The Clarens Web Service Framework for Distributed Scientific Analysis in Grid Projects

Frank van Lingen[1], Conrad Steenberg[1], Michael Thomas[1], Ashiq Anjum[2], Tahir Azim[2], Faisal Khan[2], Harvey Newman[1], Arshad Ali[2] , Julian Bunn[1], Iosif Legrand[1]

[1]*California Institute of Technology, United States*
Email:{fvlingen, julian.bunn@caltech.edu} {iosif.legrand@cern.ch} {newman, conrad, thomas@hep.caltech.edu}

[2]*National University of Science and Technology, Pakistan*
Email:{arshad.ali, ashiq.anjum, tahir, faisal.khan@niit.edu.pk}

Abstract

Large scientific collaborations are moving towards service oriented architectures for implementation and deployment of globally distributed systems. Clarens is a high performance, easy to deploy Web Service framework that supports the construction of such globally distributed systems. This paper discusses some of the core functionality of Clarens that the authors believe is important for building distributed systems based on Web Services that support scientific analysis.

1. Introduction

Scientific collaborations are becoming more and more geographically dispersed. Researchers from all over the world collaborate on new scientific discoveries and breakthroughs in many "big science" experiments such as the Virtual Observatory [40], the Large Hadron Collider (LHC) program [13], LIGO [20] and Nuclear fusion [19]. Not only do these experiments generate tera bytes to peta bytes of data. In many cases resources for analyzing and storing these large amounts of data are distributed on a national or international scale.

Some of the largest scientific collaborations today, such as CMS [21] and ATLAS [22] are building experiments for CERN's LHC program, each encompass 2000 physicists from 150 institutions in more than 30 countries. Each of these collaborations include 300-400 physicists in the US, from more than 30 universities, as well as the major US HEP laboratories.

Realizing the scientific wealth of these science experiments, presents new problems in data access, processing, distribution, and collaboration across national and international networks, on a scale unprecedented in the history of science. [23] discusses several of the information technology challenges facing these “big science” experiments.

All of these challenges need to be met, so as to provide an integrated, managed, distributed system infrastructure that can serve "virtual organizations" on a global scale. One technology that holds the promise to form the basis of such an integrated, managed, distributed system are Web Services.

The Clarens project was started in 2001 [34] to provide a scalable Web Service framework for the development of distributed applications. Initially Clarens was developed as part of the CMS experiment [21], however as Web and Grid Services became two of the de facto standards for development of distributed applications, Clarens became part of several projects: Ultralight [12], HotGrid [14], Monte Carlo Processing Service using RunJob [26], the physics shell project (PHYSH) [7], Lambda Station [24] project, IGUANA [6] and the Proof [8] Enabled Analysis Center (PEAC). Development and deployment of Clarens is also part of several large Grid collaborations such as PPDG [41], IvdGL [43], Griphyn [42], OSG [25] and Grid3 [44]. Clarens was also used in the winning SuperComputing 2003 bandwidth challenge (23 Gb/s peak), in which Clarens servers generated a peak of 3.2 Gb/s disk-to-disk streams consisting of CMS detector events.

This paper gives an overview of Clarens. More information about Clarens can be found in [1], [14], [15], [16], [17], [18], and [34]. The first sections of

Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPPW’05)
1530-2016/05 $20.00 © 2005 IEEE

this paper discuss the architecture and core functionality of the Clarens framework. This functionality is, according to the authors essential in providing a framework for developing distributed service based applications for scientific collaboration and distributed data analysis. Section 4 describes some of the portal functionality while section 5 describes several performance measurements of the Clarens server. Section 6 and section 7 describe related and future work. Section 8 finishes with conclusions.

Unless mentioned otherwise Clarens refers to both the Python and Java implementation, JClarens refers to the Java implementation and PClarens refers to the python implementation.

