A user interface framework for Grid-based computational steering and visualization.

Jason Wood[1], Mark Riding[2] and Ken Brodlie[1]
[1]University of Leeds, [2]University of Manchester

Abstract
This paper describes a flexible and extensible user interface tool for interacting with simulations and visualization applications running on the Grid. It uses an XML description of the visualization pipeline to provide parameter and location information with which to configure the user interface and connect it to remote applications using the gViz library. Messages in this XML format allow for the dynamic alteration of the interface in response to user and system behaviour. A framework is provided into which user created widgets can function alongside or replace the system widgets. This framework also allows alternate communication mechanisms to be added to allow the encapsulation of a broad set of applications within a single pipeline.

1. Introduction

Grid computing [FK99] offers the potential for users to access large distributed computing resources on demand. This may be in the form of job submission to batch queues, timed access managed through advanced reservation, or interactive access. The latter two options should offer the potential for performing interactive visualization tasks on high performance Grid computers using standard or specialist visualization software. However, in addition to the scheduling hurdle, visualization on the Grid provides us with a number of other challenges.

In typical Grid computing, irrespective of the scheduling arrangements, the physical resources provided to satisfy a request may well be different each time the request is made. In the worst case, this means a different cluster or supercomputer comprised of different hardware and with different software installed. This then limits the user's access to Grid resources for visualization if they need to wait for the right combination of resources to become available in order to run a specific piece of visualization software. Alternatively, if users were able to provide a system-independent description of the visualization they wished to achieve, then different software tools could be substituted. This would reduce the dependence on the specific hardware.

Another challenge of Grid computing is that by design all or most of a Grid application is executing remotely from the user's desktop. In general this is unimportant. A simulation which generates data as files that are retrieved at the end of a run does not require any interaction with the user while it executes. This is entirely different for visualization whose very purpose is to present information to the user in a graphical form, providing greater benefits if driven by the user for interactive exploration. This then provides a requirement for a user interface that not only displays the graphical output of the visualization but also allows the user to drive the remote visualization pipeline from their desktop.

The e-Viz project [RWB*05] has been investigating the area of high performance visualization. The system is designed to guide the user through a series of screens to help them choose the visualization techniques they wish to use for visualizing their data. From this process the e-Viz system generates an XML description of that pipeline and finds available hardware and software resources on which to implement it. The system then starts two tools on the user's desktop to provide a local user interface to remote running visualization applications. One tool is for displaying the remotely rendered graphical output, and the other is to act as a user interface for interacting with the dynamically constructed application.

This paper describes the e-Viz Client user interface tool. It gives details of how the XML description in the e-Viz system is used to automatically configure the user interface, how XML messages can be used by remote visualization or simulation components to modify the interface, and how user preferences can be used to personalise it. The paper also describes the e-Viz Client's plug-in architecture that allows user contributed communications and interface components to be integrated and selected at run-time.

2. Related Work

A range of strategies for delivering control of remote visualization processes to the user's desktop have been developed. One is to split the user interface from the computationally intensive visualization components in a client- server model. The user interface (client) component executes on the user's desktop while the visualization (server) components can be run on one or more remote compute resources. The two components are then joined by a communications protocol. This is the approach taken by Paraview [Par] which uses VTK [VTK] for visualization and Tk [Tkt] for the user interface. It operates as a turnkey system presenting the user with a simple interface to a subset of the VTK library routines. Since it is a turnkey system, the interface required is well defined and the communications mechanism to the distributed visualization components is proprietary.

While in essence we are also providing a client side GUI, our motivation is to provide a more general purpose system. We wish to create an interface that can sit above a range of visualization systems, even within the same pipeline, and be able to provide a suitable interface on the fly. Since the systems that make up the pipeline are different, the way that they send and receive user interface parameters may also be different and so our interface framework provides a plugin architecture for communications mechanisms. A further requirement for a GUI is that its appearance can be changed by the application in response to user input or internal state changes of the application. This is a difficult problem for a distributed system sitting above a variety of visualization systems but we have tackled it through the use of XML messages that allow a visualization component to update its portion of the GUI.

