

# GridLab: Enabling Applications on the Grid

Gabrielle Allen<sup>†</sup>, Dave Angulo<sup>§</sup>, Tom Goodale<sup>†</sup>, André Merzky<sup>†</sup>,  
Jarek Nabrzyski<sup>\*</sup>, Juliusz Pukacki<sup>\*</sup>, Michael Russell<sup>+</sup>,  
Thomas Radke<sup>†</sup>, Ed Seidel<sup>†</sup>, John Shalf<sup>\*\*</sup>, Ian Taylor<sup>\*\*\*</sup>

<sup>†</sup>Albert Einstein Institute Golm (AEI/MPG)

<sup>§</sup>Argonne National Lab (ANL)

<sup>\*\*\*</sup>Cardiff University

<sup>\*\*</sup>Lawrence Berkeley National Laboratory (LBNL)

<sup>\*</sup>Poznań Supercomputing and Networking Center (PSNC)

<sup>‡</sup>Zuse Institute Berlin (ZIB)

March 10, 2002

## Abstract

We present an overview of the architecture of the GridLab project, a largescale, EU-funded Grid project spanning over a dozen groups in Europe and the US. The project aims to provide fundamentally new capabilities for applications to exploit the power of the Grid computing. We describe the architecture on a very general level, focused on the main project objectives and user application scenarios.

## 1 Introduction

Computational Grids are becoming increasingly common, promising ultimately to be ubiquitous and thereby change the way global resources are accessed and used. However, presently there is a dearth of real Grid users, in part because the whole concept is new, but also because there are few applications that can exploit Grid resources. Although some application developers are interested in writing Grid-enabled applications, there are few user level tools, while application developer tools are nonexistent and well understood Grid usage scenarios are lacking.

It is therefore imperative to attract real users into the Grid community (for example the Global Grid Forum (GGF)) and ultimately onto the Grid. The Applications Research Group (ARG) of the former European Grid Forum (EGrid) has been addressing questions about users requirements, problems, and usage scenarios for several years now. In November 2000 the Group established a pan-European testbed [?], based on the Globus Toolkit, for prototyping and experimenting with various application scenarios. These testbed experiences inspired some members of the research group to submit a successful proposal for an application oriented project, called *GridLab*, to the European Commission.

The primary aim of GridLab ([www.gridlab.org](http://www.gridlab.org)) is to provide users and application developers with a simple and robust environment enabling them to produce applications that can exploit the full power and possibilities of the Grid. The GridLab project brings together computer scientists with computational scientists from various application areas to design and implement a *Grid Application Toolkit (GAT)*, together with a set of Grid services, in a production grid environment. The GAT will provide functionality through a carefully constructed set of generic high-level APIs, through which an application will be able to call the grid services laying below. The project will demonstrate the benefits of the GAT by developing and implementing real application scenarios, illustrating wild, exciting, new uses of the Grid. We will make extensive use of specific application frameworks, Cactus [?] and Triana [?] as powerful and broad reaching real world application examples for developing GridLab, but the GAT will be useful for applications and users of all types. Our aim is to make Grid computing accessible for the widest possible spectrum of applications and users.

This paper describes the some of the motivation as well as initial ideas for the architecture of the GridLab project.

## 2 Example Grid Application Scenario

The advocates of Grid computing promise a world where large shared scientific research instruments, experimental data, numerical simulations, analysis tools, research and development, as well as people, are closely coordinated and integrated in “virtual organizations”. This integration will be accomplished through web-based portals, woven together into modular wide-area distributed applications. One hypothetical scenario in astrophysics described now illustrates such an integration. Although futuristic sounding, many individual components have already been prototyped and through the GridLab project we are striving to make such a scenario a common day occurrence.

Gravitational wave detectors will rely on results from large-scale simulations for understanding and interpreting the enormous amounts of experimental data they collect. The Grid infrastructure is used both to share these expensive and centralized resources among many scientists, as well as to integrate experimental data feeds with the simulation codes necessary to analyze them. For instance, the GEO600 detector in Hanover detects an event characteristic of a black hole or neutron star collision, supernova explosion, or some other cosmic catastrophe. Astronomers around the world are alerted and stand by ready to turn their telescopes to view the event before it fades, but the location of the event in the sky must first be found. This requires a time-critical analysis of a number of templates created from full-scale simulations.

