e-Viz: Towards an Integrated Framework for High Performance Visualization

M. Riding,[1] J.D. Wood,[2] K.W. Brodlie,[2] J.M. Brooke,[1] M. Chen,[3]
D. Chisnall,[3] C. Hughes,[4] N.W. John,[4] M.W. Jones,[3] and N. Roard[3]

[1] *Manchester Computing, University of Manchester*
[2] *School of Computing, University of Leeds*
[3] *Department of Computer Science, University of Wales Swansea*
[4] *School of Informatics, University of Wales, Bangor*

Abstract

Existing Grid visualization systems typically focus on the distribution onto remote machines of some or all of the processes encompassing the visualization pipeline, with the aim of increasing the maximum data size, achievable frame rates or display resolution. Such systems may rely on a particular piece of visualization software, and require that the end users have some degree of knowledge in its use, and in the concepts of the Grid itself. This paper describes an architecture for Grid visualization that abstracts away from the underlying hardware and software, and presents the user with a generic interface to a range of visualization technologies, switching between hardware and software to best meet the requirements of that user. We assess the difficulties involved in creating such a system, such as selecting appropriate visualization pipelines, deciding how to distribute the processing between machines, scheduling jobs using Grid middleware, and creating a flexible abstract description language for visualization. Finally, we describe a prototype implementation of such a system, and consider to what degree it might meet the requirements of real world visualization users.

1 Introduction

Research into Grid-based visualization has often involved the porting of the classical visualization pipeline [1] into a Grid framework, enabling remote data acquisition, security based on the Grid Security Infrastructure (GSI) [2], distributed computation and parallel remote rendering. Many systems have been created that offer to users the ability to visualize data sets of ever increasing size, at high resolutions and interactive frame rates [3]. This is achieved through parallelism at some or all stages of the visualization pipeline, including the transmission of data between functional modules.

It is worth considering what additional requirements users might have of Grid visualization systems. The Grid encompasses a wide range of high performance machines of varying architectures, operating systems and underlying visualization applications.

A truly generic Grid visualization system should be able to fulfill a user's request for any particular visualization technique, and to select the most appropriate hardware and software in order to implement it.

In addition, we might assess a Grid visualization system in terms of quality of service. Electrical power grids have been used as an analogy when describing computational Grids [4]. While this has not met with universal approval certain characteristics of power grids, such as fault tolerance and adaptability, should be considered when creating computational Grids. System transparency is also an important issue. An end user ought to be able to consider the Grid as a black box capable of performing computational tasks without being concerned about the underlying architecture. Such a concept poses difficult challenges for real-time interactive visualization applications, where the end-user is part of a feedback loop with the remote machines. This paper introduces the e-Viz architecture for visualization on the Grid, which aims to address such issues.

2 Consideration of end-users

Visualization users come from a number of backgrounds and disciplines, and are not always skilled at creating their own visualization applications or familiar with Grid computing. We can classify potential users according to their level of knowledge in the visualization and Grid computing domains, as depicted in Figure 1. The expectations and requirements of each group will differ and should be considered if we are to offer a system which can be useful to all. In the following descriptions, we define 'visualization knowledge' to be the ability to create a custom visualization application in either a modular network-based system such as AVS/Express or IRIS Explorer, or a software development kit, such as the Visualization Toolkit (VTK). 'Grid knowledge' is defined as familiarity with the principles of

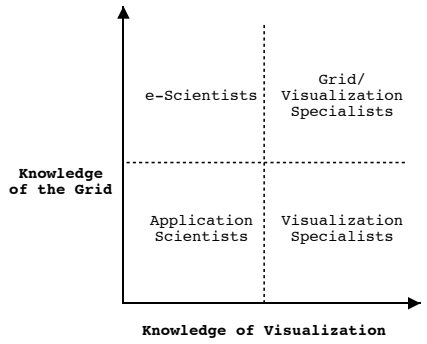

Figure 1: Users of Visualization Systems

Grid computing, and the mechanisms involved in its implementation.

The four groups are identified as follows:

- Grid / Visualization Specialists: Users with knowledge of both the Grid and Visualization: Visualization users who can construct their own applications to run on the Grid are comparatively rare. Such people are either developers of new, or users of existing, grid-enabled visualization software packages.

