Service-Oriented Approach to Collaborative Visualization

Haoxiang Wang, Ken Brodlie, James Handley, Jason Wood

School of Computing, University of Leeds, Leeds, LS2 9JT, UK

Abstract

This paper presents a new service-oriented approach to the design and implementation of visualization systems in a Grid computing environment. The approach evolves the traditional dataflow visualization system, based on processes communicating via shared memory or sockets, into an environment in which visualization Web services can be linked in a pipeline using the subscription and notification services available in Globus Toolkit 4. A specific aim of our design is to support collaborative visualization, allowing a geographically distributed research team to work collaboratively on visual analysis of data. A key feature of the system is the use of a formal description of the visualization pipeline, using the skML language first developed in the gViz e-science project. This description is shared by all collaborators in a session. In co-operation with the e-Viz project, we generate user interfaces for the visualization services automatically from the skML description. The new system is called NoCoV (**No**tification-service-based **Co**llaborative **V**isualization). A simple prototype has been built and is used to illustrate the concepts.

1. Introduction

Visualization is widely recognized as a critical component in e-science, allowing insight into the increasingly large datasets generated by simulation and measurement. In recent years a number of important visualization tools have been developed, many of these following the dataflow paradigm. This dataflow approach sees visualization as a sequence of processing steps, whereby raw data is first filtered in some way, then transformed to a geometric representation and finally this geometry rendered as an image. Visualization software provides these steps either as classes that can be embedded in a user program (vtk – vtk, 2006 - is an example of this approach), or as an overall environment with a visual programming editor that enables pipelines to be built from a supplied set of modules (here IRIS Explorer – IRIS Explorer, 2006; Walton, 2004 - is an example). These pipelines can be built, torn apart and reformed, as users experiment with different ways of looking at their data. The dataflow paradigm for visualization (first suggested 20 years ago) has stood the test of time, not least because it provides a high level of abstraction for designing visualization applications.

Major computational applications today typically involve distributed computing – with the user interface executed at the desktop, and remote resources used for computationally demanding tasks. Some traditional visualization systems have managed to evolve to this style of working: in fact IRIS Explorer was designed from the outset to have this capability, with the visual editor on the desktop controlling remote execution of modules. Recent work in the gViz e-Science project (gViz, 2006) has exploited this to adapt IRIS Explorer to grid computing.

However there is a trend today to employ Service-Oriented architectures in designing distributed computing applications. An important and pioneering effort to utilize web services in visualization was the GAPtk toolkit (Sastry and Craig, 2003) which provides specific 'turnkey' visualization services such as isosurfacing, but without the ability to chain them together in pipelines. However, the traditional dataflow visualization paradigm is an excellent match to the concept of a service-oriented architecture: the modules simply become services. Charters (Charters et al, 2004; Charters, 2006) has described the design of a visualization system based on these concepts, in particular using Web Services. Our work in this paper takes this approach further, by using stateful Web services and the facilities in Globus Toolkit 4 (GT4, 2006) for subscription and notification.

Much e-science is multi-disciplinary in nature, involving geographically distributed research teams, and so visualization tools must be designed to be used collaboratively. Again the dataflow paradigm has proved extremely flexible, and it has been exploited in a number of different ways to provide collaborative visualization systems. This work however pre-dates the emergence of Service-Oriented architectures, and so it is important to study how best to provide team-working in this newer context. We see collaboration as fundamental, and so our design incorporates multi-user working from the outset. The style of collaborative working has evolved from our experiences with collaborative visualization over the past decade.

The structure of the paper is as follows. We begin in section 2 with the vision which underpins our research programme in service-oriented visualization, followed in section 3 by an overview of NoCoV, our proposed system. The design of the system is discussed in section 4, and section 5 describes an implementation of a prototype, developed as proof of concept. Conclusions and future work are in section 6.

