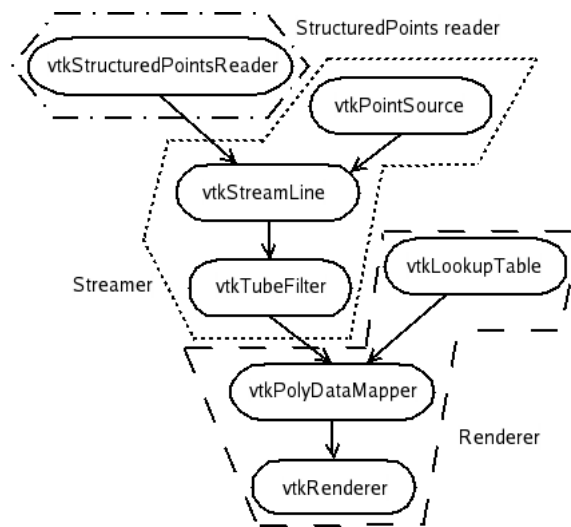


Figure 4.6: Main blocks of the streamline extraction pipeline

on different hosts) and perform the same functionality as if they run together.

As for the second pipeline, the conceptual process is similar: reading the raw dataset, generating streamlines (streamtubes), and mapping and rendering into image. The pipeline is divided into three major blocks: a structured points reader, a streamer and a renderer (See Figure 4.6).

The first block is `vtkStructuredPointsReader`. The second block consists of `vtkStreamLine`, `vtkTubeFilter` and `vtkPointSource` wrapped with a `vtkPolyDataWriter`, which writes polydata into a dataset. The third block consists of `vtkPolyDataMapper` and `vtkRenderer`, which map and render primitives into image. This block serves the same functional purpose as its equivalent in the first pipeline.

The reader and writer objects embedded in each service are key factors of the pipeline division as they serve as an interface between the pipeline and the intermediate datasets. A fragment of the Java code for a reader and a writer is provided below.

```

//A vtkStructuredPointsReader object
vtkStructuredPointsReader reader = new vtkStructuredPointsReader();
//reads dataset from file s
reader.SetFileName(s);
reader.Update();
...
...
//A vtkStructuredPointsWriter object
vtkStructuredPointsWriter writer = new vtkStructuredPointsWriter();
writer.SetInputConnection(vect.GetOutputPort());
  
```

```
//writes dataset as spw file
writer.SetFileName(output + "spw");
writer.Write();
```

In total four visualization modules have been created: a surface reader, a structured points reader, a streamer and a renderer (applicable to both pipelines). These modules were then put into service packages that bare the same names, respectively.

## 4.5 Performance Measurements

This section discusses the measurements of the performance of the VTK pipeline execution on the medical datasets of different sizes (simulated blood flow), as well as the performance of visualization services measured for the same datasets.

### 4.5.1 VTK Pipeline Execution

For performance measurements we experimented with the VTK pipeline of the blood flow visualization shown in Figure 4.3. Three structured points datasets being visualized are of different sizes, namely,

- Small: 856KBytes with 81874 point-data;
- Medium: 31MBytes with 1314648 point-data;
- Large: 67MBytes with 2495004 point-data.

The experimental testbed includes workstations with different hardware configurations:

- Owf
  - processor: Intel(R) Pentium(R) 4 CPU at 2.80GHz
  - memory: 513896 KB
  - graphics card: GeForce4 MX440 AGP8x with 65536KBytes videoRAM
- Torn
  - processor: Intel(R) Xeon CPU at 3.0 GHz
  - memory: 4128248 KB
  - graphics card: nVidia Quadro FX3000 with 262144KBytes videoRAM

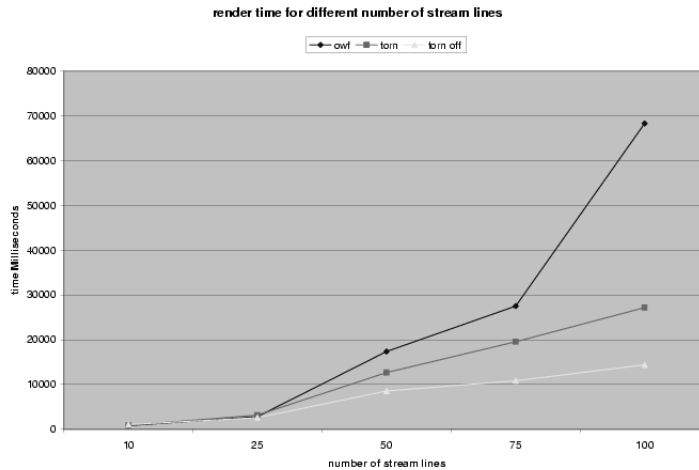


Figure 4.7: Render time of a small dataset with different number of stream-lines

The VTK pipeline has been executed on both workstations and render time has been measured. The rendered images are of 800x600 pixel and render time is reported by vtkRenderer. On workstation Torn, on-screen and off-screen rendering modes were both performed (represented by legends “torn” and “torn off”, respectively, in the following figures) while only on-screen rendering was performed on Owf (represented by the legend “owf”). Results are shown as follow.

Figures 4.7-4.9 show the results of render time depending on the number of streamlines to be visualized. For the better comparison, different values of the StepLength variable in streamline generation (vtkStreamLine) have been applied to each dataset (small: 1.0; medium: 100; large: 200).

