A Service Agreement Framework for Grid based Distributed Rendering

J. T. O'Brien and R. S. Kalawsky
Department of Electronic and Electrical Engineering
Loughborough University

Abstract

The increasing size of scientific data sets has prompted the development of a number of grid visualization tools, which provide collaborative remote access to these data sets. However, these systems have not addressed the underlying rendering challenges in providing visualization performance and quality guarantees for the user. We address this issue and present a new service agreement framework which provides quality-of-service guarantees for hardware accelerated distributed rendering of existing visualization applications. The framework is built around WSRF components, exploiting the WS-Agreement standard and the job submission description language (JSDL).

1. Introduction

Grid enabled visualization has developed into a key component of high-performance computing for online analysis of live simulation data from large computations [1] and collaborative exploration of captured data sets [2,3].

The ability to harness remote rendering resources offers the possibility to better exploit the high-cost of these resources through a larger community of visualization users, as has been achieved in computational science. However, for this scenario to succeed, visualization users and providers must be able to successfully form service level agreements (SLAs). This is particularly crucial for interactive visualization, where a user's understanding depends on a consistent, low-latency interactive scene at a frame rate greater than 10 fps [4]. Further requirements may be imposed by particular visualization domains. One example is medicine, where lossless rendering is often an immovable requirement.

This paper addresses the problem of forming distributed rendering SLAs from a user perspective, detailing an intuitive framework through which a user can enter into service agreements with providers. The framework makes it simple for service providers to setup SLA templates by automatically generating new templates on the fly using a base template.

This new framework represents a complete render management framework for launching and interacting with distributed rendering applications.

Where possible current and emerging web service standards are utilised, including WSRF, WS-Notification [5], the job service description language (JSDL) [6], and WS-Agreement [7] to achieve this. This standards based approach should allow existing grid visualization systems to utilise our framework. However, the limited resource requirements schema of JSDL has led us to develop a custom render resource extension, as permitted within the JSDL specification. Defining this new schema is outside the scope of this paper and should be superseded by a more flexible standard in the future.

The new framework was developed to enable SLA based ubiquitous access to rendering resources in a clinical environment. To achieve this, the service agreement framework has been implemented using a combination of WSRF::Lite and Java Servlets, to provide an SLA management system for our adaptive distributed rendering system [8]. The adaptive render system enables users to remotely interact with their existing distributed or standard OpenGL applications. Cross platform interactive access is provided through a novel web browser interface. The interface provides users with a native GUI representation of their application and the facility to monitor the status of the rendering agreements allocated to the application.

The paper is organised into four sections: (1) a discussion of the challenges involved in providing service agreements for heterogeneous render resources; (2) the architecture of our new SLA render framework; (3) details of our implementation of the framework to provide remote rendering access for medical visualizations; (4) and finally conclusions and future work.

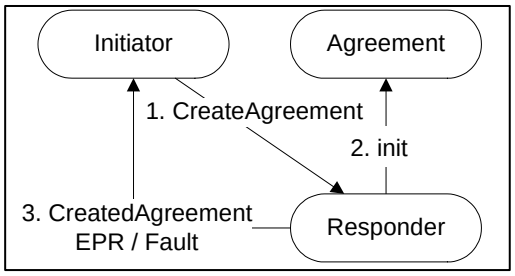

Figure 1. Synchronous agreement acceptance model with WS-Agreement.

2. Discussion

An effective agreement framework for distributed rendering must address three core problems. These are (1) the management of heterogeneous render resources; (2) establishing agreements that capture the objectives of both consumer and provider; (3) and providing effective monitoring and renegotiation of agreements.

2.1 Managing rendering resources

A resource management system in its simplest form can be reduced to a quality-of-service broker, resource discoverer, resource co-allocator, and scheduler [9]. In the context of a render management system, the quality-of-service broker component can be further reduced to the process of resolving a suitably homogeneous set of resources from a larger pool of heterogeneous resources. This is a challenging problem due to the variability in hardware accelerated rendering APIs, which in the visualization community is OpenGL. OpenGL hardware conformance does not guarantee pixel exact rendering or rendering performance between implementations or hardware [10]. This has allowed hardware vendors to target their products to the requirements of different application domains. High-end products target the demanding quality requirements of engineering and medical domains. Other vendors have optimised for the demanding frame rates of the consumer games market. OpenGL also permits vendors to develop custom API extensions, which are often quickly exploited by the scientific community to achieve novel rendering techniques [11].

