BUILDING APPLICATIONS FROM A WEB SERVICE BASED COMPONENT ARCHITECTURE

D. Gannon, S. Krishnan, A. Slominski, G. Kandaswamy, L. Fang
Department of Computer Science
Indiana University
Bloomington, IN 47401
gannon@cs.indiana.edu

Abstract This paper describes an approach to building large-scale, distributed applications based on a software component composition model that allows web services to be used as the basic units. The approach extends the Common Component Architecture used in many parallel supercomputer applications, from static composition of directly coupled processes to a system that incorporates mediated workflow between remote services. The system also allows legacy applications to be easily wrapped as a component and executed from a service factory. We motivate the work in terms of a large, distributed application for modeling severe storms. The entire system is based on a three-level architecture with a portal providing the user interface, a set of security and factory service utilities in the middle and the application services and components in the back-end.

Keywords: Grid, Web Services, Software Component Architectures

1. Introduction

The goal of building software from pluggable components is not new. Unix pipes allowed simple linear composition of programs based on composing an output stream with an input stream. In the area of computer visualization systems like AVS [1]allowed users to put together complex graphics applications by composing graphs of simple filter and rendering components. The Common Component Architecture (CCA) [8]is a modern component system that is used to build large supercomputer simulations by coupling together components such as linear solvers and boundary condition evaluators. Many other component systems exist they differ primarily in the way individual components are required to behave and in the semantics of the method of composition. Most use a variation of the ”inversion of control” pattern which is based on extracting the control of the overall application and placing it in surrounding framework.

Grid systems are distributed frameworks for sharing resources among the membership of a virtual organization. Currently Grids are designed as a collection of common services which provide security, data management, application execution scheduling, notification and logging, policy expression and system monitoring. These services are often implemented as web services and many Grid applications can be constructed by composing collections of basic services and ”atomic” application components.

For example, the LEAD project [12]is trying to build a distributed cyber infrastructure powerful enough to enable the ”better than real-time” prediction of mesoscale weather events such as tornadoes. One of the goals of LEAD is to allow a scientist to compose applications that are advanced Grid measurement and prediction scenarios of various types. For example, an application may allow a user to select a region of land such as a state or a few counties, and a data query, such as current radar data. From this information, the application begins a data mining monitoring process which searches the radar data in that region for anomalous behavior. When strange conditions are detected, compute resources are allocated and a series of simulations are launched. The simulations are monitored to see how they align with the real weather and those that are not consistently tracking reality are terminated. Addition resources may be applied to those that are accurate. For example, the radar array be asked to provide more detail so the resolution of the simulation can be increased. This type of application is a good example of a dynamic workflow that requires an extensive distributed framework of services to be composed. Specifically, the services include

- Metadata catalog services for extracting information about past storm histories and available instrument data streams for a given geographic region.

- A data mining service that can be configured to mine the instrument data.
- Resource allocators for both space and computation
- Simulation instance factories that can be used to launch version of a simulation with different parameters.
- Visualization services that can convert the output of simulations into movies and other display data the user wants.
- Logging services so the user can keep track of what is going on.

In the sections that follow we will outline a service composition model that allows these service components to be composed by scientists to enact specific experimental prediction scenarios. The system is based on a three level architecture. The user interacts with the system by means of a Grid portal that forms the first level. In the second level we have the security framework and application factories. At the third level are the specific service instances that participate in computation.

2. The Portal

The Grid portal is based on the Open Grid Computing Environment (OGCE) [16]framework. But it could also use any JSR-168 compliant portal such as the GridSphere [22]The OGCE portal allows the user to interact Grid applications and services from a standard web browser. The portal provides each user with a context of resources including a proxy identity certificate to allow the portal to authenticate with remote grid services on behalf of the user. The portal provides tools for defining geographic regions, querying and searching metadata catalogs, checking job execution logs, cataloging experimental results and defining workflows with simple graphical tools.

The portal represents the top layer of the grid stack. The bottom layer of the stack is a set of shared resources. These may be real (computers, databases, instruments) or virtual (documents, name spaces, ontoloties). Above the resource layer we have OGSI [23]or WSRF [3]which is a set of basic web service abstractions designed to provide a standard mechanism for describing resources.

From our perspective, a set of services that provide a mechanism for communicating ”events”, such as WS-Notification, is critical. We will return to this issue below.