It is clearly shown that render time increases along with the increase of the number of streamlines being visualized. The differences between two workstations start to become significant when the number of streamlines is between 40-50. When the number of lines reaches 100, the differences are even more than doubled. Another fact indicated by the graphs is that render time when using off-screen rendering is less than when on-screen rendering is used. The reason behind this lies in the fact that in the off-screen mode, Mesa library does software rendering through Xlib API using main RAM, while in on-screen mode, data has to be moved from main memory to the

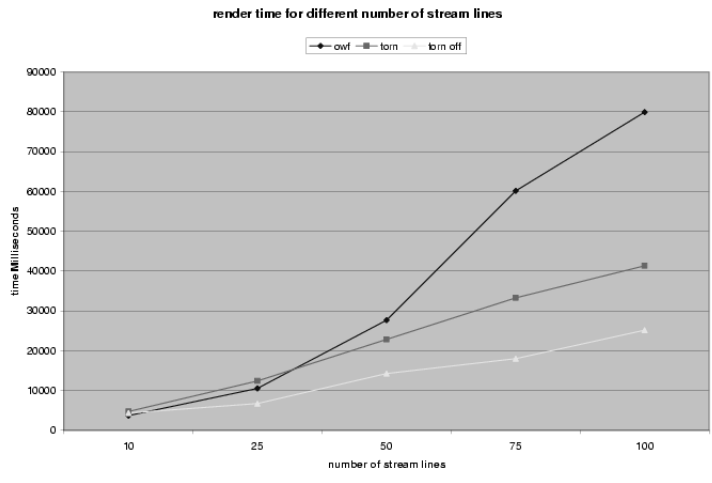


Figure 4.8: Render time of a medium dataset with different number of streamlines

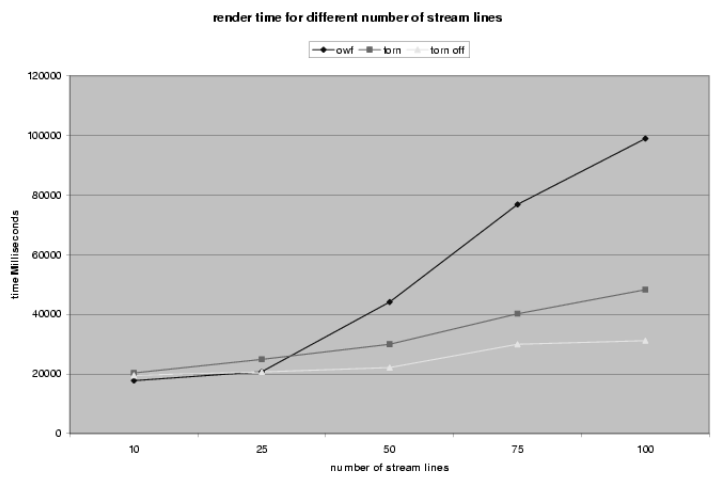


Figure 4.9: Render time of a large dataset with different number of streamlines

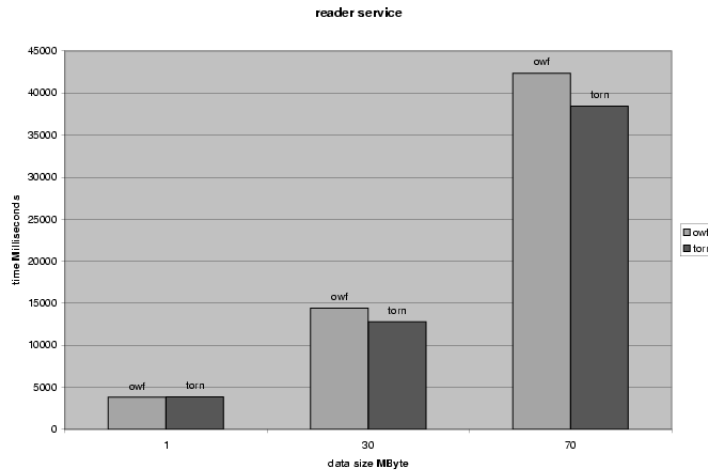


Figure 4.10: Execution time of a reader service on different datasets

graphics card memory, which adds more time.

Above findings has motivated us to execute render service on Torn using off-screen rendering mode to achieve better performance.

## 4.5.2 Service Execution

Services designed for visualizing the blood flow are deployed on Owf and Torn. Their execution time was measured and results are shown in the figures below.

For the comparison reason, the StepLength variable of the vtkStreamLine object is set to 150 for all three datasets. Figures 4.10-4.12 show that the difference of execution time on two hosts is minor when small dataset is applied, while it is rather significant when medium and large datasets are applied instead. This indicates that both hosts (Owf and Torn) are capable of performing light-weight task but the more powerful Torn suits better for heavy-weight tasks. Comparing the execution time of three services, we can find that filter and render services take more time than reader service. So, ideally both filter and render services need to be assigned to more powerful machines. However, render service still remains the most time consuming.

