Legion: An Integrated Architecture for Grid Computing[*]

Andrew S. Grimshaw
Marty A. Humphrey
Anand Natrajan

LEGION Worldwide Virtual Computer
e pluribus unum: one out of many

Abstract:

Legion and Globus are pioneering grid technologies. Several of the aims and goals of both projects are similar, yet their underlying architectures and philosophies differ substantially. The scope of both projects is the creation of world-wide grids; in that respect, they subsume several distributed systems technologies. However, Legion has been designed as a virtual operating system for distributed resources with OS-like support for current and expected future interactions between resources, whereas Globus has long been designed as a "sum of services" infrastructure, in which tools are developed independently in response to current needs of users. We compare and contrast Legion and Globus in terms of their underlying philosophy and the resulting architectures. We discuss how these projects converge in the context of new grid standards being formulated.

[*] This work partially supported by DARPA (Navy) contract #N66001-96-C-8527, DOE grant DE-FG02-96ER25290, DOE contract Sandia LD-9391, Logicon (for the DoD HPCMOD/PET program) DAHC 94-96-C-0008, DOE D459000-16-3C, DARPA (GA) SC H607305A, NSF-NGS EIA-9974968, NSF-NPACI ASC-96-10920, and a grant from NASA-IPG.

Globus, Globus Toolkit, GGF, Legion, Global Grid Forum, PBS, LSF, Avaki, Avaki Data Grid, ADG, LoadLeveler, Codine, SGE, IBM, Sun, DCE, MPI, CORBA, OMG, SSL, OpenSSL, GSS-API, MDS, SOAP, XML, WSDL, Windows (NT, 2000, XP), DCE, J2EE, NFS, AIX, Kerberos are all trademark or service mark of their respective holders.

1 Introduction

Grids are collections of interconnected resources harnessed together in order to satisfy various needs of users. The resources may be administered by different organizations and may be distributed, heterogeneous and fault-prone. The manner in which users interact with these resources as well as the usage policies for those resources may vary widely. A grid infrastructure must manage this complexity so that users can interact with resources as easily and smoothly as possible.

Our definition, and indeed a popular definition, is: A grid system is a collection of distributed resources connected by a network. A grid system, also called a grid, gathers resources – desktop and hand-held hosts, devices with embedded processing resources such as digital cameras and phones or tera-scale supercomputers – and makes them accessible to users and applications in order to reduce overhead and accelerate projects. A grid application can be defined as an application that operates in a grid environment or is "on" a grid system. Grid system software (or middleware), is software that facilitates writing grid applications and manages the underlying grid infrastructure. The resources in a grid typically share at least some of the following characteristics:

- They are numerous.
- They are owned and managed by different, potentially mutually-distrustful organizations and individuals.
- They are potentially faulty.
- They have different security requirements and policies.
- They are heterogeneous, e.g., they have different CPU architectures, run different operating systems, and have different amounts of memory and disk.
- They are connected by heterogeneous, multi-level networks.
- They have different resource management policies.
- They are likely to be geographically-separated (on a campus, in an enterprise, on a continent).

The above definitions of a grid and a grid infrastructure are necessarily general. What constitutes a "resource" is a deep question, and the actions performed by a user on a resource can vary widely. For example, a traditional definition of a resource has been "machine", or more specifically "CPU cycles on a machine". The actions users perform on such a resource can be "running a job", "checking availability in terms of load", and so on. These definitions and actions are legitimate, but limiting. Today, resources can be as diverse as "biotechnology application", "stock market database" and "wide-angle telescope", with actions being "run if license is available", "join with user profiles" and "procure data from specified sector" respectively. A grid can encompass all such resources and user actions. Therefore, a grid infrastructure must be designed to accommodate these varieties of resources and actions without compromising on some basic principles such as ease of use, security, autonomy, etc.

A grid enables users to collaborate securely by sharing processing, applications and data across systems with the above characteristics in order to facilitate collaboration, faster application execution and easier access to data. More concretely, this means being able to:

- Find and share data. Access to remote data should be as simple as access to local data. Incidental system boundaries should be invisible to users who have been granted legitimate access.
- Find and share applications. Many development, engineering and research efforts consist of custom applications – permanent or experimental, new or legacy, public-domain or proprietary – each with its own requirements. Users should be able to share applications with their own data sets.

- Find and share computing resources. Providers should be able to grant access to their computing cycles to users who need them without compromising the rest of the network.

In this paper, we will compare two pioneering grid technologies – Legion and Globus. As members of the Legion project, we naturally have a deeper understanding of Legion as compared to Globus. However, we have attempted to present a careful, balanced and symmetric comparison of the two projects from the literature we have found. To this end, we have avoided going into deep details of Legion in order to maintain parity with our understanding of Globus. In Section 2, we motivate the comparison between Legion and Globus as opposed to other technologies. Under the broad definition of a grid given so far, we will show that these two remain, at the time of this writing, the only technologies that attempt to build a complete grid. In Section 3, we enumerate a set of requirements for grids, and then describe how each project addresses each requirement, noting the relative importance of each requirement to each project. We also describe the design principles of each project. In this section as well as in the rest of this paper, we attempt to offer a constructive, informative differentiation to the community without criticizing the work of the Globus Toolkit. Despite the differences between Legion and Globus, we respect the approach and successes of the Globus project and are currently working together on the community-defined Open Grid Services Architecture (OGSA, see Section 6). The specifics of each architecture are contained in Section 4. When referring to their philosophy and architecture, we will refer to the two projects as "Legion" and "Globus" respectively, but if referring to their implementation, we will refer to them as "Legion 1.8" and "Globus 2.0" respectively, the numbers denoting the latest versions available at the time of writing. Specifically, when referring to Globus, we will discuss version 2 of the toolkit (GT2), not version 3 (GT3). At the time of this writing, GT3 has not been specified fully. In Section 5, we will present the current status of both projects, touching on commercial as well as academic deployments. In Section 6, we will discuss both projects in the context of upcoming standards, specifically the Open Grid Services Architecture (OGSA), being formulated by the Global Grid Forum (GGF). We summarize this comparison in Section 7. We encourage readers to peruse through the many Legion and Globus papers listed in the bibliography.

2 Background and Related Work

Over the years, there have been many definitions of what constitutes a grid. Below, we present a small list of the more visible definitions:

Users will be presented the illusion of a single, very powerful computer, rather than a collection of disparate machines. [...] Further, boundaries between computers will be invisible, as will the location of data and the failure of processors.

- Grimshaw [7]

We believe that the future of parallel computing lies in heterogeneous environments in which diverse networks and communication protocols interconnect PCs, workstations, small shared-memory machines, and large-scale parallel computers.

- Foster, Kesselman, Tuecke [42]

A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

- Foster, Kesselman [49]

... a Grid is a system that: (i) coordinates resources that are not subject to centralized control... (ii) using standard, open, general-purpose protocols and interfaces... (iii) to deliver nontrivial qualities of service...

- Foster [52]

A Grid is a hardware and software infrastructure that provides dependable, consistent, and pervasive access to resources to enable sharing of computational resources, utility computing, autonomic computing, collaboration among virtual organizations, and distributed data processing, among others.

- Gentzsch [65]

... the overall grid vision [is] flexible access to compute, data, and unique network resources by providing flexible access and sharing of desktop PC resources.

- Chien [62]

A Grid system is a collection of distributed resources connected by a network. ... Grid system software (or middleware), is software that facilitates writing Grid applications and manages the underlying Grid infrastructure.

- Grimshaw et al. [14]

With time, the definitions have become more and more general in order to encompass the wide capabilities we now expect from a grid. There is a clear tendency in the definitions towards a more general definition of what constitutes a "resource" in a grid and what actions may be performed on those resources. Moreover, several of the definitions present (and argue for or against) general characteristics of a grid. Evaluating currently-available commercial and academic technologies against the characteristics mentioned in these and other definitions is a valuable survey exercise; however, we will not undertake it here. Instead, we will compare Legion and Globus alone.

From the start, a common goal of both Legion and Globus was to build grids that would span administrative domains. This goal is a key differentiator between these projects and other distributed systems. For example, queuing systems such as NQS [68], PBS [61], LSF [74] [75], LoadLeveler [58], Codine [63] and SGE [63] were not designed initially to operate across administrative domains or even multiple clusters within a single domain. Although some of these queuing systems – LSF and SGE in particular – have adapted to multi-cluster configurations, their underlying design does not address grid goals. Likewise, systems such as PVM [64], MPI [66], NOW [60], DCE [70] and Mentat [5] [8] [10], were intended to enable writing parallel or distributed programs, but did not have any support for concerns such as wide-area security, multiple administrative domains, fault-tolerance, etc. Some academic projects, such as Condor [69], Nimrod [59] and PUNCH [67] were designed in much the same manner as Legion and Globus. However, these projects did not encompass a diversity of resources and actions as Legion and Globus did. For example, Condor provided support for location-independent running of a specific class of resources, namely, applications; Nimrod provided fault-tolerance and monitoring of a specific class of resources, namely, parameter-space applications; and PUNCH provided location-independent access to a specific class of resources, namely data files.

The Globe project [73] shares many common goals and attributes with Legion. Both are middleware metasystems that run on top of existing host operating systems and networks, both support implementation flexibility, both have a single uniform object model and architecture, both use class objects to abstract implementation details, and so on. However, Globe's object model is different. A Globe object is passive and is assumed to be distributed physically potentially over many resources in the system, whereas a Legion object is active and expected to reside within a single address space. These conflicting views of objects lead to different mechanisms for inter-object communication. Globe loads part of the object (called a local object) into the address space of the caller whereas Legion sends a message of a specified format from the caller to the callee. Another important difference is the notion of core object types. Legion's core objects are designed to have interfaces that provide useful abstractions that enable a wide variety of implementations. We are not aware of similar efforts in

Globe. We believe that the design and development of the core object types define the architecture of Legion, and ultimately determine its utility and success. Legion is designed to look like an OS for grids, whereas Globe is designed to look more like an application environment. On the other hand, Globe differs much more from Globus because Globus does not have an underlying object model.