Built upon these foundations we have the Open Grid Service Architecture which is the federation of services that define the core grid platform. Finally, as shown in Figure 1, the portal build upon these services to create the application level services it needs.

While the portal provides several other workflow composition tools, such as an interface to upload Dagman Condor scripts, the graphical component

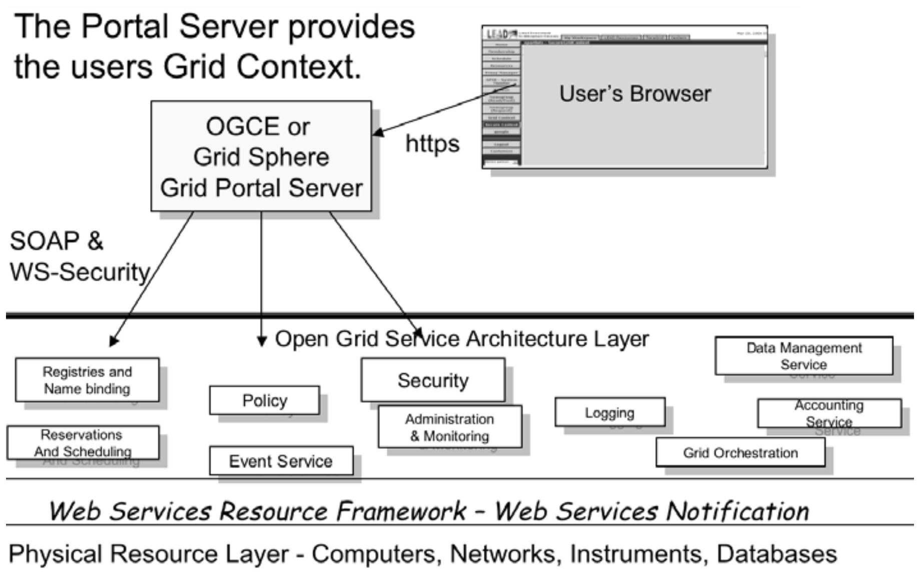

The Portal Server provides the users Grid Context.
OGCE or Grid Sphere Grid Portal Server
User's Browser
https
SOAP & WS-Security
Open Grid Service Architecture Layer
Registries and Name binding
Reservations And Scheduling
Policy
Event Service
Security
Administration & Monitoring
Logging
Grid Orchestration
Data Management Service
Accounting Service
Web Services Resource Framework – Web Services Notification
Physical Resource Layer - Computers, Networks, Instruments, Databases

Figure 1. The layered organization of a Grid. The portal is the user's access point.

composition tool is the primary mechanism for building applications based on web services and CCA components. The user interface 2 is a simple ”drop and drag” composer that is similar to the standard CCA composer or the Scirun II composer. (Scirun [24], from the University of Utah, is also based on CCA.) It is also similar to the Triana [25]and Kepler [26]interfaces.

The primary difference between our composer and others lies in the back end. The OGCE composer allows both web service and CCA components to be integrated into a single application. While this work is still in progress and not yet released, it is based on a compiler that translates the graphical specification of the application into a standard workflow language. We will return to this topic later in this paper.

There is one more important point to make about the composer: it is an example of how a Grid services provides an interface to the user through the portal. Our model for doing this is similar to the WSRP specification [20], but it takes advantage of the structure of a Grid service. For Grid services that have user interfaces, we have defined a ”standard” service data element (in OGSI terms) or resource (in WSRF) terms that provides the URL to load the GUI for that service. This GUI may be an applet (as in the case of the composer) or it may be an xhtml document with imbedded java script.

There are two difficult problems that must be solved when you want a user to interact with a service through a remote client interface. What do you do