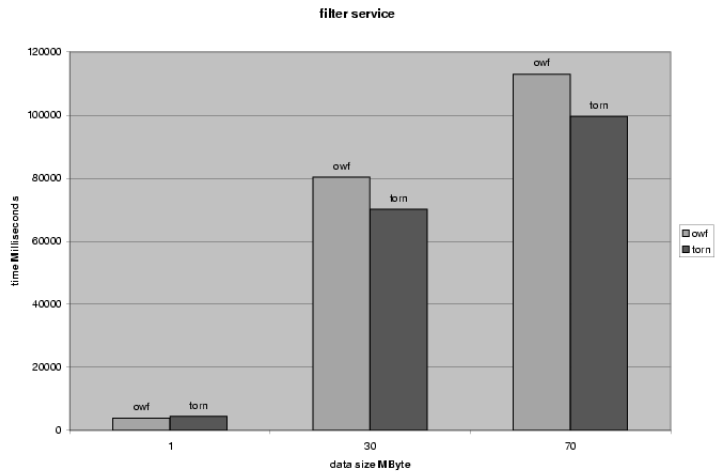


Figure 4.11: Execution time of a filter service on different datasets

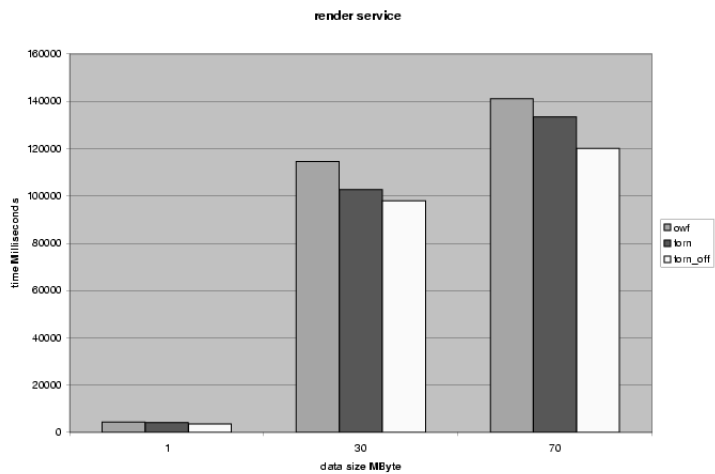


Figure 4.12: Execution time of a render service on different datasets

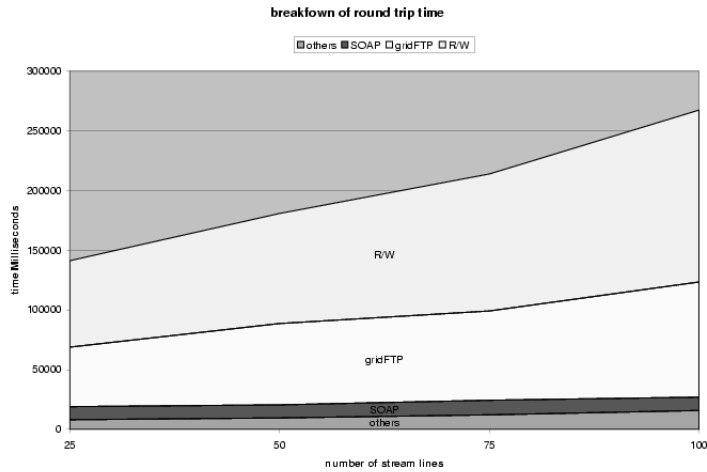


Figure 4.13: Components breakdown of round trip time

### 4.5.3 Breakdown of Round Trip Time

Motivated by experimental findings, we deployed render service on Torn and two other services on Owf. Round trip time is defined as the elapse time from the point client starts data transfer to the point client receives the final image. It can be measured by inserting timestamps into the source code using Java timing. By comparing the values reported by those timestamps, time taken by different objects and/or processes can be quantified respectively. In such way, total round trip time can be broken down to major components with their own contribution as shown in Figure 4.13.

In the breakdown of round trip time, components that have major contributions are Reading and Writing (R/W), GridFTP transfer, SOAP messaging and Others:

- R/W: reading and writing of data objects
- GridFTP: gridFTP data transfer
- SOAP: SOAP messaging
- Others: issues such as networking, VTK library loading, etc.

It is clearly shown that R/W and GridFTP are the two components that have the most contribution to round trip time. In the blood flow visualization

pipeline, intermediate data objects are generated by reader and filter services. Depending on visualization parameters such as number of streamlines being visualized and the StepLength variable, the size of these data objects can be extremely large and up to the difference of an order of magnitude comparing with the raw datasets. Reader and writer objects have to explicitly read and write these data objects from/to a file. As a result, the time spent on reading and writing could be significant.

In addition to reading and writing, data objects generated by services have to be moved from/to the data repository. This is done by GridFTP data transfer and the time spent on data movement is decided by the size of data being transferred and transfer efficiency.

In round trip time, the proportion of time spent on SOAP messaging remains almost the same. This is because only control information, such as indication of the data and parameters to be applied on data, is being sent in SOAP messages over network. Other issues like library loading also contribute to the round trip time but with a rather minor proportion.

# Chapter 5

## Interactive Composition of a Visualization Pipeline

This chapter describes the interactive composition of a visualization pipeline in a scientific workflow system with the aid of a graphical workflow editor.

### 5.1 Scientific Workflow Systems

Simply put, workflow[46] is the movement of documents and/or tasks through a work process. It involves the aspects of how tasks are structured, their relative order, and how information flows along the tasks. Workflow paradigms can be categorized into "scientific" and "business". The scientific workflow is mostly concerned with throughput of data through algorithms, applications and services.

