Life Beyond the Browser: The TeraDRE

Dave Braun
Rosen Center for Advanced Computing
Purdue University
dbraun@purdue.edu

X. Carol Song
Rosen Center for Advanced Computing
Purdue University
carolxsong@purdue.edu

Laura Arns
Envision Center for Data Perceptualization
Purdue University
larns@purdue.edu

ABSTRACT
This paper discusses the development of a multi grid-domain application for users of the Purdue TeraDRE resource. The TeraDRE (Distributed Rendering Environment on the TeraGrid) allows a user to greatly reduce the render time of their 3D animations using a cluster of distributed computers, while providing temporary storage for large animations. It has been used successfully by Purdue researchers and students in a number of projects where animations were generated, and is now a service available on the TeraGrid. As we broaden the access to this distributed rendering service, we foresee several challenges. One of the key challenges is to support users coming from different grid domains that require different authentication methods. We also face the increasing demand for the support of multiple rendering engines. To meet these and other demands, we have developed the next generation grid-aware TeraDRE as a gateway for users from TeraGrid, Open Science Grid and other organizations. Using the Java Web Start technology, the new generation TeraDRE provides a user friendly interface that supports the end-to-end workflow needed to render 3D graphics and animations on the grid. The grid-aware TeraDRE is rich with features such as automatic video file production, previews and notification on web-capable mobile devices. It also supports multiple rendering engines, including the open source renderers POV-Ray and Blender.

Categories and Subject Descriptors
I.3.2. [**Computer Graphics**]: Graphics systems – *Distributed/network graphics, remote systems.*

General Terms
Design

Keywords
Distributed rendering, 3D graphics, animations, TeraGrid, Condor, grid domains, Java web start.

1. INTRODUCTION
Digital animation uses a series of still images in sequential frames to make a movie. Digital animators render highly detailed, computationally complex graphics models for every frame. Purdue researchers have created a distributed rendering environment, TeraDRE, for scientists and students to render 3D animations using a cluster of distributed computers [6,10]. By spreading the computations across hundreds of machines simultaneously, the overall rendering time can be reduced significantly. The time for rendering a two-minute movie can easily be 120 hours on a single computer. This can be reduced to 36 minutes using grid-aware TeraDRE on 200 computers.

Initially created to serve the Purdue campus users, TeraDRE was modified and enabled for TeraGrid users. Using the Purdue Condor pool [9,15] as the distributed computational backend, TeraDRE was used successfully and impressively for the Bandwidth Challenge competition at Supercomputing 2006 [1]. Figure 1 shows a single 4K image frame of the cell structure rendered for this competition. This image is part of a two-minute, 10-gigabyte (4096x 3072 pixels, 30 frames per second video) scientific animation of the cell structure of a bacterial ribosome.

We have designed and developed the grid-aware TeraDRE to support users of university campuses, the TeraGrid and the Open Science Grid (OSG), with the goal of providing a remote, high performance, graphics resources to a broad range of users while lowering the barrier of entry.

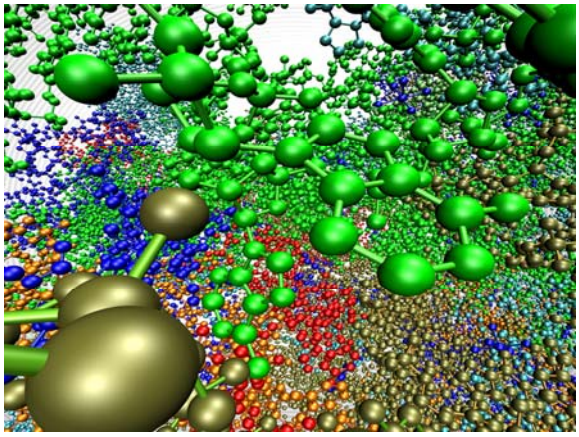

Figure 1. A rendered 4K frame of a cell structure rendered for Supercomputing'06

2. BACKGROUND
In recent years, photo-realistic animations have become the standard for a good animation. The photo realism and increased geometry sizes however do come at a price in terms of the computations needs to generate the animations. An animator faces many challenges when constructing an animation because of the iterative nature of the animation process. This process includes making tweaks to the model's geometry, textures, lighting, and motions to get a seamless animation sequence. Though the

D. Braun, X.C. Song, L. Arns, "Life beyond the browser: the TeraDRE", International Workshop on Grid Computing Environments 2007, November 11-12, 2007, Reno, NV, USA. An electronic version of this document will be available at http://casci.rit.edu/proceedings/gce2007.

animator usually renders the models with reduced resolution and features as a preview, there is always a need to go through the detailed rendering to ensure that everything is correct. In general, detailed rendering uses more computational cycles than the animator's local machine can provide and, thus, the general solution has been to farm this step of the process to a rendering cluster.