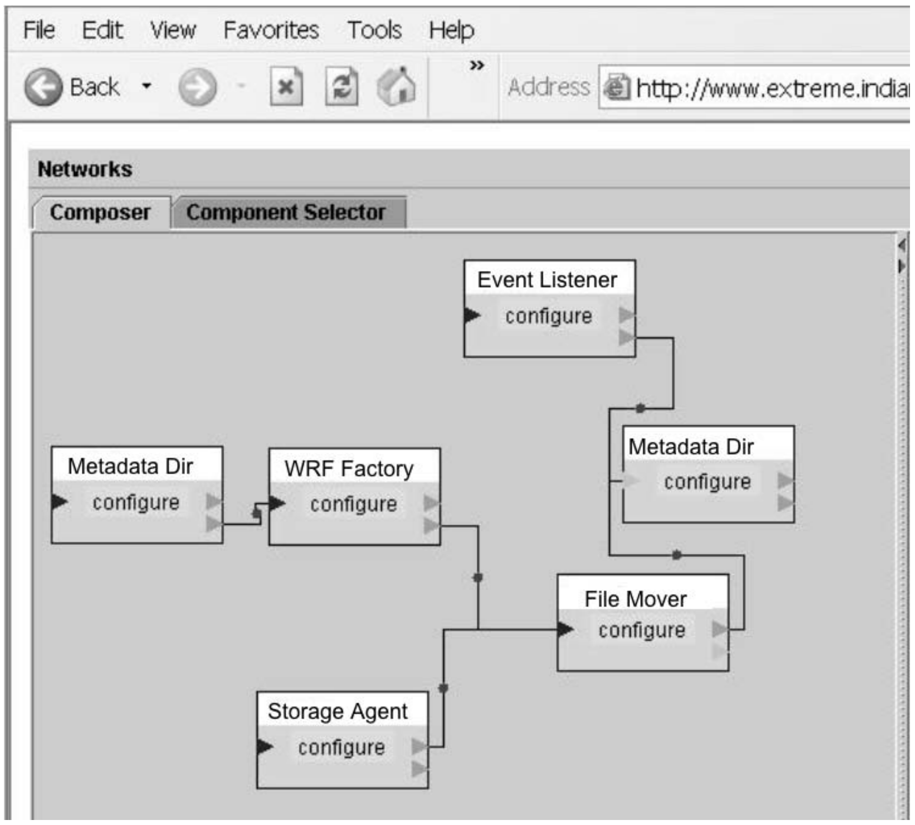


Figure 2. Component Composer Interface. This is the standard way most component systems use to assemble small workflows. This version is an applet that will be distributed as part of OGCE.

when the Grid service is behind a firewall? What do you when the Grid service requires the user to use WS-Security when it talks to the service? The user may have the interface, but not their own X.509 identity certificate. The solution to both of these problems is to filter all transaction between the service and the user through the portal, which we assume is a gateway through the firewall and also has access to the user's proxy certificiate. The protocol is as follows. When the user discovers the Grid service in the portal directory, a special servlet in the portal fetches the interface and passes it to the user. The user interact with the GUI which sends user commands back though the browser HTTPS connection to the portal. The portal then forwards those commands back to the Grid service using the user's credentials for the XML digital signature in the WS-Security-based communication with the service (see Figure 3).

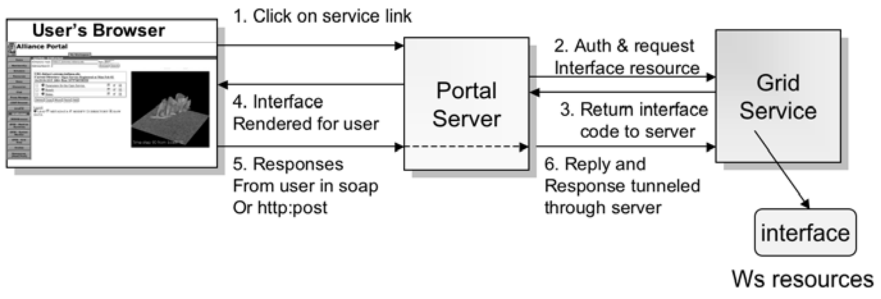

User's Browser
Alliance Portal
1. Click on service link
4. Interface Rendered for user
5. Responses From user in soap Or http:post
Portal Server
2. Auth & request Interface resource
3. Return interface code to server
6. Reply and Response tunneled through server
Grid Service
interface
Ws resources

Figure 3. Protocol for loading a remote user interface for a Grid service.

3. Wrapping Legacy Applications as Components

One of the most frequently asked questions about this system is, "I have a Fortran application that i cannot modify. Can i use it in this system?" In other words, in what way can we turn a legacy application into a service component? By legacy application, we mean a program that can be submitted through a batch queue. For example, a script that that inserts some text into a program, compiles it and then runs it while consuming some input files and producing some output files. (We do not consider the case of interactive applications, though, in some cases, it is possible to threat them in the same way we describe here.)

Our approach is to use the factory pattern. We will build a service which is capable of starting an instance of the application on behalf of the service client. The input to the factory is either through a direct web service call, or through a user interface as described in the previous section. The input consists of any configuration parameters that the factory needs to start the job running.

