Building Grid Portals for e-Science: A Service Oriented Architecture

Dennis Gannon, Beth Plale, Marcus Christie, Yi Huang, Scott Jensen, Ning Liu, Suresh Marru, Sangmi Lee Pallickara, Srinath Perera, Satoshi Shirasuna, Yogesh Simmhan, Aleksander Slominski, Yiming Sun, Nithya Vijayakumar
School of Informatics
Indiana University, Bloomington, IN.

Abstract. Grids are built by communities who need a shared cyberinfrastructure to make progress on the critical problems they are currently confronting. A Grid portal is a conventional Web portal that sits on top of a rich collection of web services that allow a community of users access to shared data and application resources without exposing them to the details of Grid computing. In this chapter we describe a service-oriented architecture to support this type of portal.

Introduction

Grid technology has matured to the point where many different communities are actively engaged in building distributed information infrastructures to support discipline specific problem solving. Often called Grids, this distributed infrastructure is based on Web and Web service technology that brings tools and data to the users in ways that enable modalities of collaborative work not previously possible. The vanguard of these communities are focused on specific scientific or engineering disciplines such as weather prediction, geophysics, earthquake engineering, biology and high-energy physics, but Grids can also be used by any community that requires distributed collaboration and the need to share networked resources. For example, a collaboration may be as modest as a handful of people at remote locations that agree to share computers and databases to conduct a study and publish a result. Or a community may be a company that needs different divisions of the organization at remote locations to work together more closely by sharing access to a set of private services organized as a corporate Grid. A Grid portal is the user-facing access point to such a Grid. It is composed of the tools that the community needs to solve problems. It is usually designed so that users interact with it in terms of their domain discipline and the details of Grid services and distributed computing are hidden from view.

In this chapter we describe a Service Oriented Architecture to support Grid portals for these, what we call "eScience collaborations". We base this discussion on our experience with a five-year project [1] to build such a framework for a community of atmospheric scientists who are engaged in changing the paradigm of severe storm prediction from static "back room" forecasts to an interactive, adaptive data-driven enterprise.

Our concept of the software manifestation that constitutes a "portal" has evolved over the years and, as we look towards the future of Web technology, we can already

see signs that it will continue to change. The simplest vision of a "portal" is a desktop application that is an all-in-one appliance for doing domain science in a way that exploits remote data and compute resources.

The most common current model for a Grid portal is based on standard web portal technology such as used by banks, airlines and social networking: a specially designed web server that allows the user to login and see his or her private and group resources. The user accesses such a portal with a standard web browser. As we move to the future, new "Web 2.0" technologies will allow the web browser to be replaced by more interactive, automatically downloaded clients that provide a more seamless integration with the user's desktop.

1. The Components of a Grid Portal

There are five common components to most Grid portals.

1. Data search and discovery. The vast majority of scientific collaborations revolve around shared access to discipline specific data collections. Users want to be able to search for data as they would search for web pages using an Internet index. The difference here is that the data is described in terms of metadata and instead of keywords, the search may involve terms from a domain specific ontology and contain range values. The data may be generated by streams from instruments and consequently the query may be satisfied only by means of a monitoring agent.
2. Security. Communities like to protect group and individual data. Grid resource providers like to have an understanding of who is using their resources. This is a specific requirement of Gateway portals such as those that access the TeraGrid. A key feature that distinguishes a portal from a website is the fact that each user has an account. The Grid portal must authenticate each user and associate the appropriate user-specific authorization rights that grant that user access the back-end resource Grid.
3. User private data storage. Because each user must "login" and authenticate with the portal, the portal can provide the user with a personalized view of their data and resources. This is a common feature of all portals in the commercial sector. In the eScience domain private data can consists of a searchable index of experimental result and data gathered from the community resources.
4. Tools for designing and conducting computational experiments. Scientist need to be able to run analysis processes over collected data. Often these analysis processes are single computations and often they are complex composed scenarios of preprocessing, analysis, post processing and visualization. These experimental workflows are often repeated hundreds of times with slightly different parameter settings or input data. A critical feature of any eScience portal is the capability compose workflows, add new computational analysis programs to a catalog of workflow components and a simple way to run the workflows with the results automatically stored in the user's private data space.
5. Tracking data provenance. The key to the scientific method is repeatability of experiments. As we become more data and computationally oriented in our approach to science, it is important that we support ways to discover exactly how a data set was generated. What were the processes that were involved? What versions of the software were used? What is the quality of the input data? A good

eScience gateway should automatically support this data provenance tracking and management so that these questions can be asked.

There are many approaches to supporting this feature set. As previously noted, one obvious solution is to implement all the required capability in a single monolithic application. The application may be a single instance of a portal server, or it may be a desktop application. In fact, our earliest attempts to build science portals followed the monolithic approach and we discovered two major problems: scalability and extensibility. Scalability is a problem with a monolithic portal server. As soon as you have more than a few dozen users logged into a portal server that tries to manage internally all the features described above, it will run out of memory and crash.

To solve this problem, we learned to make the portal server as thin as possible. As illustrated in figure 1, one can build the architecture as a set of specialized services: a service (GFAC) for incorporating new analysis algorithms into the framework, a service for cataloging data products, a workflow manager, a personal metadata catalog, and a provenance service. All of this sits on top of an enterprise service bus that conveys messages between the various components.