Within this paper we use the following definitions for Web Services and Web Service Framework: A Web Service is a component performing a task, most likely over a network. A Web Service can be identified by a URI and its public interfaces and bindings are described using WSDL. At the basis of a Web Service call (invocation) is a protocol (frequently, but not exclusively this is XML-RPC [3] or SOAP [5]). A Web Service Framework is an application that provides support for developing and deploying Web Services.

2. Architecture

Clarens aims to provide the basis for a consistent, high-performance, fault tolerant system of distributed Web Services deployment and development. By leveraging existing, widely implemented standards and software components, including HTTP, SSL/TLS (RFC 2246) encryption and X509 (RFC 3280)[1] certificate-based authentication, and SOAP/XML RPC data serialization, Clarens also aims to be easily accessible to a wide variety of client implementations with the minimum of software dependencies. This approach lowers the barriers of entry to participate in the service network, re-use of existing developer skills, and a wide choice of development tools and languages.

In order to improve scalability, the PClarens server is implemented as an extension to the Apache Web Server [2] using the `mod_python` extension in the Python byte-code compiled language. PClarens itself is both architecture and platform independent by virtue of using Python as an implementation language. Figure 1 shows the PClarens architecture. The Apache server receives an HTTP POST or GET request from the client, and invokes PClarens based on the form of the URL specified by the client (other URLs are handled transparently by the Apache server according to its configuration). Secure Sockets Layer (SSL) encrypted connections are handled transparently by the Apache server, with no special coding needed in PClarens itself to decrypt (encrypt) requests (responses).

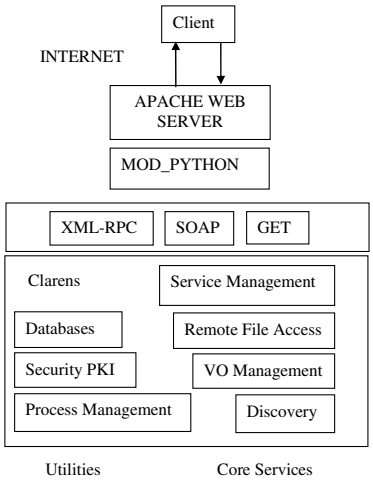

Figure 1. PClarens Architecture

After the request has been processed, a response is sent back to the client, which is usually encoded as an RPC response, but may also be in the form of binary data. GET requests return a file or an XML-encoded error message to the client, while XML-RPC or SOAP encoded POST requests return a similarly encoded response error message

In response to a preference for developing not only Python based Web Services but also Java based Web Services, a second Java based Web Service framework has been implemented (JClarens). The Java language and runtime environment have several desirable characteristics, including implementations on several platforms, a large developer community, and mature Web Service development tools.

The JClarens implementation is based on so-called servlets implemented inside a commodity container, in this case the open source Apache Tomcat server [4]. For JClarens the Tomcat server replaces the Apache web server and `mod_python` module in the architecture as depicted in Figure 1.

As mentioned in the beginning of this section, Clarens utilizes HTTP requests. Since the HTTP protocol does not require[2] persistent connections, it is important that session information is stored

[1] For Internet Engineering Task Force Request For Comment (RFC) documents, see `http://www.ietf.org/`

[2] In the HTTP 1.1 standard, persistend connections are the default for performance reasons, but the protocol is still inherently stateless, as opposed to e.g. the FTP protocol.

persistently on the server side. This has the positive side-effect of allowing clients to survive server failures or restarts transparently without having to re-authenticate themselves to the server in those cases.

2. Core Services and Utilities

The architecture discussed in the previous section can give the impression that Clarens (both the Python and Java implementation) is very similar to for example a Tomcat server and an Apache AXIS module [37] (the Java implementation actually uses Tomcat and AXIS). Both systems can be called Web Servers that host Web Services. Some of the differences are that the Clarens Web Service framework address issues such as:

- Certificate based authentication when establishing a connection.
- Access control on Web Services.
- Remote file access (and access control on files).
- Discovery of services.
- Shell service. Shell like access to remote machines (managed by access control lists).
- Proxy certificate functionality
- Virtual Organization management.
- Multiple protocols (XML-RPC, SOAP, Java RMI (only for JClarens), JSON-RPC [38]).