Although not intended for grid computing, the Common Object Request Broker Architecture (CORBA) standard developed by the Object Management Group (OMG) [71] shares a number of elements with the Legion architecture. Similar to Legion's idea of many possible object implementations that share a common interface, CORBA systems support the notion of describing the interfaces to active, distributed objects using an Interface Description Language (IDL), and then linking the IDL to implementation code that might be written in any of a number of supported languages. Compiled object implementations rely on the services of an Object Request Broker (ORB), analogous to the Legion run-time system, for performing remote method invocations. Despite these similarities, the different goals of the two systems result in different features. Whereas CORBA is more commonly used for business applications, such as providing remote database access from clients, Legion is intended for executing high-performance applications as well. This difference in vision manifests itself at all levels in the two systems – from basic object model up to the high-level services provided. For example, where CORBA provides a simple RPC-based (remote procedure call) method execution model suited to client-server style applications, Legion provides coarse-grained dataflow method execution model, called a macro-dataflow[1] model, suitable for highly-concurrent grid applications.

Finally, several commercial projects such as Entropia, United Devices, and Parabon attempted and continue to attempt building large grids. However, unlike Legion and Globus, several of these projects impose restrictions such as inability to run on a wide range of platforms, or code conversion of applications to one language or another. Such restrictions violate several of the grid principles to which both Legion and Globus adhere.

3 Requirements and Design Principles

Legion and Globus share a common base of target environments, technical objectives, and target end users, as well as a number of similar design features. Both systems abstract access to processing resources: Legion via an interface called the "host object"[2], and Globus via a service called the Globus Resource Allocation Manager (GRAM) interface [38]. Both systems also support applications developed using a range of programming models, including popular packages such as MPI. Despite these similarities, the systems differ significantly in their basic architectural techniques and design principles. Legion builds higher-level system functionality on top of a single unified object model[3] and set of abstractions to insulate the user/programmer from the

[1] Macro-dataflow [8] [24] [30] is a data-driven computation model based on dataflow, in which programs and sub-programs are represented by directed graphs. Graph nodes are either method calls (procedure calls) or sub-graphs. A node "fires" when data is available on all input arcs. In Legion, graphs are first-class and can be passed as parameters and manipulated directly [30]. Macro-dataflow lends itself to flexible communications between objects. For example, the model permits asynchronous calls, calls to methods where parameters may come in from yet other objects, and dispersal of results to multiple recipients.

[2] A Legion object runs in, or is contained, in a host object when it is executing. Thus, a host object is essentially a virtualization of what would now be called a hosting environment, as in J2EE. The host object interface [18] includes methods for metadata collection, object instantiation, killing and suspending object execution, etc.

[3] The fact that Legion is object-based does not preclude the use of non-object-oriented languages or non-object-oriented implementations of objects. In fact, Legion supports objects written in traditional procedural languages such as C and Fortran [4] as well as object-oriented languages such as C++, Java,

underlying complexity of the grid. The Globus implementation is based on the combination of working components into a composite grid toolkit that fully expose the grid to the programmer.

The Globus approach of adding value to existing high-performance computing services, enabling them to interoperate and work well in a wide-area distributed environment, has a number of advantages. For example, this approach takes advantage of code reuse, and builds on user knowledge of familiar tools and work environments. However, a challenge with this sum-of-services approach is that as the number of services grows in such a system, the lack of a common programming interface to Globus components and the lack of a unifying model of their interaction can make ease-of-use difficult. Typically, end-users must compensate for the system by providing their own mechanisms for service interoperability. By providing a common object programming model for all services, Legion enhances the ability of users and tool-builders to use a grid computing environment effectively by employing the many services that are needed: schedulers, I/O services, applications, etc. Furthermore, by defining a common object model for all applications and services, Legion permits a more direct combination of services. For example, traditional system-level agents such as schedulers as well as normal application processes are both normal Legion objects exporting the standard object-mandatory interface. We believe in the long-term advantages of basing a grid computing system on a cohesive, comprehensive and extensible design.

In this section, we contrast the Legion philosophy and architecture with our belief and understanding of the Globus philosophy and architecture. We start in Section 3.1 with the high-level requirements that we identified and targeted in the design of Legion. Next, we enumerate the design principles of the architecture and implementation that guided the development of Legion. In Section 3.2, we describe the Globus philosophy and architecture. While we believe that the Legion team and the Globus team agree largely on the requirements, the differences lie in the emphasis or approach each project places on each requirement. Therefore, in this section we present the same requirements list as in Section 3.1, but augmented with a discussion of each requirement in the context of Globus. Next, we enumerate a set of design principles that we believe guided the development of Globus. Whereas the requirements lists in Section 3.1 and Section 3.2 are identical, the design principles differ because of key differences in architectural approach. While we have attempted to differentiate the discussion of each requirement from its implementation (discussed in more detail in Section 4), we have found it necessary at times in this section to provide some implementation details to clarify the discussion.

3.1 Legion

Clearly, the minimum capability needed to develop grid applications is the ability to transmit bits from one machine to another – all else can be built from that. However, several challenges frequently confront a developer constructing applications for a grid. These challenges lead us to a number of requirements that any complete grid system must address. The designers of Legion believed and continue to believe that all of these requirements must be addressed by the grid infrastructure in order to reduce the burden on the application developer. If the system does not address these issues, then the programmer must – forcing programmers to spend valuable time on basic grid functions, thus needlessly increasing development time and costs. These requirements are high-level and independent of implementation. They are:

and Mentat Programming Language (MPL, a C++ dialect with extensions to support parallel and distributed computing) [10].

- ***Security.*** Security covers a gamut of issues, including authentication, data integrity, authorization (access control) and auditing. If grids are to be accepted by corporate and government IT departments, a wide range of security concerns must be addressed. Security mechanisms must be integral to applications and capable of supporting diverse policies. Furthermore, we believe that security must be built in firmly from the beginning. Trying to patch security in as an afterthought (as some systems are attempting today) is a fundamentally flawed approach. We also believe that no single security policy is perfect for all users and organizations. Therefore, a grid system must have mechanisms that allow users and resource owners to select policies that fit particular security and performance needs, as well as meet local administrative requirements.
- ***Global name space.*** The lack of a global name space for accessing data and resources is one of the most significant obstacles to wide-area distributed and parallel processing. The current multitude of disjoint name spaces greatly impedes developing applications that span sites. All grid objects must be able to access (subject to security constraints) any other grid object *transparently* without regard to location or replication.
- ***Fault tolerance.*** Failure in large-scale grid systems is and will be a fact of life. Machines, networks, disks and applications frequently fail, restart, disappear and behave otherwise unexpectedly. Forcing the programmer to predict and handle all of these failures significantly increases the difficulty of writing reliable applications. Fault-tolerant computing is known to be a very difficult problem. Nonetheless, it must be addressed or else businesses and researchers will not entrust their data to grid computing.
- ***Accommodating heterogeneity.*** A grid system must support interoperability between heterogeneous hardware and software platforms. Ideally, a running application should be able to migrate from platform to platform if necessary. At a bare minimum, components running on different platforms must be able to communicate transparently.
- ***Binary management and application provisioning.*** The underlying system should keep track of executables and libraries, knowing which ones are current, which ones are used with which persistent states, where they have been installed and where upgrades should be installed. These tasks reduce the burden on the programmer.
- ***Multi-language support.*** Diverse languages will always be used and legacy applications will need support.
- ***Scalability.*** There are over 400 million computers in the world today and over 100 million network-attached devices (including computers). Scalability is clearly a critical necessity. Any architecture relying on centralized resources is doomed to failure. A successful grid architecture must adhere strictly to the distributed systems principle: the service demanded of any given component must be independent of the number of components in the system. In other words, the service load on any given component must not increase as the number of components increases.
- ***Persistence.*** I/O and the ability to read and write persistent data are critical in order to communicate between applications and to save data. However, the current files/file libraries paradigm should be supported, since it is familiar to programmers.
- ***Extensibility.*** Grid systems must be flexible enough to satisfy current user demands and unanticipated future needs. Therefore, we feel that mechanism and policy must be realized by replaceable and extensible components, including (and especially) core system components. This model facilitates development of improved implementations that provide value-added services or site-specific policies while

enabling the system to adapt over time to a changing hardware and user environment.

- ***Site autonomy.*** Grid systems will be composed of resources owned by many organizations, each of which desires to retain control over its own resources. The owner of a resource must be able to limit or deny use by particular users, specify when it can be used, etc. Sites must also be able to choose or rewrite an implementation of each Legion component as best suits their needs. If a given site trusts the security mechanisms of a particular implementation it should be able to use that implementation.
- ***Complexity management***. Finally, but importantly, complexity management is one of the biggest challenges in large-scale grid systems. In the absence of system support, the application programmer is faced with a confusing array of decisions. Complexity exists in multiple dimensions: heterogeneity in policies for resource usage and security, a range of different failure modes and different availability requirements, disjoint namespaces and identity spaces, and the sheer number of components. For example, professionals who are not IT experts should not have to remember the details of five or six different file systems and directory hierarchies (not to mention multiple user names and passwords) in order to access the files they use on a regular basis. Thus, providing the programmer and system administrator with clean abstractions is critical to reducing their cognitive burden.