There are two problems that must be solved. First, it is not hard for a web-service programming professional to write a web service to build an application factory. But it is often the case that the person who wishes to build such a service is the scientific programmer who is responsible for the deployment and testing of the application. This person is not a web-services programming expert. So can we automatically generate the factory for an application from a specification provided by the application provider?

The second problem involves security. One problem with sharing legacy applications is deployment. An application may run in one user's environment, but it may take a substantial effort to deploy it in another user 's environment. Furthermore once an application leaves the developers control, that developer now has a version control problem. This is one motivation for provide the application as a service rather than as a program. But if an application provider creates an application factory that can instantiate a running instance of the application for somebody else, then how do we provide the authorization control mechanism that determines exactly who is allowed to run the app?

Taking the first question, we note that we have built an *Factory Service Generator* into the portal server. This allows the application provider to automatically generate a factory service from a relatively simple xml specification. The user need only provide

- a script that can execute the application. Any needed input files should appear as filename parameters to the script. In addition we assume that the script take a ”jobname” parameter so that it can create a private working directory for each run of the program. (The factory service assumes it can run multiple instances of the application concurrently so care must be taken that intermediate and final files are not overwritten.)
- an XML file that describes the application and the input parameters and other annotations to be placed on the user interface to the factory.

Given these two items, the portal can automatically generate a factory and start it running on behalf of the application providing user. When invoked the factory simply executes the script. There are no restrictions on this script other than the those described above. In many cases, we have the script run another web service which is a transient instance of a service that is dedicated to one user. In other cases, we have the script execute a complete workflow. In any case it is very useful if the script has the capability of sending event notifications such as “job complete”, “output file is at URL ../jobname/filename” or “job failed because ..”. Python, GridAnt [27]and other high level scripting tools have this capability. We will describe how this is used later.

Once the factory is running, the provider must decide who is allowed to invoke it. The solution we use is based on capabilities. Each application provider supplies a list of individuals or groups that he or she will allow to run his or her application factory. The portal capability manager will then, for each user and group, create a signed XML capability document that says this individual or group has permission to execute the application factories ”create instance” methods. When one of these users logs into the portal, the portal server loads the proxy cert for the user, which is then used to load that user's capabilities (see Figure 4.) If the user invokes the factory service, the appropriate capability is added to the SOAP header for the service request. The factory service verifies that the request is authentic and that the capability is authentic and that the requestor is the same person as the capability certificate.

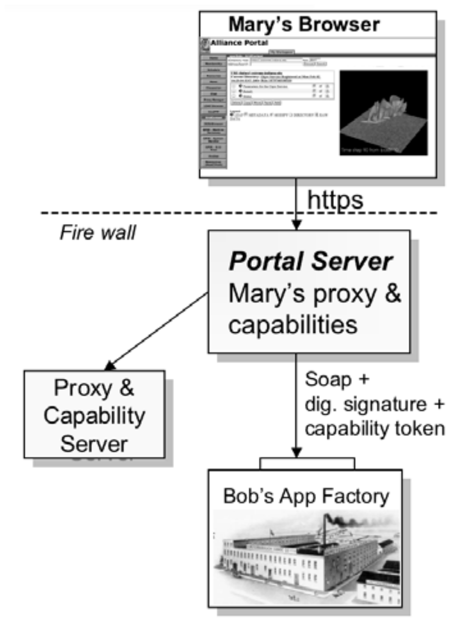

Mary's Browser
Alliance Portal
https
Fire wall
Portal Server
Mary's proxy & capabilities
Proxy & Capability Server
Soap + dig. signature + capability token
Bob's App Factory

Figure 4. capability-based authorization protocol for access to factory services.

4. XCAT3 - Integrating CCA Components and Grid Services

We briefly introduce the Common Component Architecture (CCA) [8]and the Open Grid Services Architecture Infrastructure (OGSI) before presenting the architecture of the XCAT3 [15]framework in the following subsections.

4.1 Common Component Architecture

The Common Component Architecture is defined by a set of framework services, and the definitions of the components that use them. Each component communicates with other components by a system of *ports*. Ports are defined by a type system that is expressed in the Scientific Interface Definition Language (SIDL) [2]. SIDL is similar in nature to WSDL, but differs in the support it provides to data structures common to scientific computing. There are two types of CCA ports:

- **Provides Ports** are the services offered by the component. Each provides port implements an interfaced defined in SIDL.