Having a dedicated rendering cluster is a luxury that most animators cannot afford. Thus the other solution is to purchase rendering cycles from third party vendors, which can become costly if too many iterations are required. The rendering workflow is a batch serial job where each frame of the animation can be processed independently. Batch serial jobs are a perfect match for opportunistic scheduling systems, where spare computational cycles are used to perform useful work rather than being wasted. The approach we describe in this paper is based upon the use of the Condor [9,15] technology to provide a low cost way of executing the rendering process within a multi-grid environment. The goal of this project is to provide student animators and researchers with easy access to the distributed rendering resource via the grid.

2.1 Related Work

Many commercial rendering farms are available for use. However, the fees for use can be quite high, as the following examples illustrate:

- RenderCore is a 500-machine render farm which supports a variety of 3D graphics applications/plug-ins, including Maya©, 3D Studio Max©, XSI©, VRay©, Brazil©, LightWave©, Cinema 4D© etc. The rendering price ranges from $0.45 to $0.90 per GHZ-hour depends on the applications/plug-ins. For example, for a 30 node network (3.0GHZ dual core), a Maya Mental Ray project costs $324 for a just 2 hours of rendering. [13]

- Respower is another internet render farm containing more than 750 nodes. It supports 3DSMAX©, Maya©, Brazil©, LightWave©, Blender©, VRAY©, Vue©, YAFRAY©, mental ray© etc. The price is charged on a daily basis. For example, to render a Maya Mental Ray project, it costs $600 a day, which allows one individual to work and the frames are allotted to the maximum limit of 120 minutes to complete. With more rendering time required by the project, the price ranges from $2000/4days to $6000/30days, etc. [14]

Such fees are viable for commercial entities working on large productions, but are not feasible for most students and researchers. An additional problem is that commercial render farms support commercial 3D graphics software packages, but they do not support scientific visualization tools or Open Source rendering tools. For these reasons, most researchers and educators build their own rendering farm to support their work. Unfortunately, the farms are usually quite small (perhaps half a dozen machines) and there is a great deal of overhead in setting up and maintaining these systems, as there is no software that allows non-experts to easily create render farms. This results in many small farms with wasted cycles and wasted administration time, and a continuing lack of a large number of cycles when actually needed for rendering.

TeraDRE addresses these issues by providing a service that is generally free to students, educators, and researchers. It also provides a combination of both commercial and Open Source renderers to address the unique needs of this set of users. All renderers can be accessed using the same graphical user interface, simplifying the process of submitting rendering jobs. Finally, because the TeraDRE uses Purdue's Condor pool, it has access to nearly 4,000 nodes for rendering – far more than the number available via most commercial render farms. This helps to reduce congestion and wait time.

2.2 A brief history of TeraDRE implementations

The initial TeraDRE was implemented as a set of command line utilities. An enhanced version was implemented as MEL (Maya Embedded Language) scripts that provide a graphical user interface within the Maya client. A user must have a Maya client running on the local computer, and must also configure a number of files and directories on the local computer before submitting TeraDRE jobs to a cluster of distributed computers. This implementation has the advantage for Maya users (such as student labs in many schools that have Maya software installed), providing a user interface integrated with the modeling tool they use in creating the render job.

To simplify access for TeraGrid users, a second generation TeraDRE interface was created. We developed a GridSphere-based [12] TeraDRE portlet to provide an easy-to-use web browser interface. The TeraDRE portlet removes the manual process of configuring files on the local computer, allowing the user to authenticate through the TeraGrid MyProxy server [3] within the portlet. The user uploads the job file through the browser, and the job is automatically submitted to the TeraGrid Condor resource using the Globus Resource Allocation Manager (GRAM).