2. Service-oriented Visualization – The Vision

The overall vision for our design is a next generation visualization system – based on the proven concept of dataflow pipelines linking elementary visualization modules, but exploiting modern ideas from service-oriented architectures, and involving collaboration between users and between providers, at a global level. Thus we see visualizations being created from a worldwide repository of visualization services, being assembled collaboratively by research teams, and exploiting the latest Grid computing technologies.

A fundamental concept therefore in our design is the Visualization Web Service. This corresponds to a module in a traditional Modular Visualization Environment, or MVE, such as IRIS Explorer or Open DX. In an MVE, modules can be linked in a dataflow pipeline, this being achieved typically by a visual programming ‘front-end’. Often, this front-end also provides a user interface to each module, allowing parameters to be modified interactively. We retain the front-end concept, but make a clear distinction between the editing of pipelines and the user interface to services – we envisage visualization application developers having access to pipeline editing, while visualization users only require access to parameter setting, in pipelines previously created.

A central aspect of our vision is a formal description of the visualization pipeline; this indicates the services, the user specified parameters of the services, and the linkage between services. This description, together with information about the expertise of the user, is used to build automatically tailored user interfaces.

The flow of data between modules in an MVE is handled either by shared memory (on the same machine) or by socket connection (between machines). The triggering of dataflow is handled by a firing algorithm. We are able to exploit a novel concept in web services to initiate dataflow, namely notification. A service subscribes to a data element on another service, and receives a notification when that element changes – thus data can flow from one service to another.

Visualization expertise is distributed across the world, and so our vision is to see a worldwide repository of services, maintained on a co-operative basis by specialist groups. These can be wrapped as Grid Archives (gars), listed in a central UDDI, and downloaded and deployed on a local basis by visualization service providers. Thus a flow visualization service developed by a team in the Netherlands could be deployed in Japan by a Japanese service provider.

In any pipeline, each service can be deployed on a different resource – thus allowing us to exploit dedicated rendering resources for example. In general, delivery to the desktop should allow a choice of remote rendering (delivery of images) or local rendering (delivery of geometry).

There is increasing interest in collaborative applications and so a fundamental design requirement of our system is to support team working, where the members of the team may be distributed in space and time. A number of approaches to collaborative visualization using traditional MVEs have been suggested. These include (at two ends of the spectrum): VNC (RealVNC, 2006), where the display of the MVE of one user is shared by a number of collaborators – this is fairly effective when the collaboration is passive, but is extremely awkward if several people wish to take an active role; and COVISA (Wood et al, 1997) where each user develops their own pipeline but can tap data from any point in their pipeline and make it available to all collaborators – this is extremely flexible, and almost any style of collaboration can be programmed, but the different pipelines make it difficult for any user to have a global view of the set of individual pipelines. Thus we aim for a midway position: we have a shared ability to build the pipeline, but there is a single pipeline for all users.

In addition to this *synchronous* (same time, different place) collaboration, there is a similar need to support *asynchronous* working, where the participation is spread over a period of time. The requirement here is for a persistent pipeline of services, which a collaborator can pick up and develop at a later time – important for collaboration with Australasia for example.