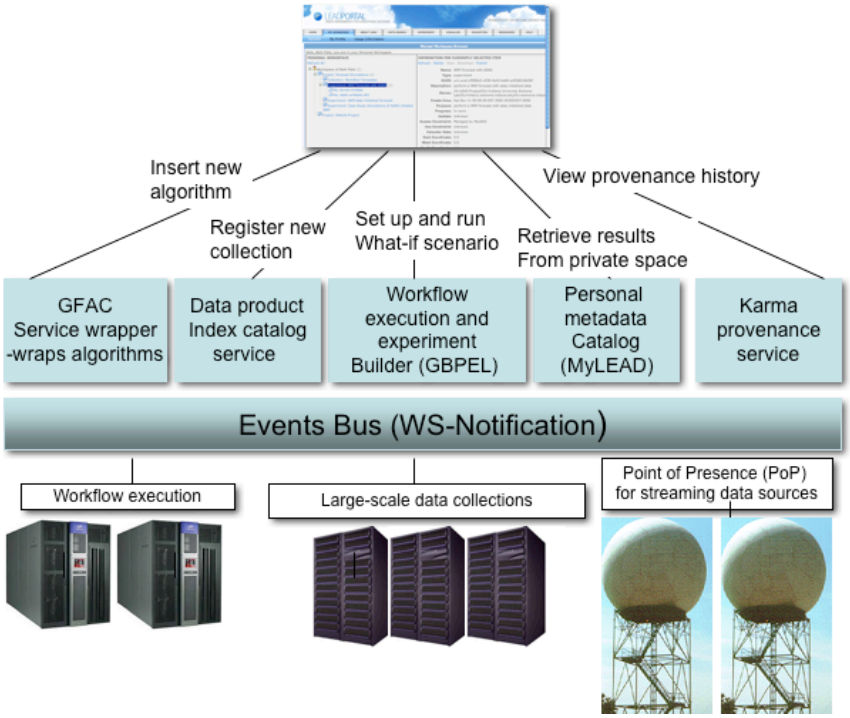

Figure 1. A distributed service-based portal server.

Extensibility means that the framework can easily incorporate new features and capabilities. This is not a strong suite for monolithic application portals unless they are designed with a strong software component model. However, even with a component architecture, the framework must know how to discover new components and load them automatically. This problem is also related to software maintenance. A change to

a client-side application requires that users update the application or periodically install a new version.

A Web portal gives greater flexibility in extending capabilities and doing maintenance. The user only needs to update the browser once or twice a year, while the back end portal server can be continuously updated.

1.1. The Portal Server.

Once the decision to use a web portal framework has been made, there are several issues that must be addressed. First, one must decide on the specific architecture to use. There are many Web portal tool frameworks available from open source and commercial sources. Many are based on a technology called the Java JSR-168 specification [2] in which the portal framework is a container for **portlets**. From the user's point of view a portlet is a windowpane in the browser whose contents defined by the portal manager. From the portal designers point of view, a portlet is a container for an application that manages a user interface. As illustrated in Figure 2, once a user logs into a portlet-based portal they see a pallet with tabs. Each tab, when "clicked" will bring up a page with one or more portlets. Users can add and remove portlets from the collection they see, but most do not bother to do so if the portal manager makes a reasonable selection as the default.

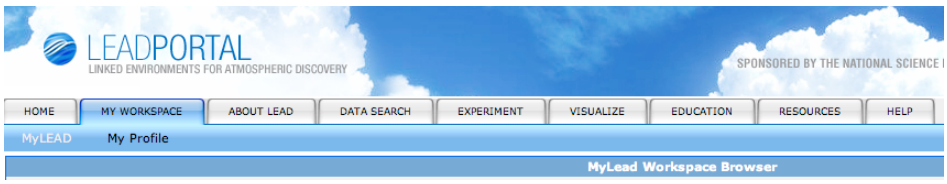

Figure 2. The collection of portlets in the LEAD portal are organized by tabs. Each major tab leads to a second level of portlets. In this case the user has selected "My Workspace" and then the user metadata catalog browser "MyLEAD".

While a portlet can contain a user interface to a substantial application running in the portal server, in our experience, the best design is for each portlet to interface to one of the core Web services that lies behind the portal server. The portlet's job is to provide a user interface for that service. When the user provides input through that interface, a Web service invocation is created by the portlet and sent to the corresponding service. The response is rendered in the user interface. In some cases the portlet is just an interface to a **mashup**, which is a graphical interface that is a combination of information from other services. Our data search interface is an example of one such mashup that we will describe later.

1.1.1. Portal and Grid Security

A major component of any Gateway Portal server is the management of security. Every off-the-shelf portal framework has a way to allow users to request portal accounts and it has the database support for maintaining consistent records of each user's portal configuration. However, the major challenge for the portal security is to connect this to the security of the back-end Grid. In the standard Globus Grid [3], users are authenticated with the back-end by means of a proxy certificate, which is a

temporary public-private key pair that can be used by other agents to act on the user's behalf with distributed Grid services and resources. Given a user's distinguished name and passphrase, a tool called MyProxy [4] can be invoked by the portal server to acquire a Grid proxy certificate for the user. Once the portal server has the proxy, it can pass it to the external web services to be used during interaction with the Grid resources. If each portal user also has a Grid account, the mapping of the user's portal identity to their Grid identity is relatively easy to manage by the portal server.

Unfortunately, most Gateway Portal users do not have a Grid account. They may not even know, or want to know, that the back-end Grid exists. In this case the solution we use in TeraGrid is to give the portal its own Grid identity and account. In this approach, all portal users are mapped to a single Grid account and it is up to the portal services to create an audit trail for each portal user, so the Grid managers can be satisfied that no misuse of the Grid resources occurred by a troublesome portal user. To accomplish this, the portal server must provide additional metadata that is propagated through the system that identifies each portal user and "proof" of his or her authorization to carry the requested Grid operations.

There is much more that can be said on the subject of security, but space does not allow us to go into more detail here.

1.2. A High Level View of the Service Organization

The primary persistent Web services in the LEAD gateway SOA are shown in Figure 3. The data search and discovery portlets in the portal server talk to the Data Catalog Service. As described in the following section, this service is an index of data that is known to the system. The user's personal data is cataloged in the MyLEAD service. MyLEAD is a metadata catalog. The large data files are stored on the back-end Grid resources under management of the Data Management Service. The myLEAD Agent manages the connection between the metadata and the data.