This section gives an overview of several of the services and utilities within the Clarens framework

2.1 Virtual Organization Management

Virtual organization management allows (geographically dispersed) users in large collaborations to be grouped together. Using this group structure it is easier for administrators of Grid resources to create fine grained access control lists for different groups and sub groups of scientists.

Each Clarens server instance manages a tree-like Virtual Organization (VO) structure, as shown in Figure 2, rooted in a list of administrators. This group, named admins, is populated statically from values provided in the server configuration file on each server restart. The list of group members is cached in a database, as is all VO information. The admins group is authorized to create and delete groups at all levels.

Each group consists of two lists of distinguished names (DNs), for the group members and administrators respectively. Group administrators are authorized to add and delete group members, as well as groups at lower levels. The group structure is hierarchical because group members of higher level groups are automatically members of lower level groups in the same branch. The example in Figure 2 demonstrates the top-level groups A,B, and C with second level groups A.1, A.2, and A.3.

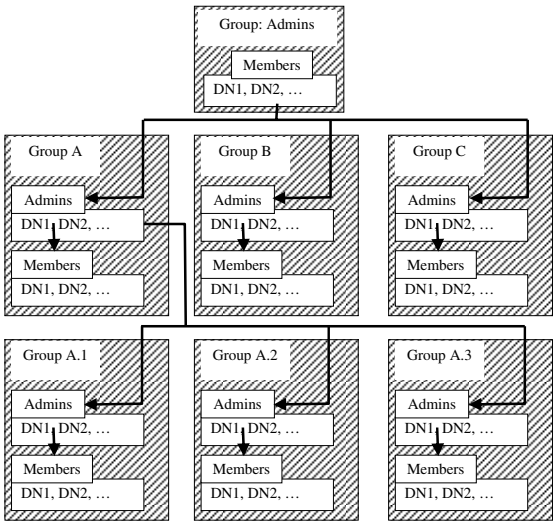

Figure 2. Clarens Virtual Organization Diagram

A further optimization, the hierarchical information in the DNs may also be used to define membership, so that only the initial significant part of the DN need to be specified in defining members of a group. DNs are structured to include information on the country (C), state/province (ST), locality/city (L), organization (O), organizational unit (OU), common name (CN), and (Email). An example DN issued by the DOE Science Grid CA for individuals is:

```
/O=doesciencegrid.org/OU=People/CN=John Smith 12345
```

For servers a DN could look like:

`/O=doesciencegrid.org/OU=Services/CN=host/www.mysite.edu` for servers.

To add all individuals to a particular group, only `/O=doesciencegrid.org/OU=People` need to be specified as member DN

2.2 Access Control Management

Access control management enables administrators to deny or allow groups (virtual organizations) of using resources. For example certain services have methods that are used for administration of the service by persons other than the local site administrator. You

do not want to give everybody access to these methods. Access Control Lists (ACLs) allow you to prevent and manage that.

Execution of Web Service methods as well as mapping of certificate DNs to users on the server system is controlled by a set of hierarchical ACLs in a similar fashion to the VO structure described in the previous sub section, and modelled after the access control (`.htaccess`) files used by Apache.

Methods have a natural hierarchical structure. Clarens places no arbitrary restrictions on the depth of this hierarchy, but a depth of two or three levels is most common, e.g. `module.method` or `module.submodule.method`.

An ACL consists of an evaluation order specification (allow, deny or deny, allow) followed by a list of DNs allowed, groups allowed, DNs denied and groups denied access. A DN or group granted access to a higher level method automatically has access to a lower level method, unless specifically denied at the lower level. The ACL specification is therefore evaluated from the lowest applicable level to the highest.