- e-Scientists: Users with knowledge of the Grid but not Visualization: There are a growing number of application scientists using the Grid to assist in their research, utilising remote high performance computing (HPC) machines to run simulations that their desktop machine or even local HPC resources would have been unable to accommodate. Such users, commonly termed 'e-Scientists' will not always have the skills necessary to create their own bespoke visualization applications, and may instead rely on a visualization specialist (see below) to do this for them.

- Visualization Specialists: Users with knowledge of Visualization but not the Grid: There are a number of groups within research institutions providing visualization services to either on-campus research groups, or external private enterprises. Such service providers would be familiar with a range of visualization software packages, but perhaps less familiar with the Grid.

- Application Scientists: Users whose expertise lies in areas other than Grid and Visualization: Probably the largest single group of candidate visualization users, and potentially the group who are least familiar with both the use of the Grid, and of the construction of visualization applications. Indeed, they are possibly not used to Unix based computer systems, instead familiar with the Windows operating system.

Users in the left hand side of Figure 1 (e-Scientists and Application Scientists) either use custom off the shelf software to visualize their data, or rely on those in the right hand side (Visualization Specialists and Grid/Visualization Specialists) to create applications for them. Similarly, users in the bottom half of the diagram are not Grid experts, but may still want to make use of Grid computing to aid in their work.

In the next section, we evaluate existing Grid visualization systems, and identify which of our user types they are most suitable for.

3 Existing Distributed and Grid Visualization Systems

The Op3D project [5], relies on the visualization capabilities of a shared memory parallel (SMP) machine with hardware accelerated graphics capabilities to remotely render volume data. Resulting frames are sent to a client machine which would itself not have had the necessary memory or computational power to accommodate the data sets involved. In this model, all computation is performed on the server, and the client acts as a dumb terminal. The Grid is used to provide a particular machine with sufficient processing power for the client to connect to. Surgeons are the users of the Op3D system, certainly we can consider such a group to fall into the 'Application Scientists' class of users.

The Visapult project [6], which also addresses the topic of remote volume rendering, introduces parallelism at the very start of the visualization pipeline. Large volume files are cached over multiple disks, and transferred to visualization servers in parallel. In this way, the Visapult system can volume render large data sets in the order of tens of Gigabytes and upwards, a capability which enabled it to win the SC Bandwidth Challenge three years on in succession (2001-2003). The application is tailored to the rendering of time-stepped volumetric data, and is not a general purpose visualization toolkit. Users of the software would most likely be application scientists, perhaps aided by visualization engineers.

The Resource Aware Visualization Environment (RAVE) project [7] has focused on the distribution of the rendering stage of the visualization pipeline, and supports a number of options; remote rendering, local rendering and a hybrid approach where some of the workload is performed locally, and some on a remote machine. The system aims to choose the most appropriate technique automatically, based on the client's graphical capabilities, and the network bandwidth. Since RAVE is currently using the Grid to distribute the rendering stage of the pipeline we can consider it to be aimed at users who we would class as 'e-Scientists'. Ap-

plicability to 'Grid/Visualization Specialists' would be achieved if the software was extended to encompass the entire visualization pipeline.

The Grid Visualization Kernel (GVK) [8] is a system which extends the OpenDX modular visualization system so that individual stages of the visualization pipeline can be executed on remote machines by means of the Globus Toolkit [9]. The system is designed to be generic enough that a remote module may be implemented in another visualization system, so long as it is wrapped up with a GVK interface. This approach yields a heterogeneous architecture, but requires that users have the knowledge to construct their own visualization pipelines, or have access to visualization engineers who can do so for them. As such, GVK users would fall into our classification of 'Grid/Visualization Specialists'.

The gViz project [10] offers a collaborative Grid visualization system built around IRIS Explorer. Similarly to the GVK system, visualization applications are constructed in a dataflow network editor, and the user can define that modules be run on remote machines. This approach is again very generic, enabling the construction of almost any type of visualization. The system is aimed at users who would be classed in our taxonomy as 'Grid/Visualization developers', but applications created with it would in turn be aimed at either 'Application Scientists' or 'e-Scientists'.