Workflow in this architecture is described in terms of dataflow graphs, were the nodes of the graph represent computations and the edges represent data dependencies. The actual computations are programs that are pre-installed and run on the back-end Grid computing resources. However, the workflow engine, which sequences the execution of each computational task, sees these computations as just more Web services. Unlike the other services described in this section, however, these application services are virtual in that they are created on-demand by an application factory and each application service instance controls the execution of a specific application on a specific computing resource. The application service instances are responsible for fetching the data needed for each invocation of the application, submitting the job to the compute engine, and monitoring the execution of the application. The pattern of behavior is simple. When a user creates or selects an experiment workflow template, the required input data is identified and then bound to create a concrete instance of the workflow. Some of the input data comes from user input and others come from a search of the Data Catalog or the user's MyLEAD space. When the execution begins, the workflow engine sends work requests for specific applications to the Application Factory. If a needed Application Service is not already instantiated, the Factory starts one and the request is sent to the service instance.

A central component of the system is an event notification bus, which is used to convey workflow status and control messages throughout the system. The application service instances generate "event notifications" that provide details about the data being

staged, the status of the execution of the application and the location of the final results. The MyLEAD agent listens to this message stream and logs the important events to the user's metadata catalog entry for the experiment. The workflow engine, provenance collection service and data management service all hear these notifications, which also contain valuable metadata about the intermediate and final data products. The Data Management service is responsible for migrating data from the Compute Engine to long-term storage. The provenance collection service records all of this information and organizes it so that data provenance queries and statistics are easily satisfied.

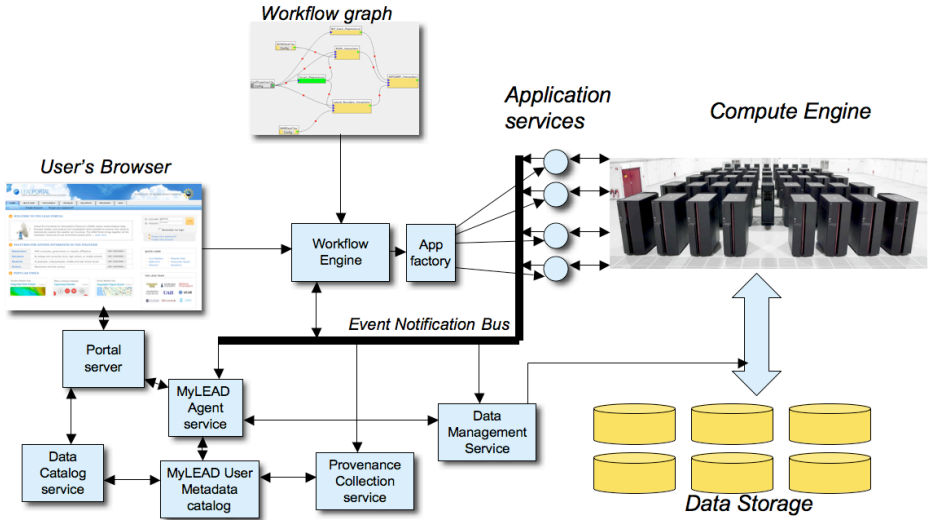

Figure 3. The LEAD persistent services.

In the following sections of this chapter we dig deeper into the role and design of each of these major services.

2. The Data Architecture

Data search, discovery and cataloging are central components of the portal. Most eScience domains have community data collections that serve as the raw information upon which knowledge is built. In the case of atmospheric science, this data comes from a wealth of remote sources including routine aviation reports, upper air soundings from balloon-born instruments, satellite image data, NEXRAD Doppler radar data and various forecast results. Much of this information is distributed through a community supported message-oriented middleware (MOM) layer called the Unidata Internet Data Distribution (IDD) system and catalogued with the Thematic Realtime Environmnetal Distributed Data Service (THREDDS) [5]. Just as a Web search engine crawls websites to build an index, the data subsystem for the portal infrastructure for LEAD crawls the IDD and THREDDS weather data systems and builds a unified index of these heterogeneous sources. However, crawling and indexing scientific data sources differs from standard Web search engine crawling, largely because of the kind of information crawled and what the user needs to do with the results. A Web search

engine need only build an index based on keywords scraped from Web pages. Scientific data is often in a binary representation that is described by metadata. Sometimes the data and metadata are stored separately. The information that must be crawled is often the metadata because that is where we find a semantic description of the contents of a data file. Typical metadata in atmospheric sciences includes the geospatial region it covers and time at which it was generated as well as the generation source (description of the instrument or model that generated it), the binary format of the data, etc.

Scientific metadata is organized according to an XML schema but, unfortunately, a scientific discipline often has multiple competing schemas for describing data. Consequently any data crawler must be able to understand all the pertinent schemas. In the LEAD project a global, extensible schema, called the LEAD Metadata Schema (LMS) [6], was designed so that any of the needed metadata described in the THREDDS catalogs and other important sources could be extracted and cataloged.

Having rich metadata enables far more meaningful data search capabilities than keyword search in a Web search index. In particular, one can build an interface that uses semantic knowledge about the attributes of the data schema. Such an interface can enable much deeper and more meaningful queries. For example, the LEAD data search portlet is shown in Figure 4.