- **Uses Ports** are the hooks that enable a component to use a service provided by another component. Uses ports are stubs that a component *uses* to invoke the services *provided* by another component. Uses ports are also defined in SIDL.

A uses port of one component can be connected to a provides port of another component as long as they implement the same SIDL interface. Connections between uses and provides ports are made at runtime. A component needs to execute a `getPort` statement to grab the most recent reference to provider, and a `releasePort` when it has finished using it. The get/release semantics of component connections enable the framework to infer if any port calls are being made at any point in time, and also enable the connections to be changed dynamically.

Apart from uses and provides ports, a component also implements a *ComponentID* interface that has methods that uniquely indentify the component, and provide metadata about it. CCA also defines a *Builder* service for creation and composition of these components.

4.2 Grid Services

The Open Grid Services Infrastructure extends the Web services model by defining a special set of service properties and behaviors for stateful Grid services. Some of the key features of OGSI that separate Grid services from simple Web services are:

- **Multiple level naming:** OGSI separates a logical service name from a service reference. A Grid Service Handle (GSH) provides an immutable name for a service, while a Grid Service Reference (GSR) provides a precise description of how to reach a service instance on a network, e.g a WSDL reference. A GSH can be bound to different GSRs over time.

- **Dynamic Service Introspection:** Grid services can expose metadata to the outside world through the use of Service Data Elements (SDE), which are XML fragments that are described by a Service Data Descriptor (SDD). SDEs can be queried by name or type, and can be used to notify state changes to clients.

- **Standardized ports:** Every Grid service implements a *GridService* port, which provides operations to query for SDEs, and manage lifetime of the Grid service. OGSI also specifies standard ports for creation, discovery, and handle resolution.

Recently, OGSI has been superceded by the WS-Resource Framework (WSRF) [3]proposal. WSRF also addresses the above issues for stateful Grid services, but tries to integrate them better with the current Web service standards.

4.3 XCAT3 Architecture

Currently, the XCAT3 framework is implemented in Java, and we plan to implement a C++ version that is interoperable with the former. In XCAT3, we implement the CCA specification in the context of Grid services. To that end, some of the key features of XCAT3 are:

- **Ports as Grid services:** As per the CCA specification, one component can have more than one provides port of the same type. Simple Grid and Web services allow multiple ports of the same `portType`; however, multiple bindings of the same port are semantically equivalent. Hence, the same operation on different ports of the same type affect the service in exactly the same way. However, unlike Web service ports, ports in CCA are designed to be stateful. Hence, every provides port in XCAT3 is implemented as a separate Grid service 5. The consequence of this is that every provides port inherits multiple level naming from the OGSI specification, and this enables the ports to be location independent. Additionally, any Grid service that is compliant with the OGSI specification can serve as a provides port.

- **ComponentID as a Grid service:** The ComponentID, as specified by the CCA specification, is also implemented as a Grid service. It exposes handles and references of all the provides ports that a component contains, and thus acts as a manager for the component. Users can query a component for the types of services provided via the ComponentID, and connect to them directly.

Some of the other useful services in the XCAT3 framework are:

- **Builder Service:** As mentioned before, the Builder service defines methods for component creation and composition. We allow remote instantiation of components via ssh or Globus GRAM provided by the Java CoG [4]kit. For composition purposes, the Builder service provides `connect` and `disconnect` methods for connecting and disconnecting a uses port to a provides port respectively. Once the ports have been connected, all communication between them is via Web service invocations provided by the XSOAP [5]toolkit.

- **Handle Resolver:** Since we employ multiple level naming for our ports and ComponentIDs, we need to use a handle resolution mechanism that translates a GSH to a GSR. This is provided by the Handle Resolver service. The Handle Resolver, as other Grid services in the XCAT3 framework, is implemented using the GSX [6]toolkit, which provides a lightweight implementation of the OGSI specification.