Without a management system that specifically targets the conformance of rendering hardware, existing grid visualization systems must assume that rendering takes place on homogenous resources which are known to be acceptable by the user.

Current visualization toolkits rely on advertising resource capabilities using resource discovery mechanisms to allocate rendering jobs. Resource discovery mechanisms include UDDI [12] and MDS [13]. Universal Description, Discovery and Integration (UDDI) permits WSDL documents associated with a particular web service to be registered with a UDDI registry. This approach is used to advertise resource capabilities within the RAVE architecture [2]. The Metacomputing Directory Service (MDS) is used by GVK [3] and defines a heavily fault tolerant protocol for use in high-performance computing environments.

These query services cannot in themselves efficiently resolve rendering conformance across resources due to the fine granularity at which the resources descriptions would have to be stated. Instead, we favour a brokered approach to resource allocation that specifically targets the requirements of distributed rendering. The broker uses the resource requirements of a job submission request to obtain a suitable collection of resources. The broker matches both rendering features (e.g. extensions and available memory) and conformance (hardware and implementation homogeneity). The details of this are discussed in Section 3. The resource allocation process is process is performed using the WS-Agreement protocol.

2.2 Service Level Agreements

The WS-Agreement [7] standard defines a process for establishing and monitoring agreements (SLAs) and the structure that agreements take. Each agreement has four components: the agreement name, context (including the initiator and provider to which the agreement relates, and the template from which the agreement was formed), service terms and guarantee terms. The most important of these parts are the terms of the agreement. Service terms define the measurable properties of an agreement and guarantee terms define the value or objectives of the service terms. Guarantee terms can have an additional business value associated with them, which is used to weight the importance of each guarantee term.

Agreements are obtained by a responder (service provider) accepting an agreement offer from an initiator (Figure 1). Acceptance is a unilateral decision taken by the responder. A responder exposes a set of agreement templates which enable the initiator to determine the agreements that a responder is likely to accept. Agreement templates take the form of an agreement with the addition of creation constraints (or negotiation constraints).

Creation constraints specify the guarantee values that may be applied to service properties (a bandwidth service property could for instance have a constraint of a maximum of 1Gb/sec and minimum of 10Mb/sec).

WS-Agreement does not attempt to define the manner in which agreement terms and constraints are defined. This allows WS-Agreement to define a process flexible enough for both banking services and distributed rendering.

2.3 Service level objectives

Service agreements are only effective when renegotiation is minimised. Renegotiation predominantly requires some form of human interaction, and may enact penalty clauses within existing agreements.

An important component for minimising renegotiation is the choice of objectives used to define an agreement. This is often a compromise between agreement initiator and responder. A responder is better able to form agreements with low level objectives measured in resource terms, such as bandwidth, number of CPUs, or memory requirements. A resource scheduler can be used to allocate such requirements and predict the availability of future resources. However, the objectives of an agreement initiator are to carry out a task with higher level objectives (e.g. completion time or transactions per second), not to acquire an allocation of resources. These higher level objectives are determined by the end user of the agreement initiator.

Uncertainty in predicting resource objectives from higher level user objectives represents risk in meeting the requirements of the end user. This can arise from asymmetric information distribution between responder and initiator about the precise characteristics of both the application and resources. However, given that at least one party has access to sufficient information, predicting resource requirements may still be an extremely costly process.

This is particularly true for the domain visualization rendering. A service provider can readily determine the polygon processing capabilities of their rendering hardware for certain OpenGL states through a benchmarking process. However, predicting how the OpenGL characteristics of a different visualization application map to these benchmark values is non-trivial.