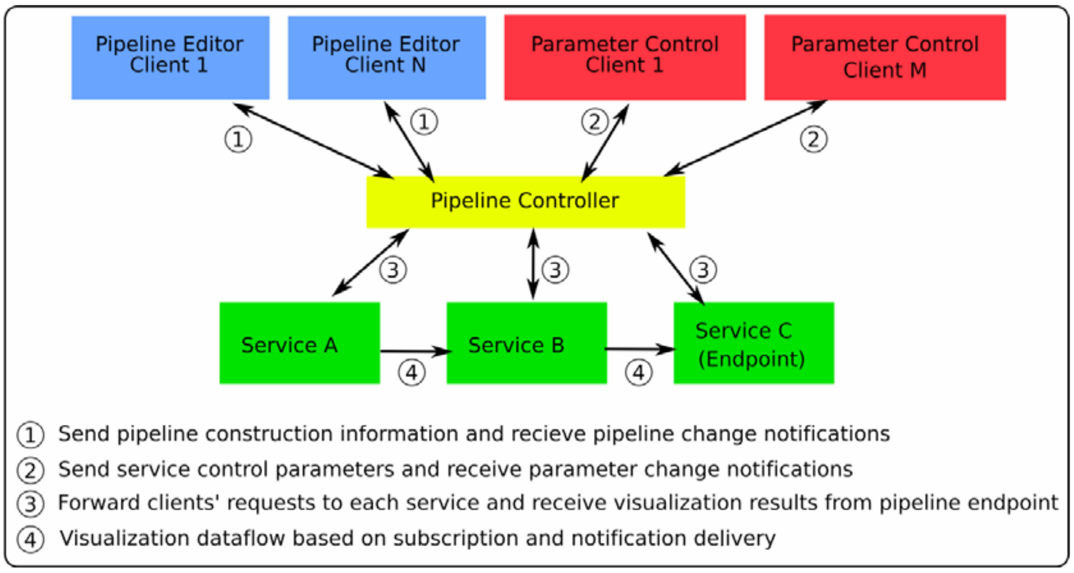
Pipeline Editor Client 1
Pipeline Editor Client N
Parameter Control Client 1
Parameter Control Client M
Pipeline Controller
Service A
Service B
Service C (Endpoint)
① Send pipeline construction information and recieve pipeline change notifications
② Send service control parameters and receive parameter change notifications
③ Forward clients' requests to each service and receive visualization results from pipeline endpoint
④ Visualization dataflow based on subscription and notification delivery

Figure 1: The overview of service-oriented collaborative system (NoCoV)

3. NoCoV System overview

NoCoV (Notification-service-based Collaborative Visualization) provides a collaborative visualization system implemented using Notification Web Services. It is an evolution of the MVE, implemented using a Service-Oriented architecture. Here, visualization services replace the visualization modules of the MVE, and the links between traditional visualization modules are implemented as subscriptions and notifications between visualization services.

As shown in Figure 1, the Pipeline Editor Client is the user interface for creating a visualization pipeline, and the Parameter Control Client provides the GUI for users to interact with parameters on each service. This separation of interfaces allows us to support various classes of users. Visualization experts may use the Pipeline Editor to create visualization applications for use by novice users who subsequently only interact with these pipelines through the Parameter Control Client. A middle class of users may be comfortable creating their own pipelines so may use both interfaces.

The Pipeline Controller Service sits between the end-user clients and the distributed visualization services thus making the visualization services transparent to end-users. It forwards the requests from clients to corresponding services and broadcasts visualization results to subscribing clients as notifications. Users only need to consider the visualization process at a logical level without being aware of the physical locations of these services or the different invocation methods required. The Pipeline Controller Service also acts as a collaborative workspace by sharing pipeline and control parameters with subscribing clients.

The visualization dataflow is implemented by making subscriptions between different visualization services (service A, B and C in Figure 1). Each visualization service publishes one or more notification topics which act as the output ports for that service through which data is sent to other connected services. There is a special service set as an endpoint within each pipeline to which the Pipeline Controller subscribes. This service triggers a notification every time a new result is generated by the pipeline.

NoCoV is an extendable visualization system since customised visualization services can be introduced into the system, so long as the Pipeline Controller service knows how to communicate with these services (i.e. it is provided with the WSDL descriptions of the services). The Pipeline Editor Client lists these customised services as available services for users to link into their pipeline (i.e. services register on a UDDI server from which the Pipeline Editor Client can retrieve the available service list).

To support collaborative working over the construction of a pipeline there is a requirement to share pipeline description information between the Pipeline Controller Service and all the clients. An extension of skML (Duce and Sagar, 2005), an XML based visualization pipeline description language, is used for this purpose. It has been extended to fit the NoCoV system with an emphasis on Service-Oriented features.

