Enabling Applications on the Grid – A GridLab Overview

Gabrielle Allen, Kelly Davis, Konstantinos N. Dolkas,
Nikolaos D. Doulamis, Tom Goodale, Thilo Kielmann[1], André Merzky,
Jarek Nabrzyski, Juliusz Pukacki, Thomas Radke, Michael Russell,
Ed Seidel, John Shalf, Ian Taylor

www.gridlab.org

Abstract

Grid technology is widely emerging. Still, there is an eminent shortage of real Grid users, mostly due to the lack of a "critical mass" of widely deployed and reliable higher-level Grid services, tailored to application needs. The GridLab project aims to provide fundamentally new capabilities for applications to exploit the power of Grid computing, thus bridging the gap between application needs and existing Grid middleware. We present an overview of GridLab, a large-scale, EU-funded Grid project spanning over a dozen groups in Europe and the US. We first outline our vision of Grid-empowered applications and then discuss GridLab's general architecture and its Grid Application Toolkit (GAT). We illustrate how applications can be Grid-enabled with the GAT and discuss GridLab's scheduler as an example of GAT services.

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 few applications have been written that can exploit Grid resources. Although some application developers are interested in writing Grid-enabled applications, there are few user-level tools, high level tools for application developers are nonexistent, and well-understood Grid usage scenarios are rarely available.

In order to catalyze Grid usage, 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 (APPS-RG) of the GGF (and formerly of the European Grid Forum, EGrid) has been addressing questions about user requirements, problems, and usage scenarios for several years now. In 2000, the group established a pan-European testbed [1], based on the Globus Toolkit, for prototyping and experimenting with various application scenarios. These testbed experiences gave inspiration for an application oriented project, called *GridLab*, funded by the European Commission.

The primary aim of GridLab 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 underlying Grid services. The project will demonstrate the benefits of the GAT by developing and implementing real application scenarios, illustrating compelling new uses of the Grid. We will make extensive use of specific application frameworks, namely Cactus [2, 3] and Triana [4], as powerful and broad reaching, real-world application examples for developing GridLab, but the GAT will be useful for many kinds of applications and users. Our aim is to make Grid computing accessible for the widest possible spectrum of applications and users.

[1]Corresponding author (kielmann@cs.vu.nl)

The development of the GAT is accompanied by the establishment of a pan-European testbed (including real production machines for the respective application user communities) and by the development of Grid services of varying complexity, tailored to the needs of our user community. These services are designed to complement and complete the existing Grid infrastructure, and to provide functionality needed by the GridLab applications in order to be usefully deployed in such environments.

In this paper, we first present our vision of Grid-empowered application scenarios. We then motivate and discuss the global architecture of the project. This is followed by a more detailed discussion of exemplary GridLab components: the GAT (the application interface to Grid environments), Triana (one of the GridLab applications), and the GridLab Scheduling Service (a GridLab service).

2 A Vision of Grid-Empowered Application Scenarios

The advocates of Grid computing promise a world where large, shared scientific research instruments, experimental data, numerical simulations, analysis tools, research and development platforms, as well as people, are closely coordinated and integrated in "virtual organizations". This integration will be fostered through web-based portals, woven together into modular wide-area distributed applications. One hypothetical scenario in astrophysics, described in the following, illustrates such an integration. Although sounding futuristic, many individual components have already been prototyped, and through the GridLab project we are striving to make such a scenario a common 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 expensive and centralized resources among many scientists, as well as to integrate experimental data sources with the simulation codes necessary to analyze them. For example, the GEO600 detector in Hanover detects an event characteristic of a black hole or neutron star collision, supernova explosion, or some other cosmic event. Astronomers around the world are alerted and stand by, ready to turn their telescopes to view the event before it fades. However, the location of the event in the sky must first be found. This requires a time-critical data analysis with 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, estimates 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 her way home, the astrophysicist's mobile phone receives an SMS message from the portal's notification unit, informing her that more templates are required and must be generated by a full-scale numerical simulation. She immediately contacts an international collaboration of colleagues who are experts in such simulations. Using a code composition tool in their simulation portal, her colleagues assemble a simulation code with appropriate physics modules suggested by the present analysis. The portal's performance prediction tool indicates that, due to memory constraints, 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. After querying a Grid information server (GIS), the simulation autonomously decides 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 to the GEO600 experimenter in Germany for analysis, which finally yields the likely source location for the gravitational wave signal. This triggers another Grid application which utilizes a different virtual organization and its infrastructure to direct the Hubble Space Telescope and various other available instruments toward this source location. 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 Requirements for a Grid Software Environment

The main goal of the GridLab Project is to provide a software environment for Grid-enabling scientific applications. It is our aim to provide an API through which applications access and use available resources. This API directly reflects application needs. Among the intended functionality is the exploration of available resources (CPU, storage, visualization, etc.); remote data access; application migration; etc. The API will be concentrated in the *Grid Application Toolkit* (GAT). The functionality behind the API will be provided by interchangeable capability providers, which may be GridLab services or third-party services.

In this section, we will briefly define important types of capability providers. We then summarize application requirements and general constraints on a software architecture for the Grid Application Toolkit, and its capability providers.

3.1 Terminology

This subsection reviews the terminology used throughout this paper. During the development of the GridLab architecture, we realized that the standard set of terms used in Grid Environments was insufficient to represent the whole range of our component types. Thus we tried to refine these definitions, hopefully without introducing incompatibilities to the original terms:

Capability Provider: A capability provider is an entity providing a specific capability. It is defined in terms of an interface used to invoke a capability, and the behavior expected in response to that invocation (i.e., capability provider = interface + behavior).

Service: "A service is a **network-enabled entity** that provides a specific capability. [...] A service is defined in terms of the protocol one uses to interact with it and the behavior expected in response to various protocol message exchanges (i.e., service = protocol + behavior)." [5]

Web Service: "The term 'web services' describes an important emerging distributed computing paradigm [with] focus on simple, Internet-based standards (e.g., eXtensible Markup Language: XML [...]) to address heterogeneous distributed computing. Web services define a technique for describing software components to be accessed; methods for accessing these components; and discovery methods that enable the identification of relevant service providers." [6]
service providers." [6]

Grid Service: "A Grid service is a web service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; the conventions address naming and upgrade-ability." [6] Grid services are defined by the emerging OGSA standard.

GridLab Service: A service provided by the GridLab project. Where possible these will be Grid (OGSA) services.

Third-party Service: A service provided outside the scope of GridLab, either from underlying Grid middleware, from legacy software, or from concurrent developments outside of the GridLab project.