When the visualization application does not alter, benchmarks can be obtained by executing the application with test datasets. This is the approach favoured by benchmarking standards like viewperf. Such an approach is also possible with distributed visualization environments. These environments, including GVK and RAVE, allow users to construct workflow projects of their data using a defined set of rendering processes. The RAVE architecture, in particular, uses a small test data set, consisting of a lit sphere, to predict polygon rendering performance of rendering resources [2].

Our approach is to use tiered agreement templates. Under this approach a responder advertises basic templates which contain low level objective terms. The objective terms consist of readily guaranteed terms such as the provision of OpenGL extensions, texture memory or a particular graphical processing unit (GPU). These basic templates are complemented by high level templates targeted at known applications, which provide guarantees on frame rate and response time given known application conditions.

2.4 Agreement monitoring

A crucial component of service level agreements is the monitoring of an agreement status. WS-Agreement defines an agreement state port type through which the status of the agreement, service terms and guarantee terms can be queried. A synchronous agreement can be in one of four states (observed, observed and terminating, complete or terminated). The state of a service term allows observers to determine if this value is currently observable. Guarantee states are defined as fulfilled, violated or not determined.

All measurable states are exposed as WS-ResourceProperties. WS-Agreement allows for domain specific information to be stored inside each state element. We exploit this by placing current service values within the guarantee term state. This allows a monitoring service to determine trends within the monitored service values as well as the state of the guaranteed term.

In the case of distributed rendering, the entity judging these trends should be the end user. They are best able to judge if a small violation of a service term means renegotiation is necessary (associated business values can also help with this determination). To achieve this, a suitable monitoring interface must be incorporated into the main application interface through which the user interacts.

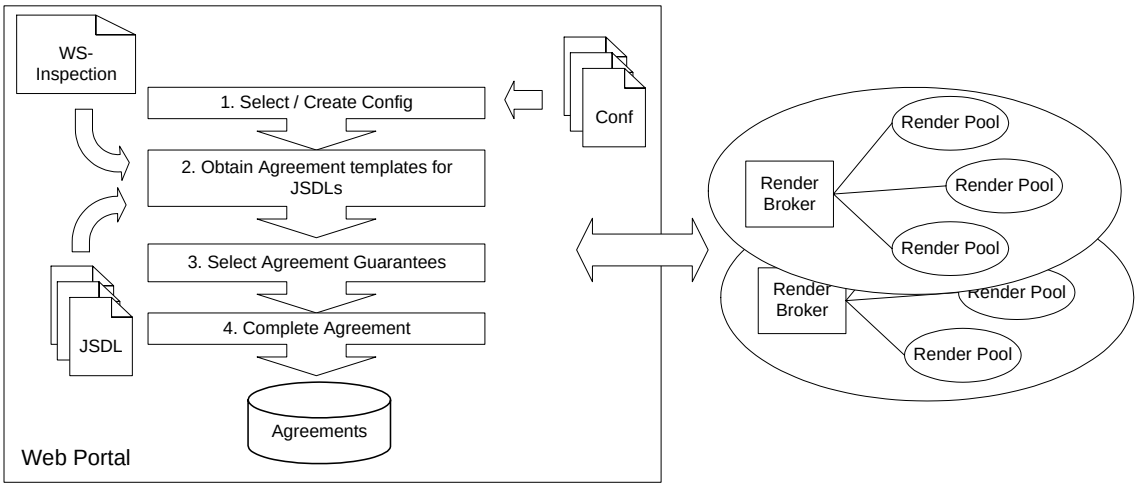

Figure 2. The render agreement process. A user initiates the process through a web portal by selecting or creating a new render configuration.

3. Architecture

Our innovative render agreement framework consists of three main components. (1) A render management system, which is responsible for brokering resources and managing agreements. (2) A web portal through which a user establishes rendering agreements. (3) A monitoring system for analysing the state of active agreements.

3.1 Render Management

The render managements system consists of a collection of WSRF services. At the core of these services is the RenderPool service group, representing a pool of homogeneous rendering resources. Each entry in a pool represents an individual hardware rendering resource with identical rendering capabilities; explicitly this means that each resource has the same rendering hardware and OpenGL implementation. Under these conditions an application launched on any set of these resources will perform identically. Any display attached to a RenderService in the pool will support the same display visuals such as colour depth and stereo support.