Another project, GAPtk [NS05], takes a similar approach to ParaView in having user interface applications running locally on the desktop to control remote processes. GAPtk is based on a service-oriented architecture and provides a number of visualization and data manipulation services which can be run on remote Grid resources. A client library is provided in a number of common languages that can be built into user interface software. It provides a simplified API through which to communicate with the remote services that make up the application. In this approach, tailored user interfaces are created by developers for use with a single Grid application, i.e. a specific selection and arrangement of the services provided by the GAPtk project.

A second approach is as follows. Rather than running the actual user interface code for a remote application locally, an image of the user interface running on a remote machine is provided on the user's desktop. The user interacts with the image as though it were the actual interface and the underlying technology, VNC [VNC] for example, translates this as interactions with the real user interface. This approach has been taken by TeraGrid at the Texas Advanced Computing Centre (TACC). Here users are allocated a Grid resource with a graphics card and a VNC session is started, exporting the desktop of that machine. The intention is to allow users access to remote high performance hardware, visualization software and their own large data sets. While this gives reasonable performance, requiring only moderate bandwidth to appear interactive, and provides access to common visualization tools, it may not utilise the Grid to the best effect. Unless the visualization tools themselves are coded to take advantage of more than a single resource (multiple processors, multiple graphics cards), then the user is merely presented with desktop tools but through a less interactive medium. In contrast we provide a user interface tool that runs on the desktop, so is always interactive, and is able to sit above the visualization software that is selected at runtime by the e-Viz system. This gives a uniform interface to a range of underlying visualization tools.

Finally, a third approach is the common strategy of using a web browser as a portal to access Grid resources for computational tasks is common [MYG]. This mechanism has been extended to visualization by projects such as nanoHub [Nan]. Here a complete environment has been created for researchers in the field of nanotechnology. It provides users with simulations that they are able to execute on remote resources and allows them to subsequently visualize the results on a remote graphics card with the image returned to the desktop. As well as being able to use the simulations provided by the system, users are able to upload their own simulations. However, these must first be integrated by the system providers before they are available. nanoHub is tailored to a specific application area where the choices of visualization have been made by the system's creators. By contrast, the e-Viz approach has been to guide the user through the creation of their own visualization pipeline and then provide a user interface that is tailored to this resulting pipeline. Also, in nanoHub the interface elements are provided by the portal, while in our system users are able to write their own interface widgets and override system choices at run time with their own.

3. XML for visualization pipeline descriptions

A traditional approach to describing a visualization task is to use the Haber and McNabb data flow reference model [HM90]. This represents the visualization task as a sequence of connected processes that transform data from its raw state to an image. A number of commercial (IRIS Explorer [Wal04]]) and open source (openDX [OPE]) systems implement this model to provide users with an environment in which to create visualization pipelines. The eViz system too uses this reference model as the basis for describing its visualization tasks. In contrast to the systems above which use proprietary file formats to encode their pipelines, eViz uses XML and outputs its pipelines using an extended version of skML [DS05] from the gViz project [BDG*04].

skML is made up of a hierarchy of elements where a single skML document may contain one or more named map elements. A map is the skML name for a data flow pipeline which contains a collection of modules and links. Each map element contains one or more named module elements and zero or more link elements. Link elements link output data ports to input data ports on referenced module elements. Each module element contains zero or more named param elements.

skML goes beyond just describing a static snapshot of the system, it provides a mechanism to dynamically change the visualization pipeline. These changes are achieved by the provision of an action attribute associated with the module and link elements. These attributes allow for creation (default), destruction and modification of these elements.

In addition to the skML component with its multiple maps, a document can also contain an RDF [RDF04] section. This is used to add "run-time" information such as the resource allocated, or the name of the software tool used for a particular component of the pipeline. This is useful in a Grid context given that hardware and software resources may vary between uses of the same visualization pipeline.

To allow the eViz client tool to generate user interfaces for the dynamically created pipeline, it has been necessary to extend the original skML description in the following ways:

- We add a `type` attribute to the parameter element of skML so that an appropriate widget can automatically be selected and presented in the user interface.
- We add an `interaction` attribute to the parameter element to allow the distinction between parameters that are for user input, and those that are for system output only. Its values can be `steer` (input and output) or `view` (output only)
- We also add a `widgetType` attribute to the parameter element, allowing a specified choice of widget to be included rather than the automatically selected one.
- We add an `action` attribute to the parameter element, allowing dynamic changes to the way parameters are presented in the user interface - for example, this allows the parameter to be hidden in the interface, or visible but inactive.