2.3 Remote File Access

In many "big science" experiments data is stored in files rather than in databases. As storage resources will be geographically distributed the data (thus the files) will be too. Scientists should be able to access remote data using well known interfaces (e.g. UNIX file system). Furthermore scientists should be able to deny or allow read or write access on these remote files, to groups of collaborators. The Clarens file service enables scientists to manage their remote data and create ACLs for it.

Clarens serves files in two different ways: in response to standard HTTP GET requests, as well as via a `file.read()` service method. A virtual server root directory can be defined for each of the above via the server configuration file which may be any directory on the server system. The `file.read()` method takes a filename, an offset and the number of bytes to return to the client. Error messages are returned as serialized RPC responses. Network I/O is handed off to the web server, which uses the zero-copy `sendfile()` system call where available to minimize CPU usage and increase throughput.

Other file access methods include `file.ls()` to obtain director listings, `file.stat()` to obtain file or directory information, and `file.md5()` to obtain a hash file for checking file integrity.

Just as with methods, files (and directories) can be subject to ACLs using the same structure as described in the previous sub section. The ACLs specification for files extend the method ACLs with two extra fields: read and write.

2.4 Dynamic Service Discovery

Within a global distributed service environment services will appear, disappear, and be moved in a unpredictable manner. It is virtually impossible for scientists, and applications to keep track of these changes. The discovery service allows scientists and applications to query for services and retrieve up to date information on the location and interface of a service. Using the discovery service, applications (and this include other services) can make service calls that are location independent by virtue of the discovery service. Binding to a location can than occur in real time. Such a discovery environment needs to scale to large numbers of servers and users without incurring prohibitively large amounts of administrative overhead.

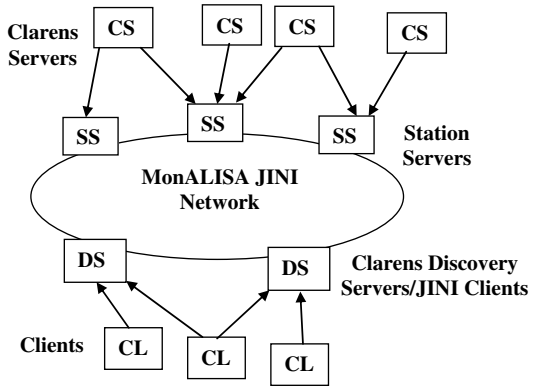

Figure 3. The MonALISA-based service discovery architecture.

The MonALISA[11] framework provides a scalable distributed monitoring service system using JINI/JAVA [35] and WSDL/SOAP technologies At the time of writing MonALISA was monitoring more than 90 sites and network connections. The sites range from 1 PC to dozens of computing farms with 100s of compute nodes[3]. Each MonALISA server acts as a dynamic service system and provides the functionality to be discovered and used by any other services or clients that require such information.

[3] You can access this information by downloading the monitoring client: http://monalisa.caltech.edu/dl_jClient.html

Information provided to MonALISA is usually arranged roughly as described by the so-called GLUE [36] schema, as a hierarchy of servers, farms, nodes and key/numerical value pairs. Although the schema is not ideal for organizing service description data, the scalable publish-subscribe network offers a ready to use service discovery environment.

Clarens servers can publish service information using a UDP-based application to so called station servers that in turn republish it to the MonALISA network.

Figure 3 shows the current discovery architecture where the JClarens server becomes a fully fledged JINI client, and aggregating discovery information from the JINI network. The JClarens server is consequently able to respond to service searches far more rapidly by using the local database.

The discovery service is one of the several services identified by the Open Science Grid consortium [25] as vital for robust distributed system development.

2.5 Shell Service

The Shell provides a secure way for authorized clients to execute shell commands on the server. The command is executed by a designated local system user.

The local system user is designated by using an ACL file located under the clarens/shell directory, named `.clarens_user_map`. The file maps user distinguished names to local system users.