As we continue to broaden access to TeraDRE, we encountered a number of challenges. The top three are: (1) supporting users from different grid domains, including the national grids such as TeraGrid and OSG, regional grids such as Northwest Indiana Computational Grid (NWICG), and various campuses (aka campus grids). We need a versatile mechanism to support the authentication methods of these grids; (2) the ability to support a rich set of user-driven workflows (e.g., previews, review and resubmit); and (3) the ability to manage multiple renderers (Maya clients require commercial licenses which many users prefer to avoid). The support of open source renderers is also the key for many users to using the TeraDRE resource.

The inner workings of the distributed rendering environment at Purdue (aka the backend of TeraDRE) were described in a previous paper [6]. The rest of this paper focuses our discussion on the design and implementation of the grid-aware TeraDRE front end that aims at making TeraDRE an effective, easy-to-use and scalable application on the grids.

3. GRID-AWARE TERADRE

The grid-aware TeraDRE is currently accessible through a web site and is being integrated with the TeraGrid Visualization Gateway [4]. From the user's perspective, they are able to access the same functionality, whether logged in at the TeraGrid Visualization Gateway or connecting to the TeraDRE web portal in a browser.

Figure 2 illustrates the hardware and network architecture that supports the TeraDRE resource. Based on access rights, a user can gain a proxy from different sources. Currently, users can gain authentication to either Purdue campus grid or TeraGrid using either their local certificate or a grid-domain MyProxy server. Once the user gains a proxy, her job is staged on a resource via `gsiftp` that will place the render project bundle in a user-specific sandbox area. The next step in the process is a fork job to a WSGRAM gatekeeper which executes a server side script to start the processing of the job. Each frame of the render job is stored in the user's sandbox and may be retrieved either from a web server or through `gsiftp`. As the project is executed, HTML files are generated with additional information on the job's status and can be retrieved from the web server.

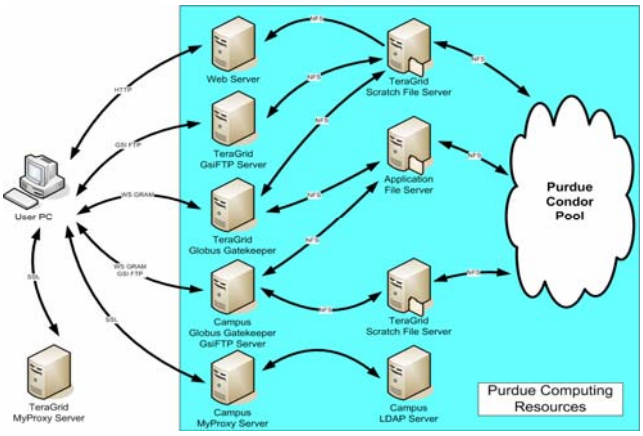

Figure 2. Hardware/Network diagram for the grid-aware TeraDRE

The TeraDRE software (Figure 3) is constructed as an abstract rendering Java project that holds a list of job files and an argument list. Each renderer is an implementation of the rendering project, which further specifies the job arguments and additional rendering parameters. A rendering panel provides a generic viewer for the abstract rendering project. At the point where the project is submitted, the rendering project is realized and a JAR (Java Archive) file is constructed that contains all the necessary job dependencies. A Condor DAG (Directed Acyclic Graph) is then generated, and a property file is generated as well. When the job is submitted, the JAR file is expanded and the property file is converted into shell environment variables.

The TeraDRE also has been architected to support multiple job submission and authentication models. This flexibility allows it to be adapted to different grid domains through the addition of new grid domain and grid site objects. At present, both a TeraGrid and Purdue campus grid domain are supported with the anticipation of many more in the future including local submissions.