Further additions are added to the RDF section as they either represent runtime information, or are system specific details and do not aid in the description of the dataflow pipeline. We have extended the RDF in the following ways:

- For communications we add a `CommsType` tag to specify the mechanism to be used when communicating with specific components. We also add a `ContactPoint` tag so that a port number can be specified for those communications mechanisms that require it. These apply at both the map and the module level.
- To enhance the automatic layout of control panels we have added `Layout` and `LayoutElement` tags. This allows for a basic grid arrangement of parameters to be described to give the control panel some structure and organisation.

skML is not the only XML format for visualization pipeline description, VisTrails [BCC*05] for example provides an alternative transactional style XML format. The choice of skML was based on past experience with the language and the knowledge that it required only minimal changes.

4. Implementation

4.1. Architecture

The e-Viz Client is an application written in Java that takes a skML document and uses it to enable the generation of a user interface for remote simulations and visualization tools. The e-Viz Client takes either a filename or a URL to the skML document.

In addition to the XML parser component, a number of basic widgets are provided. These include: text boxes; integer and floating point sliders; menu boxes; radio buttons; simple colour and transparency editors. These are selected as appropriate when the interface is constructed based on the information provided about the pa-

rameters in the skMl descrption. As well as providing a widget set, a Java interface (e-Viz Client Widgets Interface in Figure 1) is defined that allows widgets to be created and added to the e-Viz system. They can be referenced by name as the `widgetType` attribute of a parameter in the skML document or defined in a local configuration file. Conforming to the interface allows the widget to interact seamlessly with the e-Viz Client.

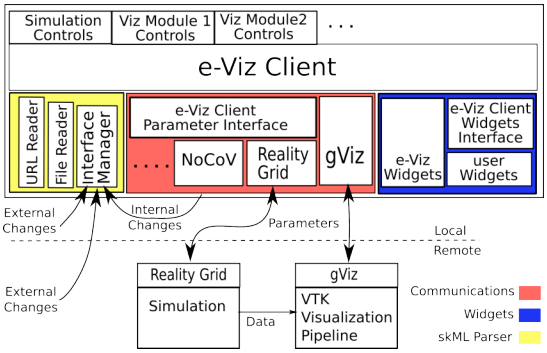


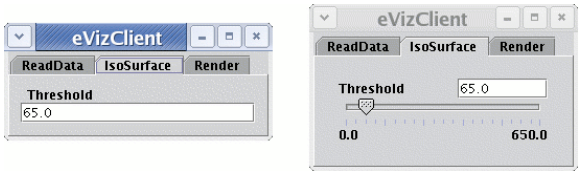
Figure 1: Architecture of the user interface tool.

Communications are provided by default through a Java implementation of the client side component of the gViz library. It is assumed that the remote components have been integrated with the server component of this library. For tools that are already integrated with other communications libraries, a Java interface (e-Viz Client Parameter Interface in Figure 1) is provided for accessing parameter values on the e-Viz Client. This allows plugins to be created that sit between the e-Viz Client and the remote component to provide communications in the native format. These are referenced in the RDF section of the skML file on a per module basis and are instantiated at run time.

4.2. Generating the Interface

The e-Viz Client starts as a blank template upon which a user interface for a particular application is constructed using a skML document. A tabbed interface is created where each component of the pipeline is represented as a separate tab. Widgets that represent the parameters of a visualization component are placed on that component's tab. Figure 2(a) shows the effect of parsing the simple skML example shown below which is in the non extended skML format. The Threshold parameter is represented as a text box since no type attribute is present in this default skML description. With the addition of type information, more sensible choices of widget can be made automatically. Figure 2(b) shows the result of the same skML but with a `type="Float"` attribute added to the Threshold parameter. This time a float slider is selected as the appropriate widget.

```
<skml>
 <map id="isosurface">
  <module id="DR" name="ReadData"
                    out-port="Output">
   <param name="Fname">Dglazing.dat</param>
  </module>
  <module id="IS" name="isoSurface"
     in-port="DataIn" out-port="Geomtery">
    <param name="Threshold">65.0</param>
  </module>
  . . . . .
  <link id="DRtoIS">
   <module ref="DR" out-port="Output" />
   <module ref="IS" in-port="DataIn" />
  </link>
  . . . . . .
 </map>
</skml>
```