There may be a physical association between displays attached to services in a RenderPool, examples of this include the separate display units used in large display walls and CAVE environments. This association is made explicit by a RenderPool WS-ResourceProperty. A schema similar to that of Distributed Multihead X project [14] is used to describe the display mapping between RenderServices.

RenderPools can be ad-hoc, consisting of individual render resources individually associated to a pool, or cluster based representing a collection of render resources managed by an internal resource scheduler. Collecting RenderServices into homogenous pools is a performance optimisation inspired by Condor-G [15], which advertises collections of homogenous computational resources.

Each RenderPool has a unique set of agreement templates which it is prepared to enter an agreement on. These templates can be generic, offering guarantees on low-level properties such as polygon rate or texture mapping performance. Templates can also be offered for known job descriptions. These are job descriptions that have previously been executed on nodes within the RenderPool, and the performance of which has been previously measured. Consequently, a RenderPool maybe prepared to enter into higher-level performance guarantees, such as frame render time and even penalty clauses for this.

Obtaining templates and making an offer to a RenderPool is achieved through a broker, which aggregates a collection RenderPools.

3.2 Agreement Process

Render agreements are initiated by an end user through a web portal (Figure 2). The web portal has been developed as a collection of Java Servlets deployed inside a Tomcat container. Communication with the WSRF render services is performed using the Globus core java libraries [16].

The portal exposes a set of XML render configurations to the user. Each configuration defines a directed acyclic graph of render nodes,

```
<jsdl:resources xmlns:jsdl-render=
        "http://lboro.ac.uk/JSDL-
Render>
  <jsdl-render:Api>
      OpenGL_1_2
  </jsdl-render:Api>
  <jsdl-render:ApiExtension>
      GL_EXT_texture3D
  </jsdl-render:ApiExtension>
  <jsdl-render:TextureSize>
      7561216
  </jsdl-render:TextureSize>
</jsdl:resources>
```

Figure 3: Sample JSDL description of render resources for a volume rendering application.

representing a particular distributed rendering topology. Associated with each node is a JSDL identifier (currently a UID specified within the JobAnnotation element is used). This information is used to create a list of unique job descriptions for which agreements must be obtained.

Each job description describes the application to be executed including application arguments using the POSIXApplication schema. In addition to this, the job description provides information on the type of rendering resource that the application will require.

The current standard for JSDL includes a limited resource requirements schema, which is focused on general computation, describing the operating system, CPUs, file system, networking and memory required. We augment this schema with a render requirements extension, a sample of which is shown in Figure 3. This allows the JSDL to capture the specific rendering requirements of a visualization application. It is envisaged that a future resource requirements language will have sufficient scope for such descriptions.

Once JSDL descriptions have been retrieved for each node in the rendering configuration, agreement templates must then be obtained. Agreement templates are retrieved from known render brokers, referred to in a local WS-Inspection document [17]. WS-Inspection resource discovery was chosen over simple text storage or more complex UDDI because it provides a simple, structured, method for storing references to web services. A WS-Inspection document can link to other inspection documents and UDDI servers. The retrieved render broker endpoints are then queried to obtain agreement templates.

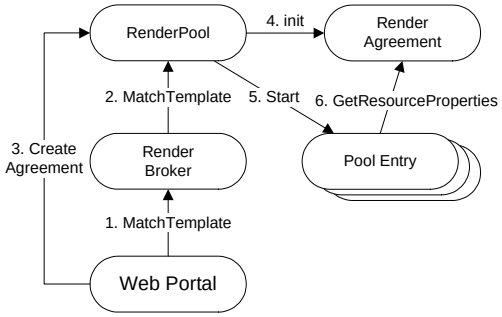

Figure 4: Web service operations in obtaining agreement acceptance.

3.3 Obtaining agreement templates

In the WS-Agreement protocol, an agreement initiator requests a set of template agreements from a service responder through the Template WS-ResourceProperty. This allows the initiator to determine the types of agreements that a responder maybe prepared to enter into.

Within our rendering framework some of these templates will be more relevant to a visualization job than others. For example if a template exists for the particular job description this should be chosen over all other templates.