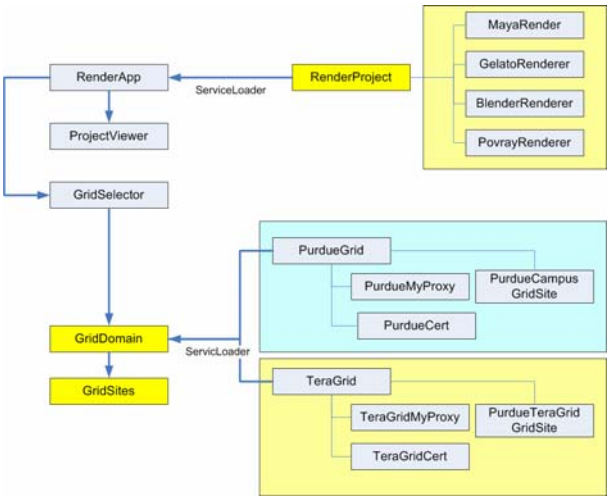

Figure 3. Software class diagram for the grid-aware TeraDRE

To use TeraDRE, a user goes through the following steps to submit rendering jobs and receive notification of the results:

1. User creates a new project.
2. User adds files from their local PC to the project. The files are stored as references in the virtual sandbox.
3. User fills in the project properties for the particular renderer, including selecting files from the sandbox.
4. User logs into a grid domain of their choice.
5. User submits a file to a grid resource within that domain.

At this point, the TeraDRE performs the following:

1. The user project is packaged as a JAR file and all files are included that were in the sandbox. A Condor DAG is generated as part of this.
2. The JAR file is staged at the grid resource.
3. A fork job is submitted to a collection of back end scripts that will extract the JAR file, fill in several resource specific environmental variables as defined in the resource, and submit a condor DAGMan job.
4. When the user requests status information, a fork job is submitted to generate a status file containing information about all active rendering projects.
5. When the user wishes to preview the current project, an output file containing a list of the names of completed files is retrieved from the grid resource. At the time of preview, the previewer thread pool uses this file to load the preview files to the user's local computer.

Figures 4(a) through 4(d) are screen images from the grid-aware TeraDRE application.

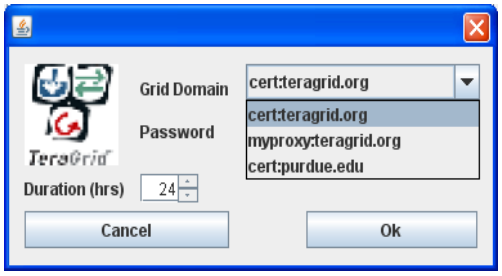

Figure 4(a). Grid domain selection screen

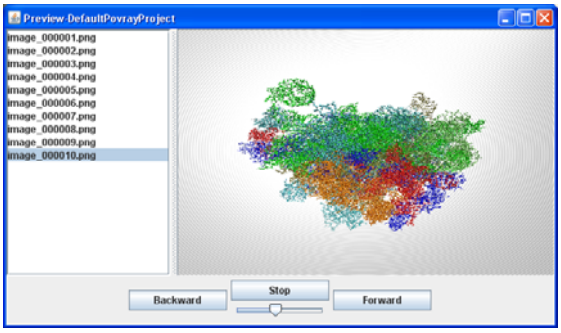

Figure 4(b). Preview screen (left: frames, right: animation)