In a research institute in Berlin, an astrophysicist accesses the GEO600 portal and, using the performance tool, finds the resources required for cross-correlating the raw data with the available templates. The brokering tool finds the fastest affordable machines around the world. Merely clicking to accept the portal’s choice initiates a complex process by which executables and data files are automatically moved to these machines by the scheduling and data management tools. Then the analysis starts.

Twenty minutes later, on his way home, the astrophysicist’s mobile phone receives an SMS message from the portal’s notification unit, informing him that more templates are required and must be generated by a full-scale numerical simulation. He immediately contacts an international collaboration of colleagues who are experts in such simulations. Using a code composition tool in their simulation portal, his colleagues assemble a simulation code with appropriate physics modules suggested by present analysis. The portal’s performance prediction tool indicates that the required simulation cannot be run on any single machine to which they have access. The brokering tool recommends that the simulation be run across two machines, one in the U.S. and the other in Germany, that are connected to form a large enough virtual supercomputer to accomplish the job within the required

time limit. The simulation begins, and after querying a Grid information server (GIS), decides by itself to spawn off a number of time-critical template generating routines, and to run asynchronously on various other machines around the world. An hour later, the network between the two machines degrades and the simulation again queries the GIS, this time deciding to migrate to a new machine in Japan while still maintaining connections to the various template generators at other sites. All the while, the international team of collaborators monitor the simulation's progress from their workstations or wireless devices from an airport (where several team members happen to be), visualizing the physics results as they are computed. The template data are assembled and sent the GEO600 experimenter in Germany for analysis. The entire process, which could not be performed on any single machine or at any supercomputing site available today, takes only a few hours.

### 3 The Gridlab Architecture and Implementation Issues

#### 3.1 Requirements for the Grid Application Toolkit

To support user scenarios like the example described above, GridLab must be designed in a flexible way, supporting without compromising possible future developments in addition to the existing Grid protocols and technologies which can be used today. Applications should be able to make use of the broad range of grid services, independently of their location and implementation technology, incorporating simplified use of grid resources, such as service discovery and access to OGSA [?] or Corba, as these are developed by the Grid community.

Key design requirements for the Grid Application Toolkit include (i) applications must have a flexible, robust and efficient interface to the Grid; (ii) the interface must allow applications to be developed irrespective of the actual state of deployment of Grid services; (iii) the toolkit should allow fast prototyping, testing and production use with current smaller virtual organizations, while scaling seamlessly to future larger scale resource pools [?]; (iv) the interface should consist of short, simple and clear APIs which are designed to meet application needs, but yet allow access to the full functionality of underlying services; (v) services should be provided by a dynamic plug-in architecture, with multiple service implementations able to provide the same functional need through a common API interface; (vi) decision making and fault tolerance should be present at all levels, with the user able to decide how much autonomy to delegate to lower levels; (vii) finally the GAT API must satisfy the obvious requirements such as being portable, threadsafe, and well documented.

#### 3.2 High Level View

To meet these requirements we plan to implement the GAT using a service-based architecture, with four base levels with clearly defined interfaces, as illustrated in Figure ??:

- *Application Layer* Applications, such as simulation codes (e.g. Cactus, Triana), portals, scripts, developed and run by users, use the GAT API as the primary (ideally only) way to access the lower level grid services and infrastructure.
- *GAT API Layer*, the gateway through which services and GAT enabled applications communicate.
- *Services Layer* containing the services which use underlying grid infrastructure to provide the applications needs as requested through the GAT API.
- *System Layer* containing basic grid infrastructure (e.g. Globus GRAM/MDS/GridFTP, Condor, .NET, RPC-services, EJB)