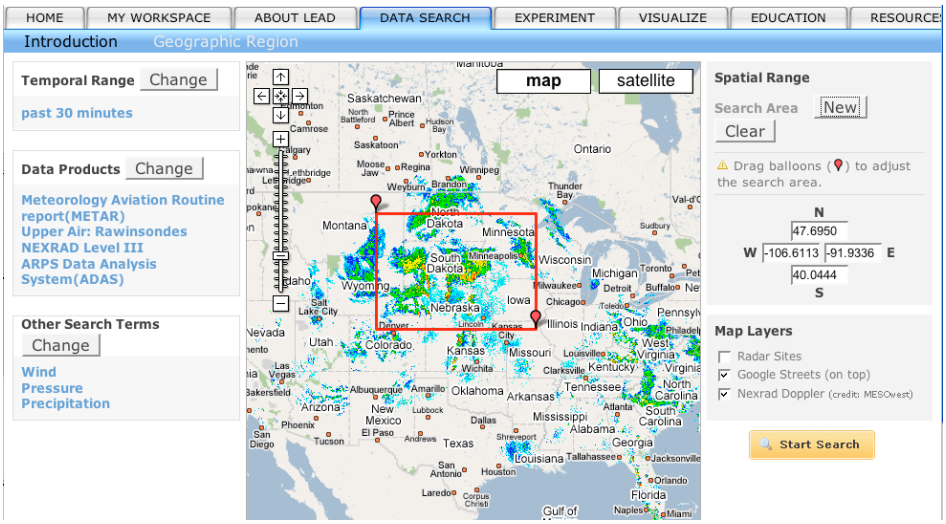

Figure 4. The LEAD data search mashup portlet.

In the figure, the user has constructed a query requesting data produced in the last 30 minutes (upper left) from the Meteorology Aviation report (METAR), the upper air balloon instruments, NEXRAD Level III and the ARPS Data Analysis system data sources. It is further refined to be limited to data about wind, pressure, and precipitation, and to products that contain data about the US prairie states. As can be seen in Figure 4, the current weather radar indicates that stormy weather is currently developing in the US prairie state region. Hitting the "Start Search" button returns a complete list of all the data that matches the query. For each item in the returned list, the user has the option of downloading it to the desktop, viewing it in a graphics package or saving it to their MyLEAD space for further analysis.

MyLEAD manages the user's personal workspace experience [7,8]. From the user's point of view, it is a browser, as shown in Figure 5, organized as a hierarchical set of folders. For the atmospheric science domain, the top level entities in the workspace are projects. Projects have members called "experiments", and project-wide materials stored to a "collection". An experiment represents a single execution of a analysis workflow. As shown in the illustration, selecting an experiment name in the left hand pane brings up a history of the events that took place during the execution of the experiment. Opening the experiment folder brings up all the metadata records for the input data, intermediate data products and results as well as error reports.

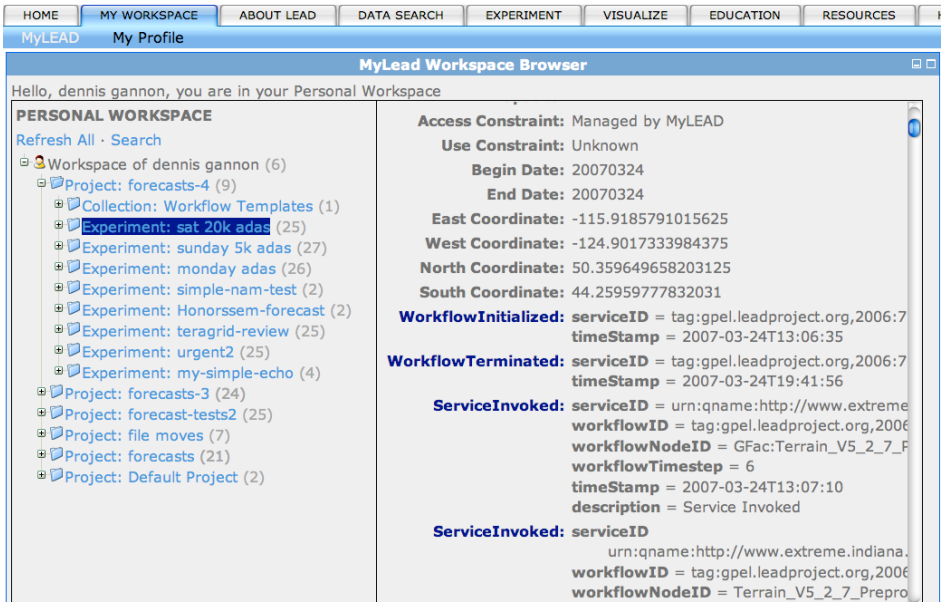

Figure 5. The MyLEAD personal workspace.

Because the MyLEAD workspace conforms to the LEAD metadata schema, it can provide the user a familiar yet highly sophisticated search capability. This search interface allows the user to search over files, experiments and projects, for parents and children, and for any combination of attributes in the metadata schema. For example, one can search for all experiments run on or after April 1, 2007 that used a 2km grid spacing in the simulation and also encountered a fault during execution. This is an extremely powerful feature that is not possible without rich semantic information provided by the metadata schema and the workspace notions of context and organization.

2.1. Data Management.

Data transfer, replica management, and naming are related functionalities required of a storage service in a cyberinfrastructure. While third-party data transfer allows clients to move data from one host to another without having to itself monitor the transfer, replica management allows optimal selection of replicas when transferring the data. Naming provides location transparency by using logical identifiers for the files and mapping them to the physical locations of the replicas Data movement, replication,

and naming is provided by the Globus Toolkit DRS, and by the simpler Data Movement and Naming service that was inspired by DRS. The Data Movement and Naming (DaMN) service, for instance, provides support for data transfers through simple GET and PUT service operations. The GET operation to transfer data out of a data repository takes a logical name for the source data and a logical or physical destination where the data needs to be transferred to. As a precursor to the transfer, DaMN uses the Replica Location Service (RLS) [9] to resolve the logical data name to the physical locations of the data products, optionally using algorithms for optimal replica selection. DaMN separately maintains a list of logical myLEAD repository locations to use as destination locations and resolves them to pre-configured physical storage locations. The capabilities of Reliable File Transfer (RFT) [10], based on GridFTP, is leveraged to effect third-party data transfers between the resolved source physical location and the destination physical location in a secure and reliable manner. After the transfer is complete, the new replica's location is registered with RLS, ensuring accurate tracking of replicas. The DaMN service provides polling, callback, and notifications as means of notifying the client of the successful data transfer and replica registration.