Each mapping tuple consists of a system user name string, followed by a list of user distinguished name strings, a list of group name strings, and a final list reserved for future use. For example, this file maps the user: `/DC=org/DC=doegrids/OU=People/CN=Joe User` to the user *joe*:

Execution takes place in a sandbox owned by the local system user. This sandbox can be created or re-used for subsequent commands and is visible to the file service. Using the `shell.cmd_info` command the user gets back the top directory of the sand box that it can use to issue file service commands such as uploading and downloading files, but also commands like `file.ls`, and `file.find`.

2.6 Proxy Service

The proxy service provides a secure way to store and retrieve so-called "proxy" certificates on a Clarens server. Proxy certificates consist of a temporary certificate (public key) and unencrypted private key that can be used to log into remote servers without the inconvenience to type in the private key password over and over. This has the side benefit that it allows the proxy to be used on behalf of the user by others, so-called *delegation*.

This service also allows the user to use a previously stored proxy as a way of logging into the server by only knowing the certificate distinguished name and password that was used to store it.

Additionally, a stored proxy can also be "attached" to an existing session, thereby renewing an existing proxy, or bringing the benefits of delegation to sessions that were initiated without a proxy certificate - e.g. browsers use CA-issued certificates to initiate sessions.

3. Portal Functionality

According to [29] a Grid portal is a user's point of access to a Grid system. It provides an environment where the user can access Grid resources and services, execute and monitor Grid applications, and collaborate with other users. Portals can also lower the barrier for users (the scientists) to access Web Services and using Grid enabled applications.

As modern browsers have native support for SSL-encrypted connections and client-side certificate authentication, it provides a platform for constructing Grid portals on top of Clarens services.

Clarens is able to serve web pages in response to HTTP GET requests, the portal is implemented as a series of static web pages that embed JavaScript scripts to handle communication and web service calls using dynamic HTML. Such a portal implementation eliminates the need for users to install any additional software apart from a web browser, which most people already have.

Functionality currently provided by the browser interface include: browsing remote files, access control management, virtual organization management, service discovery, job submission.

The remote file access portal component has a look and feel similar to conventional file browsers such as MS explorer (within a JavaScript context). Users can browse the parts of the remote file system to which they have been granted access to.

Similar to the file access portal component, the discovery portal component enables users to query for servers, and services within a browser context, and by point and click on the query results can navigate to portal components.

Clarens does not offer a framework to build portals other than the already available tools that enable

developers to embed JavaScript components that execute Web Service calls to Web Services.

4. Performance

Important for the success of Web Services is the performance of the web server, hosting the service. Although badly engineered Web Services can still have a slow performance, the server itself should not become the bottleneck. Poor performance can lead to high hardware costs for larger servers or unacceptable long response times. A performance and scalability test for *PClarens* was performed using a system consisting of a dual 2.8 GHz Xeon server with 1 GB of memory, accessed using a 100 Mb/s local area network

In this test a configurable number of unencrypted client connections were opened and set to access the `system.list_methods` Web Service method as rapidly as possible. The client was run on a 2.6 GHz Pentium 4 workstation as a single process opening connections to the server and completing requests asynchronously,

The clients would make 1000 web service calls to the server, after which the total response time was measured (e.g. 0.5 seconds for 1000 calls means 2000 calls per second). This process was repeated 2000 times for each number of asynchronous clients (we varied the number of asynchronous clients between 1 and 79). After varying the clients from 1 to 79 the process described above was repeated to verify the results. A grand total of 316 million requests where successfully completed without any client or server errors.

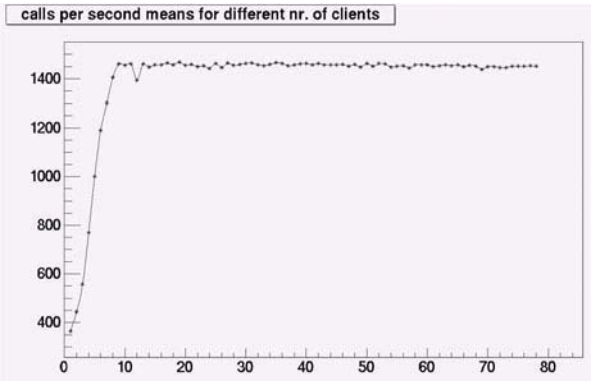

Figure 4. Clarens performance

Each request passed through two access control checks involving access to several databases, namely checking whether the client credentials are associated with a current session, and whether the client has access to the particular method being called. No caching was performed on the server, with each request incurring a database lookup for all registered methods in the server, and serializing the resultant list of more than 30 strings as an array response in XML-RPC. The Python client de-serialized each response to a native list object that could be used in the rest of the script.