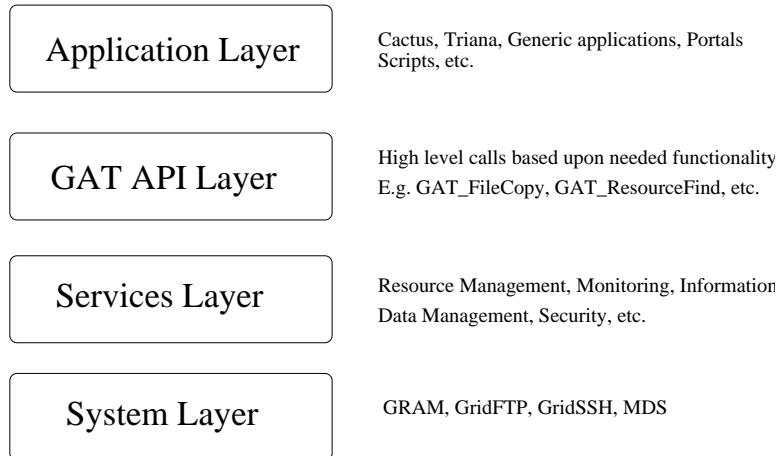


Figure 1: The service oriented architecture of the GridLab *Grid Application Toolkit* (GAT) consists of four clearly defined and hierarchical layers, described in Section ???. The GridLab project will develop software for the GAT API and Services Layers, which will be prototyped and tested in a range of applications.

### 3.3 A More Detailed View of the GridLab Layers

#### 3.3.1 The Application Layer

The highest layer in Figure ?? contains the applications, which could for example be user written codes (such as Cactus or Triana), portals, or scripts. Ideally these applications only communicate with the Grid environment through the GAT API, using interface calls which request application oriented functionality (e.g. move a file, spawn a subtask, perform a parameter survey, obtain performance metrics). GAT compliant applications (that is, accessing grid infrastructure only through the GAT API), will be highly portable and able to run in any environment, whether or not grid services are available, for example running even on a laptop without a network connection! Furthermore GAT compliant applications will have easy access to new Grid services and technologies, without further change to user written code, and be able to choose dynamically (or have chosen for them!) between different Grid services providing the same functionality at run time.

#### 3.3.2 The GAT API Layer

The GAT API layer consists of two components: the *GAT Engine* and a set of *GAT Adaptors*. The GAT Engine implements the GAT API, and for each GAT API call chooses an appropriate service and dispatches the task to this services adaptor. A *GAT Adaptor* provides the interface between the GAT Engine and one or more services. These adaptors translate the user requests into the appropriate interface syntax for accessing given services, and may also utilize grid service discovery mechanisms. The set of active adaptors can change dynamically, based upon the availability of services.

#### 3.3.3 The Services Layer

The Services Layer is made up of modules which provide various services. Services, here usually network-enabled, provide some capability, such as data management, brokering, monitoring, information or scheduling. The services are the core components providing applications with the requested capabilities for exploiting the dynamic Grid infrastructure.

The application gains access to the services in this layer through the GAT API, and these services need to provide the functionality requested by particular API calls. These services may, in turn, use other services, or functionality from the underlying System Layer. Note that while a service may be directly accessing a specific grid infrastructure in the System Layer, for example by using Globus APIs, the application is insulated from knowledge of specific infrastructure implementation by the GAT API.

The GridLab architecture is designed to seamlessly use all relevant services, including services developed in the workpackages of the GridLab project which are being specifically developed to provide application-driven functionality.

### 3.3.4 The System Layer

The System Layer is composed of what is now viewed as “standard” Grid middleware, such as the infrastructure already provided by Globus (e.g. GSI, third party file transfer, resource allocation, MDS). The GridLab project itself is not concerned with implementing any components of this layer, but only with interfacing to and utilizing these components and the resources they represent.

## 4 GAT Example: Resource Location

Here we illustrate an example application use of the GAT, where the application could be a simulation code or programming framework (such as Cactus or Triana) a portal, or some user-written script. The chosen example is the location of a compute resource, where the user simply wants to find a machine capable of delivering 1 GFlop of processing power. Note that this example uses initial ideas for the GAT architecture and implementation (concrete details will be provided at HPDC11), and is purposely simplistic, for example the same interface should be capable of providing multiple machines which can be connected together to provide the needed resources.