Adaptor: The adaptor pattern provides programmers with interfaces. Adaptor components implementing an interface abstract other components, which can have a wide variety of interfaces. Adaptors are vital inside the GAT to allow interfacing applications to heterogeneous capability providers.

3.2 Requirements

The ultimate goal of the GridLab project is to provide application programmers and users with an environment that enables scenarios such as the one from Section 2. From such scenarios, the main application

requirements to the GridLab architecture can be drawn. A number of additional user requirements have been specified by the GridLab application groups. They mainly deal with the usability of the whole system and, in general terms, express their wish for some degree of involvement in the internals of the environment. It may not hold for all groups, but there are for sure users, but more importantly application programmers, who would not accept working with a complex system if that system presents itself as a black box *only*.

Other, more technical requirements are also to be applied to our project, originating from the type of environments our system wants to enable, and from general arguments about administration, security and such. This subsection lists these requirements, which are the base for the definition of the GridLab architecture, covered in Section 4.

(i) abstraction of the environment :
By definition, the ultimate requirement to the GAT is to provide an abstraction of the underlying Grid infrastructure, its services, and its communication layers.

(ii) adaptivity to the environment :
One of the most important requirements of the GridLab user community is that applications utilizing the GAT should be able to run in a wide variety of real world environments. These include the Grid environments of both present and future, disconnected environments (e.g. laptops, developer machines), and firewalled resources.

(iii) interchangeability of capability providers :
Application executions, whether inside a Grid installation or upon isolated, disconnected resources, should become alternate cases of a unified application, supporting execution upon whichever resources are available There should not be a separation between "normal" and a "Grid-enabled" versions of the application code. This requirement demands a single GAT-API with which applications can be developed. The capability providers relevant to the current configuration can then be instantiated at runtime.

(iv) complete control on all levels :
Our users, and even more our application developers, strongly request the ability to control the utilized environment to as much an extent as possible. This does not mean that the application programmer herself wants to take care of every detail of the environment, but rather enables the programmer to detect errors, to prototype, and to split complex operations into smaller, simpler pieces if necessary and so on.

(v) smart adaptivity on all levels :
Grid environments are ever changing, and very heterogeneous, which is a constant challenge to Grid middleware. Our solutions have to be able to adapt to such environments.

(vi) robustness and fail safety, error tracing facilities :
Complex systems as computational Grids and distributed software systems inherit multiple points of failure. The middleware has to be able to recover from failures gracefully, and to report failures. Application programmers as well as administrators need to be able to verify the systems functionality on all levels.

(vii) independence from communication/architecture models :
With the advent of the Open Grid Service Architecture (OGSA) [6], GridLab's architecture and services will revolve around OGSA's notion of "Grid services." For the time being, this seems the most widely accepted architecture and communication model for the Grid community. However, in order to enable support both for legacy services and future development, the GridLab architecture itself should be independent from any specific architecture/communication model (e.g. OGSA, Legion, CORBA, RMI, RPC etc.).

(viii) cleanly layered architecture :
To achieve a complete abstraction of the underlying (Grid) environment, a cleanly layered architecture seems appropriate.

(ix) security mechanisms on a global level :
In order to get accepted in a wider community and to reach production state, the general GridLab architecture must enforce a flexible but tight global security model.

(x) third-party services are to be easily incorporated :
The GridLab project will by no means be able to provide all types of services to its user community – but other groups and projects do and will continue to provide various Grid components useful to the users. Also, we hope that the lifetime of the GridLab architecture by far exceeds that of the project itself. For these reasons, the easy utilization of third party services is crucial to the success of our approach.

4 The GridLab Architecture

The GridLab project aims to bridge the gap between applications and Grid middleware by providing a software layer in between. We will now describe the architecture of this software layer, consisting of the Grid Application Toolkit (GAT) and the GridLab services.

4.1 Mandates versus Policies

In some respects, the above requirements and constraints partially contradict each other. In particular, the mandate to incorporate security mechanisms for all components at a global level (ix) is incompatible with support for third-party services (iii, x) and disconnected environments. To elaborate, it is difficult to enforce security policies for third-party services; furthermore, Grid security fundamentally depends on network access. For third-party services, this contradiction might be resolved by requiring accesses to any third-party service to be performed via a (secured) GridLab service. But this is not possible if the application is running in a minimalistic environment (ii). It is also impossible if the application deliberately chooses to contact a suspicious (e.g., legacy) component on its own behalf, as applications need full control on all levels (iv).

In fact, contradictions of this type are very common in Grid environments. Middleware developers want to have control over all aspects of the environment. This is especially true for security issues, but also for scheduling, network interfaces, communications, and so on. In turn, the end user desires the ability to run an application in any environment, benefiting from a Grid infrastructure if available, but also ignoring Grid resources if appropriate. In particular, our experience shows that the users must be able to deliberately ignore aspects of the environment without loosing the benefits of the environment at large. For many user communities, an "all–or–nothing" bargain is not acceptable.

In the GridLab project, we explicitly distinguish between abstract architecture concepts and specific realizations of these concepts. Our *"General Architecture"* reflects this effort. By design, our architecture allows the incorporation of global security (ix) and the encapsulation of the communication models (vii) by providing a layered (viii) framework for all its components. On the other hand, such specifications are based only on policies, not mandates. Consequently, developers and users are encouraged to utilize these features, but are also free to ignore these and to incorporate components incompatible with these policies where necessary (ii, iii, iv, vii).

As a result of these considerations, the general GridLab architecture consists of the overall design (Figure 1) and a set of implementation policies. This combination of design and policies is the result of intense and controversial discussions. This approach is considered a main characteristic of GridLab, attributable to the strong involvement of real application groups.