In effect this test reports the overhead that the PClarens server system imposes on service request, with control passing through all parts of the server used by a typical service.

This test under-reports the actual server performance for at least two reasons: in a more realistic environment multiple client machines would be accessing the server, and the Apache server configuration was used unmodified on a Linux 2.4-based kernel which is known to be a sub-optimal setup.

A final summary of the results of this test is given in Figure 4 showing an average of 1450 requests per second served.

During the test the controlling Apache server process that is responsible for accepting new requests and opening new connections constantly used all available CPU time on of the two CPUs of the test server. This is probably due to the way that the file descriptors use for network connections are handled by the Linux 2.4-kernel.

Future tests will be repeated using more optimized Apache configurations and SSL/TLS-encrypted network connections. Informal test show the latter to reduce performance by up to 50%. No formal tests were setup for other Web Service frameworks such as Globus. However 3rd party test results with the Globus toolkit 3 differed[4] substantially with Clarens performance

5. Related Work

Several other Web Service and portal frameworks have been developed in the last couple of years. The latest versions of Globus [9] have been service oriented frameworks based on the open grid service architecture (OGSA). Although Globus offers secure and authenticated access using the concept of grid map files, it has a much coarser authentication and access control granularity than the Clarens ACL and VO management. Furthermore, the server performance (calls/second) for Globus 3 are not as high as the Clarens server.

[4] A trivial method 100 times (ignoring first invocation) across a 100Mbps LAN using GTK 3.0 and GTK 3.9.1 resulted in 5 to 1 calls per second.

IBM Websphere [10] is a commercial product that enables you to develop and deploy Web Services and is therefore not a desirable candidate to be deployed in large science collaborations that rely on open source.

OGCE [30] provides open source software for building Grid Computing Portals. Several of its portlets are based upon Globus services. A drawback (at the time of writing this paper) is the large size of the download, setup and configuration functionality. Several OGCE portlets have a conceptual commonality with Clarens based portals. For example: remote file access and job submission. Furthermore, OGCE portlets are based on Java, rather than JavaScript which offers a richer development environment.

Glite[31] is a Perl based service framework based on Alien [32] that is used by the EGEE project [33] to develop web services. Several of the interfaces developed for services in this project are similar (but not the same) to Clarens service interfaces. The first versions of the Glite framework were based on Alien which initially was not designed to be a generic service oriented framework, while Clarens was. Substantial work has been carried out however to make Glite, less dependent on Alien.

6. Future Work

Future work will focus (amongst others) on: GUI framework, message based protocols, mass storage integration, improved service discovery functionality.

The JavaScript browser based GUI has limitations in creating easy to use and functional graphical user interfaces. Instead Java offers more flexibility in creating sophisticated user interface to support scientists in interaction with a distributed service environment. The current Clarens Web Service implementation was designed for a request response mode of operation, making it ill-suited for the type of asynchronous bi-directional communication required for interactions between users and the jobs they are running on private networks protected by network address translation (NAT) and firewalls. An instant messaging (IM) architecture provides the possibility to overcome this limitation. Since messages can be sent and received by jobs asynchronously, jobs can be instrumented to act as Clarens servers, or clients sending information to monitoring systems or remote debugging tools.

Although Clarens provides remote file access through a Web Service, it does not support interfaces to mass storage facilities yet. Work is under way to provide an SRM service interface [27] to dCache [28] such that Clarens can support robust file transfer between different mass storage facilities.