To address these basic grid requirements we developed the Legion architecture and implemented an instance of that architecture, the Legion run-time system [17] [18] [25]. The architecture and implementation were guided by the following design principles that were applied at every level throughout the system:

- ***Provide a single-system view.*** With today's operating systems we can maintain the illusion that our local area network is a single computing resource. But once we move beyond the local network or cluster to a geographically-dispersed group of sites, perhaps consisting of several different types of platforms, the illusion breaks down. Researchers, engineers and product development specialists (most of whom do not want to be experts in computer technology) are forced to request access through the appropriate gatekeepers, manage multiple passwords, remember multiple protocols for interaction, keep track of where everything is located, and be aware of specific platform-dependent limitations (e.g., this file is too big to copy or to transfer to that system; that application runs only on a certain type of computer, etc.). Re-creating the illusion of single computing resource for heterogeneous, distributed resources reduces the complexity of the overall system and provides a single namespace.
- ***Provide transparency as a means of hiding detail.*** Grid systems should support the traditional distributed system transparencies: access, location, heterogeneity, failure, migration, replication, scaling, concurrency and behavior. For example, users and programmers should not have to know where an object is located in order to use it (access, location and migration transparency), nor should they need to know that a component across the country failed – they want the system to recover automatically and complete the desired task (failure transparency). This behavior is the traditional way to mask details of the underlying system.
- ***Provide flexible semantics.*** Our overall objective was a grid architecture that is suitable to as many users and purposes as possible. A rigid system design in which policies are limited, trade-off decisions are pre-selected, or all semantics are pre-determined and hard-coded would not achieve this goal. Indeed, if we dictated a

single system-wide solution to almost any of the technical objectives outlined above, we would preclude large classes of potential users and uses. Therefore, Legion allows users and programmers as much flexibility as possible in their applications' semantics, resisting the temptation to dictate solutions. Whenever possible, users can select both the *kind* and the *level* of functionality and choose their own trade-offs between function and cost. This philosophy is manifested in the system architecture. The Legion object model specifies the functionality but not the implementation of the system's core objects; the core system therefore consists of extensible, replaceable components. Legion provides default implementations of the core objects, although users are not obligated to use them. Instead, we encourage users to select or construct object implementations that answer their specific needs.

- ***Reduce user effort.*** In general, there are four classes of grid users who are trying to accomplish some mission with the available resources: end-users of applications, applications developers, system administrators and managers. We believe that users want to focus on their jobs, e.g., their applications, and not on the underlying grid plumbing and infrastructure. Thus, for example, to run an application a user may type

  legion_run my_application my_data

  at the command shell. The grid should then take care of messy details such as finding an appropriate host on which to execute the application, moving data and executables around, etc. Of course, the user may optionally be aware and specify or override certain behaviors, for example, specify an OS on which to run the job, or name a specific machine or set of machines, or even replace the default scheduler.
- ***Reduce "activation energy".*** One of the typical problems in technology adoption is getting users to use it. If it is difficult to shift to a new technology then users will tend not to take the effort to try it unless their need is immediate and extremely compelling. This is not a problem unique to grids – it is human nature. Therefore, one of our most important goals was to make using the technology easy. Using an analogy from chemistry, we kept the activation energy of adoption as low as possible. Thus, users can easily and readily realize the benefit of using grids – and get the reaction going – creating a self-sustaining spread of grid usage throughout the organization. This principle manifests itself in features such as "no recompilation" for applications to be ported to a grid and support for mapping a grid to a local operating system file system. Another variant of this concept is the motto "no play, no pay". The basic idea is that if you do not need a feature, e.g., encrypted data streams, fault resilient files or strong access control, you should not have to pay the overhead of using it.
- ***Do no harm.*** To protect their objects and resources, grid users and sites will require grid software to run with the lowest possible privileges.
- ***Do not change host operating systems.*** Organizations will not permit their machines to be used if their operating systems must be replaced. Our experience with Mentat [8] indicates, though, that building a grid on top of host operating systems is a viable approach. Furthermore, Legion must be able to run as a user level process, and not require root access.

Overall, the application of these design principles *at every level* provides a unique, consistent, and extensible framework upon which to create grid applications.

3.2 Globus

The philosophy underlying Globus can be found in the technical literature available on Globus as well as several public pronouncements by key members of the design team. In this section, we describe the Globus philosophy by providing a similar structure to the previous section, which described the Legion philosophy. Again, we will focus on GT2, not GT3, which is under development at the time of writing. We will first discuss the Globus requirements (using the same list as in Section 3.1) and then discuss Globus principles. In this section, we will show that there are many differences between Legion and Globus, both in the overall design principles and the emphasis and approach to each of the requirements. The requirements are:

- ***Security.*** Security is an important facet of the Globus approach; both Legion and Globus have recognized and agreed that there will be diverse security mechanisms and policies in any grid. Globus addresses authentication and data integrity, but intentionally does not define an authorization model. Instead, Globus explicitly defers authorization to the underlying operating system.
- ***Global name space.*** Naming is a key philosophical difference between Legion and Globus. The Globus approach is that a global name space is not a requirement for grids; rather, the combination of local naming mechanisms (e.g., UNIX file system and UNIX process IDs), URLs (for locating remote files), IP addresses/DNS (for naming remote resources), and DNs (for humans) is sufficient. Location transparency is not a goal.
- ***Fault tolerance.*** While both recognize that the "system" must be fault-tolerant, the projects differ on the degree to which the grid infrastructure itself masks the failures/errors. Globus focuses on low-level protocols for grid computing, arguing that the creation of robust, core low-level protocols enables *other projects* to create higher-level tools and protocols which will mask faults. Therefore, the Globus approach is not necessarily to implement, but to facilitate, fault–tolerance.
- ***Accommodating heterogeneity.*** Both projects agree that accommodating heterogeneity is a fundamental requirement.
- ***Binary management and application provisioning.*** This requirement is viewed as a higher-level function, therefore not part of the Globus toolkit directly.
- ***Multi-language support.*** Both projects agree that multiple languages must be supported.
- ***Scalability.*** While both projects recognize the need for scalability, the mechanisms and/or focus of scalability differ significantly. Since Legion provides more high-level services than Globus (such as a grid-enabled distributed file system, an integrated grid scheduler for all grid activities, etc.), the Legion developers have had to directly face the issues of scalability perhaps more often than the Globus developers. Since the focus with the Globus toolkit has been "closer to the hardware", in many ways the scalability of Globus is a direct result of the scalability of the Internet. However, clearly both projects believe that scalability is an important issue.
- ***Persistence.*** Persistence, much like scalability, is achieved in different ways in Legion and Globus. Generally, the approach in Legion is that persistence in grids requires grid-specific mechanisms, whereas persistence in Globus is largely achieved through non-grid-specific means, such as *inetd.* When a Globus user "logs onto" the grid, he realizes that he can access prior days' data because he knows where he left it, and he knows that "gridftpd" is available there.
- ***Extensibility.*** The toolkit approach explicitly allows for extensibility – as new requirements are identified, new toolkit components can be created. Arguably, the design of the toolkit itself is *not* for an individual user to be able to choose a particular component in the toolkit and re-implement or customize it for the user's individual requirements. In Legion, every functionality is intentionally extensible through the object-based design. In practice, this extensibility is achieved by means of operator

overloading, inheritance, and (republishing) a new, open interface if the default implementation is not sufficient.

- ***Site autonomy.*** Both projects strongly agree on the need for strong site autonomy. In fact, both projects have devoted a significant amount of time arguing that "being part of a grid" does not mean that a user can execute anything they want on any site. Rather, it is crucial that sites not have to give up any of their rights in order to participate in a grid.
- ***Complexity management***. It is not clear to what extent Globus incorporates complexity management as a fundamental requirement of the grid infrastructure. For example, Globus users are expected to remember a lot more than Legion users, for example, where their computations are currently executing, where their files are, etc. In other words, if a Globus user cannot remember where certain files are, it is not clear how she might find them. In contrast, a Legion user is presented a location-independent, distributed file system abstraction, which is searchable. Although the Legion user can find the physical location if she wants, we argue that the physical location is *not actually* what the user wants anyway! Abstracting away the physical location of a resource is part of complexity management.

Given that there are differences with regard to the focus and attainability of the requirements, it is not surprising that the guiding principles are different for the Legion project and the Globus project. From our interpretation, the philosophy of Globus is based on the following principles:

- ***Provide a "toolkit" from which users can pick and choose.*** A set of architectural principles is not necessary, because it ultimately restricts the development of "independent solutions to independent problems." Similarly, having all components share some common mechanisms and protocols (perhaps above IP) restricts individual development and the pick-and-choose deployment possibilities. By contrast, the Legion developers believe that there is a core set of protocols that are fundamental to most "grid services"; as such, most new components in Legion should to some degree re-use the core protocols and functionality. This is perhaps the single biggest difference in philosophies between the two projects.
- ***Focus on low-level functionality, thereby facilitating high-level tools (and general usability).*** Globus is based on the principles of the "hourglass model". In the Globus architecture, the neck of the hourglass consists of the *Resource* and *Connectivity* protocols. Since the lower-level protocols are so critical to the success of the grid, the focus within the core Globus project itself is on these protocols. Other projects can then build higher-level services, such as a File Replica Manager and grid schedulers. In Legion, we believe that these higher-level functionalities are absolutely critical for the *usability* of the grid, so we provide default implementations (which can be replaced) of many of these higher-level functionalities.
- ***Use standards whenever possible for both interfaces and implementations.*** The Globus developers have been very diligent in identifying candidate existing software or protocols that might be appropriate for grid computing, albeit with some necessary modifications. Key examples are a reliance on FTP for data movement, PKI as an authentication infrastructure (and *gssapi* as an API for authentication), *OpenSSL* for low-level cryptographic routines, and LDAP as the basis for information services. In contrast, Legion uses existing standards whenever possible; however, we have decided that certain standards such as *gssapi* are not necessarily appropriate, either because of limited endorsement outside the grid community, because their complexity was judged to be not worth the potential value, or because

ultimately there was not a smooth fit (even with modifications) between the interface/protocol and the unique, heterogeneous, dynamic nature of grids.