This is both optimal for the responder and initiator. Therefore as well as exposing the Template WS-ResourceProperty, a RenderPool also exposes a MatchTemplate web service operation. The operation uses a hash map of job description identifiers and associated agreement templates. If no agreement is found the operation returns a basic agreement template.

The MatchTemplate operation is used by a RenderBroker service to retrieve possible templates on behalf of the web portal initiator. When a pool returns a basic agreement template, the broker must ensure that the resources specified by the JSDL document do not exceed those of the pool. This is achieved by inspecting the conformance WS-ResourceProperty of the RenderPool. Each resource term is matched with a term in the conformance document.

3.4 Agreement Creation

Agreement templates are translated into agreement offers by selecting appropriate guarantee terms for each of the measurable service properties in an agreement. The constraints section of each template defines the values that the guarantee terms of an agreement can have.

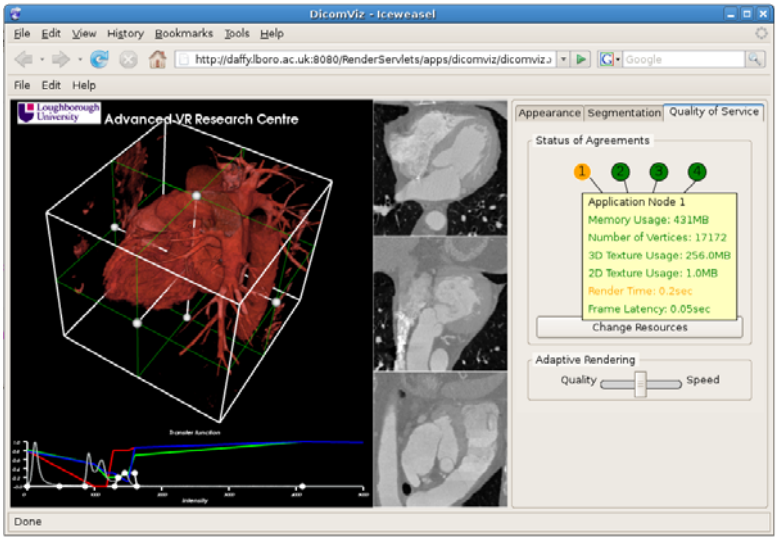

Figure 5. A screenshot of our DicomViz application utilising the agreement framework. The application client is operating inside the Firefox browser. The right hand box shows a tool tip displaying the current agreement status of node 1. The orange colour informs the user that a guarantee term has almost been violated.

Our web portal transforms the possible agreement templates for each JSDL into a HTML form. The terms of the agreement are exposed as user input elements in the form. A user is then able to quickly complete the forms and submit the agreements for acceptance.

Currently our system supports synchronous agreement acceptance (Figure 4). This means that when a successful agreement is made the agreement must be in an observable state.

As such the render job associated with the agreement must be started prior to the RenderPool service returning. If an agreement is not accepted, the user is requested to submit an alternate agreement from another agreement template.

3.5 Agreement Monitoring and Completion

The render agreement service implements both the Agreement and AgreementState port types, allowing other services to query both the agreement and the agreements current state. The state of an agreement is updated using publish-subscribe messages, as defined in WS-Notification [5]. The RenderPool passes a subscription request to each allocated entry as part of the Start operation. This subscribes the render agreement resource to performance topics generated by each RenderService resource. The method through which performance statistics is obtained depends on the deployment and implementation of the RenderPool. Modern ATI and NVIDIA GPUs maintain internal performance counters that can be queried through an exposed API [18].

An agreement reaches completion when a termination request is received. This can be generated by an associated render service (in the event of a fault, or lifetime expiration) or by the user (agreement initiator). The termination request is then relayed to all associated render services; these services then re-associate themselves with the render pool for allocation.

The associated RenderPool uses statistics gathered by the agreement to update agreement templates before removing the agreement. This allows new JSDL specific agreements to be automatically generated from the base template agreement.