Work is underway to provide interoperability between the Clarens discovery service and Globus MDS [39] such that both systems can publish/retrieve information in the other system. Other activities include collaboration with the EGEE project on a common discovery interface.

7. Conclusions

The Clarens Web Service framework is gaining acceptance in the science community to support the development of a scalable distributed service environment. Several of the projects have chosen Clarens as it offers a good service response performance and yet provides powerful features such as ACL and VO management, service discovery, and remote file access.

Clarens provides a growing functionality for distributed analysis in a Grid-based environment, coupled with a set of useful client implementations for physics analysis. The projects that utilize Clarens also provide valuable feedback, which enable the Clarens team to enhance and improve the core functionality of Clarens and reuse components across projects that focus on scientific analysis.

8. Acknowledgements

This work is partly supported by the Department of Energy grants: DE-FC02-01ER25459, DE-FG02-92-ER40701, DE-FG02-04ER25613, DE-AC02-76CH03000 as part of the Particle Physics DataGrid project, National Science Foundation grants: ANI-0230967, PHY-0218937, PHY-0122557, PHY-0427110 and by Department of State grant: S-LMAQM-04-GR-170. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Department of Energy, the National Science Foundation, or the Department of State.

9. References

[1] Steenberg, C., Aslakson, E., Bunn, J., Newman, H., Thomas, M., Van Lingen, F., "The Clarens Web Service Architecture", Computing for High Energy Physics, La Jolla, California, 2003