The NoCoV system has been implemented with GlobusToolkit 4 (GT4). The stateful Web Services provided by GT4 offer the capability of maintaining visualization pipeline information between sessions. This allows users to save the current status of the pipeline on the Pipeline Controller Service for use the next time they reconnect. We also exploit Notification Web Services in GT4. Moreover, GT4 provides a set of Grid security specifications which can be seamlessly applied to the NoCoV system in its future development to address one of the desired issues in collaboration: security.

4. NoCoV System design

4.1 Using Notification Web Service

WS-Notification includes a set of specifications (OASIS, 2006) and a whitepaper (Publish/Subscribe, 2006) which describe the use of a topic-based publish/subscribe pattern to achieve notification with standard Web Services.

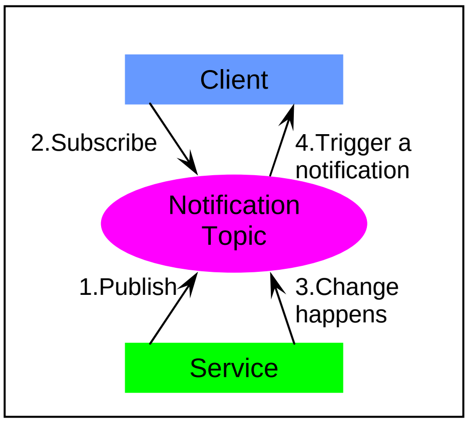

Figure 2:The working pattern of Notification Web Service

With the normal request/response communication mode between service and client, the client has to keep polling the service in order to get the latest changes. By contrast, the notification approach works in the manner shown in Figure 2. When the service publishes a set of notification topics, clients can subscribe to relevant topics according to their different interests. Every time a change happens in these notification topics, the service will automatically deliver the changed information to the subscribing clients.

The main reason for choosing Notification Web Services to implement visualization is their ‘push’ feature. It fits well with the requirement of a visualization pipeline which needs the services to send their results to other services connected to them ‘downstream’, each time they produce a new result.

The publish/subscribe pattern provides a data sharing approach for collaboration. The notification topics published by the visualization service can be either the data or the control parameters. Actions from participants (such as changing parameter values) are also published as notifications, allowing these to be shared.

The stateful feature inherited from GT4 Web Services makes it possible to achieve asynchronous collaboration as the status of the pipeline is persistent and users can retrieve the saved pipeline to carry on their previous work – or new users can take over and continue the development.

The publish/subscribe pattern can also reduce network traffic by only delivering changed information to clients. Moreover it can cut down service and client workloads as clients only need to subscribe to the service once and then just wait for the notification messages. It is similar to previously used technologies such as socket communication, but with the facility of WS-Notification, the system developers do not need to consider the details of physical communication ports, and the communication is relatively easy to set up compared to socket communication.

As XML/SOAP messages have a restriction on their maximum size, when large data (e.g. update of geometry) needs to be sent as a notification, only a reference to the data is included in the notification message, and the data itself can be transferred using http, ftp or gridftp separately. Another possible solution is to add data as an attachment (e.g. DIME) to the notification message.

There are however some disadvantages of notification services. When a client subscribes to a notification service, the client needs to be able to function as a listening service waiting for the notification. In the case of a GT4 implementation, it requires GT4 to be installed on the machine where the notification client runs. In addition, every time a change happens on a notification service, the service needs to start a connection to the subscriber. If the notification client (subscriber) sits behind a firewall, the firewall may block all the connections initiated from outside. In this case, although the client can subscribe successfully to the service, it can not receive any notifications from outside of the firewall. WS-Messenger (Huang, et al, 2006) proposed an approach to address the issue of the delivery of notification through a firewall by using a MessageBox service to store all notification messages and having consumers periodically pull notification from the message box. Another alternative, which is less secure but more straightforward, is to set a small range of open ports in the firewall and configure the local notification system to only use ports within this range.