4.2 The GridLab Architecture

Global Design

As discussed above, the overall architecture diagram, presented in Figure 1, defines a cleanly layered environment providing an abstract view for the applications. The applications, located on the highest level of

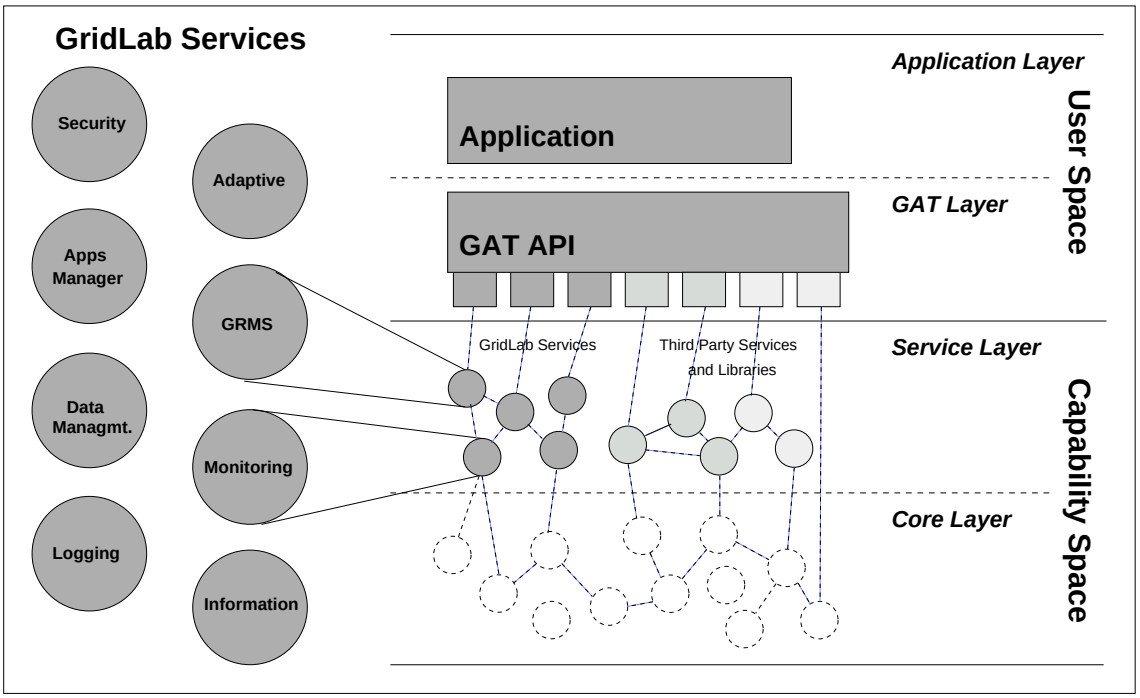

Figure 1: **The General GridLab Architecture:** The GAT will be designed to interact with all types of capability providers, via various communication channels. Security and uniformity for both types of components are recommended, not mandated (see text).

the user space, can access *all* capability providers they need via the Grid Application Toolkit (GAT) API. The GAT also resides in user space, providing interfaces to the capability providers in the capability space.

A detailed discussion of how the GAT is intended to provide a set of well defined capabilities by accessing the various capability providers is presented in Section 5. For the present discussion, only some features of the GAT are important: the ability to access various kinds of capability providers via adaptors specific to their communication mechanism, and the semantics of invoking specific capability providers.

As discussed above, the GridLab project distinguishes between GridLab services and third-party services (e.g. low-level Grid services like GIS or GRAM, system services, libraries). Following the requirement (iv), the GAT is able and allowed to access all types of capability providers, on all levels.

Implementation Policies

In addition to the presented diagram, a set of policies ensure that the outcome of the GridLab project itself consists of compatible, uniform, and consistent components. Deviations from these policies within the GridLab project require explicit justification. The implementation policies cover the following areas:

(1) Communication Model All capability providers designed and implemented in the scope and course of the GridLab project are, by default, required to be *Grid Services*. However, until OGSA implementations become widely available (in particular for C++), GridLab services are being implemented as *Web Services* and will be converted to Grid services as soon as possible.

(2) Capability Access The GAT is required to access capabilities via GridLab services, wherever possible and sensible. GridLab services are to be tailored to the needs of the GAT API. GridLab services will utilize lower-level services to implement their capabilities (like smart adaptivity, control, and fail safety).

(3) Security All components designed and implemented in the scope and course of the GridLab project honor the GridLab Security Infrastructure.

These policies support the development of a clearly layered, consistent design of the GridLab software. Also, the combination of a general architecuture with strict policies preserves the simplicity and flexibility of the framework itself, which is necessary for the overall success of both the GridLab approach and the GAT beyond the lifetime of the project itself.

The next sections describe various components of the presented architecture in more detail: the GAT itself (Section 5), one of the major Gridlab applications, Triana (Section 6), and a sample GridLab service, the scheduling service (Section 7).

5 The GAT: Design and Prototype

The Grid Application Toolkit (GAT), as the main deliverable of the GridLab project, shall provide an application programmer with a single interface to the (potentially Grid enabled) environment in which her program is running. As described, this is achieved by letting the GAT delegate all interaction to external capability providers, hence abstracting all external capabilities.

This section describes how we are designing the GAT to achieve this goal. We then discuss the capability registry, introduce several example calls from the GAT-API, and suggest how these may be utilized by application programmers.

5.1 GAT Design Objectives

The term *GAT* incorporates two distinct elements: the interface definition (the GAT-API) and the embodying software entity (the GAT implementation). Once the GAT-API is defined, any compliant implementation should be usable in the domain for which the GAT was designed. However, due to a number of constraints, we suspect the design of the GAT as a software entity may be restricted to the design we present here.

The constraints applying to the GAT design draw largely from the following goals:

(1) GAT should provide and abstraction of capability provider interfaces

(2) GAT should be flexibility

(3) GAT should be fail safe

The abstraction of interfaces is well supported by the adaptor pattern. "The Adaptor Pattern tells [one] how to convert the interface of a class into another interface clients expect. An adaptor lets classes work together that couldn't otherwise because of incompatible interfaces." [7]. In the GAT design we present, adaptors represent each capability by interfacing them to corresponding capability provider.

In order to provide flexibility and interchangeability of capability providers at runtime, the GAT must be able to choose from a pool of adaptors and invoke different adaptors as needed. For example, multiple adaptors may interface to various file transfer capability providers. Each of these adaptors will present the same interface to the GAT, and the GAT must be able to choose between them at runtime, and invoke its selection on the fly. In our design, this is achieved by introducing a *capability registry*. On initialisation, all adaptors register themselves with the capability registry. The GAT Engine (the core of the GAT) is then able to query the registry, select the appropriate adaptors for the requested capability, and invoke this capability by calling the corresponding adaptor.

If the invocation of an adaptor fails, the GAT will, unless instructed otherwise, invoke the next most suitable adaptor providing the required capability, returning failure to the user only in the case that all suitable adaptors fail. This can be used to provide a measure of fail-safety.

The GAT will offer an adaptor which provides a simple implementation of all capabilities, and this can be used to test or run the application as long as no non-local operations are requested, thus allowing rapid development and testing with the GAT in non-Grid environments.