(a) *(b)*

Figure 2: *Two images showing the same tab of the user interface, (a) is using original (b) is generated from the extended skML*

Types that exist are text, float, int, textArray, floatArray, intArray and enum. The type enum has a second attribute of labels which allows text to be presented to the user for choice widgets. This type is represented either as a set of radio buttons, or if the number of options is large then it switches to using a drop down menu. All widgets have a label to identify them which comes directly from their `name` attribute in the skML file. A further attribute associated with widgets is `interaction` which can be one of two values, `steer` or `view`. Most user interface widgets are of type `steer` (to indicate they can be altered by the user), but `view` would be used to indicate they can only be changed by the system and are for information only - for example, a parameter such as the current time-step of a simulation.

The e-Viz Client is only capable of basic layouts. If the number of widgets is small then the layout is set as a row; if the number is large then it forms the widgets into a grid to try and keep the interface square. To generate interfaces with a particular arrangement of widgets, a Layout tag can be supplied with the RDF section for a module.

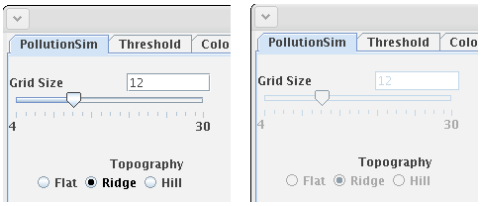
4.3. Reactive User Interfaces

User interfaces to applications are not static, they change in response to changes in the application, or changes made by the user. One novel feature of our system is that we seek to replicate this behav-

iour but through the use of the XML description of the system. By this we do not just mean changes to the pipeline by adding or removing modules, or by simply changing parameter values. These features are offered by systems such as VisTrails or the skML pipeline editor. We wish to use the XML description to change the representation of the parameters in response to user or system behaviour, hence the addition of the `action` attribute described in section 3.

In terms of user behaviour, for example, it is useful to reduce the cognitive load on users by dynamically modifying the list of options shown to users based on their past behaviour. For example if a user never modifies the `decimate` parameter for a Marching Cubes module then its widget can be hidden from the control panel:

```
<map id="MCMap" >
  <module id="2" name="MarchingCubes"
                          action="modify">
  <param name="Decimate" action="hide" />
  </module>
</map>
```



(a) *(b)*

Figure 3: XML messages are used to control the appearance of widgets on the user interface, (a) enabled or (b) disabled.

The e-Viz Client also allows changes in response to application behaviour. This can be done either at the whole interface level, where messages from some external controlling agent inform the e-Viz Client that a new pipeline component has been started and therefore requires a tab on the interface and communications to be implemented; or at an individual module level where a component in the current pipeline is able to modify its own parameter representations.

In computational steering, for example, there are typically parameters associated with the setting up of the numerical solution that must not be changed during the course of a run. An example might be the grid size over which the solution is computed; or some parameter defining the problem, such as pressure under which a simulation is carried out. In contrast, other parameters (such as frequency of output of results) are changeable at any time. Thus the interface needs to react in response to information from the simulation, concerning its present state: for example, is it at its initial step or midway through?

```
<map id="PollSim">
 <module id="2" name="PollutionSim"
                          action="modify">
 <param name="Grid Size" action="disable"/>
 <param name="Topography" action="disable"/>
 </module>
</map>
```

Here we have set the `action` attribute to be `disable` to allow the user interface to display these widgets, but indicate that they are not currently active (see Figure 3). As well as being able to hide and change the interactive state of a widget, the system allows for replacing the representation of a widget being used, or even modifying the layout of a whole module.

4.4. Personalised Interfaces

It is important to be able to adapt the user interface to the experience and preferences of individual users. We address this in two ways with the e-Viz Client. Firstly, the user can create a configuration file in which they can record preferences for default settings. This can also be used by the system to record recently unused widgets so that a compact version of the interface can be displayed as an option (as discussed above). Secondly, the system is extensible and allows for the addition of user created widgets (see architecture in Figure 1). An interface specification is provided and any widget that conforms to that specification can be used. The configuration file contains a list of mappings from module/parameter name combinations to widget names. When a skML document is parsed and the system finds an occurrence of a module/parameter name combination it substitutes the widget chosen by the user in place of the system-prescribed widget. This substitution happens at run time.