We can categorise the above Grid visualization systems according to two characteristics; the degree of distribution in the pipeline, and the range of visualization techniques offered. If we reconsider our plot of users and their visualization and Grid knowledge (Figure 1), we can see that there are parallels between the classification scheme for users, and for the software. The degree of Grid knowledge of users equates to the amount of distribution in existing systems, and similarly the degree of visualization knowledge equates to the range of visualization techniques afforded by the system architectures (their degree of specialisation or generality). Those systems which are the most flexible and the most distributed are aimed at the most knowledgeable of users, whilst the systems aimed at the least experienced users are also the most specific and the least distributed. This is perhaps to be expected, since the more complicated systems by their nature require the user to have more knowledge. But does this mean that it would be impossible to construct a system aimed at non-expert users and still offer a high degree of distribution and generality? Is it impossible for visualization users to treat the Grid as a black box? And can we build a visualization system that can harness the power of the Grid, and deliver it to the desktop of the application scientist? The e-Viz project attempts to answer these questions.

4 Design Considerations

The GVK and gViz projects described above both offer a generic architecture for visualization on the Grid, flexible enough to create a wide range of visualization pipelines implemented in a distributed manner. End users, however, must either have the skills and knowledge to construct their own pipelines, or else rely on visualization specialists to do so for them. The e-Viz project attempts to offer a similar generic architecture, but by abstracting away from the underlying implementation it aims to provide a mechanism for end users to visualize their data without having to consider the implementation details of the underlying pipeline.

To help us achieve this aim, we need to consider the concept of a common interface to visualization systems, and the technologies that can be used to stage visualizations on the Grid.

4.1 An Abstract Interface to Visualization Software

If we are to create an abstract interface to visualizations software, we must consider the problem from three viewpoints. Firstly, we must be able to describe visualization pipelines at an abstract level. Secondly, we must be able to communicate with pipeline implementations in a generic manner. Finally, the user must be presented with a generic user-interface that supports multiple underlying systems. We now introduce technologies which can help us to achieve these requirements.

4.1.1 Abstract Visualization Description Language

The gViz project developed a visualization description language known as skML [11], which contains a description of the functional components that comprise a visualization process, where these processes exist, how they are connected together, and, in a collaborative context, which roles have access to them. The abstract nature of the skML language has been demonstrated by its use as an intermediate data format in the translation of an IRIS Explorer network into its OpenDX equivalent. skML is therefore a useful technology for e-Viz as an abstract visualization description language. Within the gViz project, it was influenced by IRIS Explorer in terms of the vocabulary used for functionality (for example, module names) but it is possible to generalise the vocabulary to encompass developing work on visualization ontologies [12].

It is worth considering at this point the viability of an abstract visualization description language. It is important to ascertain if two different

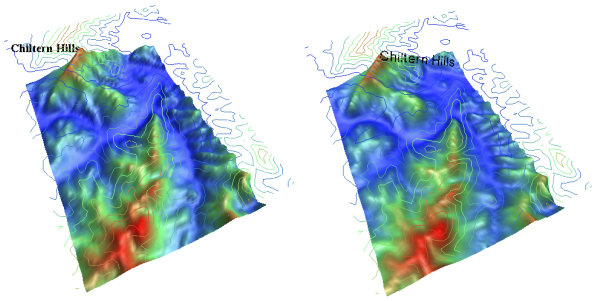

Figure 2: The same visualization implemented with AVS/Express (left) and VTK (right)

visualization packages can produce identical images when implementing the same abstract pipeline. In order to investigate this concept, an example skML pipeline, itself a visualization from an IRIS Explorer tutorial, was re-implemented using AVS/Express and VTK, as depicted in Figure 2. Aside from some small differences in specular lighting, camera angles and the annotating text, the two visualizations are identical. This is encouraging and supports the hypothesis that we can create a single abstract description of a visualization, and use it to automatically generate equivalent implementations using a variety of software tools.

4.1.2 Computational Steering for Visualization

Recent development of computational steering libraries such as those provided by the RealityGrid [13] and gViz [10] projects offer a solution to the issue of implementing a common communication interface to control visualization pipelines. A visualization pipeline is a computational task with a number of controllable parameters, therefore computational steering libraries are an appropriate technology to interface with. If we instrument visualization applications to expose the parameters of an active pipeline via a computational steering library, then we have satisfied the requirement for a common communications interface for user input. A generic user interface could be created based on the concepts of a computational steering client, but effort would have to be made to ensure the application would be intuitive to visualization users, and not just a list of steerable parameters. We discuss an approach to solve this in section 5.1.2.