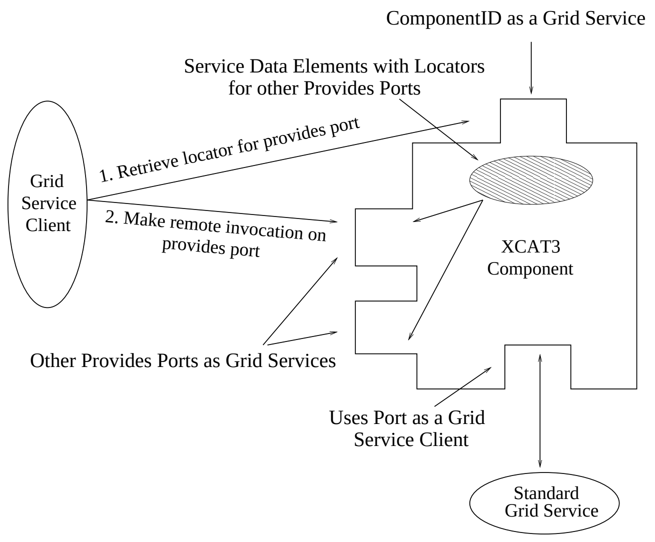

ComponentID as a Grid Service
Service Data Elements with Locators for other Provides Ports
Grid Service Client
1. Retrieve locator for provides port
2. Make remote invocation on provides port
XCAT3 Component
Other Provides Ports as Grid Services
Uses Port as a Grid Service Client
Standard Grid Service

Figure 5. Every XCAT3 component port is a Grid service. It contains SDEs with locators for all provides ports, which are also Grid services themselves

5. An Example

As an example that integrates most of these ideas together, we consider an application that is part of a much larger LEAD scenario. Much of LEAD requires large simulations based on weather data input. The output of these simulations consists of data fields that represent severe thunder storms and tornadoes. It is useful to have a tool that can generate visualizations from these simulation outputs. We have an application factory that launches a OGRE script [17]for which sets up the LAM MPI on a cluster and runs a parallel rendering program on the output from the WRF job. Once the parallel rendering is complete, it launches a conversion program which translates the rendered output to a GIF movie that can be viewed from the browser. OGRE scripts are capable of publishing events into the notification system (which is currently being converted to work with WS-Notification). An event listener, listens for all events published under the topic defined by the name of this OGRE execution. These events are logged into a special directory service visible from the portal. The entire workflow is depicted in Figure 6. (Note that the workflow for this example predates the composer tool described earlier. The workflow was hand crafted and not generated from the picture.)

The user can discover the status of each execution of this workflow by going to the portal ”Grid Context” directory service. There entry for each execution looks like a directory which contains a list of all parameters used to launch the workflow, the log of execution events and a reference to the output GIF movie.

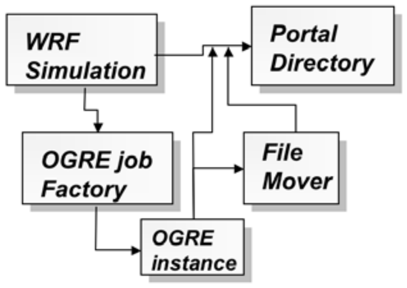
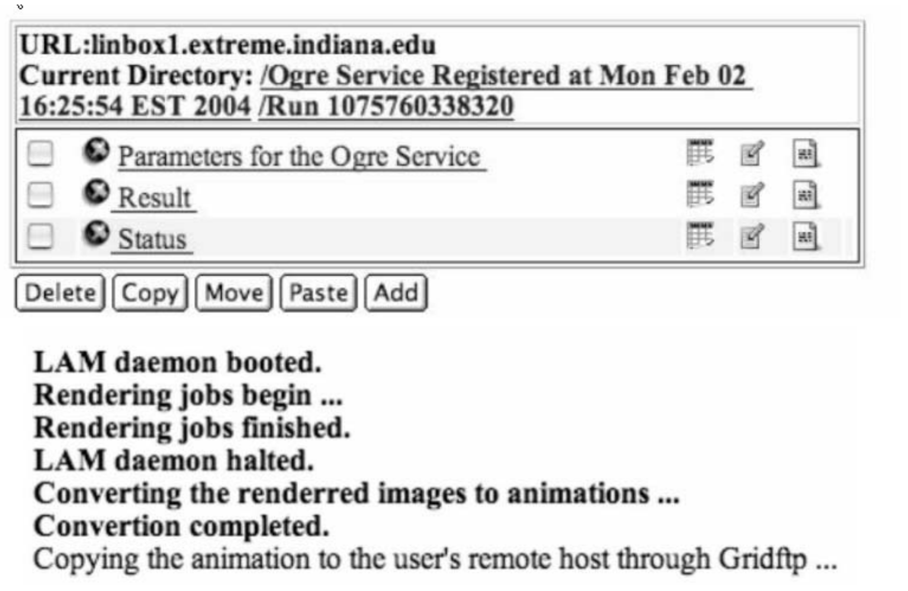


Figure 6. Complete distributed WRF output animation application. The output from the wrf simulation is used as input to the OGRE animation script. All components log events to the event channel. The event listener captures them and pushes them to the directory service.