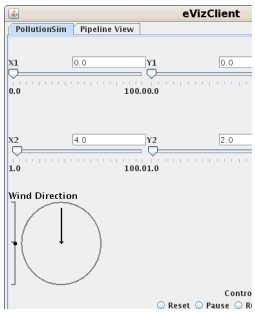


Figure 4: The Wind Direction parameter represented by a user supplied 2D compass widget capable of delivering a vector of 3 floats.

An example of this is illustrated in Figure 4, where by default three sliders would be provided to represent a vector of three floats, used to implement the parameter called Wind Direction. These have been replaced by a simple 2D compass type widget plus an elevation control. As well as being able to substitute individual widgets

at run time, the user can elect to remove widgets from the control panel or even edit the layout of a control panel through directives set in the local configuration file.

4.5. Communications Mechanism

The e-Viz Client is designed to be an interface for Grid-based visualization applications. This necessarily means that it will be interacting with remotely executing components and must therefore provide a communications mechanism. By default the system uses the gViz library [BDG*04] for parameter communications and locates visualization components through information provided by their `PhysicalLocation` and `ContactPoint` tags in the RDF section of the skML document.

While we use the dataflow pipeline model to abstractly define the visualization that we wish to achieve, in reality it may not be implemented as a series of independently executing modules linked together. A single piece of software may have been identified that could implement all or many of the individual steps in the pipeline and may have been started on the remote server. This means that in the RDF section the `PhysicalLocation` and `ContactPoint` tags for all or many of the pipeline components will actually be the same. In these situations, rather than generating many communications channels to the same piece of software, the e-Viz Client groups these components together and manages their communications through a single connection.

Although gViz is the default communications mechanism, an interface is provided to allow other communications libraries to be used. The `CommsType` tag in the RDF section is used to indicate, on a per module basis, the mechanism that is to be used. This allows maximum flexibility when adding new remote visualization components. Rather than recoding the remote application, a plugin can be written for the e-Viz Client, using the parameters interface, to link the two together. Two other communications mechanisms have been implemented for the e-Viz Client. One is for the RealityGrid [PHP*04] steering library and the other is for a web services based system called NOCOV [WBH*06].

5. The gViz Computational Steering Library

Work done in the gViz project [BDG*04] looked at allowing users to interact with and visualize data from simulations running on Grid resources. The outputs from this project included a Grid enabled version of IRIS Explorer, the skML visualization pipeline description language (described in section 3) and the gViz steering library.

The extension to IRIS Explorer allowed components of the pipeline to be placed on selected Grid resources, authenticated through a Globus [Fos05] certificate. This allowed visualization processes to be moved to the location of the data, rather than bringing the data to the desktop. In the context of computational steering, the gViz library was used to connect modules in the visualization pipeline to running numerical simulations for steering of parameters, while the data was directed to the remotely executing visualization components.

While IRIS Explorer was used as the main exemplar in the gViz project, the gViz library was intended to be system independent. It was designed to be easily integrated with simulations written in C/C++/Fortran to: allow the simulation to present multiple parameter and data streams; handle connections to streams from multiple clients; transfer data and parameters to clients when made available by the simulation; accept parameter changes from clients and queue them until requested by the simulation; provide standard TCP socket connections as well as authenticated connections through the Globus API; and provide a web services interface.

This functionality worked well for computational steering, but further work has been done to allow this API to be used in the more general setting of user interface communications. Much of the work has been to implement a native Java implementation of the client component of the gViz library. This allows the e-Viz Client to communicate with components instrumented with the original C implementation. We have also extended parameter XML schema to include enumerated types and array types as well as provide a commit flag (described later).

While the queue approach for parameters is appropriate for simulations, where we need to guarantee that all parameter changes delivered are received by the simulation, it can be a hindrance for other applications. We use the gViz library to connect not only the e-Viz Client to the back-end visualization components, but also to connect the viewer window to the back-end renderer. Here we are delivering view position changes as the user drags the mouse, and receiving images of remotely rendered results. Many mouse positions can be generated in the time it takes to render and deliver a single frame. These unrequired view positions need to be ignored, taking only the most recent view position for the next frame. Extracting the last view position from the general queue of parameters is problematic so an option was created that allows the developer to nominate parameters to be ‘non-queuing’. The values of these

parameters are continuously overwritten with newly arriving values so that when the renderer requests a new viewpoint position it is guaranteed to get the latest.