- ***Emphasize the identification and definition of protocols and services first, and APIs and SDKs next.*** The contribution of Globus, in some sense, is not the software but rather the protocol: “A standards-based open architecture facilitates extensibility, interoperability, portability, and code sharing; standard protocols make it easy to define services that provide enhanced capabilities” [50]. Protocols are essential for interoperability. In contrast, the Legion emphasis has been, arguably, on the software itself – while we believe that the success of this approach is our ability to deliver a highly-usable software product, historically we have not placed a strong emphasis on direct interoperability with other grid approaches (this emphasis is changing, particularly within Avaki).
- ***Provide open-source community development.*** Recognizing the tremendous impact of the open-source movement, particularly Linux, Globus has always strongly endorsed an open-source community development for the Globus toolkit. Legion has been open-source for much of its development; however, we have generally found that in practice people would rather have deployable binary versions rather than source code itself.
- ***Provide immediate usefulness.*** Legion required a sophisticated, working “core functionality” that would be utilized in many of the grid services. As such, it was very difficult to deliver *only pieces* of a grid solution to the community, so we could not provide immediate usefulness to the community without the entire product. In contrast, Globus recognized that certain short-term problems (such as single sign-on) would largely be solved by a small number of software artifacts. The result was that Globus provided immediate usefulness to the grid community. Other examples of this kind of immediate usefulness include: for computation, focusing on high-performance application requirements; for data, focusing on replicated, large datasets essentially accessible by FTP, downplaying the importance of non-local access to "small" files; focusing on authentication instead of authorization; treating “computation” differently from “data”, and promoting separate “computational grids” and “data grids”.
- ***Do NOT provide a “virtual machine” abstraction.*** Apparently the virtual machine abstraction was considered in the early days of the Globus project. However, the virtual machine abstraction was viewed as an inappropriate model, “inconsistent with our primary goals of broad deployment and interoperability”.[4] Additionally, the “traditional transparencies are unobtainable” [50]. In contrast, in the Legion project, the virtual machine is *precisely* what is needed to mask complexity in the environment. This is a fundamental difference in approaches in the two projects.

Overall, while arguably the requirements are equivalent for both Legion and Globus, the emphasis on these requirements within each project are quite different. More importantly, the principles of each project as discussed in this section clearly indicate the stark differences in how to best achieve the goals. In the next section, we provide more details on how each project attempts to satisfy the requirements of grid computing.

[4] This principle of NOT providing a virtual machine abstraction has been a consistent theme in the design of Globus Toolkit 2 (GT2). In Globus Toolkit 3 (GT3), this theme has been discarded in favor of a container model that is essentially a virtual machine.

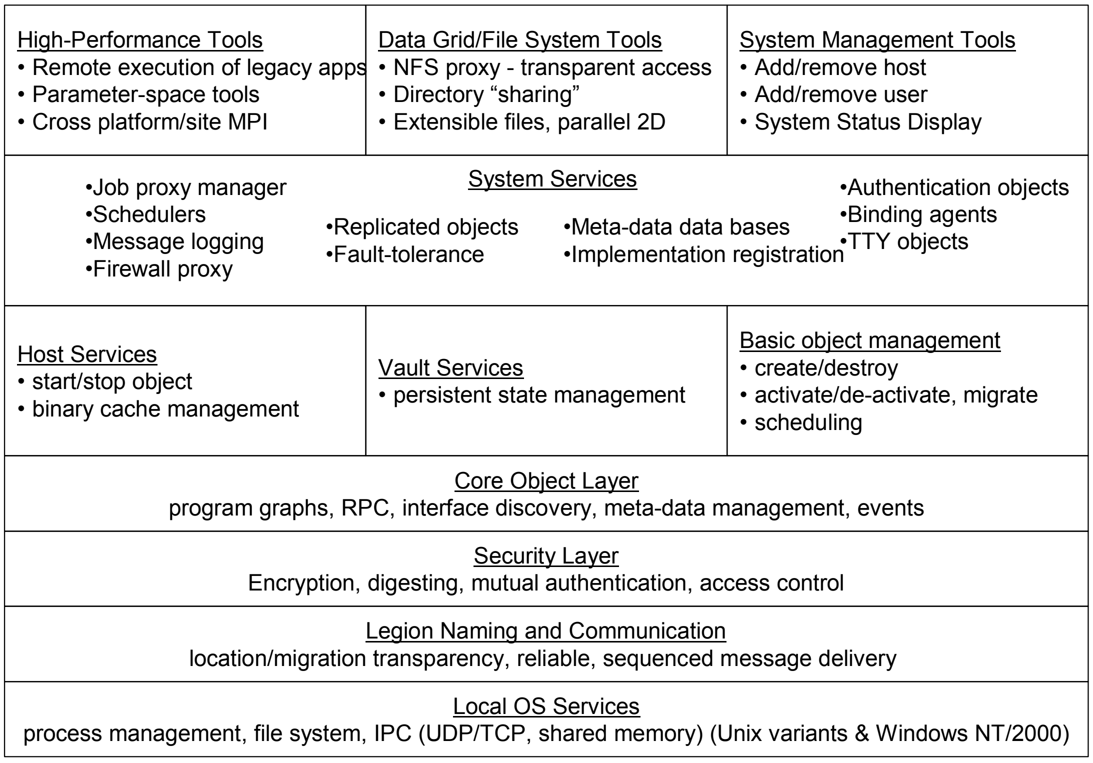
4 Architectural Details

As is obvious from the previous sections, Legion and Globus overlap significantly in their goals. In fact, the overlap between these projects is far greater than the overlap between either and any of the queuing systems or parallel systems mentioned in Section 2. The philosophical and architectural differences between Legion and Globus are significant, as we discussed in Section 3. In this section we will contrast those differences further by providing details on the implementation of each system.

4.1 Legion Architectural Details

Legion started out with the "top-down" premise that a strong architecture is necessary to build an infrastructure of this scope. This architecture is based on communicating services. These services are implemented as objects. An object is a stateful component in a container process. Much of the initial design time spent in Legion was in determining the underlying infrastructure and the set of core services over which grid services could be built [1] [6] [9] [31]. Tasks involved in this design included:

- designing and implementing a three-level naming, binding, and communication scheme. Naming is crucial in distributed systems and grids. The Legion scheme includes human-readable names such as attributes and directory-based pathnames to name objects – abstract names that do not include any location, implementation, or “number” information and concrete object addresses,
- building a remote method invocation framework based on dataflow graphs,
- adding a security layer to every communication instance between any services,
- adding fault-tolerance layers, and
- adding layers for resource discovery through naming and adding mechanisms for caching name bindings (examples of bindings being IP addresses and ports for processes corresponding to services).



High-Performance Tools
• Remote execution of legacy apps
• Parameter-space tools
• Cross platform/site MPI
Data Grid/File System Tools
• NFS proxy - transparent access
• Directory “sharing”
• Extensible files, parallel 2D
System Management Tools
• Add/remove host
• Add/remove user
• System Status Display
System Services
•Job proxy manager
•Schedulers
•Message logging
•Firewall proxy
•Replicated objects
•Fault-tolerance
•Meta-data data bases
•Implementation registration
•Authentication objects
•Binding agents
•TTY objects
Host Services
• start/stop object
• binary cache management
Vault Services
• persistent state management
Basic object management
• create/destroy
• activate/de-activate, migrate
• scheduling
Core Object Layer
program graphs, RPC, interface discovery, meta-data management, events
Security Layer
Encryption, digesting, mutual authentication, access control
Legion Naming and Communication
location/migration transparency, reliable, sequenced message delivery
Local OS Services
process management, file system, IPC (UDP/TCP, shared memory) (Unix variants & Windows NT/2000)

Figure 1. The Legion architecture viewed as a series of layers.

In Figure 1, we show a layered view of the Legion architecture. The bottom layer is the local operating system – or execution environment layer. This corresponds to true operating systems such as Linux, AIX, Windows 2000, etc. The bottom layer along with parts of the layers above are also addressed by containers in hosting environments such as J2EE. We depend on process management services, file system support, and inter-process communication services delivered by the bottom layer, e.g., UDP, TCP or shared memory. Above the local operating services layer we build the Legion communications layer. This layer is responsible for object naming and binding as well as delivering sequenced arbitrarily-long messages from one object to another. Delivery is accomplished regardless of the location of the two objects, object migration, or object failure. For example, object A can communicate with object B even while object B is migrating from Charlottesville to San Diego, or even if object B fails and subsequently restarts. This is possible because of Legion's three-level naming and binding scheme, in particular the lower two levels.

The lower two levels consist of location-independent abstract names called LOIDs (Legion Object IDentifiers) and object addresses specific to communication protocols, e.g., an IP address and a port number. The binding between a LOID and an object address can, and does, change over time. Indeed it is possible for there to be no binding for a particular LOID at some times if, for example, the object is not running currently. Maintaining the bindings at run-time in a scalable way is one of the most important aspects of the Legion implementation [18].

Next is the security layer on the core object layers. The security layer implements the Legion security model [2] [15] for authentication, access control, and data integrity (e.g., mutual authentication and encryption on the wire). The core object layer [17] [18] [25] addresses method invocation, event processing (including exception and error propagation on a per-object basis [26]), interface discovery and the management of meta-data. Objects can have arbitrary meta-data, such as the load on a host object or the parameters that were used to generate a particular data file.

Above the core object layer are the core services [18] that implement object instance management (*class managers*) and abstract processing resources (*hosts*) and storage resources (*vaults*). These are represented by base classes that can be extended to provide different, or enhanced implementations. For example, the *host* class represents processing resources. It has methods to start an object given a LOID, a persistent storage address, and the LOID of an implementation to use, stop an object given a LOID, kill an object, and so on. There are derived classes for UNIX and Windows called *UnixHost* and *NTHost* that use UNIX processes and Windows spawn respectively. There are also derived classes that interact with backend queuing systems, *BatchQueueHost*, and that require the user to have a local account and run as that user, *PCDHost* [3] [15]. There are similar sets of derived types for vaults and class managers that implement policies (for example replicated objects for fault-tolerance, or stateless objects for performance and fault-tolerance [24] [26]) and interact with different resource classes.

Above these basic object management services are a whole collection of higher-level system service types and enhancements to the base service classes. These include classes for object replication for availability [26], message logging classes for accounting or post-mortem debugging, firewall proxy servers for securely transiting firewalls, enhanced schedulers [1], databases called collections that maintain information on the attributes associated with objects (these are used extensively in scheduling), job proxy managers that "wrap" legacy codes for remote execution [19] [23], and so on.

Finally, an application support layer contains user-centered tools for parallel and high-throughput computing, data access and sharing, and system management.