4. Case Study: DicomViz

Modern medical imaging represents an ideal visualization domain with which to test an agreement framework for distributed rendering. Visualization has become a key component of medical practices. MRI and CT scanners have advanced rapidly in both resolution and speed of scanning. This has led to improvements in medical diagnosis, particularly cardiovascular conditions, where it is now possible to analyse cardiovascular function using time varying data sets.

Visualising these data sets in 3D involves a ray casting process known as direct volume rendering. Rendering methods have been developed to achieve this on commodity graphics hardware [11]. However, the large size of these data sets demand the performance of hardware specific solutions [19] or parallel

rendering [20] to achieve full resolution interactive rendering. When direct access to these systems is not available, radiologists currently revert to 2D planar images of the volumetric data sets or fixed viewpoint videos.

Remote access to rendering hardware with well defined performance characteristics would alleviate this problem.

4.1 Adaptive Distributed Volume Rendering

Our unique adaptive distributed rendering system allows OpenGL applications to be remotely accessed [8]. The adaptive system operates transparently on the OpenGL stream of an application, utilising feedback control and quality-of-service messages. This allows the system to adapt and stage the OpenGL stream on the most appropriate parts of the distributed rendering resources according to the user's requirements.

We utilise this system to deploy a custom DICOM volume rendering application. The application provides sort-last parallel direct volume rendering support. Each node renders a sub section of the overall data-set. The resulting partial images are composited together to create the final rendered scene using our adaptive rendering system.

4.2 Rendering configurations and resources.

We have deployed render pools consisting of NVIDIA Quadro graphics cards and GeForce2 cards. The latter only provides 2D texture support, resulting in lower performance volume rendering with more visible artefacts. The aim of these test pools was to examine the frameworks ability to automatically generate template agreements when new JSDL documents were encountered and to test agreement monitoring.

A number of test configurations have been deployed in our web portal. These utilise the sort last application in 1, 2, and 4 node configurations.

4.3 Client Interface

Cross-platform interactive GUI access to the volume rendering application is achieved by utilising a combination of Mozilla XUL [21] and a custom plug-in (Figure 5.). A similar process has been utilised for computational steering using asynchronous java script and xml (AJAX) [22]. XUL provides the same asynchronous communication as AJAX. However, XUL also defines a rich set of GUI widgets and allows complete stand alone applications to be described using Javascript, CSS and XML. These are interpreted by a XUL runtime package, such as the Firefox web browser.

Our XUL client utilises Mozilla's asynchronous SOAP support to monitor agreements and communicate GUI interactions with a web service plug-in attached to our volume rendering application. A block diagram of the complete communication process is shown if Figure 6.

Each application supported by the web portal has a custom XUL document associated with it. This allows the client interface to adopt the look and feel of the application's native GUI. Common components are provided through a set of XUL extensions and Javascript libraries. These provide support for render agreement monitoring widgets and WSRF communication.

Access to the visualization is provided through a custom Mozilla plug-in that acts as part of the adaptive render framework. The plug-in captures mouse and keyboard events occurring inside the plug-in window and broadcasts them through the render system.

Separately managing visualization events allows the client to offer a low latency interaction loop for the visualization. GUI interaction occurs locally providing instant interaction feed back to the user and allowing many GUI events (such as navigating menus and tooltips) to occur entirely locally.

The plug-in is the only aspect of the client that is platform specific (allowing it to achieve native hardware rendering support). Mozilla's plug-in management system enables the user to obtain the correct version of the plug-in for their platform upon initialisation of the XUL client.

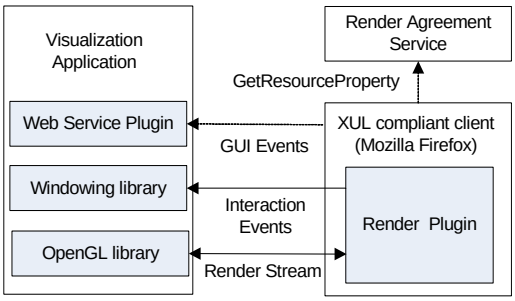

Figure 6. Communication between visualization application, XUL client, and a render agreement service

5. Conclusions & Future Work

We have outlined a framework for establishing service agreements for distributed rendering. The novel approach we have taken in auto generating new template agreements from base agreements allows the system to offer guarantees in user orientated terms. This removes the need to develop elaborate prediction models for the performance of visualization applications. Additionally, we have presented a novel cross platform client interface that allows the user to monitor the status of their agreements.