2.2. Data Provenance: The key to the modern scientific method.

Provenance is a form of metadata that describes the derivation history of data products derived from data transformation pipelines [11]. The Karma provenance framework [12] collects runtime information from scientific workflows and the services that constitute them. Karma defines an information model and messages that are generated by instrumented services and workflows at control transfer boundaries and data creation and usage points in the workflow execution. These messages are pushed from the instrumentation points to the Karma service through the portal's publish-subscribe system. From the information it receives, the provenance service can construct a global causal graph of the data derivation trail that can span across individual workflows when data is shared between them. Service APIs to query for different forms of provenance graphs [13] are exposed and graphical clients that can view complete provenance graphs or monitor the realtime provenance activities are available.

The provenance system supports queries of the form "Find the process that led to Atlas X Graphic, excluding everything prior to the averaging of images with softmean" or "Find all invocations of procedure align warp using a twelfth order nonlinear 1365 parameter model that ran on a Monday" are possible.

3. Turning Scientific Applications into Web Services

The concept of application-as-a-service is one that has gained considerable interest in the commercial software world. In this model a service provider is responsible for maintaining, updating and executing the application on-demand for clients. The client need only access the application interface through a Web service. However, this has not been widely used in the eScience domain until recently. In the LEAD project we have number of stable "community codes" which can be combined in various ways to do data analysis or to run complex weather forecast scenarios. The problem is that these are Fortran codes that are executed as command-line programs. To make them

composable, we turned them into virtual Web services. The services are virtual for a simple reason. If we had a persistent Web service for each science application on each supercomputer, we would have dozens of services to keep alive. To solve this problem we have a single persistent factory service capable of creating an instance of a needed application service on-demand. The specific application service instance need only persist as long as it has work to do. Because the process of creating a virtual service from a command-line program is so simple, we have automated it as a service available through the portal known as the Generic Application Factory GFAC [14].

The portal guides the user through the creation of two documents.

1. The service registration document contains the service name, the executable path for the application, the input parameters names and types and types, the output parameter names and types and details about the execution parameters such as the number of processors to use when running the application.
2. A host description document, which contains the details about the host where the GFAC service has been deployed and the application services will run. (This document is only needed if the host is not previously known to the portal. In the case of a Grid deployment, the host on which the service instance is running need not be the same as the host upon which the application runs. Globus GRAM is used to remotely invoke the application.)

With this information, the GFAC service generates a standard Web Service Description Language (WDL) document and can instantiate the service on-demand.

When an application service is invoked with a given set of input parameters, it runs the application with those input parameters (possibly on a cluster of resources), monitors the application and returns the results to the user. For debugging GFAC also provides a generic web service client that allows users to securely access any application service created by the toolkit from the convenience of a portal. When a user accesses an application service, the user is presented with a graphical user interface (GUI) to that service. The GUI contains a list of operations that the user is allowed to invoke on that service. After choosing an operation, the user is presented with a GUI for that operation, which allows the user to specify all the input parameters to that operation. The user can then invoke the operation on the service and get the output results.

4. Composing service: Workflows and Exploration Patterns.

Scientists conduct experiments and the portal is designed to support that. As we have seen, the MyLEAD personal metadata catalog is organized around this concept. We have identified two ways users may wish to conduct computational experiments. The first of these we refer to as task-driven, in which the user wants to run a particular type of data analysis or simulation and then gathers the needed data, and the other is data-driven, where the user wants to search and discover data and then find the tools that can be applied to it. In this chapter we address the former model, but the later is one we are currently investigating.

The way an experiment is initiated in the portal is through the Experiment Builder. Based on the idea of a software “wizard”, a program which leads the user through the steps for installing or using software, the experiment builder guides the user through

the setup and execution of an analysis workflow. As illustrated in Figure 6, the first thing the experiment builder asks the user is to identify the project that the experiment belongs to, the name of the experiment and a brief description.

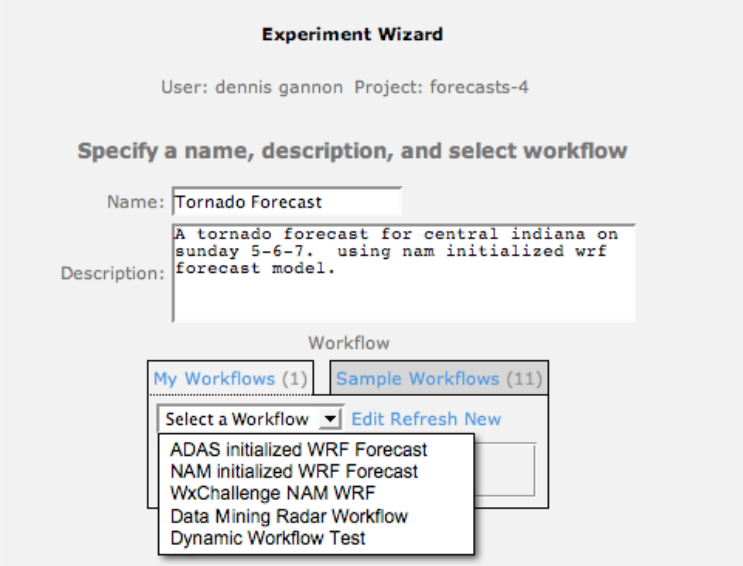

Figure 6. The Experiment Builder Wizard.