In the high-performance tool set there are tools to wrap legacy codes (*legion_register_program*) and execute them remotely (*legion_run*) both singly and in large sets (as in a parameter-space study [23]). Legion MPI tools support cross-platform, cross-site execution of MPI programs [21], and BFS (Basic Fortran Support) [4] tools wrap Fortran programs for running on a grid.

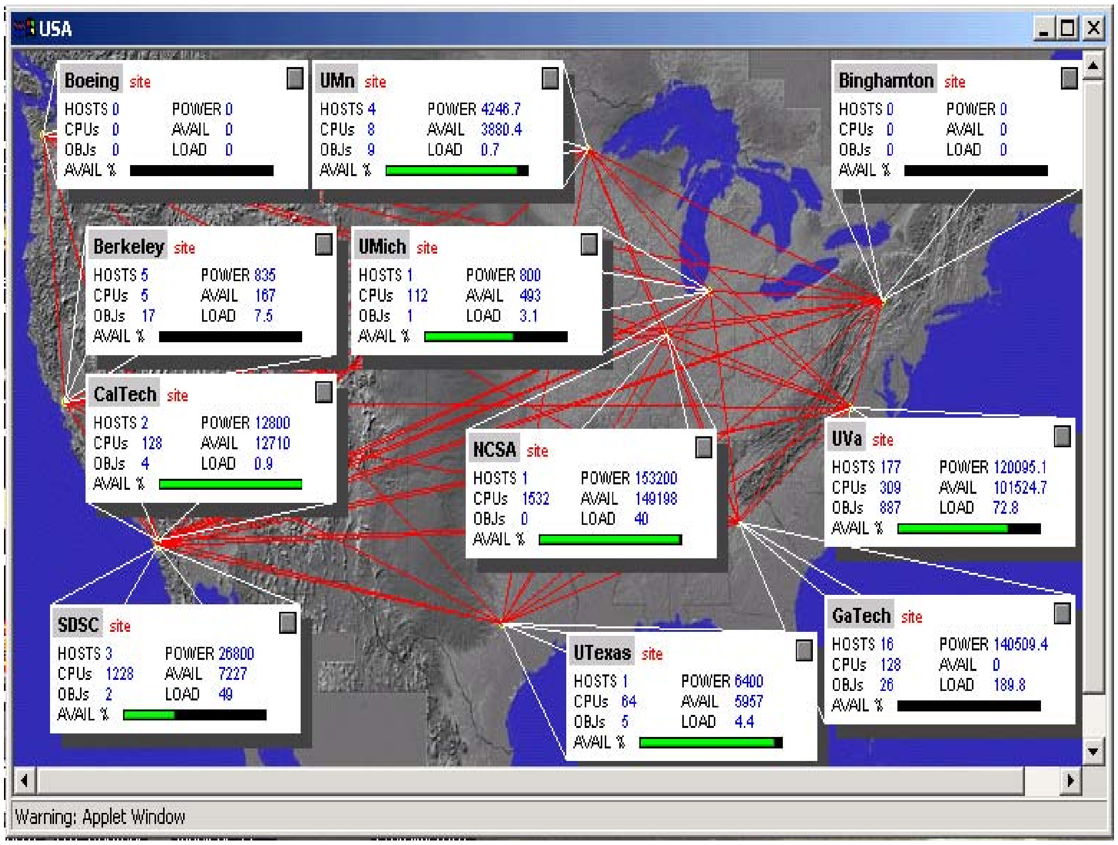
Legion's integrated data grid support is focused on both extensibility and reducing the burden on the programmer [28] [29] [32]. In terms of extensibility there is a basic file type (basicfile) that supports the usual functions – read, write, stat, seek, etc. All other file types are derived from this type. Thus, no matter the file type it can still be treated as a basic file, and for example piped into tools that expect sequential files. There are 2D files that support read/write operations on columns, rows, and rectangular patches of data (both primitive types as well as "structs"). There are file types to support unstructured sparse data, as well as parallel files where the file has been broken up and decomposed across several different storage systems.

Data can either be copied into the grid, in which case Legion manages the data, deciding where to place it, how many copies to generate for higer availability, and where to place those copies. Alternatively, data can be "exported" into the grid. When a local directory structure is exported into the grid it is mapped to a chosen path name in the global name space (directory structure). For example, a user can map data/sequences in his local Unix file system into /home/grimshaw /sequences using the legion_export_dir command, *legion_export_dir data/sequences /home/grimshaw/sequences*. Subsequent access from anywhere in the grid (whether read or write) are done directly against the files in the user's Unix file system (subject to access control of course.)

To simplify ease of use, the data grid can be accessed via a daemon that implements the NFS protocol. Therefore, the entire Legion namespace, including files, hosts, etc., can be mapped into local operating system file systems. Thus, shell scripts, Perl scripts, and user applications can run unmodified on the Legion data grid. Futher, the usual Unix commands such as "ls" work, as does the Windows browser.

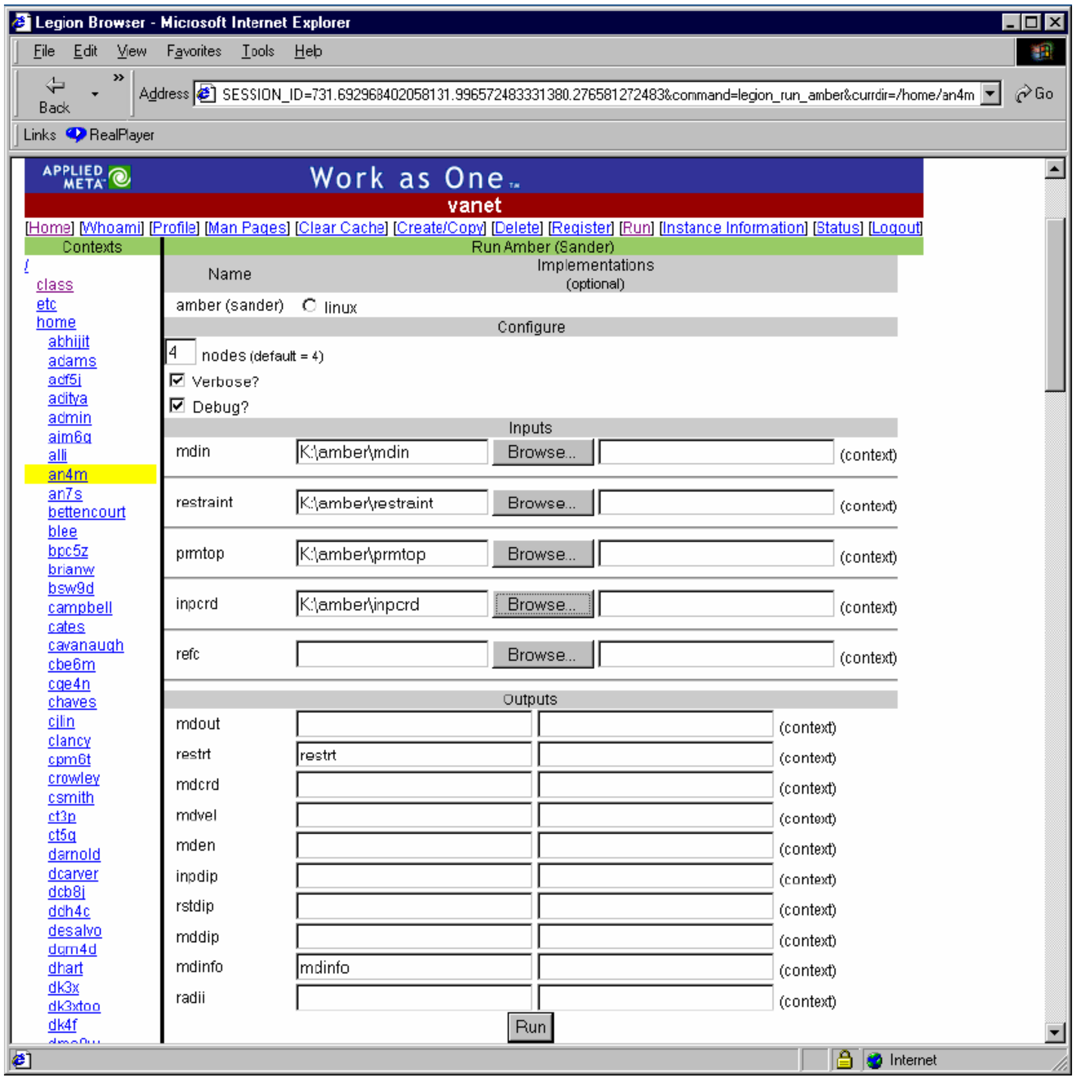
Finally, there are the user portal and system management tools to add and remove users, add and remove hosts, and join two separate Legion grids together to create a grid of grids, etc. There is a web-based portal interface for access to Legion [22], as well as a system status display tool that gathers information from system-wide meta data collections and makes it available via a browser (see Figure 2 for a screen-shot).

The web-based portal (Figures 3-6) allows an alternative, graphical interface to Legion. Using this interface, a user can submit an Amber job (a 3D molecular modeling code) to NPACI-Net and not care where it executes at all. In Figure 4, we show the portal view of the intermediate output, where the user can copy files out of the running simulation, and in which a 3D molecular visualization plug-in called Chime is being used to display the intermediate results.



```
Boeing site
HOSTS 0     POWER 0
CPUs  0     AVAIL 0
OBJs  0     LOAD  0
AVAIL %

UMn site
HOSTS 4     POWER 4246.7
CPUs  8     AVAIL 3880.4
OBJs  9     LOAD  0.7
AVAIL %

Binghamton site
HOSTS 0     POWER 0
CPUs  0     AVAIL 0
OBJs  0     LOAD  0
AVAIL %

Berkeley site
HOSTS 5     POWER 835
CPUs  5     AVAIL 167
OBJs  17    LOAD  7.5
AVAIL %

UMich site
HOSTS 1     POWER 800
CPUs  112   AVAIL 493
OBJs  1     LOAD  3.1
AVAIL %

CalTech site
HOSTS 2     POWER 12800
CPUs  128   AVAIL 12710
OBJs  4     LOAD  0.9
AVAIL %

NCSA site
HOSTS 1     POWER 153200
CPUs  1532  AVAIL 149198
OBJs  0     LOAD  40
AVAIL %

UVa site
HOSTS 177   POWER 120095.1
CPUs  309   AVAIL 101524.7
OBJs  887   LOAD  72.8
AVAIL %

SDSC site
HOSTS 3     POWER 26800
CPUs  1228  AVAIL 7227
OBJs  2     LOAD  49
AVAIL %

UTexas site
HOSTS 1     POWER 6400
CPUs  64    AVAIL 5957
OBJs  5     LOAD  4.4
AVAIL %

GaTech site
HOSTS 16    POWER 140509.4
CPUs  128   AVAIL 0
OBJs  26    LOAD  189.8
AVAIL %

Warning: Applet Window
```

Figure 2. Legion system monitor running on NPACI-Net in 2000 with the “USA” site selected. Clicking on a site opens a window for that site, with the individual resources listed. Sites can be composed hierarchically. Those resources in turn can be selected, and then individual objects are listed. “POWER” is a function of individual CPU clock rates and the number and type of CPUs. “AVAIL” is a function of CPU power and current load; it is what is available for use.