Future work will examine the use of asynchronous agreement creation, which will allow a human user to decide on an agreement offer on behalf of a service provider. We will also investigate improving our conformance matching broker through OpenGL conformance benchmarks. This will allow conformance to be matched between heterogeneous rendering hardware.

References

[1] J. Shalf and EW Bethel, “Cactus and Visapult: An Ultra-High Performance Grid-Distributed Visualization Architecture Using Connectionless Protocols,” *IEEE Computer Graphics and Applications*, vol. 23, pp 51 – 59, 2003

[2] I. J. Grimstead, N. J. Avis and D.W. Walker, “Automatic Distribution of Rendering Workloads in a Grid Enabled Collaborative Visualization Environment”, *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, 2004

[3] P. Heinzlreiter, and D. Kranzlmüller, "Visualization Services on the Grid: The Grid Visualization Kernel", *Parallel Processing Letters*, vol. 13, No. 2, pp 135-148, 2003

[4] R. S. Kalawsky, J. O'Brien and P. Coveney, "Improving scientists' interaction with complex computational-visualization environments based on a distributed grid infrastructure", *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, vol. 363, pp 1867-1884, 2005

[5] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, T. Storey, and S. Weerawaranna, “Modeling Stateful Resources with Web Services”, *Globus Alliance*, 2004.

[6] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, AS. McGough, D. Pulsipher, and A. Savva, "Job Submission Description Language (JSDL) Specification, Version 1.0", *Draft Recommendation, Global Grid Forum (GGF)*, 2005

[7] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, “Web Services Agreement Specification (WS-Agreement)”. Version 2.0 Draft, 2006

[8] J. O’Brien and R. S. Kalawsky, “A novel control mechanism for distributed stream rendering”, *Theory and Practice of Computer Graphics*, June 2007.

[9] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing", *Software Practice and Experience*, vol. 32, pp 135-164, 2002

[10] J. Neider, T. Davis, M. Woo, “OpenGL Programming Guide: The Official Guide to Learning OpenGL 1.4”, 2003

[11] J. Kniss, and G. Kindlmann, and C. Hansen, "Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets", *Visualization, 2001. VIS'01. Proceedings*, 2001

[12] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, “Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI”, IEEE Internet Computing, vol. 6, pp 86-93, 2002

[13] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing", *10th IEEE International Symposium on High Performance Distributed Computing*, pp 181-184, 2001

[14] ”Distributed Multihead X”, Available on-line at: *http://dmx.sourceforge.net/*, May 2007

[15] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", *Cluster Computing*, vol. 5, pp 237-246, 2002

[16] B. Sotomayor, and L. Childers, "Globus Toolkit 4: Programming Java Services", *Morgan Kaufmann*, 2006

[17] K. Ballinger, P. Brittenham, A.Malhotra, W.A. Nagy, and S. Pharies, "Web Services Inspection Language (WS-Inspection) 1.0", Available on-line at: *http://www.ibm.com/developerworks/library/specification/ws-wsilspec/*, May 2007

[18] J. Kiel and D. Cornish, "Optimize Your GPU with the Latest NVIDIA Performance Tools", *SIGGRAPH’ 06*, 2006

[19] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler, "The VolumePro real-time ray-casting system", *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp 251-260, 1999

[20] K.L. Ma, JS Painter, CD Hansen, and MF Krogh, "Parallel volume rendering using binary-swap compositing", *Computer Graphics and Applications*, vol. 14, pp 59-68, 1994

[21] D. Boswell, “Creating Applications with Mozilla”, *O'Reilly*, 2002

[22] A. Sen, J. Brooke, B. Harbulot, M. Mc Keown, S. Pickles, and A. Porter, "Combining AJAX and WSRF for Web-browser based Grid clients", *http://www.realitygrid.org/publications/wsrfajax.pdf*, November 2006