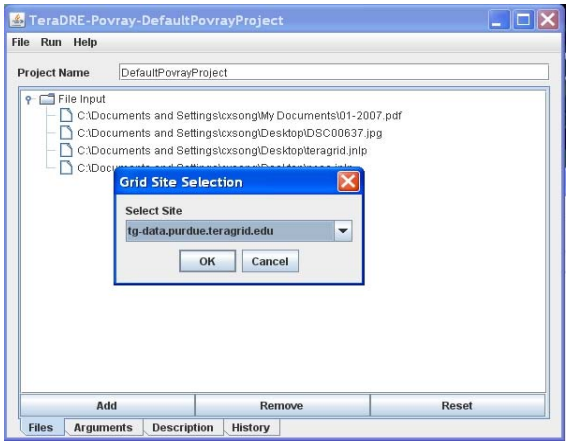

Figure 4(c). Grid site selection screen

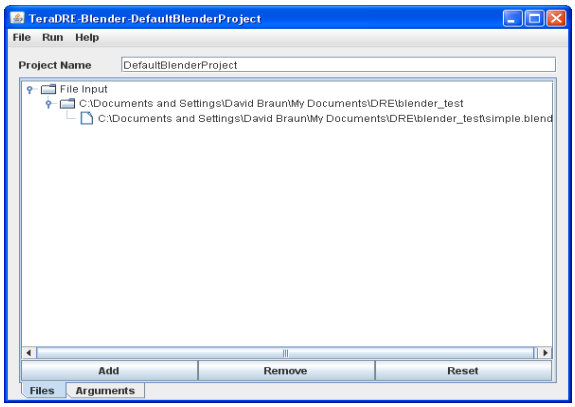

Figure 4(d). Project files that can be drag-n-dropped into GUI

4. Key implementation issues

To keep the implementation simple, the TeraDRE server components are a collection of rendering applications and scripts that provide integration to the computational resources. A key feature is making the TeraDRE service available to users of both the popular national grids and local campus grids. The following subsections describe the challenges and our solutions in the implementation of the grid-aware TeraDRE.

4.1 Multiple Grid Domain Authentication

TeraDRE uses the CogKit [5] to implement GSI Authentication for grid domains. Upon logging in, a user may select a grid domain where s/he can receive a proxy certificate. Users gain access to TeraDRE using X.509 certificate-based authentication, a popular mechanism used by various grids today. Within TeraDRE, we use the notion of grid domains. For example, TeraGrid is a grid domain, the Purdue campus grid is another. Grid domains may include particular methods of authentication, Java submission, and may contain one or more grid gatekeepers that allow for job submission. A user can be authenticated via an X.509 proxy generated on the user's local computer or through a MyProxy server for a particular grid domain. In both cases, a proxy is generated and held in the application memory instead of being saved as a local file.

In our implementation to support campus grids, a user requests a proxy from a campus MyProxy server. Based upon the principle and credential which the user presents, the MyProxy server is configured to gain authentication from the campus LDAP server. An anonymous proxy is returned. The returned proxy may also be issued to a group DN (Distinguished Name), e.g., students can be mapped to a single user thus simplifying account management.

4.2 Rich Client Deployment

Rich client is a term commonly used to describe applications that are not browser-based applications, and that can directly access local resources. Because rich clients have access to local resources, they typically require the user to go through an install process and perform further installs with each revision.

Java Web Start is a technology developed by Sun to deploy Java rich clients to a user's computer in a secure manner. Applications that are deployed via Java Web Start are nearly identical to the applications that users would install on their desktop except that they are downloaded to a secure sandbox area and run in an external JVM (Java Virtual Machine) that is not part of the browser.

There are numerous examples of Java Web Start applications that are used by scientists to do their daily work. The Java NEXRAD Viewer [11] and Integrated Data Viewer [8] are two examples of visualization applications deployed via Web Start. These applications allow users to visualize data from both local and remote sources.

Application deployment is described in a JNLP (Java Network Launch Protocol) file. The JNLP file is requested by the end user's Java Web Start environment and the corresponding JAR files are downloaded to a cache area on the local computer. Depending on the JNLP configuration, updates to the cached JAR files are downloaded each time a user executes the application.

The newest version of Web Start technology includes a servlet that will detect changes in the deployed JAR files.

Although this technology is related to Java Applet technology popular in web browser applications, Web Start does not require a browser to function. This technology works well for a Java application that needs more local abilities and does not require browser interaction. In the latest releases of the Web Start technology, Sun expanded the application launch protocol, and included the concepts of resources and a smart JNLP download servlet. These additions make deploying Web Start applications easier. One of the major complaints about Web Start was the time it takes to download the application to the cache. Indeed if the application is large, this would take some time. The new release allows an application to be broken up into a set of resources which are individually cached. There is little impact on an application to do this. An application that implements a pluggable architecture can take advantage of this technology at deployment. Applying this technique has improved the download speed of the TeraDRE submission client.

4.3 Performance tuning

Condor DAGMan can be used to throttle job submissions when one submits an extremely large number of batches jobs. The performance of Condor Job Manager (`schedd`) degrades when the number of jobs exceeds the resource availability. Throttling is used to prevent large submissions from disabling the shared schedds. Throttling does have an impact, however, on the overall performance because it is static.