As GAT-API calls may not always be mapped to a specific capability provided by supporting services, an adaptor may need to invoke a number of capability providers in order to implement the exact capability

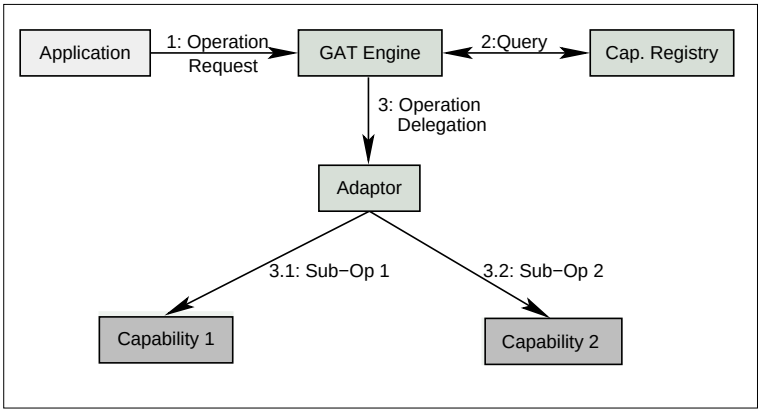

Figure 2: **GAT Adaptor Call:** The application invokes the GAT via some GAT-API call. The GAT engine, as core of the GAT, queries the capability registry for a suitable adaptor for that API call, and then invokes the selected adaptor. The adaptor interfaces to one or more capability providers in order to provide that functionality. On failure, the GAT Engine will the select the next most suitable adaptor from the registry.

requested by the GAT-API call. For example, an adaptor interfacing to a Grid scheduling service may need to interact with some form of Grid security service prior to the invocation of the scheduling service.

The design we have described, which is graphically represented in Figure 2, complies to the GridLab design requirements (Section 3) and to the requirements imposed by the goals of the GAT (as listed above). There are likely numerous designs for the functionality we have described, but we feel that our approach is both sufficiently simple and generic to serve various incarnations of GAT implementations for the foreseeable future.

5.2 GAT Capability Registry

This section describes the capability registry used by the GAT to map between GAT-API calls made by an application and specific functions provided by adaptors.

On initialisation each adaptor registers the capabilities it can provide access to with the registry, by providing an appropriate function to be invoked if the corresponding GAT-API call is made, and a description of the capability which may be used to distinguish between different adaptors providing the same capability, if more than one is present. This is shown in Figure 3.

When a GAT-API call is made, the capability registry is queried for a set of appropriate adaptor functions, and these are then invoked as described in the previous section. By default the capability registry will return all functions providing the required capability; however, the application may force a subset or a function from a specific adaptor by providing information which will then be matched against the descriptions registered with each function. This is shown in Figure 4.

5.3 GAT-API Definition

The GAT-API is designed to provide capabilities to the application programmer, allowing her to utilize the computing environment for the application. The design of the API is hence driven by our application developer and user groups, and not by the underlying Grid environment and the capabilities it actually offers.

As core GridLab project partners are rather active in the GGF Application and Testbed Working Group (APPS-RG), we hope to be able to specify an API with a much wider scope than the GridLab applications. We also hope to get significant feedback on this definition, and to refine and extend it through continuing discussions with other user groups.

To give a sense for the GAT-APIs appearance and use, we will list several preliminary GAT calls and

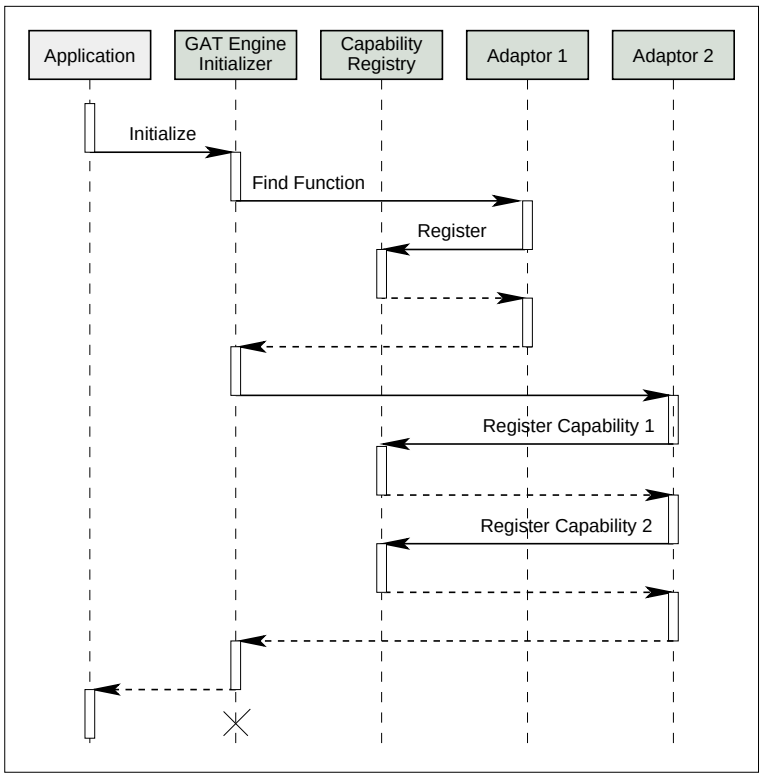

Figure 3: **GAT Initialization:** On initialization, all adaptors register the capabilities they provide with the GAT Capability Registry.

describe their functionality. Note that these examples are intentionally simplistic, and are not formulated independently from the language bindings; we assume a `C`-application here. The API is currently being defined; so, these examples are schematic only, and specific arguments will almost certainly be different in the final version.

For example, the following call initializes the GAT:

```
context = GAT_Init (Requirements);
```

Here, `context` denotes an opaque object holding persistent GAT information (such as security credentials and the internal state of the GAT Engine), and `Requirements` specifies any specific information needed for initialisation, such as the initial adaptors to be loaded, etc. As described in the previous sections, this includes the setup of the capability registry. After setup, the GAT knows which capabilities it can actually provide. A rebuild of the registry may be requested at later times.