Legion Browser - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address
Go
Links RealPlayer
APPLIED META
Work as One
vanet
[Home] [Whoami] [Profile] [Man Pages] [Clear Cache] [Create/Copy] [Delete] [Register] [Run] [Instance Information] [Status] [Logout]
Contexts
Internet

SESSION_ID=731.692968402058131.996572483331380.276581272483&command=legion_run_amber&currdir=/home/an4m

Contexts: / class etc home abhijit adams adf5j aditya admin aim6q alli an4m an7s bettencourt blee bpc5z brianw bsw9d campbell cates cavanaugh cbe6m cge4n chaves cjlin clancy cpm6t crowley csmith ct3p ct5q darnold dcarver dcb8j dch4c desalvo dem4d dhart dk3x dk3xtoo dk4f

Run Amber (Sander)

Name	Implementations (optional)
amber (sander)	○ linux

Configure

4 nodes (default = 4)

☑ Verbose?

☑ Debug?

Inputs

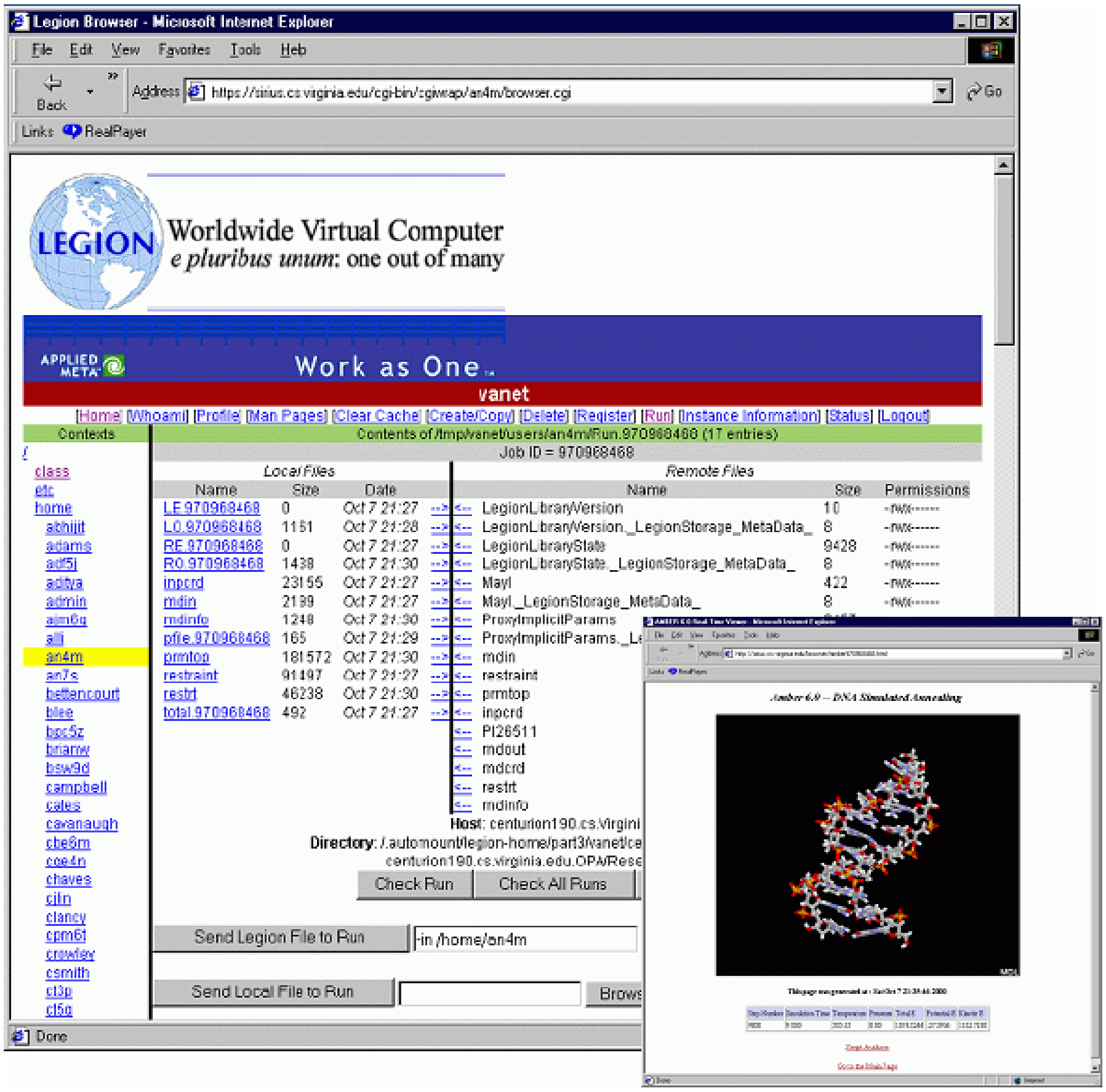
mdin	K:\amber\mdin	Browse..		(context)
restraint	K:\amber\restraint	Browse..		(context)
prmtop	K:\amber\prmtop	Browse..		(context)
inpcrd	K:\amber\inpcrd	Browse..		(context)
refc		Browse..		(context)

Outputs

mdout			(context)
restrt	restrt		(context)
mdcrd			(context)
mdvel			(context)
mden			(context)
inpdip			(context)
rstdip			(context)
mddip			(context)
mdinfo	mdinfo		(context)
radii			(context)

Run

Figure 3. A job submission window for Amber using the Legion web portal.


Legion Browser - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address https://sirius.cs.virginia.edu/cgi-bin/cgiwrap/an4m/browser.cgi
Go
Links RealPlayer
LEGION
Worldwide Virtual Computer
e pluribus unum: one out of many
APPLIED META
Work as One
vanet
[Home] [Whoami] [Profile] [Man Pages] [Clear Cache] [Create/Copy] [Delete] [Register] [Run] [Instance Information] [Status] [Logout]
Contexts
/
class
etc
home
abhijit
adams
adf5j
aditya
admin
ajm6q
alli
an4m
an7s
bettencourt
blee
bpc5z
brianw
bsw9d
campbell
cales
cavanaugh
cbe6m
cce4n
chaves
cjln
clancy
cpm6t
crowley
csmith
ct3p
ct5q
Check Run
Check All Runs
Send Legion File to Run
Send Local File to Run
Brows
Done
Amber 6.0 – DNA Simulated Annealing

Contents of /tmp/vanet/users/an4m/Run.970968468 (17 entries)

Job ID = 970968468

Local Files

Name	Size	Date
LE.970968468	0	Oct 7 21:27
L0.970968468	1151	Oct 7 21:28
RE.970968468	0	Oct 7 21:27
R0.970968468	1438	Oct 7 21:30
inpcrd	23155	Oct 7 21:27
mdin	2199	Oct 7 21:27
mdinfo	1248	Oct 7 21:30
pfile.970968468	165	Oct 7 21:29
prmtop	181572	Oct 7 21:30
restraint	91497	Oct 7 21:27
restrt	46238	Oct 7 21:30
total.970968468	492	Oct 7 21:27

Remote Files

Name	Size	Permissions
LegionLibraryVersion	10	-rwx------
LegionLibraryVersion._LegionStorage_MetaData_	8	-rwx------
LegionLibraryState	9428	-rwx------
LegionLibraryState._LegionStorage_MetaData_	8	-rwx------
Mayl	422	-rwx------
Mayl._LegionStorage_MetaData_	8	-rwx------
ProxyImplicitParams		
ProxyImplicitParams._Le		
mdin		
restraint		
prmtop		
inpcrd		
PI26511		
mdout		
mdcrd		
restrt		
mdinfo		