Next the user selects a workflow from a list of predefined templates. The user may edit the template or create a new template using the workflow graphical editor. The workflows are represented as a combination of dataflow and control flow. Most scientists are comfortable with the concept of dataflow represented as a graph. Each node of the graph represents an application service with multiple inputs and one or more outputs. The inputs and outputs may be as simple as a numerical or string value, or as complex as a descriptor of a set of files, which must be fetched and staged on the compute host that is running the application. (This staging is done automatically by the application service instance.) An example is illustrated in Figure 7 from the LEAD project. In this case the application service is a data interpolator that produces lateral boundary conditions for a forecast simulation. It take three input file descriptors. One describes the terrain of the forecast domain. The second is a file containing a set of

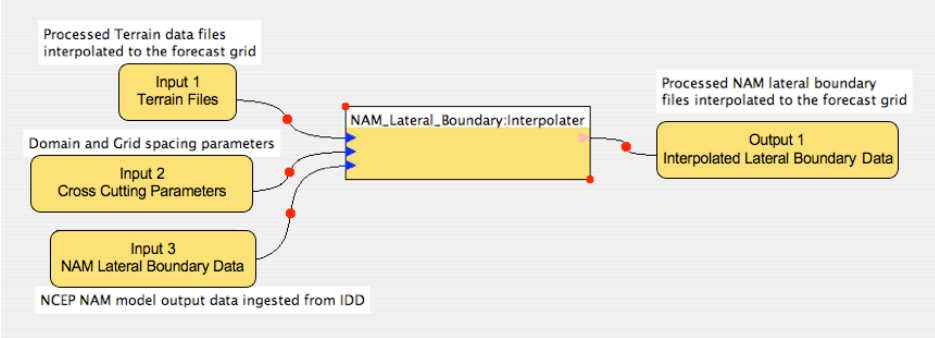

Figure 7. An application service from a weather forecast workflow. The service is an interpolator for the lateral boundary conditions for the forecast simulation domain.

parameters that describe the domain and the mesh spacing for the computation. The third is data that comes from the National Centers for Environmental Prediction (NCEP) North American Mesoscale (NAM) forecast. The output is an interpolated lateral boundary condition that is used as part of a storm forecast.

To compose a complete workflow, the portal user can invoke a tool called XBaya that provides a graphical interface to build the workflow. Shown in Figure 8, XBaya is a conventional "drop and drag" graph composer. The list of available workflow components and application services are shown on the left. Each of these components can be placed on the composition pallet to the right and wired into the workflow, which is shown add a left-to-right directed acyclic graph.

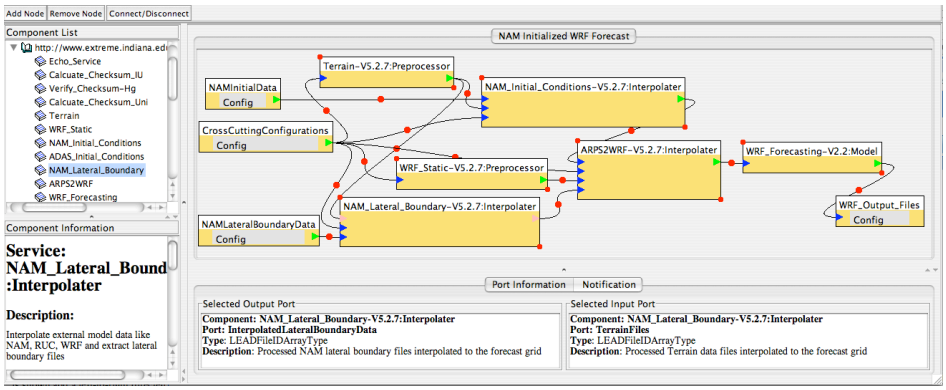

Figure 8. The XBaya workflow composer and monitoring tool.

The workflow shown in Figure 8 is a LEAD project storm forecast simulation that takes inputs consisting of a set of configuration property files that initialize parameters for different parts of the weather simulation and a set of North American Model (NAM) data files that define the initial conditions of the forecast.

XBaya is a compiler which translates this graphical description into a lower level workflow language. Because XBaya is so high level it is possible to map the graphical workflow description into many different workflow language and two are supported now. For long running workflows, BPEL [15] is the most completely supported Web service workflow language. If the workflow is short running, then the user may wish to

execute the workflow directly from their desktop machine. In this case, XBaya can generate the Java dialect of Python, called Jython.

When the user builds a workflow with XBaya, he or she is actually building an abstract workflow template. What is missing is the specific input data and the specific instances of the application web services. That binding is left to execution time for two reasons. First, in the case of workflows that must consume data that is timely (such as weather data), one does not want to bind the data until the workflow is almost ready to run. Second, because the applications runs on many different host computers, it is best to select the application services instance that corresponds to a host that has the lightest load or greatest capacity. In this case, the selection may be deferred until just prior to invoking the abstract service. The way this is done is the actual service request is sent to the application factory service, which through an intermediate service, makes a call-out to a resource broker that determines the best choice for a selected instance of the application service. When this best service is known the factory forwards the Web service there and the results return to the workflow engine.

4.1. The Workflow Engine.

There are a large number of possible workflow systems that can be and are being used in eScience projects and portals. In the case of Web service based frameworks, we have found that the BPEL standard is well suited to our needs for several reasons. BPEL is designed for long running workflows that are common in eScience applications that must use services that are widely distributed over a Grid. We have implemented a version of the standard we call GPEL [16] with several special features that are significant for applications like LEAD. Specifically, we require that

1. A workflow may run for hours or, in the case where the workflow is waiting for significant external events such as storm warnings to trigger actions, the workflow may run for days or weeks. Consequently the workflow instance state must be saved in stable storage.
2. The workflow must be able to respond to dynamically changing conditions including crashes of parts of the Grid or massive new demand created by a scenario, such as a major storm, or even the desire of the scientist to interrupt it and modify it or redirect it.
3. In case additional requirements are needed by the scientist, it may be necessary for a workflow to be restarted or cloned from an intermediate state.

In GPEL the state of every workflow instance is represented by an XML document stored in a database. A change of state, such as advancing the workflow to a new state because of the completion of some action, is just a transformation of that document. Consequently the three conditions listed above are easily met. One capability that we did not plan on was the ability to restart workflows that failed because of back-end Grid failures. Because the workflow state and all intermediate data products are saved by the data system, it proved to be a simple task to retrieve the state of a failed workflow and "resurrect" it using different resources.