An application linked to the *Grid Application Toolkit* requests a resource by making a GAT API call, for example

```
GAT_ResourceFind (GAT_State, Requirements, Resource)
```

Here the `GAT_State` denotes an opaque object holding persistent GAT information (such as security credentials and the internal state of the GAT Engine). The `Requirements` parameter encodes the information that 1 GFlop is required (perhaps representing this information in some property table, for example a ClassAds object). Finally `Resource` is a parameter returned by the GAT API which holds information about the requested resource. This information could then be passed into a following GAT API call, for example to start a job on the machine.

The GAT Engine examines the resource location services of which it is aware. If it finds multiple services, it uses some mechanism to choose between them. For example, it could simply iterate over each service until a machine is found. Importantly, to provide robustness for users and flexible application development, a default set of GAT adaptors can be used which return defaults (here for example, `localhost`).

In our example case the GAT Engine finds three possible services for finding a machine:

- *A Known Resource Broker (RB)*: The GAT Engine calls the RB adaptor which knows how to contact the service (e.g. protocol, argument list, location and port). The adaptor translates the `GAT_ResourceFind` API call into the appropriate request for the RB.

The RB attempts to find the requested resource, and returns the result (for example a machine, or some failure message). The adaptor returns this information to the GAT Engine, which can either return the resource to the application, or can continue to the next service.

- *A Service Dynamically Discovered Through OGSA:* As above the GAT Engine calls the OGSA adaptor, which following the Open Grid Services Architecture (OGSA) [?] specification, queries a UDDI. A list of Grid Service Handles for resource discovery services is returned. The adaptor then queries a service, parses the returned WSDL documents and uses this information to call the service. In this way a dynamically discovered resource broker is contacted by the mechanism described above.
- *Simple Testing Broker:* For prototyping and testing applications, developers need to be able to easily customize the information returned by services. In this case the machine information is fixed, for example a list of resources in a local machine pool, and the GAT adaptor could simply cycle through them.

## 5 Conclusions

In this paper we have described the overall layered architecture of the GridLab project, which aims to provide application oriented Grid services for users and developers alike. These services will be made accessible to any applications running on the Grid through a Grid Application Toolkit (GAT), that will abstract various services needed by Grid applications. In this way applications can utilize service discovery, at runtime, making use of whatever services are available, including different implementations of the same service. This will enable users and application developers to easily develop and run powerful applications on the Grid, without having to know in advance what the runtime environment will provide. Such applications should then run on a laptop or an intercontinental Grid, taking advantage of whatever services are available (or not) at runtime. Although we will make extensive use of powerful application frameworks such as Cactus and Triana in the development of GridLab, the project is designed to enable *any* application not only to run on the Grid (or without a Grid), but to endow it with new capabilities uniquely possible on a Grid, such as those described above in our usage scenario.

The project also will develop various application driven services, such as resource brokering, monitoring, etc, that will be needed by the applications. The components of the GridLab project itself all sit on top of what we would call the “System Layer”, which consists of various Grid infrastructure components (e.g., provided by Globus). With such an architecture, we expect many new and powerful applications to be developed to exploit the Grids of today and tomorrow alike.

One important role of GridLab is the testing of novel Grid application scenarios, using real applications on real Grids. Hence, GridLab plans to set up a large testbed spanning Europe and also transatlantic sites. At the lowest level this testbed will exploit existing Grid middleware, namely Globus, and its services. Services in the Services Layer described above need to be able to interface to these core services. A major task is to adapt the testbed and the middleware services to new core technologies as they arise, such as services provided by other projects such as Data Management Services from the DataGrid project[?], or new frameworks emerging on the middleware level as OGSA[?].

## Acknowledgments

We are pleased to acknowledge support of the European Commission 5th Framework program, which is the primary source of funding for the GridLab project, but also the German DFN-Verein, Microsoft, the NSF ASC project (NSF-PHY9979985), and our local institutes for generous support for this work. We also thank Ewa Deelman, Thomas Dramlitsch, Ian Foster, Carl Kesselman, Jason Novotny, Gerd Lanfermann and various members of the Globus team for many discussions and support that have led to the current project.