Figures 5 and 6 depict the performance of two rendering applications. The red color represents the wait time (the time from which the job was submitted to the time of execution). The green color represents execution time. In the Condor world, the green section also includes job evictions, holds and suspends. The blue color represents the post-processing time as defined by the post processing script in the DAG. The post-processing is executed on the submit machine and includes computations such as scaling an image to create a thumbnail. The linear trend in the start times of each job is due to the throttling imposed by the DAG. For these applications, a throttling of 100 running jobs was imposed based on our analysis of average resource availability and empirical results.

The performance of the rendering application depends on several variables, including the availability of the Condor resource, whether the application has the ability to restart, and the rendering time per frame. The worst case occurs when the Condor resources are busy and the application does not have a self check-pointing capability - a job would constantly be evicted until a resource can be acquired. Blender's performance curve was taken during a time when the Condor pool was very busy. We believe the block structure of the curve is due to priority policy and machine lease time. The variation in the post processing times is due to the load on the submit machine as the jobs complete. In POV-Ray's performance curve, the Condor pool was not busy and POV-Ray has self check-pointing capability.

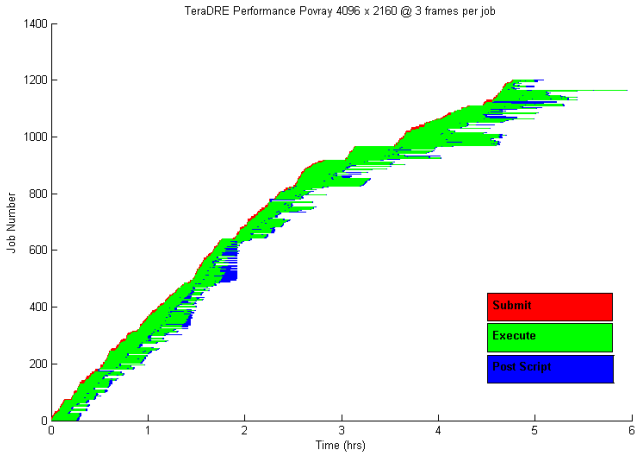

Figure 5. TeraDRE Performance in rendering a POV-Ray animation

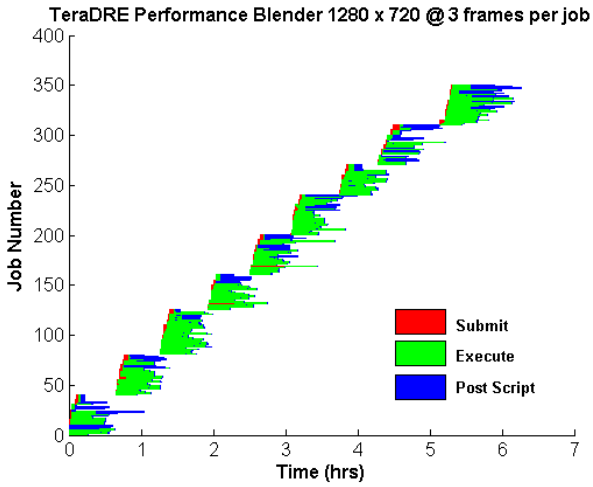

Figure 6. TeraDRE Performance in rendering a Blender animation

4.4 Flexible Packaging for Dynamic Environments

Reducing the size of the JAR files that are deployed at runtime increases the download speed of the application. Since the application only uses a small portion of the supporting software stack, *Autojar* was employed to repackage the application JAR file to include only the software (classes, directories, and libraries) that is or could be used by the application [2].

We used another technique to increase the first-time application launch speed - partitioning the JAR files into packages in the JNLP file. This allows only the pieces that change to be downloaded instead of a very large file. The downside of increasing the number of JAR files is that each JAR file must be signed. Though the process of signing JAR files is quite easy, it adds another layer of management because certificates used to sign the JARs may expire.

JAR assemblies allow for a greater flexibility in application development. Each JAR may contain one or more related plug-ins to a computer application. In the case of the TeraDRE, JAR

assemblies are used to encapsulate various renderers and grid domains. TeraDRE uses some of the new features in Java 1.6. ServiceLoader is a very interesting technology that reduces the complexity of implementing a plugable architecture. Within the JAR file, classes are listed as implementing a particular interface. The service loader will load and unload this class easily without the need for the programmer to implement a specific class loader. Using this technology in combination with the JNLP resource extension allows for rapid deployment of new features and grid domains.