Scientific workflow is widely accepted in the fields of bioinformatics and cheminformatics as they met the need for multiple interconnected tools, handling of multiple data formats and large data quantities. Scientific workflows are process networks that describe series of structured activities and computations aroused in scientific problem-solving, which usually involves the invocation of a number of analysis tools, typically in a routine manner. With the emergence of the term scientific workflows, the need of toolkits and environments for describing and managing workflows are also raised. This has led to the birth of another term, scientific workflow systems.

Scientific workflow systems[49], ideally, allow scientists to plug-in scientific data resource and computational service into a scientific workflow, inspect and visualize data as it is computed, apply changes to manipulate the final products. Thus, a scientific workflow system can be seen as a scientific problem-solving environment applicable to distributed and/or service-

oriented infrastructures.

A basic requirement for scientific workflow is seamless access to remote resources and services. Since Web services are emerging as the standard for remote service execution in loosely coupled distributed systems, scientific workflow systems are mostly enabled to handle web services. Examples of such are KEPLER[12] and VLAM-G[14].

In this project, VLAM-G was chosen for the integration of developed visualization services. We decided not to use KEPLER because it is mostly oriented for building big distributed frameworks, which is in our case certainly an overkill.

## 5.2 VLAM-G

### 5.2.1 Introduction

The Grid-based Virtual Laboratory AMsterdam (VLAM-G) provides a science portal for distributed analysis in applied scientific research. It allows scientists to conduct their experiments in a familiar environment, with remote experiment control, data management facilities and access to distributed resources and services. The main goal is to provide a unique integration of existing standards and software packages. It incorporates and integrates the most recent developments in Grid computing (such as the Globus Toolkit), database technology and visualization techniques.

VLAM-G provides a modular data flow-based system. The platform is designed to be easily adapted to include new application domains and different groups of users. A modular architecture is essential to establish such a flexibility. Supporting inter-disciplinary interactions implies the composition of (software) modules, which may have been developed independently. In order to ensure that only proper connections can be established, the constituting modules communicate using a strongly typed communication mechanism.

The VLAM-G design promotes the use of services offered by the Globus Toolkit, which is the de facto standard in Grid computing. It provides a GUI; structures to encapsulate and interface with experimental specific software and hardware; data storage and visualization systems. VLAM-G offers an assistant to guide scientists through the process of structuring their studies by giving templates to outline the different steps to perform the entire data handling process.

VLAM-G introduces the concept of study[14] to aid scientific experiments. A study is about the meaning and the processing of data. It includes descriptions of data elements and process steps for handing the data. A study is

defined by a formalized series of steps, also known as a process flow, intended to solve a particular problem in a particular application domain. A Process Flow Template (PFT) is used to represent such a formalized workflow. A study is then activated by instantiating such a PFT. This instantiation is called a process flow instantiation (PFI).

VLAM-G has developed a database model for the Experiment Environment (EE). The EE data model underlies the definition of a PFT and allows for a random ordering of processes and data elements. While the process steps in the PFT represent the actual data flow in an experiment, experiment itself is represented by a data flow graph (DFG). DFG usually contains experiment specific software entities and generic software entities. These self-contained software entities are in turn called modules. A scientist may connect these modules to conduct experiments. Separate modules may be grouped together to form a workflow. Construction of an experiment topology (workflow) takes place in a GUI, which is the only component that users interact with. The GUI, together with a run time system (RTS), form the core components and interface to the underlying entities. VLAM-G includes a PFT database that contains data models designed for various application domains.

### **5.2.2 From Services to Modules**

The VLAM-G RTS manages the topology (workflow) of an experiment, which is a graphical representation of a group of modules composed in the GUI. From the service point of view, RTS is essentially a generic dynamic client that is able to invoke a series of service modules.

To include software entities and/or services in experiments, VLAM-G modules need to be prepared and stored in the RTS database. A module editor is provided, as a part of the GUI, to aid this process. The procedure is as follows. First, a module package needs to be obtained from module developers. Then the module properties can be modified using the module editor. The module's input/output port, executable and parameter properties are specified to define the module. After verifying the module, it has been saved in the VLAM-G RTS database.

In this project, visualization services described in chapter 4 have been converted to VLAM-G modules and saved in the RTS database. To prepare modules for Web services, WSDL files of the services have been made available to VLAM-G. It is from the WSDL file that VLAM-G gets the information about a service. For any deployed web service, URI (Uniform Resource Identifier) carries the information of where the service resides, and the point of service invocation. It is also the point where VLAM-G obtains

services' WSDL files and invokes the services. Therefore, the URI must be kept consistent in the module preparation and in the actual service invocation. To ensure VLAM-G gets the correct information about a service, the name of the WSDL file has to be specified in the service's WSDO file so that service container can expose it to the outside world. The availability of the WSDL file can be checked by using a standard Web browser (e.g., Mozilla Firefox[20]). A Web browser is able to display the contents of WSDL file if it is made available by a service container.

Once the WSDL file is obtained, the module properties are then determined accordingly. The methods a service object carries are derived from the operation section. The input/output parameters are derived from the message section. The issue of SOAP encoding styles is worth to be mentioned here. SOAP stands for Simple Object Access Protocol. It is a protocol for exchanging XML-based messages over network. SOAP uses XML to serialize data that is transported to a software application with different encoding methods to convert data from a software object into XML and back. Three encoding styles have become the most popular ones:

- Remote Procedure Call(RPC) encoding,
- Remote Procedure Call Literal encoding(RPC-literal),
- and document-style encoding, also known as document-literal encoding.