While the gViz library supports multiple connections to parameter and data streams, it could not be said to be providing a truly collaborative environment. One user may become aware of another's actions only after the result of these actions becomes apparent in the visualized output. Work has been done [WW05] to extend the gViz API to support a collaborative mode of working. A commit tag has been added to the parameter XML schema and a new committed parameter stream added to the library. When this stream type is used, any uncommitted parameters sent to the simulation are merely reflected back to all connected clients. This allows these parameter changes to be seen by collaborators and discussed before a final decision is reached. Once a new parameter value has been agreed, it is submitted in a committed state and is passed to the running simulation.

To support the dynamic changes to the interface by remote visualization components, XML messages need to be sent detailing these changes. Rather than mixing interface and parameter updates in the same XML message, a new stream has been added to the gViz library labelled Interface. This is activated by remote codes that wish to use the interface stream through a gViz library API call. The e-Viz Client simply makes a connection to the Interface stream using the same connection information as for the parameter stream. Inputs that arrive from this stream are directed by the gViz communications plugin to the interface manager, rather than the parameters manager. This functionality is provided as part of the parameters interface so is therefore available to other communications mechanisms.

6. Applications

6.1. Environmental Pollution

One of the motivating scenarios for the gViz project was that of an environmental pollution disaster. In this scenario an accidental release of a chemical has taken place and it is necessary to compute in faster than real-time the concentrations of the pollutant in the environment, to inform decisions with respect to evacuation of population centres. The pollutant is moved under the action of the wind which may change as time progresses and alter the levels of concentration in different locations. This scenario is modeled using a PDE-based numerical simulation generating data that is visualized and presented to the user in real time. The user is able to change the direction of the wind while the simulation is running and see the effects of these changes as they are computed.

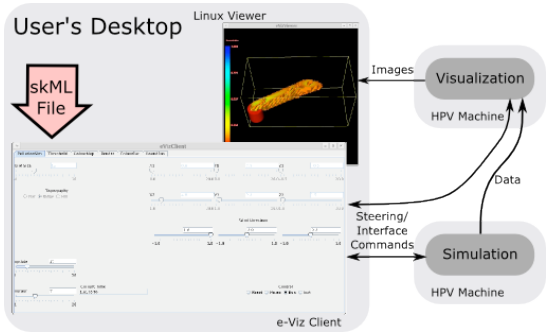


Figure 5: Environmental Pollution Demonstrator

In the gViz project a number of different visualization systems were used as clients, each one requiring a hand coded user interface to be created for the demonstrator. By contrast Figure 5 shows the automatically generated user interface for the same pollution application. Any changes to the underlying visualization pipeline do not require re-coding of the interface, merely reading the new description is enough to provide a new interface.

6.2. Molecular Visualization

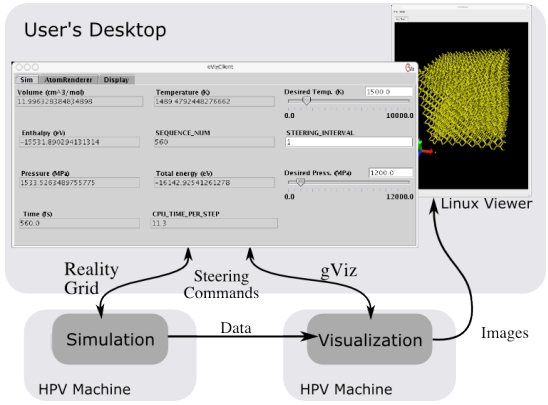


Figure 6: Molecular Visualization using VMD

The RealityGrid library for computational steering has been used by researchers at Penn State University to instrument a molecular dynamics code used to simulate the annealing of crystalline silicon. At the end of each time step the simulation sends molecular data over a socket to a remote instance of the Visual Molecular Dynamics (VMD) application for visualization [HDS96]. The VMD application has been modified using the RealityGrid library to connect to a simulation as a data source and using the gViz library to expose visualization parameters.

By creating an XML description of this distributed visualization pipeline, the e-Viz client can be

used to automatically generate a single user interface to both the simulation and the visualization, using the RealityGrid and gViz communication mechanism respectively. The simulation and visualization appear as separate tabs in the user interface, as depicted in Figure 6.