As another example, the following call queries the environment for a resource matching a number of requirements.

```
GAT_FindResource (context, Requirements, Resource);
```

The `Requirements` parameter encodes resource requirements of the application (e.g., 1 GFlop). This is likely to be stored in some property table. `Resource` is a parameter returned by the GAT-API, and holds information about a matching resource. This information could then be passed into the following GAT-API call; for example, to start a job on the machine.

```
GAT_SubmitJob (Resource, Job);
```

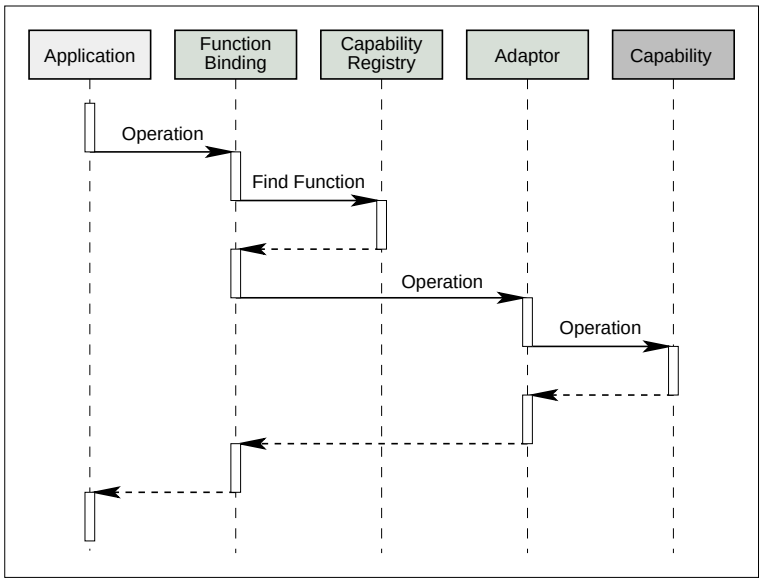

Figure 4: **GAT-API Call:** On invocation, the GAT queries the capability registry for matching adaptors, and returns the best fitting one with respect to some metric. The GAT Engine invokes that adaptor in order to service the API call. The adaptor forwards this call to the actual capability provider implementing the associated functionality.

Here, `Job` is again an opaque object holding all information needed to execute the job. This object may obtain additional information on execution useful for subsequent GAT calls (e.g., a job id or other runtime information).

As the GAT engine finds different adaptors providing the functionality to run a job on some resource, the engine will select an adaptor based upon any metric specified in the `Job` description; e.g., reliability. For example, if a first adaptor interfacing to the local UNIX system fails because it cannot handle job descriptions containing remote resources, the engine invokes the next adaptor; this in turn may interface to a GridLab service providing scheduling capabilities.

These simple examples hopefully give an initial impression of what we are working to provide application programmers. The use of the GAT should be as simple as possible. It should not matter if the environment is Grid enabled – but the user should be able to use the same, unmodified, application, if it is. By providing a flexible adaptor scheme, the GAT can be easily evolved to accommodate changing infrastructures, and easily extended to provide capabilities required by new user groups.

One of our major tasks is to design the GAT-API in a way that, from the beginning, as large and diverse a group as possible can benefit from our efforts (and in turn, benefit our efforts from their feedback). Here, dissemination via our user groups and the GGF is crucial.

6 GridLab Applications

We will now describe the applications initially being used to shape the construction of the GAT interface. These applications are both generic frameworks, and as such provide a wide range of fields and possible usage scenarios, covering a broad domain of the future GAT user base.

The principle objective of the GridLab project is to allow the easy integration of applications with emerging Grid technologies. GridLab aims to provide an environment that allows application developers to use the Grid without having to understand, or even being aware of the underlying technologies. The GAT effectively shields the application developers from the current, ever-changing Grid world by providing an application-friendly interface that contains the functionality required by applications. For this purpose,

GridLab is driven by two well known and widely used application frameworks to help prototype the GAT interface, Cactus [2, 3] and Triana [4].

The primary reason for this choice is that both Cactus and Triana are application environments. They both contain a toolkit of pluggable components (*thorns* in Cactus, *units* in Triana) that allow users to compose applications for a particular problem area. That is, they are both Problem-Solving Environments (PSE) that are complete, integrated computing environments for composing, compiling, and running applications in a specific area [8]. The key point here is that both applications are not focused on one particular task or user and therefore provide a rich collection of reference application scenarios that will be run on the Grid. Consequently, from these two applications alone we have at our disposal literally hundreds of possible user scenarios. Here we now provide an overview of Triana, and a flavor of some of the types of applications that it supports. We have not provided a similar overview of Cactus as this is described in detail in [3].

Triana

Triana is an open source PSE written in Java. It has a flexible and intuitive design that can be used in many different problem domains and at many different levels. For example, it can be used as a workflow for Grid applications or as a data analysis system for image, signal or text processing applications. It can also be used as a high-level graphical script editor for creating a number of task-graph formats including, but not limited to, WSFL, DAG, BPEL4WS and Petrinet formats.

Recently, Triana has been redesigned and sectioned into a set of modularized components, which can be used collectively to construct Triana, or independently for specific tasks e.g. for providing remote control, for writing task-graphs and for distributing third party units, etc. In this new version, the GUI has been disconnected from the underlying subsystem for both the functionality of the main system and for every Triana unit and its associated GUI. This has several ramifications. Firstly, clients (i.e. those running a GUI) can log into a Triana engine remotely. Secondly, the user interfaces for the individual components all employ the same type of separation and can also therefore be viewed remotely. For example, users can build and run a Triana network and then visualize the result on their device (e.g., laptop, PDA) even though the visualization unit (and therefore its calculations) is run remotely. Users can log off without stopping the execution of the network and then log in at a later stage to view the progress. In this context, Triana could be used as a visual environment for monitoring the workflow of Grid services or as an interactive portal by running the Triana engine as a servlet on a web server and running the applet version of the Triana GUI. Further, since any Triana network can be run with or without using the Triana GUI, Triana networks can be run as executables in a stand alone mode, i.e., Triana is an application designer tool as well.

There is support for a variety of Triana users. It is used as a quick-look data analysis system for the GEO600 gravitational wave detector and therefore contains a number of signal-processing units like file I/O, frame readers, FFT's, correlation, noise simulators, statistics units, and a wide range of mathematical functions and visualization units. It contains a complete image processing toolkit and has many tools for text processing, e.g., text filtering, substitution and searching facilities (locally, and web crawling, etc.) Further, it has tools for the visualization of galaxy formations, whose distributed implementation using the GAT has been demonstrated recently [9]. Lastly, we are intending to implement a Triana-Cactus interface. This will enable Cactus to be run as a Triana unit and further, allow Cactus parameter files to be generated by connecting various Triana unit thorns. The integration of Cactus and Triana will have countless uses and will demonstrate the flexibility of a multiple-tier GAT implementation.

In the next section, we will use Triana to illustrate its integration with Grid environments via the GAT interface.