The difference between three encoding styles lies in their complexities. RPC style offers the most simplicity and document-style imposes extra burden on developers, while RPC-literal lies in between.

The SOAP encoding style of a service is also specified in its WSDL file. If a client wants to invoke a desired service, it must use the same encoding style of the service to be able to deserialize the data in the SOAP messages.

Among three encoding styles, VLAM-G handles RPC and RPC-literal. In this project the visualization services have been initially developed using the document-style encoding. Therefore, we converted them to the RPC-literal style in order to meet the requirements of VLAM-G. This conversion involved the modification of definitions of operations and messages in the WSDL file.

When more than one parameters need to be declared by a method of a service object, document-style encoding boxes multiple parameters inside a single object as stub classes, while RPC-literal encoding simply declares parameters in corresponding messages. Fragments of two different WSDL descriptions of the same service are provided below. As an example, the

service takes one string and one integer as input and generates one string as output. The first WSDL file uses document-style encoding.

```

<!--document-style encoding declares a complex data type in the section types-->
<wsdl:types>
<schema elementFormDefault="qualified"
  targetNamespace="http://www.globus.org/namespaces/examples/core/stream"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://www.globus.org/namespaces/examples/core/stream">
  <xsd:element name="addstream">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="url" type="xsd:string"/>
        <xsd:element name="pNumber" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="addstreamResponse" type="xsd:string"/>
</schema>
</wsdl:types>
<wsdl:message name="addStreamResponse">
  <wsdl:part element="tns:addStreamResponse" name="addStreamReturn"/>
</wsdl:message>
<wsdl:message name="addStreamRequest">
  <wsdl:part element="tns:addstream" name="addStreamRequest"/>
</wsdl:message>
<wsdl:portType name="StreamerPortType">
  <wsdl:operation name="addStream">
    <wsdl:input message="tns:addStreamRequest" name="addStreamRequest"/>
    <wsdl:output message="tns:addStreamResponse" name="addStreamResponse"/>
  </wsdl:operation>

```

The second WSDL file uses RPC-literal encoding (supported by VLAM-G).

```

<!--RPC-literal encoding requires no type section for string and/or integer
  type, parameters are specified in the message section-->
<wsdl:message name="addStreamResponse">
  <wsdl:part type="xsd:string" name="addStreamReturn"/>
</wsdl:message>
<wsdl:message name="addStreamRequest">
  <wsdl:part type="xsd:string" name="url"/>
  <wsdl:part type="xsd:int" name="pNumber"/>
</wsdl:message>
<wsdl:portType name="StreamerPortType">
  <wsdl:operation name="addStream" parameterOrder="url pNumber">
    <wsdl:input message="tns:addStreamRequest" name="addStreamRequest"/>
    <wsdl:output message="tns:addStreamResponse" name="addStreamResponse"/>
  </wsdl:operation>
</wsdl:portType>

```

## 5.3 Graphical Editing

After the modules of visualization services are prepared and stored in the RTS database, they are ready for users to compose the visualization pipelines via the VLAM-G GUI and to actually invoke these services. In the GUI, the

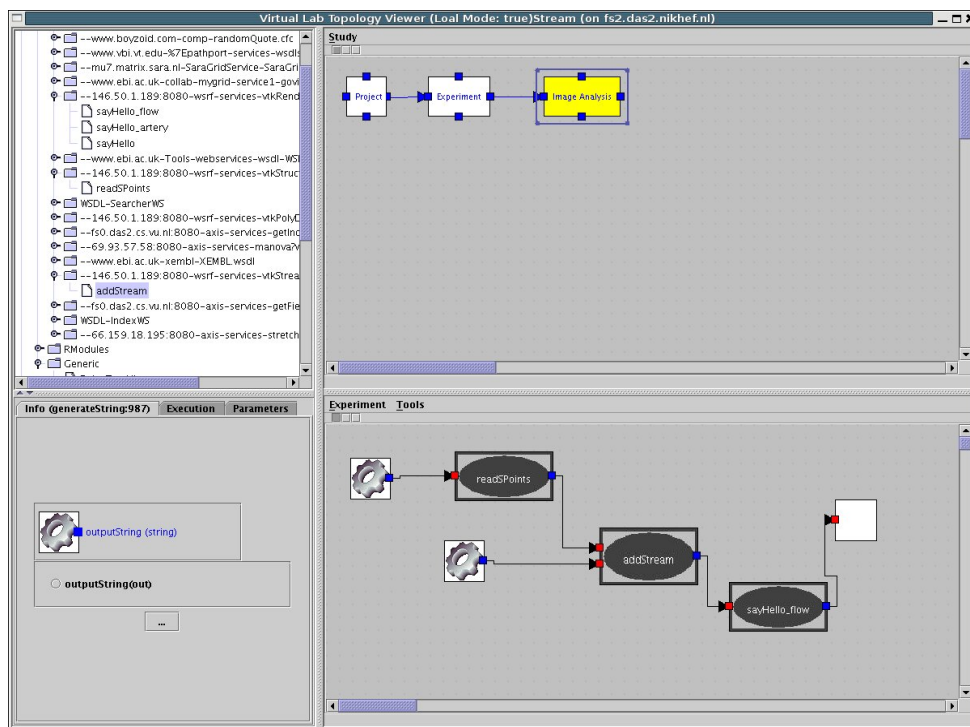


Figure 5.1: Topology composition of a visualization pipeline

available service modules are listed for users to choose and can be dragged and dropped to the composition window to form a workflow topology. At this point, the type match between input and output of modules that are connected is checked to ensure the validity of workflow topologies. Figure 5.1 shows an example of a successful topology composition and Figure 5.2 illustrates the result of the execution of a user-defined visualization service pipeline (streamline based visualization of the LB-simulated data).