4.2 The Extended skML

As the proposed NoCoV system is expected to enable the collaborative creation and configuration of visualization pipelines, and the persistence of pipeline information, the visualization pipeline must be represented in such a way that it can be stored as service resource properties.

skML is an XML-based dataflow description language (Duce and Sagar, 2005) which describes visualization pipelines in a generic way, so that the skML description can be independent of the implementation of the pipeline. The skML language was developed as part of the gViz e-science project, and was heavily influenced by MVEs such as IRIS Explorer. However it lacks certain features to describe characteristics of Service-Oriented collaborative visualization pipelines. Rather than create a different description language, we have chosen simply to modify and extend skML.

An 'instance' element replaces the 'module' element in the skML to represent visualization service instances, but the 'link' element in skML is kept to represent the subscriptions to notifications. One of the significant differences is that the extension aims to present richer information about the visualization pipeline. For example, all the output and input ports for each service are recorded: this is then made visible at the user interface, to allow users the ability to change the connections to/from that service. In contrast with the original skML, all control parameters for a service instance must be explicit in the description, again so that they can be presented in an automatically generated user interface. Another difference is the adding of new properties such as 'owner' and 'sharable', which identify who owns this visualization service instance and who are allowed to access this service instance. These new properties will make it possible to add security control in NoCoV.

4.3 Pipeline Controller Service

A Pipeline Controller Service is placed between the end-users and the visualization notification services, in order to enable users to collaboratively build the pipeline and configure each visualization service linked within the pipeline. Figure 3 displays the components of the Pipeline Controller Service.

The Pipeline Controller stands between the distributed visualization services and the end-users as a proxy, through which users can send requests to create/destroy, connect/disconnect service instances and set parameters of these instances. The removing or adding of visualization services or any changes inside visualization services are transparent to end-users.

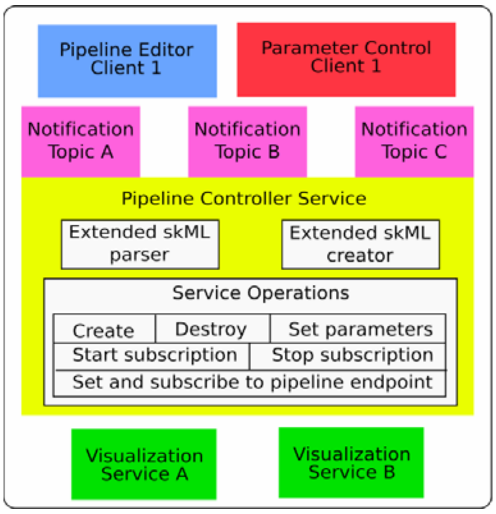

Figure 3: The Pipeline Controller Service

The Pipeline Controller also functions as a shared workspace for all the participants. For visualization experts who create visualization pipelines, the Pipeline Controller keeps a description of the current pipeline in the extended skML, which can be retrieved for the later joiners to the collaborative session. The Pipeline Controller is implemented as a notification service, in which a notification topic about the latest status of the jointly created pipeline is published so that users can share the same pipeline and join in synchronous collaboration of the construction of a pipeline.

Notification topics about control parameters and the latest visualization result generated from the pipeline, are published for scientists who do not want to be involved with the building of the pipeline but simply wish to modify control parameters. By subscribing to these notification topics, scientists can jointly control the distributed visualization services and view the resulting geometry (in case of local rendering) or image (in case of remote rendering).

4.4 Collaborative Visualization Clients

As mentioned in the previous section, the end-user interfaces are separated into a Pipeline Editor Client and a Parameter Control Client, which only provides users with a parameter control interface rather than the view of whole pipeline.