6.1 Grid-enabling Triana with the GAT

Briefly, Triana implements its distributed functionality as a collection of Triana services. Each Triana service includes a client and server component and a user may connect to a Triana service using a command line or GUI interface. The client can distribute code to many servers depending on where execution is required. The distributed implementation is layered above the GAT and therefore above the Grid middleware layer (see Figure 5).

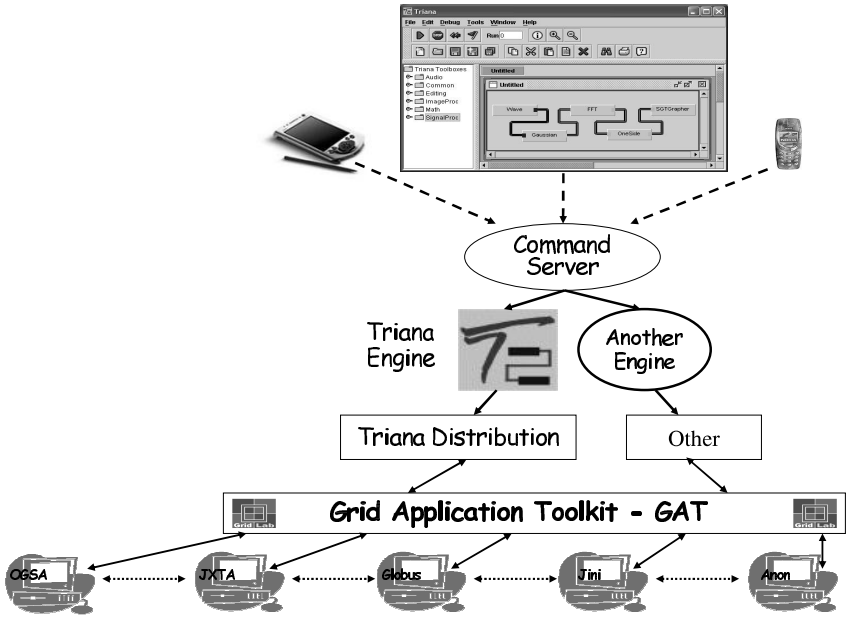

Figure 5: The Triana Grid architecture using the Grid Application Toolkit.

Distribution is based on the concept of Triana Group units (i.e., aggregate tools) which have similar properties to normal units; namely, they have input/output nodes, properties. etc. Therefore, they can be connected to other Triana units using the standard mechanism. Group units are distributed according to a distribution policy, implemented using a control unit. A control unit is implemented as a standard Triana unit, although it is invisible to the user. Control units dynamically re-route the data to and from the various distributed services that are currently available, according to the particular policy they are adhering to. There are currently two distribution policies implemented: parallel and peer-to-peer. Parallel is a *task farming* mechanism and involves no communication between hosts. Peer-to-peer is distributing the group vertically, each unit in the group is distributed onto a separate resource and data is passed between them.

Triana uses the GAT interface to distribute its services. Figure 5 shows where the GAT fits into the distributed Triana architecture. Notice that the Triana distribution mechanism is only one possible implementation and other engines can be inserted easily. Our current prototype implementation uses the JXTA binding for the GAT that provides hooks into the JXTA middleware to implement the necessary GAT functionality. However, this specific binding to JXTA is completely hidden from Triana's distribution mechanism by using the GAT interface. In essence: Triana discovers other Triana services by using the JXTA binding of the GAT discovery protocol; they identify each other by using the GAT ID, implemented as a JXTA ID; and they communicate with each other by using the GAT communication interface, implemented as JXTA pipes. A similar level of abstraction will exist in the Web Service/OGSA implementation of the GAT and will provide Triana with this new distribution mechanism, without the need for changing a single line of the current Triana code. The real programming effort in this stage is to implement the GAT Web Service/OGSA binding.

7 GridLab Services

Developing Grid services is an important aspect of the GridLab project. These services are designed to complement and complete the existing Grid infrastructure, and to provide functionality needed by the GridLab applications in order to be usefully deployed in such environments. As an example, we describe

here the GridLab Resource Management Services, a system of interoperating services providing resource management, scheduling, and job execution capabilities on the Grid. Besides the scheduling services, GridLab provides a portal toolkit, services for authorization, for user notification, data management and visualization, for monitoring of resources and job status, for metadata and for application behavior adaptation.

7.1 GridLab Resource Management Services

Resource management is an important issue for the success of computational Grids, especially in Grids capable of supporting commercial applications. Ideally, resource management services are the only path of interaction to all Grid resources, and hence are the primary pathway for expressing critical parameters such as Quality of Services (QoS) [10] specifications. Here, one of the main questions concerns how to map activities such as computation or data transfer onto sets of resources belonging to different organizations, in ways that will meet user requirements for performance, cost, security, and other metrics corresponding to quality of service. In GridLab, one of our major goals is to create a superscheduler that will be responsible for making decisions to achieve the best possible resource utilization, as well as to meet the user preferences mentioned above.

Resource management in Grids is difficult because of the following reasons:

- Users and resources are usually highly distributed and physically remote from each other.
- Resource status changes rapidly and dynamically.
- Both users and resources form dynamic groups of entities, changing in size over time. The quality of connectivities between these users and resources also changes in time.
- There is no common scheduling policy among the Grid resources; all resources (and users) may be subject to various policies depending on their host organizations.

There are potentially many users competing for Grid resources. Also, since the same resource may belong to many virtual organizations, users may use different superschedulers. Moreover, local policies can not be disregarded when scheduling a Grid user's job on a remote resource.

Our resource management system is designed to provide such functionality. In the course of this section, we describe the overall design of that system, which is currently under construction, and further list and describe the components of the system.

7.1.1 The GridLab Resource Management System

The purpose of the *GridLab Resource Management System* (GRMS) is to provide scheduling mechanisms that fit the needs of users and their applications with regard to local policies. The creation of different algorithms that can satisfy completely different needs depending on the specific charging policy is very important here. For example, virtual organizations dealing with commercial applications prefer to apply the fair scheduling policy so that high-demand users do not displace low-demand users. In contrast, scientific applications can benefit from the workload-based scheduling policy. Details of these scheduling policies are out of scope of this paper. They are described in [11].