One question that remained was scalability. While in production the system was able to manage over 1200 workflows a month and in the same time the data system managed over 2.6 terabytes of data product. A more detailed study of the performance of the entire system will be conducted over the year ahead.

4.2. The Pub-Sub Service Bus.

Web services are designed to receive messages and act upon them in the most stateless manner possible. However, if every service must rely upon other services to directly engage them for every piece of vital information, the architecture becomes very brittle and will not scale. The standard approach to solving this problem is to allow services to respond to events that represent changes to the state of workflow instances or Grid deployment status or external demands on the system. To address this problem most scalable distributed systems rely on a "publish-subscribe" event backbone. The idea is very simple. Persistent services such as the data subsystem or the providence tracking system that must know when certain thing happen, such as when a file has been moved, or an application service has terminated or encountered an error can do so by creating a subscription to an event service that will notify them by a simple message when something important happens. A subscription consists of a message to the event service indicating that a particular "listener" service is interested in receiving an invocation anytime a message is "published" on a particular topic. A topic may be any event associated with a specific user's experiment workflow state transition or an application service status message. Or it may be a message indicating that the Grid has undergone a subsystem failure. It may also be a notification that a severe storm has been detected.

There are two main Web service standard for pub-sub event systems: Ws-Eventing and Ws-Notification. The underlying Grid infrastructure, if based on Globus, uses WS-Notification, but a more widely adopted standard is WS-Eventing. Our approach has been to mediate between these two systems. Our system is based on a tool we call WS-messenger [17] that supports both standards and is an internet-scale reliable publish-subscribe system. This system allows both services and user client programs to always maintain a current picture of the state of each workflow. For example, the data providence and MyLead broker services rely on these notifications. XBaya, is able to track the state of a user's workflow by subscribing to the event stream. As the state of the workflow changes the graphical view of the workflow can show the user the current status. This has proved to be an invaluable in debugging and the users find the feedback about experiment status to be extremely helpful. The system has proven to be extremely reliable, but a quantitative study has not yet been completed.

5. Responding to the external world.

As described above, an event system is essential for monitoring the state of the service architecture and the progress of workflows. However, the external world generates events that are of great interest to scientists.

Atmospheric scientists have since the beginning of the LEAD project wanted the grid portal to give them a way to launch a regional weather forecast model in response to developing severe storms. In order to provide this, the portal has been extended with support for ingesting model- or radar-generated data in real time, for mining and filtering the streaming data using continuous query processing to filter and aggregate data [18] and through software classification algorithms that can continuously mine the results [19]. When the classification algorithms detect severe storm signatures exceeding a threshold, a notification message is sent.

As an example, suppose a user wishes to keep an eye on the storm front moving into Chicago later that evening. He/she logs into the portal, and configures an agent to observe NEXRAD Level II radar data streams for the next 4 hours looking for severe weather developing over the Chicago region. The request is in the form of a continuous query that executes the mining algorithm repeatedly. The query execution service will subscribe on-the-fly to streams of observational data. The events arrive as bzipped binary data chunks and are broken open to extract metadata. The resulting XML activities are subject to filtering and aggregation and the zipped product passed to the classification tool. Data mining will result in a response trigger, such as ``*concentration of high reflectivity found centered at lat=x, lon=y*'', that is sent to the workflow engine using the WS-Eventing notification system. WS-Eventing uses an internal event channel for the transfer of data streams.

6. Related Work.

The architecture we describe here was not created in a vacuum. There has been a community engaged in the design of Grid portals for the last five years and a substantial number of contributions have been made. The have been a long line of Grid portals aimed at managing the basic user interaction with a Grid including GridPort [20] and GENIUS [21]. The Gridsphere project [22] was an early pioneer that moved to bring the JSR-168 and the Gridsphere portal framework has been the most widely used of the portlet containers for eScience applications including the one described here.

There have been dozens of eScience Grid portals built for different communities and scientific disciplines. In the domain of biology there are several including the RENCI BioPortal [23]. The GEON Portal [24] is a gateway to geosciences data and applications. The BIRN Portal [25, 26] is the leading eScience gateway for biomedical imaging. The Cactus Portal [27] is a front end for Grid tools for the study of black holes. The area of earthquake science is covered by ServoGrid Portal [28] and the NEES Grid Portal [29] was built for earthquake engineers to share access to specialized instruments. NanoHub [30] is a Grid portal for nanotechnology and nanoscience and it is the most heavily used and successful of all Grid portals.

The National Science Foundation TeraGrid Project is now the leading organization in eScience Grid Portals. Their TeraGrid Gateways program [31] now has over 20 Grid portals for various scientific disciplines that are accessing the compute and data resources of TeraGrid.

7. Acknowledgements

The work described here on the LEAD portal has had the benefit of substantial design, evaluation, testing and encouragement of many people. Notable contributions came from Kelvin Droegemeier and Dan Weber and the rest of the team from the Center for Analysis and Prediction of Storms at the University of Okahoma and the other LEAD project principal investigators including Dan Reed, Sara Graves, Mohan Ramamurthy, Sepi Yalda, Richard Clark, Everette Joseph and Bob Wilhelmson. Katherine Lawrence and Il-Hwan Kim provided invaluable user interface evaluation and design. The testing of the portal with real users would not have been possible without the help of Richard Clark and Tom Baltzer. The LEAD metadata schema was created by a team that

included Rahul Ramachandran and Anne Wilson. Gopi Kadaswamy was critical in the GFAC design and current work on fault tolerance with Dan Reed. This work was supported by NSF awards ATM 0548692, ATM 0648857, OCI 0503697, and CNS 0330613.