As shown in Figure 7, selecting ”status” displays the log of events received. In Figure 8, we see that selecting ”results” will run the GIF movie in the right hand window.

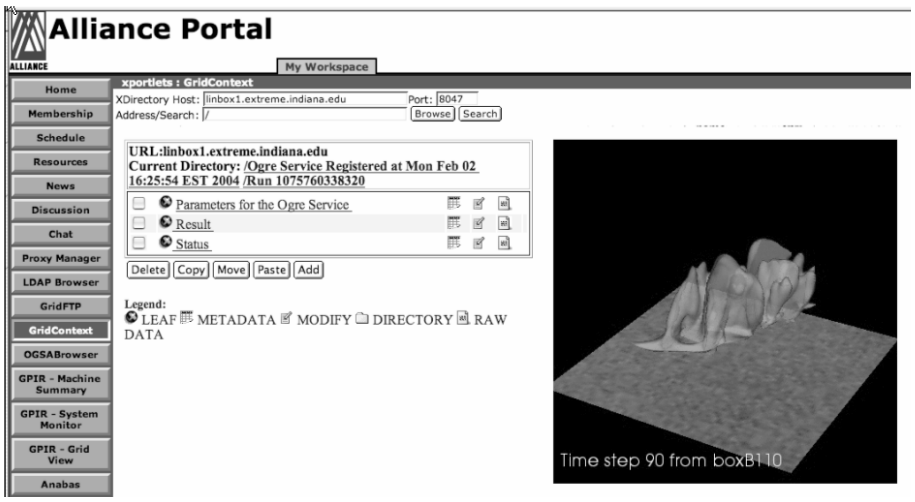


URL:linbox1.extreme.indiana.edu
Current Directory: /Ogre Service Registered at Mon Feb 02 16:25:54 EST 2004 /Run 1075760338320

Parameters for the Ogre Service
Result
Status

```
LAM daemon booted.
Rendering jobs begin ...
Rendering jobs finished.
LAM daemon halted.
Converting the renderred images to animations ...
Convertion completed.
Copying the animation to the user's remote host through Gridftp ...
```

Figure 7. A Directory service record for this execution of the workflow. Selecting ”Status” shows the even log.



XDirectory Host: linbox1.extreme.indiana.edu Port: 8047
Address/Search: /

URL:linbox1.extreme.indiana.edu
Current Directory: /Ogre Service Registered at Mon Feb 02 16:25:54 EST 2004 /Run 1075760338320

Parameters for the Ogre Service
Result
Status

Legend:
LEAF METADATA MODIFY DIRECTORY RAW DATA

Figure 8. Selecting ”Results” shows the output movie on the right.

6. Conclusion

This paper has illustrated a three level architecture for distribute Grid applications. At the top level we have a Grid portal which contains a suite of tools for creating grid services and composing application from them. This portal provides the secure interface to the back end Grid which may run behind a firewall. The middle tier is a set of services that support our Grid applications. These include security services, such as a proxy certificate repository, an authorization system based on capability tokens, directory services and a notification system. One important tools is a factory service generator that allows users to describe a legacy application and the portal will generate a factory service to access and launch instances of that application. The back end is composed of the application resources and services.

We also describe how we transformed the Common Component Architecture into a Grid service-based framework. This allow CCA components to be used as Grid services and composed into service-based workflows. It also allows regular web services to be integrated into CCA distributed applications.

In the year ahead we will on building applications from services and CCA components. While much has been done, we feel there is much more to be learned and the most important discoveries will come from applying this technology to real problems.

Acknowledgments

We would like to thank our collaborators on the LEAD project for their help in understanding the issues, and the NSF and DOE for their support of the projects that funded this work.

References

[1] I. Curington and M. Coutant. AVS: A flexible interactive distributed environment for scientific visualization applications. Proceedings of 2nd Eurographics Workshop on Scientific Visualization, 1991.

[2] Elliott, N., Kohn. S., Smolinski, B. Language Interoperability for High-Performance Parallel Scientific Components. International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE). 1999.

[3] Globus Alliance, IBM, and HP. Web Service Resource Framework. http://www.globus.org/wsrf. 2004.