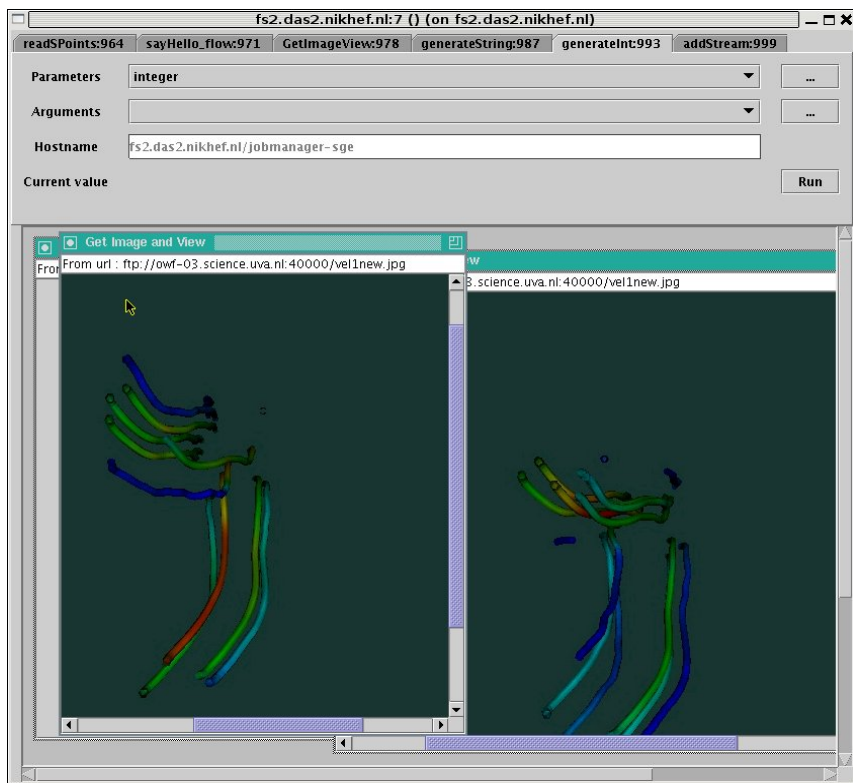


Figure 5.2: Result of a visualization pipeline in VLAM-G

# Chapter 6

## Conclusions

Processing, visualization and integration of information from various sources play an increasingly important role in modern healthcare. Information sources may be widely distributed, and the data processing requirements can be highly variable, both in the type of resources required and the processing demands made upon these systems. Grid technology is a major cornerstone of today's computational science and engineering. By offering a unified means of access to different and distant computational and instrumental resources, unprecedented possibilities and benefits are expected. Connectivity between distant locations, interoperability between different kinds of systems and resources, and high levels of computational performance are some of the most promising characteristics of the Grid.

For medical applications, the remote access to patient data, medical knowledge bases and specialized medical instruments are of utmost importance. In addition, clinical data driven experiments and simulations have become very advanced and complicated. To allow medical specialists to deal with large medical datasets, aside from developing efficient extraction techniques, it is necessary to have available computational facilities to visualize and interact with the results of an extraction process.

Having this in mind, we developed an interactive service-oriented visualization framework aimed to facilitate the image-based analysis of medical data.

Due to the fact that distributed computing is oriented mostly towards service-based architectures such as WSRF (Web Services Resource Framework), OGSA (Open Grid Services Architecture) and Globus, traditional visualization models do not comply completely our needs and requirements. To insure efficient resource allocation and user-friendly interactivity over the Grid, a service-oriented visualization model has been developed and applied in this project.

The model originates from a traditional visualization pipeline of Harber and McNabb[23], where the transformation sub-processes are combined into Web services that can be run on the Grid. In total four visualization services have been implemented in the project. Two reader services, one filter service and a render service.

The proposed architecture is well suited for the work of both visualization experts and end-users (medical specialists in our case). By means of the interactive visualization framework experts can compose visualization routines for end-users, who can later exploit them irrespective to their geographical location and available computational resources. In addition, resource sharing provides nice opportunities for collaborative work and especially for the collaboration between experts and end-users.

A service-oriented architecture makes it possible to generate multiple representations simultaneously, while the projection equipment on the client may vary from a common desktop PC monitor to an immersive stereoscopic system or even an auto-stereoscopic display. This became possible thanks to the bi-directionality of a flow. The data about display parameters and changes in the viewing conditions can be passed to other services both in the top-down and bottom-up manner[2].

In this project, all visualization services have been developed using Globus Toolkit version 4 (GT4), where GT4's Java WS Core is an implementation of the Web Services Resource Framework. Web services standards such as WSDL interface and XML-based SOAP messaging are embraced. Each service is implemented in a client-server manner. The steering interface on the client has been developed in Java.

The image based analysis of vascular disorders is the test case of this research. In particular, we experimented with two different types of medical datasets:

1. experimental data of the patient's vascular condition;
2. simulated blood flow data of the selected region of interest.