4.1.3 Remote Rendering for Visualization

Unfortunately a computational steering library cannot provide us with a complete solution to the problem of creating a generic user interface and a common communications interface. Most visualizations output a sequence of rendered images, and in a distributed pipeline, the machine responsible for rendering is not necessarily directly connected to the display device. If rendering is to be performed remotely, we need a technology to transport frames back to the client's machine, a task computational steering libraries are not ideally suited for.

There are instead a number of technologies that deal with this issue. Besides low level protocols such as X11 and GLX which perform local rendering of remote applications, remote rendering capabilities are offered by systems such as SGI's OpenGL Vizserver, and the MIDAS [14] component of the Merlot project. Similarly VNC allows users to interact with remote virtual desktops.

In order to create a transparent system for visualization on the Grid, we need a remote rendering technology that can support multiple remote renderers, but display rendered frames in a single client window. Furthermore, users must be able to interact with the render window in an implementation independent manner. A visualization rendered using VTK, for instance, should behave identically to one rendered with IRIS Explorer. Unfortunately, no existing technology can completely satisfy these requirements. Vizserver, MIDAS and VNC integrate with the remote server at the operating system level, and so the unique look and feel of the remote software is re-created on the local desktop. Additionally, each remote application is displayed in a separate window locally, destroying the illusion of a black box for Grid visualization.

To solve this problem, we require a library for remote rendering that can be tightly integrated with visualization applications, in much the same way that the gViz and RealityGrid libraries allows us to instrument code to support computational steering. The remote rendering software should allow the transmission of rendered frames from multiple servers to a single client instance, offering to the user the ability to seamlessly switch between visualization streams. We discuss such a software implementation in section 5.1.3.

4.2 Grid Middleware

Initial Grid middleware focused on batch processing, where the model of computation is one of executing a job and checking back for the results some time later. More recent developments such as the Open Grid Services Architecture (OGSA) [15] and the Web Services Resource Framework (WSRF) have attempted to introduce a service orientated architecture to Grid computing, but uptake by HPV service providers has not yet been widespread. As a result there is something of a bifurcation between the former usage of the Grid mainly for compute and file transfer tasks embodied in middleware such as GT2, and the more recent emphasis on service orchestration via workflows, as in projects such as ^{my}Grid [16].

Distributing visualization dataflows and pipelines on the Grid requires a combination of both approaches, since the stages of the pipeline are much more tightly coupled than in conventional workflows, and issues of parallelism and concurrency both within and between the components of the pipeline are also raised. As such visualization raises issues of deployment and orchestration of components and services. The RealityGrid project has addressed these issues specifically and has created lightweight middleware for such component deployment in WSRF::Lite [17]. This approach also brings the possibility of coupling an application's data output directly to the visualization components, without first having to incorporate that application into any specific visualization framework. An important issue this highlights is the necessity to express concurrent resource co-allocation. The various components must be deployed concurrently and coupling via socket-based data transfer since the simulation is refreshing the data to be transformed and rendered by the visualization. This requirement for concurrency is not currently satisfactorily addressed in Grid middleware although some super-schedulers such as the Maui silver scheduler [18] can provide this, albeit in a non-standard manner.

We now discuss a framework that will allow the creation of our abstract Grid visualization system.

5 The e-Viz System

We have developed a prototype system which implements the ideas introduced above. The system consists of three main components; the client machine, the remote HPV machine(s), and a broker. Each component is introduced below, together with associated software modules. Additionally, the relationship between modules is depicted in Figure 3.

5.1 Client

The user's machine is where the visualization pipeline is controlled from and outputs to, though not necessarily where the data resides. There are two client side software modules: a launcher application and a generic user interface used to both control pipeline parameters and to view visualization output.

5.1.1 Launcher

The launcher is a C++ application, written using the QT GUI libraries from Trolltech in order to provide cross-platform portability. It is the user's entry point to the

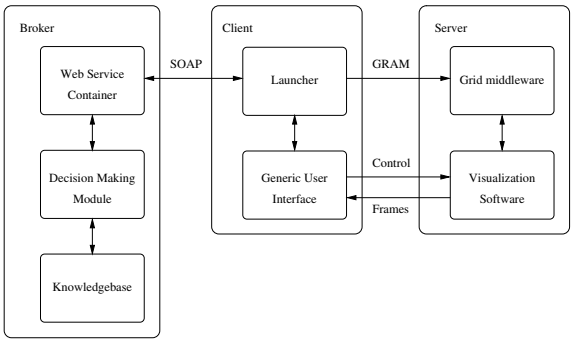

Figure 3: Architecture of the e-Viz system

e-Viz system, and provides a wizard based interface to allow users to specify their jobs in terms of input data sets and desired visualization output. The launcher uses gSOAP to make calls to the web services on the broker machine, and is bundled with the Java Commodity Grid (CoG) Kit. By removing a dependency on the user to have a Globus client pre-installed, we hope to maximise the potential user base of the e-Viz system.