7. Conclusions and Future Work

This paper has demonstrated a user interface tool for Grid-based visualization systems that is automatically configured based on the XML description of the pipeline. This system is extensible to allow users to: add layout information for control panels; create and add their own widgets to the system for instantiation at run-time; create communications plugins to allow control of software instrumented with communications libraries other than gViz. It goes beyond just providing a static interface from the original description, but allows snippets of XML to update and manipulate the interface.

The extension to skML allows for the generation of interfaces, but the vocabulary used for the modules is undefined. Work is needed to create an ontology for visualization that will allow truly system independent descriptions of visualization pipelines. This will lead to improved automatic selection of applications with which to implement a user's visualization request. With its parameter and widgets interfaces, the e-Viz Client is flexible enough to sit above any visualization application that has an external means of control.

References

[BCC*05] BAVOIL L., CALLAHAN S. P., CROSSNO P. J., FREIRE J., SCHEIDEGGER C. E., SILVA C. T., VO H. T.: VisTrails: Enabling Interactive Multiple-View Visualizations. In *Proc. of IEEE Visualization* (2005).

[BDG*04] BRODLIE K., DUCE D., GALLOP J., SAGAR M., WALTON J., WOOD J.: Visualization in Grid Computing Environments. In *Proc. IEEE Visualization 2004* (2004), pp. 155–162.

[DS05] DUCE D.A., SAGAR M.: skML: A Markup Language for Distributed Collaborative Visualization. In *Proceedings of Theory and Practice of Computer Graphics* (2005), pp 171-178.

[FK99] FOSTER I., KESSELMAN C.: The Grid: Blueprint for a New Computing Infrastructure. In Computational Grids, chapter 2, (1999). Morgan-Kaufman.

[Fos05] FOSTER I.: Globus toolkit version 4: Software for service-oriented systems. In *Proc. IFIP International Conference on Network and Parallel Computing* (2005), pp 2–13.

[HDS96] HUMPHREY W. DALKE A. SCHULTEN K.: VMD Visual Molecular Dynamics. In *Journal of Molecular Graphics* (1996)

[HM90] HABER R. B., MC NABB D. A.: Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. In *Visualization In Scientific Computing* (1990), Shriver B., Neilson G., Rosenblum L., (Eds.), *IEEE Computer Society Press*, pp. 74–93.

[MYG] RENNICK EGGLESTONE S.: A portal interface to myGrid workflow technology. *In the Proceedings of the NETTAB workshop on Workflow Management* (2005).

[Nan] nanoHub. `http://www.nanohub.org/`

[NS05] NAGELLA S., SASTRY L.: Visualization on the UK National Grid Service using GAPtk, a generic toolkit. In *Proc. AllHands Conference* (2005)

[Ope] OpenDX. `http://www.opendx.org/`

[Par] ParaView. `http://www.paraview.org`

[PHP*04] PICKLES S., HAINES R., PINNING R., PORTER A.: Practical tools for computational steering. In *Proc. of UK e-Science All Hands Meeting* (2004).

[RDF04] Resource Description Framework (RDF). `http://www.w3.org/RDF/`, 2004.

[RWB*05] RIDING M., WOOD J., BRODLIE K., BROOKE J., CHEN M., CHISNALL D., HUGHES C., JOHN N., JONES M., AND ROARD N.: e-Viz : Towards an integrated framework for high performance visualization. In *Proc. of UK e-Science All Hands Meeting* (2005).

[Tkt] TK toolkit: `http://www.tcl.tk/`

[VNC] tightVNC. `http://www.realvnc.com/`

[VTK] VTK. `http://www.vtk.org`

[Wal04] WALTON J. P. R. B.: NAG's IRIS Explorer. In *Visualization Handbook* (2004), Johnson C. R., Hansen C. D., (Eds.), Academic Press.

[WBH*06] WANG H., BRODLIE K., HANDLEY J., WOOD J.: Service-Oriented Approach to Collaborative Visualization. In *Proc. of UK e-Science All Hands Meeting* (2006).

[WW05] WOOD J., WRIGHT H.: Steering via the Image in Local, Distributed and Collaborative Settings. In *Proc. of UK e-Science All Hands Meeting* (2005).