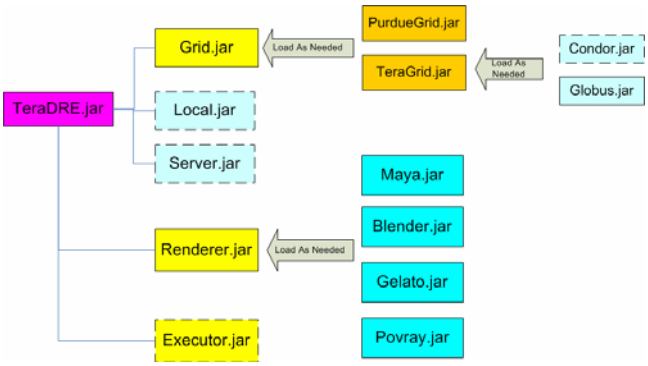

Figure 7. TeraDRE Packaging

4.5 User-Driven Workflow Support

Animation developers see an application as a collection of features and functions that they assemble to execute a task, rather than a fixed workflow that they follow. TeraDRE allows users to access the features and functions in a natural way by providing interactive capability. Users can choose to execute functions such as previewing an animation, at any point instead of waiting for the full animation to complete. TeraDRE accomplishes this by streaming the images back to the user's local computer through a grid connection. The images are pre-fetched as the movie previewer displays the sequence of images. The current solution also uses Java's volatile image component which supports hardware accelerated graphics in order to increase the previewer speed.

4.6 Handling Large Image Frames

In the past few years, a new 4K movie standard has emerged. As defined by this standard, the dimension of each 4K image frame is 4096x2160. An animation may contain thousands of frames, resulting in a very large dataset for the render job. The challenge is to keep the large datasets on the remote resource but allow the user to gain access and eventually move them. The previewer in TeraDRE uses a much smaller set of images to stream back to the user as previews. The user may use GSIFTP tools to move the data to either the local computer or other grid storage.

5. DISCUSSIONS AND FUTURE WORK

We presented the design and implementation of the TeraDRE distributed rendering service for a multi-grid environment. The development of this grid-aware TeraDRE application answers the need to access this rendering service by users from various grids and institutions. Our contribution lies in building a light-weight, grid-aware interface that enables users to access the rendering resource seamlessly without having to learn grid tools or the intricate details of submitting a rendering job to distributed resources. The lessons we have learned include:

Light-weight Globus Client - We created a light-weight Globus client in an attempt to reduce the size of the Globus install footprint. Our goal was to use a minimum set of the Globus API and let the Autojar tool figure out which other Globus packages were needed. This proved to be a daunting task because Autojar is unable to follow dynamic class loads. Instead of using the entire CogKit abstraction, the code primarily uses jGlobus, Auth, and WSRF packages. In the future the extension resource from JNLP will be employed because of the complexity of the re-jar process with each revision of Globus.

Signing JARs - Signing JAR files is a very straightforward process. The Java development kit includes tools that will sign a JAR file based upon a certificate. However, the policy to address the question of who signs the JAR file is a different issue. The current solution to this problem is to sign the JARs using private certificates issued by NCSA for the TeraGrid. In the ideal case an organizational certificate would be used to sign deployed JARs.

Customized Applications for Multiple Grid Domains - Another future use of JAR assembly is the ability to dynamically create a Web Start JNLP based upon a user's or site's configuration. For example, if a rendering application has a site license that only allows use by the site personnel, the TeraDRE would deny access from users outside the IP address space by excluding certain JAR files. This would lead to an *ala carte* type of application assembly. Additionally, a JAR assembly allows only the JAR files that are need to be downloaded when the users need them. This would also employ the ServiceLoader, but now the reference to the service is a URL. As a result, this allows the application to dynamically load a JAR when the service is needed or as a parameter-based configuration, e.g., where the user can choose plug-ins that they are interested in. In future releases of TeraDRE, we plan to incorporate the ability to deploy applications that are either tailored to the individual or to a policy.