The Pipeline Editor Client allows users to collaboratively select suitable visualization services and link them in an appropriate way. The Parameter Control Client initializes its GUI from the pipeline description retrieved from the Pipeline Controller, presenting a separate tab corresponding to each service instance created in the pipeline. Users can view parameters on the services and steer the service by changing the parameters through the Parameter Control Client. Code from the e-Viz project (Riding et al, 2005) is used to generate the GUI from the extended skML pipeline description – see section 5.4 for more information.

5. Prototype Implementation

A prototype was implemented as a proof of concept for the NoCoV system. The prototype involves a set of simplified visualization services, a Pipeline Controller Service with basic functions as proposed in the design, a Pipeline Editor Client for visualization experts and a Parameter Control Client for scientists.

5.1 Visualization Services implemented as Notification Web Services

In order to demonstrate the capability of building and controlling a visualization pipeline through end-user interfaces, four visualization services were created with simple visualization functions.

The *data service* retrieves data from a source according to the file name or URL provided by the user. The output of the data service can be subscribed to by an *isosurface service* or a *slice service*, which can generate output geometries in VRML format. The *inline service* can subscribe to both isosurface and slice services to combine multiple geometries into a single scene.

5.2 Pipeline Controller Service

The Pipeline Controller acts as an agent for end-users, releasing users from the burden of dealing directly with visualization services. In the prototype, the Pipeline Controller service can create instances from visualization services, connect instances to build up a visualization pipeline and subscribe to the endpoint of visualization pipeline to receive newly generated visualization results.

The Pipeline Controller has the following notification topics: a representation of the current pipeline information including which visualization instances have been created, how these instances are connected to each other and the setting of control parameters for each instance; the latest result generated from the pipeline endpoint; and a representation of the changes of control parameters which enables collaborative parameter control.

5.3 Pipeline Editor Client

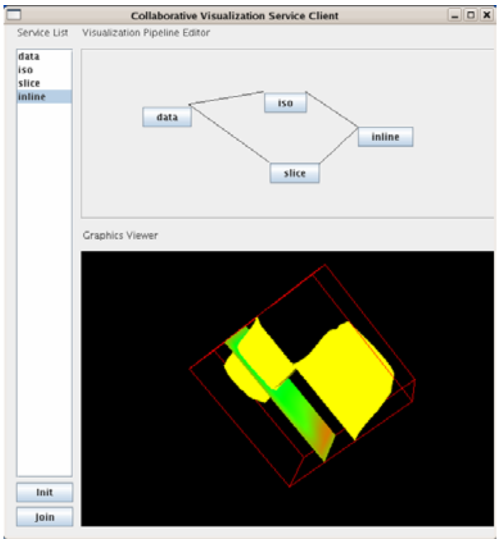
Collaborative Visualization Service Client
Service List
Visualization Pipeline Editor
data
iso
slice
inline
data
iso
inline
slice
Graphics Viewer
Init
Join

Figure 4 – Pipeline Editor Client

The Pipeline Editor Client (see Figure 4) is initialized with a list of available visualization services shown on the left hand side from an XML format service list file (which makes it possible to retrieve a list of available services from a UDDI server). By clicking on the visualization services, a corresponding instance will be created and displayed as a box in the editor window on the right hand side. The editor window supports drag-and-drop operation. By right clicking on the instance in the editor window, the user can specify the output or input to that instance in order to connect different instances together to create a pipeline. All the participants in the collaboration will see the same pipeline on their editor windows, as they subscribe to the same notification topic – the definition of the current pipeline. At this stage, we simply assume clients know notification topics published by each visualization service, but in our future work, each visualization service will provide a method which returns a description of notification topics published for the client to subscribe.

5.4 Parameter Control Client

The Parameter Control Client (PCC) provides a user interface for the control parameters of the individual components of the visualization pipeline created by the Pipeline Editor Client. The PCC is implemented using the user interface component from the e-Viz system (Riding et al, 2005), called the e-Viz Client. This component takes an XML description of the visualization pipeline and generates a user interface for each component based on its description. It also provides connectivity from the user interface to the remote visualization component to send and receive parameter values.