To allow interactive composition of visualization pipelines, we integrated all service developed in this project with the VLAM-G workflow management system developed as part of the VL-e project[9].

Currently three visualization pipelines can be turned into distributed service frameworks:

- isosurface extraction pipeline (applied to the datasets of type 1);
- streamline based visualization pipeline (applied to the datasets of type 2);

- combined visualization pipeline (datasets of type 1 and 2).

The actual visualization part of the project has been performed using the Kitware Visualization Toolkit.

## 6.1 Discussion

The service-oriented visualization framework designed and implemented in this Master project is the proof of concept that the distributed architecture across the Grid is also applicable to the area of interactive scientific visualization.

The utility of our approach is that end-users (medical specialists) are no longer restricted by hardware and software limitations and can perform image-based analysis on very large datasets. Thanks to the interactive visualization framework developed in this project, the high-quality interactive visualizations of large medical datasets (of 70 MBytes and even bigger) can be delivered to the physically remote users, whose local computational resources would be otherwise overwhelmed.

The performance measurements indicated that different computational resources are required for running reader, filter and render services. In particular, the last two services require more powerful machines and faster graphics cards. Measurements show that when image rendering is assigned to a powerful graphics station, much better performance can be gained. In addition, visualization parameters (e.g., the number of streamlines and the step length of the streamline extraction visualization) have an effect on the computational needs related to different stages of the visualization pipeline.

By spreading visualization sub-processes over the Grid as independent services, visualization tasks can be assigned to different computational resources. This motivated resource allocation allows to minimize the under utilization of certain machines for time longer than actually visualization runs, which is normally the case when all stages of the visualization pipeline are linked together (traditional visualization models). We do not argue that our approach provides with the most effective resource allocation scheme. But it definitely is a valuable solution when a distributed environment contains heterogeneous computer systems. Then for instance, filter and render services can be assigned to ultra-high performance machines.

Currently the interactive visualization framework supports only services developed in this project. To allow functionalities from different vendors to be brought together and combined within a single pipeline, we integrated our visualization services with the VLAM-G workflow management system.

It allows to browse and select among available services, as well as to check the connection validity between selected services. So, any individual component in a visualization pipeline can be swapped by an alternative one (also integrated with the VLAM- G) in case it is faster or cheaper or simply more reliable.

Another concern of the project is interactivity in terms of human-computer interaction. Through the integration of visualization services with the VLAM- G workflow manager, the creation of visualization pipelines can now be easily done by non-computer experts. End-users are able to compose adequate visualization pipelines for their specific needs. Moreover, steering of each intermediate stage of a pipeline (its input and output) allows visualization specialists to experiment with different visualization schemes for finding optimal solutions to different visualization problems.

## 6.2 Future Work

The creation of interactive visualization tools for the image-based medical analysis over the Grid still remains a challenge, which requires plenty of future related research and development.

With regard to this Master project, the first possibility of the ongoing work would be to investigate possibilities to speed up the visualization process. Considering limitations in the currently chosen platform for the deployment of visualization services and the data transfer mechanism, we would like to look into alternatives to minimize the latency and to achieve the better performance.

Next, due to the time constraints of this project, we experimented only with a conventional desktop PC monitor as the final visualization-interaction front-end. However, the developed infrastructure allows diverse projection equipment to be utilized, including stereoscopic and auto-stereoscopic immersive systems. Therefore, with regard to the display diversity, the next step of this project could be the investigation of possibilities to generate visualizations in an adaptive manner based on display parameters and the viewing situation (for auto-stereoscopic monitors[17]).

While visualization of a problem can be performed by services, the result of visualization may not be always aligned correctly to the objectives of end-users. In order to provide users with the knowledge of a problem to be visualized the client can be enriched accordingly. For instance, based on the properties of datasets at hand, suggestions and directions on how to practice visualization can be provided to users to aid the composition and steering of pipelines. In such way, the number of cycles invested on tailoring the visual-

ization result towards the users needs can be significantly reduced. Another open question is how to organize multiple steering such that each user may hold the information about his or her visualization pipeline separately.

It is clear that some components of the current development show the need in future work towards a more robust framework. Issues as security mechanism in distributed environments and fault tolerance[19] have raised the interest of further investigation.

# Bibliography

- [1] Common Object Request Broker Architecture(CORBA). <http://en.wikipedia.org/wiki/CORBA>. Wikipedia.
- [2] D. E. Avison and G. Fitzgerald. Information systems development: Methodologies, techniques and tools. *McGraw-Hill Book Company London*, 1995.
- [3] R. G. Belleman, B. Stolk, and R. de Vries. Immersive virtual reality on commodity hardware. *Proc. of the 7th annual conference of the Advanced School for Computing and Imaging*, pages 297–304, 2001.
- [4] Stuart M. Charters, Nicolas S. Holliman, and Malcolm Munro. Visualisation in e-demand: A grid service architecture for stereoscopic visualisation. *Proceedings of UK e-Science Second All Hands Meeting.*, 2003.
- [5] Luca Chittaro. Information visualization and its application to medicine. *Artificial Intelligence in Medicine*, 22:81–88, 2001.
- [6] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. *SIGGRAPH*, pages 135–142, 1993.
- [7] Migrating Desktop. <http://ras.crossgrid.man.poznan.pl/crossgrid/>. The CrossGrid Project.
- [8] W. e. Lorensen and H. E. Cline. Marching cubes: a high resolution 3d surface construction algorithm. *Computer Graphics*, 21, 1987.
- [9] VL e project. <http://www.vl-e.nl/>. University of Amsterdam.
- [10] E. Florian and W. Schmidt. Development of a time-resolved optical tomography system for neonatal brain imaging. *Medical Physics*, 27, 2000.