5.1.2 Generic User Interface: Pipeline Control

The e-Viz user interface consists of two components; one to control the pipeline parameters by means of the gViz computational steering library, and one to display the visualization output.

The generic user interface for pipeline control has been implemented using Java, and is comprised of a communications framework, a widget set and an XML processing component. The interface starts as a blank container and is configured using the abstract description of the pipeline, which is generated by the broker. This description, encoded in XML and currently based around skML, contains such information as the list of components in the visualization pipeline, how they are connected, where they are running, what user interface elements they have and hints on what types of widgets satisfy those user interface elements.

A control panel is built for each of the components in the pipeline, providing appropriately labeled and typed widgets, setting their initial values based on the contents of the XML description. The widgets used are currently selected from a small set of basic sliders, buttons and text boxes but it is expected in the future that a richer set of user interface elements will be provided. This could include colour map and transfer function editors, or possibly user defined widgets. The selection of widget type is made based on possible hints provided in the XML description or chosen based on the parameters data type.

The now tailored user interface is necessarily decoupled from the actual remote visualization pro-

cesses and both delivery of user selected parameters and receipt of visualization process-derived parameter changes are required. For example, the user may set a desired level of geometry compression, but the process delivers back the actual value achieved, requiring an update to the user interface. This communication process is managed using a new Java implementation of the client component of the gViz steering library. For each distinct connection point defined in the XML description (several pipeline components may in reality be implemented by a single process and hence share a connection point) a gViz connection is set up to send and receive changes related to named parameters, updating the user interface accordingly.

5.1.3 Generic User Interface: Visualization Output

A remote rendering library has been created which can be integrated with visualization applications to enable multiple servers to deliver frames to the same client window. The library has been written in C in order to maximise the number of visualization applications with which it can be integrated. A number of codecs are supported, each offering different trade-offs between compression ratio, image quality, and the time taken to encode and decode; frames can transmitted raw and uncompressed, or encoded using JPEG, PNG, Colour Cell Compression Delta and Run-Length Encoding algorithms. Meta-data is gathered relating to the time taken to encode, transmit and decode frames, so that in the future the system can be extended to automatically select the most appropriate codec for the current environment. As stated, the library supports multiple visualization servers, and so can switch between alternate frame streams at runtime. Servers can be added and removed at any point in the life-time of a session, meaning that an e-Viz visualization can seamlessly migrate from server to server. Additionally, the library is fault tolerant, and can cope gracefully with the unexpected loss of a remote server. This goes some way to achieving an implementation that supports the system transparency, reliability and adaptability features highlighted by the electrical power grid analogy. In the future, the client side of the library could be extended to support local rendering of triangles from remote servers, and the server side to support output to the Access Grid.

5.2 Server

5.2.1 Grid Middleware

In the general case, the only software which must be installed on a server machine is the Grid middleware, since visualization software could potentially be transferred onto the machine at runtime. In reality, many visualization applications have per-machine license requirements and non-trivial installation procedures. Furthermore, automated resource discovery and software installation procedures are outside the remit of the e-Viz project, and so we currently make the assumption that software is pre-installed onto our HPV machines. In the future though, we look to include such capability through maturing third party software. In the meantime, we use GT2.

5.2.2 Visualization Applications

As a first step towards the development of a prototype e-Viz system, we have chosen to focus on two different visualization platforms; VTK, and the Real Time Ray Tracing (RTRT) software [19], sometimes known as *ray. VTK is a complete software development kit for visualization applications, supporting a wide range of algorithms and techniques, and is available on a large number of platforms. Being an SDK, it is aimed at software developers and not application scientists. RTRT on the other hand is a parallel ray tracing application with support for volume rendering through ray casting. As such it could be said to be aimed more at end users than developers. The difference between the applications should help to evaluate the degree to which the e-Viz system can abstract away from implementation technologies.

The visualization applications have been instrumented with both the gViz computational steering and e-Viz remote rendering libraries.