[2] Apache Web Server, Apache Software Foundation, http://www.apache.org

[3] XML Remote Procedure Call Website, http://www.xmlrpc.com

[4] The Tomcat Servlet Engine, http://tomcat.apache.org
[5] Simple Object Access Protocol, W3 Consortium, http://www.w3.org/2002/ws/
[6] I. Osborne, S. Muzaffar, L. Taylor, L. Tuura, G. Alverson, G. Eulisse, "IGUANA Interactive Graphics Project: Recent Developments", In proceedings of CHEP 2004, Interlaken
[7] Physh, http://cmsdoc.cern.ch/cms/aprom/physh/
[8] M. Ballintijn, "Global Distributed Parallel Analysis using PROOF and AliEn", In Proceedings of CHEP 2004 Interlaken
[9] Foster, C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit" Intl. J. Supercomputer Applications, 11(2):115-128, 1997
[10] Websphere, http://www.websphere.org/
[11] I. Legrand, "MonALISA - MONitoring Agents using a Large Integrated Service Architecure" International Workshop on Advanced Computing and Analysis Techniques in Physics Research, Tsukuba, Japan, December 2003
[12] UltraLight Collaboration "UltraLight: An Ultrascale Information System for Data Intensive Research" proposal Submitted to NSF MPS/Physics: "ITR" February 2004. See also: http://ultralight.caltech.edu/
[13] LHC Project, http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/
[14] R. Williams, C. Steenberg, J. Bunn, " HotGrid: Graduated Access to Grid-based Science Gateways", In Proceedings of IEEE Supercomputing Conference, Pittsburgh USA, 2004
[15] C. Steenberg, E. Aslakson, J. Bunn, H. Newman, M. Thomas, F. van Lingen "Clarens Client and Server Applications" CHEP, La Jolla California 2003
[16] C. Steenberg, J. Bunn, I. Legrand, H. Newman, M. Thomas, F. van Lingen, A. Anjum, T. Azim "The Clarens Grid-enabled Web Services Framework: Services and Implementation" In Proceedings of CHEP, Interlaken Switzerland 2004
[17] A. Ali, A. Anjum, R. Haider, T. Azim, W. ur Rehman, J. Bunn, H. Newman, M. Thomas, C. Steenberg. "JClarens: A Java Based Interactive Physics Analysis Environment for Data Intensive Applications" in the Proceedings of ICWS, the International Conference of Web Services, San Diego, USA 2004
[18] Clarens homepage, http://clarens.sourceforge.net
[19] Join European Torus, http://www.jet.efda.org/
[20] LIGO, http://www.ligo.caltech.edu/
[21] The Compact Muon Solenoid Technical Proposal, CERN/LHCC 94-38 (1994) and CERN LHCC-P1; see also: http://cmsdoc.cern.ch/
[22] The ATLAS Technical Proposal, CERN/LHCC 94-43 (1994) and CERN LHCC-P2; see also: http://atlasinfo.cern.ch/ATLAS/TP/NEW/HTML/tp9new/tp9.html
[23] J. Bunn and H. Newman "Data Intensive Grids for High Energy Physics", in "Grid Computing: Making the Global Infrastructure a Reality", edited by Fran Berman, Geoffrey Fox and Tony Hey, March 2003 by Wiley.
[24] Lambda station, http://www.lambdastation.org/
[25] Open Science Grid, http://www.opensciencegrid.org/
[26] P. Love, I. Bertram, D. Evans, G. Graham, "Cross Experiment Workflow Management: The Runjob Project", In proceedings of CHEP, Interlaken Switzerland 2004
[27] A. Shoshani, A. Sim, J. Gu, "Storage Resource Managers: Middleware Components for Grid Storage", In proceedings of Mass Storage Systems conference, Maryland USA 2002
[28] P. Fuhrmann, "dCache the commodity cache", In proceedings of the Twelfth NASA Goddard and Twenty First IEEE Conference on Mass Storage Systems and Technologies, Washington DC 2004
[29] D. Gannon, G. Fox, M. Pierce, B. Plale, G. von Laszewski, C. Severance, J. Hardin, J. Alameda, M. Thomas, J. Boisseau, "Grid Portals: A Scientist's Access Point for Grid Services" Sept. 19 2003, GGF working draft
[30] OGCE http://www.ogce.org/
[31] M. Lamanna, B. Koblitz, T. Chen, W. Ueng, J. Herrala, D. Liko, A. Maier, J. Moscicki, A. Peters, F. Orellana, V. Pose, A. Demichev, D. Feichtinger, "Experiences with the gLite Grid Middleware" In proceedings of CHEP, Interlaken Switzerland, 2004. see also: http://glite.web.cern.ch/glite/
[32] P.Buncic, A.J. Peters, P.Saiz "The AliEn System, status and perspectives",In proceedings of CHEP, La Jolla, California 2003
[33] E. Laure, F. Hemmer, F. Prelz, S. Beco, S. Fisher, M. Livny, L. Guy, M. Barroso, P. Buncic, P. Kunszt, A. Di Meglio, A. Aimar, A. Edlund, D. Groep, F. Pacini, M. SGaravatto, O. Mulmo, " Middleware for the next generation Grid infrastructure", In proceedings of CHEP, Interlaken, Switzerland 2004.
[34] C. Steenberg, J. Bunn, T.Hickey, K. Holtman, I. Legrand, V. Litvin, H. Newman, A. Samar, S. Singh, R. Wilkinson (for the CMS Collaboration), "Prototype for a Generic Thin-Client Remote Analysis Environment for CMS" Proceedings of CHEP, paper 3-044, p. 186, H.S. Chen (ed.), Beijing China, 2001
[35] JINI, http://www.sun.com/software/jini/
[36] GLUE, http://www.cnaf.infn.it/~sergio/datatag/glue/index.htm
[37] AXIS, http://ws.apache.org/axis/
[38] JSON RPC, http://oss.metaparadigm.com/jsonrpc/index.html
[39] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, "Grid Information Services for Distributed Resource Sharing", Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
[40] National Virtual Observatory http://www.us-vo.org
[41] Particle Physics Data Grid, http://www.ppdg.net/
[42] Grid Physics Network, http://www.griphyn.org/
[43] International Virtual Data grid laboratory, http://www.ivdgl.org/
[44] Grid3, http://www.ivdgl.org/grid2003/