The original e-Viz Client used the gViz library as its sole communications mechanism to send/receive parameter changes. This tool has now been extended to allow alternative communications mechanisms to be used in place of the gViz library. This is managed by specifying in the RDF section of the XML description file what mechanism is to be used for each pipeline component. In this case, a Notification Services mechanism has been specified to link the e-Viz Client with the NoCoV system.

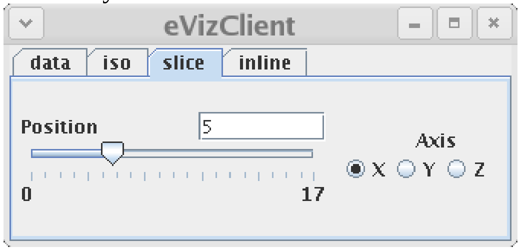

Figure 5: Parameter Control Client for pipeline shown in Figure 4

When a user changes the visualization pipeline using the Pipeline Editor Client, these changes are passed using XML to all of the attached PCCs. The e-Viz Client is designed to respond to snippets of XML that represent alterations to the pipeline and to adjust the user interface accordingly. Thus, when the user adds a module to the pipeline, the e-Viz Client grows a tab to accommodate its control parameter widgets. When a user interacts with a widget on the control panel, these changes are passed to the Pipeline Controller Service which in turn reflects these changes to all PCCs as well as to the intended visualization service. Figure 5 shows the PPC corresponding to the pipeline displayed in Figure 4.

5.5 Simple illustration

We illustrate the use of the NoCoV system with a very simple example. In Figure 6, one user has built a pipeline of two services to visualize a volumetric dataset (*data* – to read in the data; *slice* to display a cross-section).

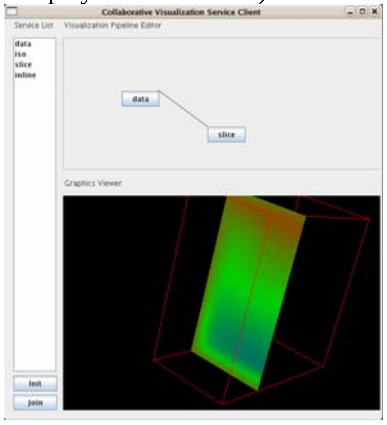
Collaborative Visualization Service Client
Service List
Visualization Pipeline Editor
data
iso
slice
inline
data
slice
Graphics Viewer
Init
Join

Figure 6 – Slice Visualization

A collaborator then joins the session, and initially sees the same slice, but with ability to view from a different angle. Figure 7 shows the two screens, displayed side-by-side here to show the effect.

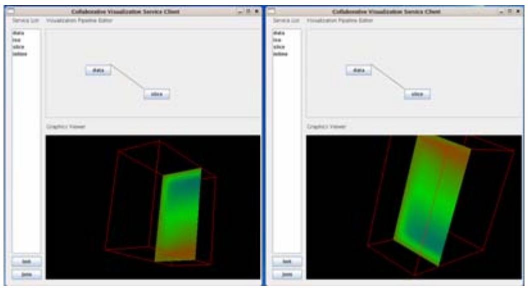

Figure 7 – Collaborative Slice Visualization

The collaborator then suggests that the addition of an isosurface would enhance the understanding of the dataset. They use the pipeline editor to collaboratively alter the pipeline, and the resulting visualizations are shown in Figure 8, again showing the ability of each collaborator to select their own viewing direction.