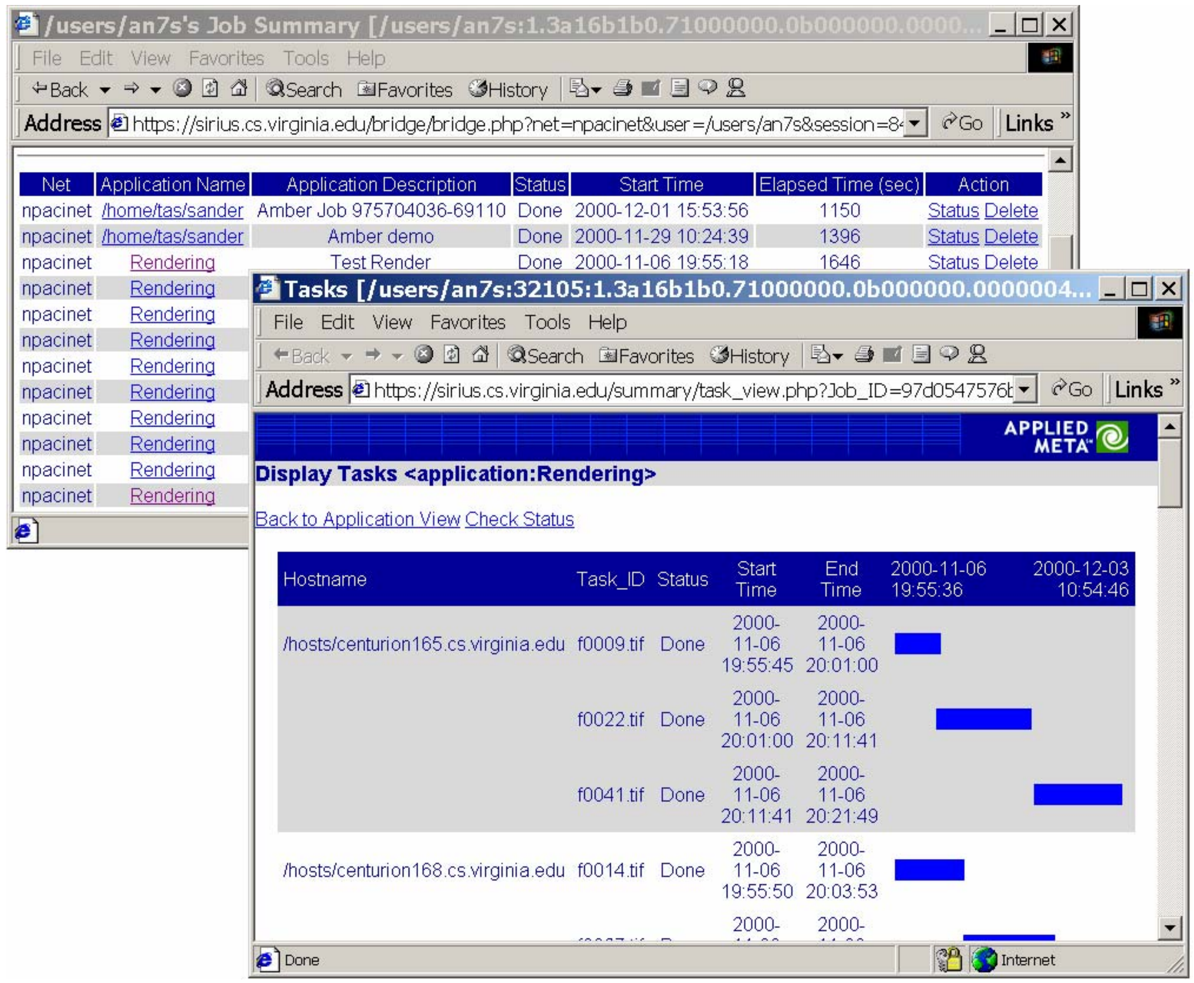
Host: centurion190.cs.Virgini

Directory: /.automount/legion-home/part3/vanet/ce
centurion190.cs.virginia.edu.OPA/Rese

-in /home/an4m

Figure 4. Updated application status from Legion and molecule status from Chime.

In Figure 5, we display the Legion job status tools. Using these tools the user can determine the status of all of the jobs that they have started from the Legion portal – and access their results as needed.


/users/an7s's Job Summary [/users/an7s:1.3a16b1b0.71000000.0b000000.0000...
File Edit View Favorites Tools Help
Back Search Favorites History
Address https://sirius.cs.virginia.edu/bridge/bridge.php?net=npacinet&user=/users/an7s&session=8
Go Links
Tasks [/users/an7s:32105:1.3a16b1b0.71000000.0b000000.0000004...
File Edit View Favorites Tools Help
Back Search Favorites History
Address https://sirius.cs.virginia.edu/summary/task_view.php?Job_ID=97d0547576
Go Links
APPLIED META
Done
Internet

Net	Application Name	Application Description	Status	Start Time	Elapsed Time (sec)	Action
npacinet	/home/tas/sander	Amber Job 975704036-69110	Done	2000-12-01 15:53:56	1150	Status Delete
npacinet	/home/tas/sander	Amber demo	Done	2000-11-29 10:24:39	1396	Status Delete
npacinet	Rendering	Test Render	Done	2000-11-06 19:55:18	1646	Status Delete
npacinet	Rendering					
npacinet	Rendering					
npacinet	Rendering					
npacinet	Rendering					
npacinet	Rendering					
npacinet	Rendering					
npacinet	Rendering					
npacinet	Rendering					
npacinet	Rendering					

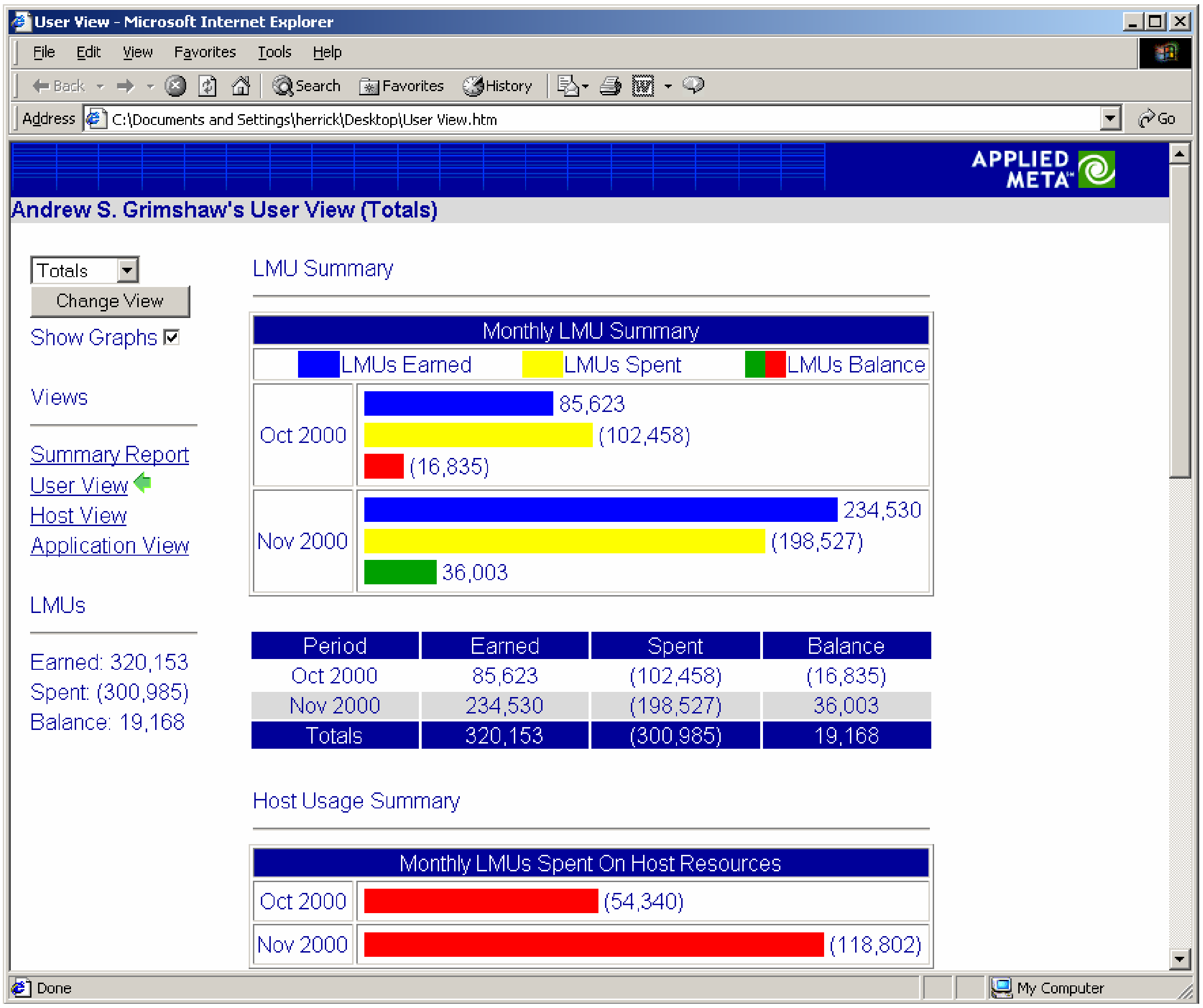
Display Tasks <application:Rendering>

Back to Application View Check Status

Hostname	Task_ID	Status	Start Time	End Time	2000-11-06 19:55:36	2000-12-03 10:54:46
/hosts/centurion165.cs.virginia.edu	f0009.tif	Done	2000-11-06 19:55:45	2000-11-06 20:01:00		
	f0022.tif	Done	2000-11-06 20:01:00	2000-11-06 20:11:41		
	f0041.tif	Done	2000-11-06 20:11:41	2000-11-06 20:21:49		
/hosts/centurion168.cs.virginia.edu	f0014.tif	Done	2000-11-06 19:55:50	2000-11-06 20:03:53		
			2000-	2000-		

Figure 5. Legion job status tools accessible via the web portal.

In Figure 6, we show the portal interface to the underlying Legion accounting system. We believed from very early on that grids must have strong accounting or they will be subject to the classic tragedy of the commons, in which everyone is willing to use grid resources, yet no one is willing to provide them. Legion keeps track of who used which resource (CPU, application, etc.), starting when, ending when, with what exit status, and with how much resource consumption. The data is loaded into a relational DBMS and various reports can be generated. (An LMU is a "Legion Monetary Unit", it is one CPU second normalized by the clock speed of the machine.)


User View - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address C:\Documents and Settings\herrick\Desktop\User View.htm
Go
APPLIED META
Totals
Change View
Done
My Computer

Andrew S. Grimshaw's User View (Totals)

Show Graphs ☑

Views

Summary Report
User View
Host View
Application View

LMUs

Earned: 320,153
Spent: (300,985)
Balance: 19,168

LMU Summary

Monthly LMU Summary

🟦 LMUs Earned 🟨 LMUs Spent 🟩🟥 LMUs Balance

Oct 2000	85,623 (102,458) (16,835)
Nov 2000	234,530 (198,527) 36,003

Period	Earned	Spent	Balance
Oct 2000	85,623	(102,458)	(16,835)
Nov 2000	234,530	(198,527)	36,003
Totals	320,153	(300,985)	19,168

Host Usage Summary

Monthly LMUs Spent On Host Resources

Oct 2000	(54,340)
Nov 2000	(118,802)

Figure 6. Legion accounting tool. Units are normalized CPU seconds. Displays can be organized by user, machine, site or application.