5.3 Broker

The broker machine is the heart of the e-Viz system, and is where decisions are made on the hardware and software that can best create an appropriate pipeline to meet a user's requirements. The client interacts with the broker by means of calls to a web service instance created for each e-Viz session. The web service is integrated with a decision making module, which itself interacts with a knowledgebase.

5.3.1 Web Services Container

The WSRF::Lite[17] implementation of the WSRF standard has been used to implement the web service container. This provides us with a framework that facilitates the rapid development of WSRF compliant web services.

5.3.2 Decision Making Module

The decision making module is a software library called by web service instances, which parses

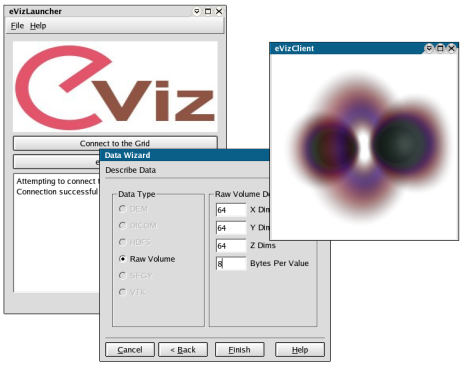

Figure 4: e-Viz Wizard and User Interface, showing the volume rendering demonstration

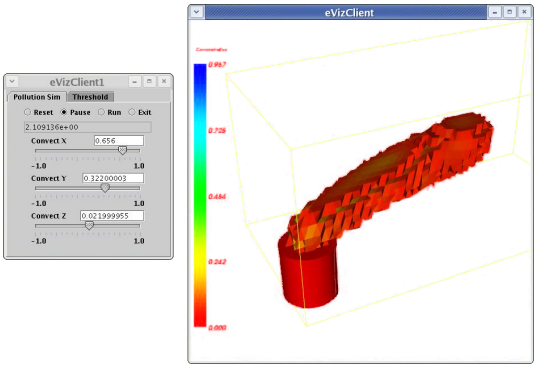

Figure 5: e-Viz User Interface, showing the pollution demonstration

XML messages containing the user's visualization requirements, and interrogates a knowledgebase to determine the most appropriate visualization pipelines. At the time of writing, this component of the project has yet to be developed, and so the e-Viz prototype is hard-wired to offer a limited selection of visualization applications.

5.3.3 Knowledgebase

A database has been implemented using PostGRE SQL in order to provide the knowledgebase component of the e-Viz system. Again, at the time of writing, this component is not integrated with the rest of the e-Viz prototype.

6 User Scenarios

We now consider how the e-Viz system could be used in practice, through two exemplar applications.

6.1 Volume Renderer

A volume rendering e-Viz pipeline has been created to allow users to visualize volume data sets. First the launcher is used to define the input data set, visualization technique, desired resolution and target frame-rate. Since we can implement volume rendering in parallel with RTRT, the system can potentially choose the number of processors that would be required in order to meet the user's desired frame-rate, if possible. The RTRT job is staged onto a SMP machine (in our case 'Green', CSAR's 256 processor Origin), and enters the batch queue. In order to provide the user with some visualization as soon as possible, a second job is staged onto an interactive machine, but this time using VTK and a downsized version of the data set, rendered at a lower resolution. The user can interact with this cut-down visualization while the full resolution job passes through the batch queue. When this happens, the client automatically connects to the more powerful machine, synchronises the visualization state, and switches the visualization streams so the user can begin interacting with the full resolution pipeline. In keeping with the notion of system reliability, the system can automatically fall back to the VTK pipeline should the RTRT job fail, or run out of processing time. If all pipelines fail, the simulation state remains on the client, and new servers can be automatically added and synchronised so that the visualization session can continue.

6.2 Pollution Demonstrator

In contrast to the volume rendering application described above in which pre-computed data is being visualized, the ability to visualize dynamically created data from running simulations is also important. This is managed in e-Viz through the launcher process which not only presents the user with the option to select a specific data set, but also allows the user to select a simulation as a data source. After choosing this option the launcher contacts a directory service to find a list of running simulations from which the user may choose. Next, the choice of visualization is selected and the XML description of this pipeline generated (in future this step will be handled by the e-Viz knowledge base). This XML description not only describes the visualization processes but also the numerical simulation as components in the overall pipeline. This complete XML description allows the generic user interface to create controls for the simulation in order to allow computational steering. In addition, user interface controls for the visualization components are provided as described earlier 5.1.2.