- [11] Alfredo Tirado-Ramos et al. Integration of blood flow visualization on the grid: the flowfish/gvk approach. *Grid Computing: Proceedings of the 2nd European AcrossGrids Conference, Lecture Notes in Computer Science*, pages 77–79, 2004.
- [12] B.Ludscher et al. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 2005.
- [13] C Upson et al. The application visualisation system: A computational environment for scientific visualisation. *IEEE Computer Graphics and Applications*, 9:30–44, 1989.
- [14] H. Afsarmanesh et al. Vlam-g: A grid-based virtual laboratory. *Future Gener. Comput. Syst.*, 19, 2003.
- [15] Ken Brodlie et al. Visualization in grid computing environments. *IEEE Visualization, Proceedings of the conference on Visualization*, pages 155 – 162, 2004.
- [16] Z. Zhao et al. Scientific workflow management: between generality and applicability. *QSIC*, pages 357–364, 2005.
- [17] Zudilova-Seinstra E.V. and Yang N. Towards service-based interactive visualization. *Proc. of the International Symposium on Ambient Intelligence and Life, San Sebastian, Spain*, 2005.
- [18] NAG IRIS Explorer. <http://www.nag.co.uk/>. Numerical Algorithms Group.
- [19] Schneider F. Implementing fault-tolerant services using the state machine approach. *ACM Computing Surveys*, 22, 1990.
- [20] Mozilla Firefox. <http://www.mozilla.com/firefox/>. Mozilla Corporation.
- [21] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. Computer graphics principles and practice. *Addison-Wesley Publishing Company*, 1990.
- [22] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.
- [23] R. B. Haber and D. A. McNabb. Visualization idioms: A conceptual model for scientific visualization systems. *Visualization in Scientific Computing*, pages 74–93, 1990.

- [24] AVS home page. <http://www.avs.com/>. Advanced Visual Systems Inc.
- [25] Globus home page. <http://www.globus.org/>. Globus Alliance.
- [26] Unicore home page. <http://www.unicore.de/>. Central Institute of Applied Mathematics, Forschungszentrum Juelich, Germany.
- [27] VTK home page. <http://www.vtk.org/>. Kitware, Inc.
- [28] I.Foster, C. Kesselman, J. Nick, and S. Tuecke. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. <http://www.globus.org/research/papers/ogsa.pdf>, 2002.
- [29] Jini. <http://www.jini.org/>. Sun Microsystems.
- [30] C. Knight and M. Munro. Mediating diverse visualisations for comprehension. *Proc. of the 9th International Workshop on Program Comprehension*, 2001.
- [31] Andreas Kolb. Visualization over the internet. *Institute of Computer Graphics, Vienna University of Technology, Austria*.
- [32] The Mesa 3D Graphics Library. <http://www.mesa3d.org/>. On the Web.
- [33] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38:114–117, 1965.
- [34] A Ozkurt and K Ozmehmet. Interactive medical volume visualization for surgical operations. *Dokuz Eylul University, Dept. of Electrical and Electronics*, 2001.
- [35] J.M. Ragas. Interactive visualization of flowfields in an immersive virtual environment - applied to the test case of simulated abdominal vascular reconstruction. *MSc thesis, University of Amsterdam*, 2002.
- [36] R.A Robb. *Handbook of Medical Imaging: Processing and Analysis*, ed. Isaac N. Bankman. Academic Press, 2000.
- [37] S.Chen and G.D.Doolen. Lattice boltzmann method for fluid flows. *Annu. Rev. Fluid Mech*, 30:329–364, 2001.
- [38] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit An Object-Oriented Approach To 3D Graphics, 3rd Edition*. Kitware, Inc, 2002.

- [39] SOAP. <http://www.w3.org/TR/soap/>. The World Wide Web Consortium.
- [40] B. Sotomayor. *The Globus Toolkit 4 Programmer's Tutorial*. <http://gdp.globus.org/gt4-tutorial/>, 2005.
- [41] E. F. Toro. Riemann solvers and numerical methods for fluid dynamics. *Springer-Verlag, Berlin*, 1997.
- [42] M. Tory and T. Moller. Rethinking visualization: A high-level taxonomy. *IEEE Symposium on Information Visualization*, pages 151–158, 2004.
- [43] T. van Walsum and F. Post. Selective visualization of vector fields. *Tutorial on Visualization and Topology of Vector and Tensor Fields*, 1995.
- [44] VIPAR. <http://www.man.ac.uk/MVC/research/vipar/>. CSAR Applications.
- [45] VRML. <http://www.w3.org/MarkUp/VRML/>. The World Wide Web Consortium.
- [46] Workflow. <http://en.wikipedia.org/wiki/Workflow>. Wikipedia.
- [47] WSDL. <http://www.w3.org/TR/wsdl>. The World Wide Web Consortium.
- [48] Yao, James S. T., Pearce, and William H. Current techniques in modern vascular surgery. *McGraw-Hill Professional*, 2000.
- [49] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *Grid Computing and Distributed Systems (GRIDS) Laboratory, The University of Melbourne*, 2005.