The GRMS is composed of a collection of system services. These services are outlined below, Figure 6 depicts the overall architecture of the system. GRMS services include the following:

- *System configuration management:* CIM-based functional and performance monitoring is necessary for all elements of the system. This monitoring should be conducted in a timely fashion, and events should be delivered according to policies or criteria set by the client.
- *Job execution management:* It handles time, priority, and space based scheduling of jobs. In case of application failure, jobs are restarted based on the applicable policy.
- *Resource management:* Dynamic and flexible resource management is essential. At the same time, resource isolation between different jobs is crucial. This is true not only for access control, but also to ensure that there are no unexpected performance dependencies.

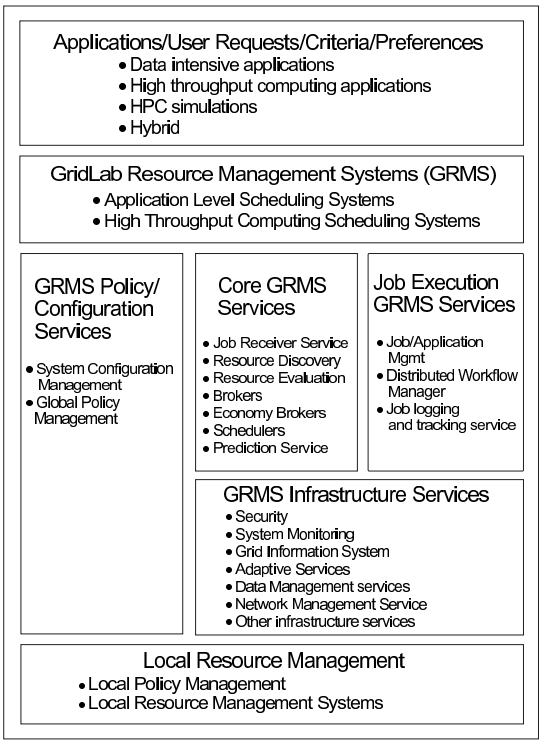

Figure 6: Reference architecture of the GridLab Resource Management System (GRMS).

- *Infrastructure services:* Thses services handle user management, accounting management, security etc.

7.1.2 GRMS Configuration and Policy Services

GRMS Configuration Services are responsible for GRMS configuration. These services allow users (GRMS administrators) to enforce different resource management policies, which can then be mapped onto different groups of users. Moreover, the basic configuration of GRMS will be supported by this service, including selection of various scheduling algorithms and strategies.

Additionally, *GRMS Policy Service* has been introduced to handle a wide variety of policy needs, with a special focus on policies governing resource allocation in a particular virtual organization. The policy service will support single domain policy management as well as multi-domain policy management and control. In management involving multiple domains, the policy service will discover and take into account the policies of different (virtual) organizations as part of their resource publishing and matching process. Both kinds of services will be developed in the last stage of the GridLab project.

7.1.3 Core GRMS Services

GRMS services include Job Receiver, Resource Discovery, Resource Evaluation, Brokering, Prediction, QoS, Resource Reservation and Resource Estimation services.

- Job Receiver (JR) is responsible for receiving the job requests and queuing them for the time of processing the first job request sitting in a queue. The Job Receiver Queue (JRQ) is a distributed service (there may exist many instances of this service) with a notion of job arrival time. This means that there are no job requests coming to two different Job Receiver services with the same timestamp.
- Resource Discovery Service (RDS) is a service that is responsible for discovering valid resources that could potentially be allocated for a given job request. In general, RDS returns an unranked list of

the resources for a particular job request. Those resources will be further evaluated by the Resource Evaluation Service.

- Resource Evaluation Service (RES) is responsible for determining which of the given resources are the best ones from the point of view of the user's job request. Among others, RES takes the following parameters into account:

  - Quantifiable User Criteria (cost and time criteria)
  - Non-quantifiable User Criteria (resource architecture, operating system etc.)
  - Preferences among the user criteria
  - Application performance characteristics
  - Historical data about completed jobs (history based predictions)
  - Statistical information
  - Resource characteristics

  The RES is supported by an intelligent decision making system, and can also be supported by the *Adaptive Component Service* of the GridLab project. The RES will be elaborated further in future GridLab publications. There, we will strictly distinguish between production- and research-based components of the RES.

- Brokering Service (BS) is a central service of the GRMS. It is responsible for allocation of resources to user jobs. Once the job is submitted, the broker, a *Job Management Service (JMS)*, and a *Distributed Workflow Service (DWS)* are jointly responsible for controlling the job. The broker can send various signals to a job, such as *JobCheckpoint*, *JobKill*, *JobMigrate*, *JobStop*, etc. These signals can be sent automatically by the broker, can be enforced by the job manager, or can be triggered directly by a user. This range of behaviors supports the implemention of the many dynamic application scenarios required by GridLab applications.

- Prediction Service (PS) is responsible for short-term predictions of the resource behavior, as well as for job run time and queue wait time predictions. The prediction service attempts to reason about jobs based on past execution of identical or similar jobs. The usage of prediction mechanisms within GRMS will change as a function of the future improvements to this technology. During the early project stages, the Logging and Tracking Service (LTS) will gather information about jobs and store this information in a database (local to GRMS). This information will be used in the future for history-based predictions.

- Advanced Reservation Service (ARS) is responsible for making advance reservations of resources, making them available for jobs in a given (but potentially negotiated) time frame in the future. This functionality will be available on selected local resource management systems only.

- QoS Service (QOS) is a service that allows a user to negotiate the quality of service provisioned to the job.

- Resource Estimation Service (REST) may be used to predict, for a given job, what kind of resources are needed to process the job in the absence of proper job description. The REST produces a resource description file. This may be an ASCII text file using a meta-language that can be interpreted by the BS and the scheduler.

7.1.4 Job Execution Services

The *Job Execution Service* addresses resource scheduling and computing access. The *Job Management Service (JMS)* must deal with many types of job requests. JMS must manage jobs during their lifetime. It carries out initial processing needed to prepare jobs for execution, submits jobs to the scheduler, and carries out any necessary post-processing. This service is responsible for monitoring the job schedule condition before and during the job execution. If a job has not yet been executed, the schedule may become invalid

(for instance, if resources fail). If an event occurs that requires re-scheduling, the scheduling service is contacted. If this is not possible or not desired for the job, only the user is informed. Additionally, the JMS is responsible for initiating job execution or for the setup of job parts. Again, this service can interoperate with the data, network, and resource services to include information about the data and network status.

The *Scheduler* must arrange for job execution based on job requirements (e.g., time, priority, cost, place, user preferences). The scheduler must maintain jobs and their schedule in a persistent scheduling database. The *Distributed Workflow Service (DWS)* must manage job workflow based on standard workflow specifications (e.g., WSFL, XLANG, and WSCI).