References

[1] B. Plale, D. Gannon, J. Brotzge, K. Droegemeier, J. Kurose, D.McLaughlin, R. Wilhelmson, S. Graves, M. Ramamurthy, R. Clark, S. Yalda, D. Reed, E. Joseph, V. Chandrasekar: CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting. *IEEE Computer* 39(11): 56-64 (2006)

[2] Java Community Process, JSR 168: Portlet Specification, http://jcp.org/en/jsr/detail?id=168

[3] The Globus Alliance. www.globus.org/

[4] J. Novotny, S. Tuecke, V. Welch, An Online Credential Repository for the Grid: MyProxy, *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 '01)* p. 104, 2001.

[5] Unidata http://www.unidata.ucar.edu

[6] B. Plale, R. Ramachandran, and S. Tanner, Data Management Support for Adaptive Analysis and Prediction of the Atmosphere in LEAD, *22nd Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology (IIPS)*, January 2006.

[7] Y. Sun, S. Jensen, S. Lee Pallickara, Personal Workspace for Large-scale Data-driven Computational Experimentation. *International Conference on Grid Computing (Grid'-6)*, 2006.

[8] B. Plale, D. Gannon, J. Alameda, B. Wilhelmson, S. Hampton, A. Rossi, and K. Droegemeier Active Management of Scientific Data *IEEE Internet Computing* special issue on Internet Access to Scientific Data, Vol. 9, No. 1, Jan/Feb 2005, pp. 27-34.

[9] Globus, Replica Location Service (RLS), http://www.globus.org/rls/

[10] Globus Toolkit version 4.0. Reliable File Transfer (RFT), http://www.globus.org/toolkit/docs/4.0/data/rft/

[11] Y. Simmhan, B. Plale, D. Gannon, A Survey of Data Provenance in e-Science. *SIGMOD Record* (Special Section on Scientific Workflows), Vol. 34, No. 3, pp. 31-36, 2005

[12] Y. Simmhan, B. Plale, D. Gannon, A Framework for Collecting Provenance in Data-Centric Scientific Workflows. *International Conference on Web Services (ICWS)*, 2006

[13] Y. Simmhan, B. Plale, D. Gannon,. Querying capabilities of the karma provenance framework. *Concurrency and Computation: Practice and Experience*, 2007.

[14] G. Kandaswamy, L. Fang, Y. Huang, S. Shirasuna, S. Marru, and D. Gannon. Building Web Services for Scientific Grid Applications. *IBM Journal of Research and Development*, 50(2/3):249-260, 2006

[15] Business Process Execution Language Specification Version 1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[16] A. Slominski, Adapting BPEL to Scientific Workflows, Chapter 14 in *Workflows for e-Science*, Tayler, Deelman, Gannon, Shields, eds. Springer, 2007.

[17] Y. Huang, D. Gannon, A Flexible and Efficient Approach to Reconcile Different Web Services-based Event Notification Specifications, *International Conference on Web Services (ICWS)*, Chicago, 2006. pp. 735-742.

[18] Y. Liu, N. Vijayakumar, and B. Plale, Stream Processing in Data-driven Computational Science *7th IEEE/ACM International Conference on Grid Computing (Grid'06)*, Barcelona, September 2006.

[19] T. Hinke, J. Rushing, H. Ranganath and S. Graves.. Target-Independent Mining for Scientific Data. in *Proceedings: The Third International Conference of Knowledge Discovery & Data Mining*. 1997.

[20] M. Dahan, et. al., Build grid portals with Grid Portal Toolkit 3, 19 Oct. 2004. http://www-128.ibm.com/developerworks/grid/library/gr-gridport/

[21] A. Andronicoa, R. Barbera, A. Falzonec, G. Lo Rea, A. Pulvirentia, and A. Rodolicoc, GENIUS: a web portal for the grid, Nuclear Instruments and Methods in *Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* Volume 502, Issues 2-3, 21 April 2003, Pages 433-436.

[22] J. Novotny, M. Russell, O. Wehrens, *Concurrency and Computation: Practice and Experience*. Volume 16, Issue 5 , Pages 503 – 513, Mar 2004.

[23] A. Blatecky, K. Gamiel, L. Ramakrishnan, D. Reed, and M. Reed. Building the Bioscience Gateway. In *Science Gateways: Common Community Interfaces to Grid Resources* Workshop at Global Grid Forum 14 (GGF14), June 2005.

[24] U. Nambiar, B. Ludaescher, K. Lin, C. Baru, The GEON portal: accelerating knowledge discovery in the geosciences, Workshop On Web Information And Data Management archive. *Proceedings of the eighth ACM international workshop on Web information and data management*., 2006.

[25] A. Lin, et al., The Telescience Project: Applications to Middleware Interaction Components (2005) Proceedings of The 18th *IEEE International Symposium on Computer-Based Medical Systems*, p. 543 -548.

[26] S. Peltier, A. Lin, D. Lee, S. Mock, S. Lamont, T. Molina, M. Wong, M. Martone, M. Ellisman, The Telescience Portal for Advanced Tomography Applications. *Journal of Parallel and Distributed Applications*, Special Edition on Computational Grids 63(5): 539 – 550. 2003.

[27] I. Kelley, O. Wehrens, M. Russell, J. Novotny. The Cactus Portal. In proceedings of *APAC 05: Advanced Computing, Grid Applications and eResearch*, (2005).

[28] M. Pierce, M. Aktas, G. Aydin, G. Fox, H. Gadgil, Simulating Earthquakes: the QuakeSim/SERVOGrid Project, Chapter 7 in *Grid Portals*, Gannon, Pierce, Plale eds., to appear 2008.

[29] C. Severance, T. Haupt, NEES - Network for Earthquake Engineering and Simulation, Chapter 2 in *Grid Portals*, Gannon, Pierce, Plale, eds. 2008.

[30] M. McLennan, R. Kennell, D. Ebert, G. Klimeck, W. Qiao, Hub-based Simulation and Graphics Hardware Accelerated Visualization for Nanotechnology Applications, IEEE Transactions on Visualization and Computer Graphics archive Volume 12 , Issue 5 (September 2006) pp. 1061-1068, 2006.

[31] TeraGrid. http://www.teragrid.org/programs/sci_gateways/