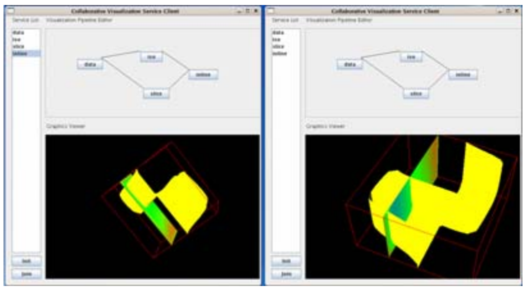

Figure 8 – Adding an Isosurface Service

6. Conclusions and future work

We have presented the vision, design and prototype implementation of a next generation visualization system. It is built using service-oriented concepts, and envisages a worldwide collaboratory in which a research team can come together to visualize data – the collaboration can be at the same time, or at different times. A key feature is exploitation of Grid and Web services technologies, in particular notification services. The publish/ subscribe pattern is used to link the visualization services in a pipeline.

A middle-layer service, the Pipeline Controller Service – acting as a proxy for distributed visualization services - is included to provide collaborative functions for different levels of end-users. Work from the gViz and e-Viz e-science projects is exploited to provide a formal description of the visualization pipeline, and automatically created user interfaces, respectively. A prototype of the proposed system has been implemented as a proof of concept.

The following aspects need to be explored in the next stage to create a comprehensive collaborative visualization system.

Security is an important issue for all collaborative systems. As the prototype is implemented with GT4, the GT4 security mechanisms can be seamlessly applied to NoCoV. The security issues that need to be considered in future work include: setting different roles for users; setting different access permission to each visualization instance in the pipeline; and dynamically changing the valid user list to control the joining and leaving of users in a collaborative session.

The discovery of available visualization services is another important strand. In the prototype, the client gets an available service list from an XML file which contains details of each visualization service – but it is possible to introduce a UDDI server into the system to provide this functionality.

Other issues include the standardization of the data format passed between different types of visualization service, and automatic updating of the Pipeline Controller when new services are brought into the NoCoV repository.

Acknowledgements

Thanks to Stuart Charters (Univ of Durham) for making available an early version of his thesis; and to John Hodrien (Univ of Leeds) for advice on GT4 matters. Jason Wood developed the GUI aspects as part of the EPSRC e-Viz project.

References

Charters, S.M, Holliman, N.S and Munro, M (2004) Visualization on the Grid: a Web Service Approach. Proceedings of the UK e-Science All Hands Meeting, pp202-209.

Charters, S.M (2006) Virtualising Visualisation. PhD thesis, University of Durham.

Duce, D.A, Sagar, M (2005) skML a Markup Language for Distributed Collaborative Visualization. EG UK Theory and Practice of Computer Graphics pp171-178.

GT4 (2006) Globus Toolkit Web site,. http://globus.org/toolkit/

gViz project web site, (2006) http://www.comp.leeds.ac.uk/vis/gviz/

Huang.Y, et al, (2006) WS-Messenger: A Web Services-based Messaging System for Service-Oriented Grid Computing. CCGrid 2006, IEEE Computer Society, pp166-173.

IRIS Explorer web site, (2006) http://www.nag.co.uk/Welcome_IEC.asp

OASIS Web Services Notification TC, (2006). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

Publish-Subscribe Notification for Web services, (2006) .http://www.ibm.com/developerworks/library/ws-pubsub/WS-PubSub.pdf

realVNC web site, (2006) http://www.realvnc.com

Riding, M, et al, (2005) e-Viz: Towards an Integrated Framework for High Performance Visualization, Proceedings of the UK e-Science All Hands Meeting, pp1026-1032. See also e-Viz project website, URL: http://www.eviz.org.

Sastry, M and Craig, M (2003) Scalable application visualization services toolkit for problem solving environments. In Proceedings of the UK e-Science All Hands Meeting, pp 520-525.

vtk web site, (2006) http://www.kitware.com.

Walton, J (2004) NAG's IRIS Explorer. In Visualization Handbook, pp 633--654, Academic Press. Available at: http://www.nag.co.uk/IndustryArticles/ch32.pdf

Wood, J, Wright, H and Brodlie, K, (1997) Collaborative Visualization, Proceedings of IEEE Visualization 1997 Conference, edited by R. Yagel and H.Hagen, pp 253-260, ACM Press.