[4] Argonne National Lab. Commodity Grid Toolkit. http://www.globus.org/cog. 2004.

[5] Slominski, A., Govindaraju, M., Gannon, D., Bramley, R. Design of an XML based Interoperable RMI System : SoapRMI C++/Java 1.1. IPDPS 2001.

[6] Grid Service Extensions (GSX). http://www.extreme.indiana.edu/xgws/GSX. 2004

[7] Agarwal, M., and Parashar, M. Enabling Autonomic Compositions in Grid Environments. Proceedings of the 4th International Workshop on Grid Computing (Grid 2003), Phoenix, AZ, USA, IEEE Computer Society Press, pp 34 - 41, November 2003

[8] Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S. McInnes, L., Parker, S., and Smolinski, B.. Towards a common component architecture for high performance scientific computing. In Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing, 1998.

[9] Business Process Execution Language for Web Services Version 1.1. http://www-106.ibm.com/developerworks/library/ws-bpel/

[10] Casanova, H. and Dongarra, J, NetSolve: a network server for solving computational science problems. Proceedings SC 96.

[11] Condor Dagman, http://www.cs.wisc.edu/condor/dagman/

[12] Droegemeier, K.K., V. Chandrasekar, R. Clark, D. Gannon, S. Graves, E. Joseph, M. Ramamurthy, R. Wilhelmson, K. Brewster, B. Domenico, T. Leyton, V. Morris, D. Murray, P. Plale, R. Ramachandran, D. Reed, J. Rushing, D. Weber, A. Wilson, M. Xue, and S. Yalda, 2004: Linked environments for atmospheric discovery (LEAD): A cyberinfrastructure for mesoscale meteorology research and education. Preprints, 20th. Conf. on Interactive Info. Processing Systems for Meteor, Oceanography, and Hydrology, Seattle, WA, Amer. Meteor. Soc.

[13] Foster, I., Kesselman, C., Nick, J., Tuecke, S., The Physiology of the Grid An Open Grid Services Architecture for Distributed Systems Integration, www.globus.org/research/papers/ogsa.pdf

[14] Anthony Mayer, Steve McGough, Nathalie Furmento, Jeremy Cohen, Murtaza Gulamali, Laurie Young, Ali Afzal, Steven Newhouse, John Darlington, ICENI: An Integrated Grid Middleware to support e-Science. Workshop on Component Models and Systems for Grid Applications, Saint-Malo, June 26, 2004.

[15] S. Krishnan and D. Gannon. XCAT3: A Framework for CCA Components as OGSA Services. In HIPS 2004, 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments. IEEE Computer Society Press, April 26, 2004.

[16] Open Grid Computing Environment (OGCE), http://www.ogce.org.

[17] J. Alameda, Orchestrating Applications on Remote Resources, a powerpoint presentation, www.grids-center.org/train/GRIDS-Alameda.ppt.

[18] GridLab, The GridSphere Portal http://www.gridsphere.org

[19] JSR-168 Portlet Specification. http://www.jcp.org/aboutJava/communityprocess/final/jsr168/

[20] Kropp, A., Leue, C., Thompson, R., Web Services for Remote Portlets (WSRP), OASIS http://www.oasis-open.org

[21] Matsuoka, et. al., Ninf: A Global Computing Infrastructure, http://ninf.apgrid.org/welcome.shtml

[22] Navotny, J. Developing grid portlets using the GridSphere portal framework, http://www-106.ibm.com/ developerworks/grid/library/gr-portlets/

[23] The Open Grid Services Infrastructure Working Group. http://www.gridforum.org/ogsi-wg, 2003.

[24] S.G. Parker and C.R. Johnson. SCIRun: A scientific programming environment for computational steering. In Supercomputing'95. IEEE Press, 1995.

[25] Matthew Shields, Ian Taylor, Programming Scientific and Distributed Workflow with Triana Services, GGF Workflow Workshop, to appear, Special Issue of Concurrency and Computation, 2005.

[26] Kepler: A System for Scientific Workflows, http://kepler.ecoinformatics.org/

[27] Gregor von Laszewski, Kaizar Amin, Mihael Hategan, Nestor J. Zaluzec, Shawn Hampton, Albert Rossi, GridAnt: A Client-Controllable GridWorkflow System, proceedings 37th Hawai'i International Conference on System Science, Jan 5-8, 2004

[28] The Weather Research and Forecasting (WRF) Model. http://www.wrf-model.org/