For future development, we are adding a middleware level to TeraDRE. The intent is to migrate some functionality from the client to the server, producing an even lighter weight client with a smaller footprint for faster initial download. This separation will be implemented as a collection of web services. These web services will support a browser-based implementation, enabling third parties to integrate TeraDRE into their own community portals to utilize local computing resources as desired. In the browser-based implementation, users will still be able to invoke Java Web Start applications to perform tasks that are better suited for local client execution, e.g., file transfer, preview, and local tool access.

Another important function the middleware could provide is user data management. In addition to this, the user needs to be presented with a consistent environment that includes both their local disk and the remote disk. The idea of a distributed sandbox would allow users to keep a local view and a remote view concurrently where files could be marked according to where the file actually resides and other methods could be employed to move or access them. By providing users a consistent view of the data space, application usability will increase.

We plan to develop a Condor master-worker [7] implementation for executing the TeraDRE workflows. In the current system, DAGMan is used to throttle applications in a static way with a hard limit on the number of running jobs. We are proposing a Condor master-worker type of model where the master will maintain a list of work and feed jobs to a collection of pre-scheduled workers. At present the number of frames that a worker renders is fixed. If the worker is evicted, the completed portion of the job is lost. By using a pre-scheduled collection of workers, we could create a much smarter wrapper application to dynamically change the number of frames that are attempted to be rendered.

6. ACKNOWLEDGMENTS

The authors wish to thank Jack Moreland, Preston Smith, Meiqi Ren and Jungha Woo for their work and support in the TeraDRE project.

7. REFERENCES

[1] Arns, L., R. Pedela, M. Shuey, P. Smith, J. Tillotson, T. Hacker, D. Braun. “Streaming uncompressed 4K scientific meda”, Bandwidth Challenge/Supercomputing, Nov. 2006. http://www.envision.purdue.edu/4kstream.

[2] Autojar project, http://autojar.sourceforge.net.

[3] Basney, J., M. Humphrey, and V. Welch. The MyProxy Online Credential Repository. Software: Practice and Experience, Volume 35, Issue 9, July 2005, pages 801-816

[4] Binns, J., J. DiCario, J.A. insley, T. Leggett, C. Lueninghoener, J.P. Navarro, and M.E. Papka. “Enabling community access to TeraGrid visualization resources”, Concurrency and Computation: Practice & Experience, Vol.19, Issue 6 (April 2007), pages 783-794.

[5] G. von Laszewski, I. Foster, J. Gawor. “CoG kits: a bridge between commodity distributed computing and high-performance grids”, *Proceedings of the ACM 2000 conference on Java Grande*, 2000.

[6] Gooding, S.L., Arns, L., Smith, P. and Tillotson, J. “Implementation of a distributed rendering environment for the TeraGrid”, *2006 IEEE Challenges of Large Applications in Distributed Environments* (CLADE), Paris, France, June 19, 2006.

[7] Heymann, E., M. A. Senar, E. Luque, and M. Livny, "Adaptive Scheduling for Master-Worker Applications on the Computational Grid". in Proceedings of the First IEEE/ACM International Workshop on Grid Computing (GRID 2000), Bangalore, India, December 17, 2000.

[8] Integrated Data Viewer (IDV), http://www.unidata.ucar.edu/software/idv.

[9] Litzkow, M.J. Livny, M. Mutka, M.W. “Condor-a hunter of idle workstations”, *8th Int’l Conf. on Distributed Computing Systems*, 13-17 June 1988.

[10] Madhavan, K.P.C., Arns, L., & Bertoline, G.R. A distributed rendering environment for teaching animation and scientific visualization. *IEEE Computer Graphics & Applications* (*Special Issue on Computer Graphics in Education*), 33 – 38, 2005.

[11] NCDC Java NEXRAD Tools, http://www.ncdc.noaa.gov/oa/radar/jnx.

[12] Novotny, J., M. Russell, O. Wehrens, "GridSphere: An Advanced Portal Framework, " EUROMICRO 2004, 412-419.

[13] RenderCore, http://www.rendercore.com/rendercoreweb/index.do.

[14] Respower, https://www.respower.com.

[15] Tannenbaum, T., D. Wright, K. Miller, and M. Livny, "Condor - A Distributed Job Scheduler", in Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*, The MIT Press, 2002. ISBN: 0-262-69274-0.