Please note that the scheduling service is not a centralized component. There may be many instances of different scheduling services available. For example, the scheduling services may be adapted for special application-dependent purposes. The selection strategies may differ according to the specific requirements for a certain resource type. They interact and cooperate with the other information and management services. This allows a scenario in which the scheduling service is set up by a user application.

7.1.5 Infrastructure Services

The GRMS interacts with a number of other services in the Grid – those services we call Infrastructure Services, since they provide an interface to the Grid infrastructure. This interaction enables the GRMS to query information from the Grid environment, to utilize lower level capabilities for the GRMS services (such as security), and to interact with the Grid resources. Some of these infrastructure services are described in the list below.

- Data Management Service: This service maintains information about the scheduled availability and location of data. This information is included in the extended information service and can be accessed from there.

  It is also responsible for answering queries on the availability of data for a given resource at a certain time. This includes the calculation of whether and how data should be transferred between resources. This requires that the data management service can evaluate where data sets are available, and what effort would be necessary to move the data. Hence, the data management service itself queries the scheduling service and network management service for certain resources and their reservations.

  In an application scenario in which certain data sets are commonly used by different users and are available at different locations, the service calculates and maintains replication and transfer schedules. The data management service includes a data migrator which monitors and executes scheduled data transactions.

- Network Management Service: Similarly to the data management service, the network management service maintains information about network resources and corresponding reservations. The network management service includes a uniform interface to network quality of service features, in combination with an infrastructure to collect information about network properties. This requires, for example, the inclusion of common standards (e.g., the GARA service or the NWS for bandwidth information) into a uniform interface. The network management service responds to queries about connection properties between given resources. It determines autonomously if network reservations are necessary and if they can actually be provided.

- Information Service: The current Grid services must be extended for maintaining dynamic information about current and future resource allocations, network status etc.. This information is available through interaction with the corresponding scheduling, data and network services. This service has a uniform interface to obtain this information. Nevertheless, its implementation as a directory service (in conformance with the current Grid information services) still allows distributed services to query for the actual resource information.

- Accounting and Billing Services: Execution and scheduling of jobs requires an accounting and billing mechanism to charge users. The querying and tentative reservation of resources can also be associated with costs. The user can include budget and billing information in his job request. The actual billing

must be accomplished via a secured transaction interface. This service needs a trusted partner hierarchy and the ability to delegate authorization for certain time spans and limited budgets. The scheduling service requires the ability to grant billing authorization in conjunction with the resources. As only the user can authorize such a transaction, this must be done interactively during the scheduling process, or at its beginning by incorporating a trusted accounting and billing service.

We hope this section gave a first impression on which problem domain the GRMS will operate on. This description did not give much technical details about the various components and their interaction – this will be the subject of future publications.

8 Conclusions

In this paper, we have presented the overall architecture of the GridLab project, which aims to provide application-oriented Grid services for users and developers alike, covering the whole range of Grid capabilities as required by applications, such as resource brokering, monitoring, data management etc. These services will abstract lower-level Grid functionality and will hence ease the development and deployment of Grid-aware applications. This consistent service infrastructure will be the first major deliverable of the GridLab project.

These services will be made accessible to any applications running on the Grid through a Grid Application Toolkit (GAT), the second main project deliverable. The GAT will abstract those services needed by the 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 both run on a single laptop or on an intercontinental Grid, taking advantage of whatever services are actually available (or unavailable). In particular, the GAT will not depend on the existence of any specific GridLab services.

Although we collaborate with the developers of powerful application frameworks such as Cactus and Triana for 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 available on a Grid, such as those described above in our usage scenario. With such an architecture, we expect many new and powerful applications to be developed to exploit the Grids of today and tomorrow alike.

Acknowledgments

We are pleased to acknowledge support of the European Commission 5th Framework program (grant IST-2001-32133), which is the primary source of funding for the GridLab project, but also the German DFN-Verein (grant TK 602 - AN 200), 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, Gerd Lanfermann, Jason Novotny, and various members of the Globus team for many discussions. Many thanks to Brygg Ullmer for his thorough proofreading.

References

[1] G. Allen, T. Dramlitsch, T. Goodale, G. Lanfermann, T. Radke, E. Seidel, T. Kielmann, K. Verstoep, Z. Balaton, P. Kacsuk, F. Szalai, J. Gehring, A. Keller, A. Streit, L. Matyska, M. Ruda, A. Krenek, H. Frese, H. Knipp, A. Merzky, A. Reinefeld, F. Schintke, B. Ludwiczak, J. Nabrzyski, J. Pukacki, H.-P. Kersken, and M. Russell, "Early experiences with the EGrid testbed," in *First IEEE/ACM International Symposium on Cluster Computing and the Grid*, (Brisbane, Australia), pp. 130–137, May 2001.

[2] Cactus Computational Toolkit home page: `http://www.cactuscode.org`.

[3] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf, "The Cactus Framework and Toolkit: Design and Applications," in *Vector and Parallel Processing – VECPAR'2002, 5th International Conference*, (Berlin, Germany), Lecture Notes in Computer Science, Springer, 2003.

[4] Triana home page: `http://www.triana.co.uk`.

[5] S. Tuecke, I. Foster, and C. Kesselman, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, 2001.

[6] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration." Draft Document, `http://www.globus.org/research/papers/ogsa.pdf`, June 2002.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison-Wesley, 1995.

[8] E. Gallopoulos, E. Houstis, and J. Rice, "Computer As Thinker/Doer: Problem-Solving Environments for Computational Science," *IEEE Computational Science and Engineering*, vol. 1, no. 2, pp. 11–23, 1994.

[9] I. Taylor, M. Shields, and R. Philp, "GridOneD: Peer to Peer visualization using Triana: A Galaxy Formation Test Case," in *Proceedings of the UK eScience "All Hands Meeting"*, (Sheffield), September 2-4 2002.

[10] W. Smith, I. Foster, and V. Taylor, "Scheduling with Advanced Reservations," in *Proceedings of the IPDPS Conference*, pp. 127–132, IEEE Computer Society, May 2000.

[11] J. Brzezinski, J. Nabrzyski, J. Puckacki, T. Piontek, K. Kurowski, L. Ludwiczak, R. Strugalski, M. Hapke, N. Doulamis, A. Doulamis, M. Varvarigos, and K. Dolkas, "Technical Specification of the GridLab Resource Management System." `http://www.gridlab.org/Resources/Deliverables/D9.2.pdf`, July 2002.