This has been demonstrated by re-working the gViz pollution simulation scenario [10] using e-Viz. Previously this simulation was steered and its data visualized on the desktop using IRIS Explorer with a user created visualization pipeline. Now, the e-Viz system selects appropriate resources for the

visualization components, the generic user interface is configured from the XML description of the pipeline and the thin client delivers the rendered results (see figure 5).

7 Future Work

We intend to improve the system by expending research effort in a number of areas. Firstly, the creation of an abstract visualization language is essential to the success of the project, and work has already begun on this topic. Additionally, as highlighted in the text, a decision making module must be implemented in order to present users with a choice of visualizations, and to instantiate pipeline instances. Another area of future research relates to the field of expert systems. It is envisaged that intelligent software agents can be used to monitor a user's visualization session, and to recommend alternative pipeline configurations in the form of new camera positions, transfer functions etc. It is hoped that the system could use feedback from previous user's experiences to configure the pipelines of current users.

8 Conclusions

The e-Viz framework we have introduced and the prototype implementation we have described aim to create a Grid-based abstract visualization system for application scientists. The system abstracts away from implementation technologies, and in doing so creates a system for Grid visualization which is transparent, adaptive, and fault tolerant.

9 Acknowledgments

Financial support for this work was provided by the Engineering and Physical Sciences Research Council through grant numbers GR/S46567/01, GR/S46574/01 & GR/S46581/01.

References

[1] R.B. Haber and D. A. McNabb. Visualization idioms: A conceptual model for scientific visualization systems. *Visualization in Scientific Computing, IEEE Computer Society Press*, pages 74–93, 1990.

[2] I.T. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[3] K.W. Brodlie, J.M. Brooke, M. Chen, D. Chisnall, A. Fewings, C. Hughes, N.W. John, M.W. Jones, M. Riding, and N. Roard. Visual supercomputing - technologies, applications and challenges. *Computer Graphics Forum*, 24(2), 2005.

[4] I. Foster and C. Kesselman. Computational grids. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 15–25. Morgan Kaufmann, 1999.

[5] N.W. John, R.F. McCloy, and S. Herrman. Interrogation of patient data delivered to the operating theatre during hepato-pancreatic surgery using high performance computing. *Computer Aided Surgery*, 9(5/6), 2005.

[6] E.W. Bethel, B. Tierney, J. Lee, D. Gunter, and S. Lau. Using high-speed wans and network data caches to enable remote and distributed visualization. In *SC*, 2000.

[7] I.J. Grimstead, N.J. Avis, and D.W. Walker. Automatic distribution of rendering workloads in a grid enabled collaborative visualization environment. In *SC*, page 1. IEEE Computer Society, 2004.

[8] D. Kranzlmüller, P. Heinzlreiter, H. Rosmanith, and J. Volkert. Grid-enabled visualization with gvk. In *European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, pages 139–146. Springer, 2003.

[9] I. Foster and C. Kesselman. Globus: A toolkit-based grid architecture. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, page 259. Morgan Kaufmann, 1999.

[10] K.W. Brodlie, D.A. Duce, J.R. Gallop, M.S. Sagar, J.Walton, and J.D. Wood. Visualization in grid computing environments. In *IEEE Visualization*, pages 155–162. IEEE Computer Society, 2004.

[11] D.A. Duce and M. Sagar. skml: A markup language for distributed collaborative visualization. In *Proceedings of Theory and Practice of Computer Graphics*, pages 171–178, 2005.

[12] D.J Duke, K.W. Brodlie, D.A. Duce, and I. Herman. Do you see what i mean? *IEEE Computer Graphics and Applications*, 25(3):6–9, 2005.

[13] S. M. Pickles, R. Haines, R. L. Pinning, and A. R. Porter. A practical toolkit for computational steering. *Philosophical Transactions of the Royal Society*, 2004.

[14] The MIDAS project: `http://www.llnl.gov/icc/sdd/img/midas.shtml`.

[15] I.T. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

[16] The ^{my}Grid project: `http://www.mygrid.org.uk`.

[17] WSRF::Lite: `http://www.sve.man.ac.uk/Research/AtoZ/ILCT`.

[18] David B. Jackson. Grid scheduling with maui/silver. *Grid resource management: state of the art and future trends*, pages 161–170, 2004.

[19] S.G. Parker. Interactive ray tracing on a supercomputer. *In Practical Parallel Rendering*, 2002.