4.2 Globus Architectural Details

Globus started out with the "bottom-up" premise that a grid must be constructed as a set of tools developed from user requirements. This architecture is based on composing tools from a kit. Consequently, much of the initial design time spent in Globus was in determining the user requirements for which grid tools could be built. Tasks involved in this design included building a resource manager to start jobs (assuming users had procured beforehand accounts on all the machines on which they could possibly run), a tool and API for transferring files from one machine to another (used for binary and data transfer), tools for procuring credentials and certificates and a service for collecting resource information about machines on a grid. The designers of Globus have always believed that new services and tools must be added to the existing set in a manner that the members of the new set can be composed in order for a user to get work done. Much of the later development in Globus has been in terms of composing these tools in order to achieve a specific goal.

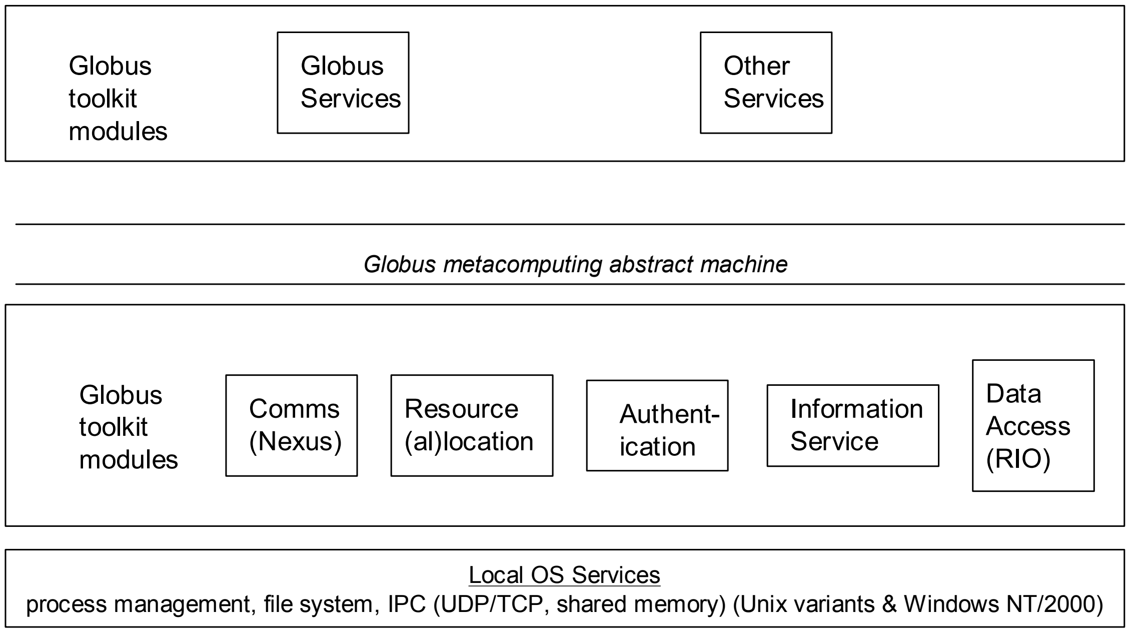
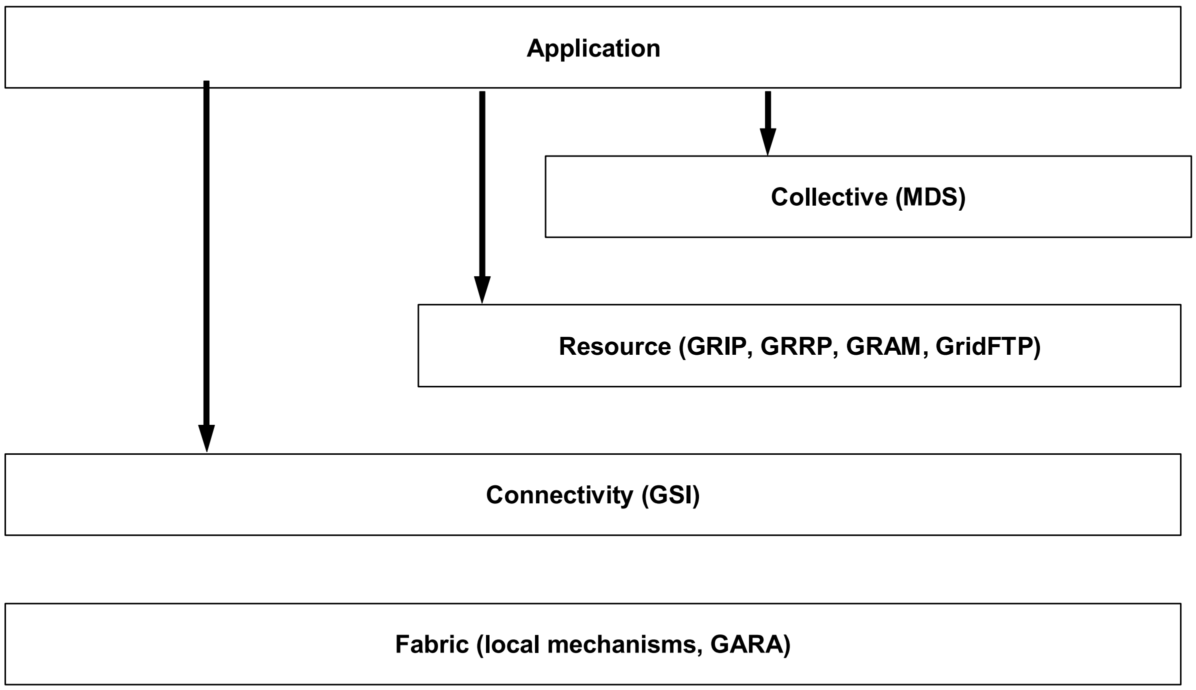


Figure 7. Globus toolkit circa 1997 (adapted from Foster and Kesselman [36])

In Figure 7, we show the early version (1997) of GT2 – the Globus toolkit, version 2 (adapted from Foster and Kesselman [45]). The *communications* module provides network-aware communications messaging capabilities. The implementation of the communications module in the Globus toolkit was called *Nexus* [42]. The *resource location and allocation* module provides mechanisms for expressing application resource requirements for identifying resources that meet these requirements, and for scheduling resources after they have been located. The *authentication* module provides a means by which to verify the identity of both humans and resources. In the Globus toolkit, the GSS-API was utilized in an attempt to be agnostic of the actual underlying authentication technique, be it Kerberos or SSL (PKI). The *information service* module provides a uniform mechanism for obtaining real-time information about metasystem structure and status. The implementation of this module in the Globus toolkit is called the *Metacomputing Directory Service (MDS)* [41], which builds upon the data representation and API of the Lightweight Directory Access Protocol (LDAP). The *data access* module is responsible for providing high-speed remote access to persistent storage such as files. All of these modules lay on top of local OS services, as in Legion. Higher-level services utilize the components of the toolkit. Such higher-level services include parallel programming interfaces (e.g., MPICH/G2).

A more recent description of the Globus toolkit, reflecting the evolution of the approach, is shown in Figure 8 (adapted from Foster, Kesselman and Tuecke [50]). At the lowest layer is the *Grid Fabric Layer,* which provides the resources to which grid protocols mediate access. The Globus developers consider this to be generally lower than the components of the toolkit, with the exception of the Globus Architecture for Reservation and Allocation (GARA). The *connectivity layer* defines the core communication and authentication protocols required for grid-specific network transactions. Included in this layer from the Globus toolkit is the Grid Security Infrastructure (GSI) [46]. Above this layer is the *Resource Layer,* which defines protocols for secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. The Globus toolkit functionality at this level includes a Grid Resource Information Protocol (GRIP), a resource information protocol; the Grid


Application
Collective (MDS)
Resource (GRIP, GRRP, GRAM, GridFTP)
Connectivity (GSI)
Fabric (local mechanisms, GARA)
Figure 8. Globus toolkit circa 2001, in the context of the “Grid Protocol Architecture) (adapted from Foster, Kesselman and Tuecke [45])

Resource Registration Protocol (GRRP), used to register resources with the Grid Index Information Servers; the Grid Resource Access and Management (GRAM) protocol, used to allocate and monitor resources; and GridFTP, which is used for data access. The *Collective Layer* is used to coordinate access to multiple resources, which in terms of the Globus Toolkit, refers to directory services *MDS* (supported by GRRP and GRIP). Finally, grid applications are at the very top.

Overall, the benefits and risks of either approach – top-down versus bottom-up – to software design are well-known. With respect to grids, a bottom-up approach tends to result in early successes simply because the approach targets immediate user requirements. However, a risk with this approach is that the infrastructure may not be able to accommodate changing requirements. Another risk is that this approach may not scale – as the number of tools or services increases, an increasing number of pairwise protocols are necessary in order to ensure that the tools compose seamlessly. In contrast, the risk with a top-down approach is that initial successes are hard to come by because initially, the designers focus on building an infrastructure with little or no end-user capability. Another risk is that the infrastructure being built itself could be so divergent from what users need that subsequent tools will not be useful. However, if the infrastructure is designed well, it tends to be flexible and amenable to a variety of tools and interfaces, all built over a common substrate. Moreover, the implementation of a new service or tool tends to be quick since much of the complexity of the underlying substrate is abstracted away.

5 Future Directions

Grid technology is becoming mature enough to move out from the indulgent environment of academia into the demanding world of commercial usage. In a commercial environment, non-technological concerns, such as standards acceptance, support personnel, open sources and deployment model compete with the technological issues which we have discussed so far. Grids are large, infrastructure-style deployments. Therefore, while much of the technological discussion of the last decade or so have been valuable, it may now be time to focus on the non-technological issues as well. In this section, we present the initial approach of Legion and Globus to deployments and standards.

5.1 Future Directions in Legion

From the outset of the Legion project a “technology transfer” phase had been envisioned in which the technology would be moved from academia to industry. We felt strongly that grid software would move into mainstream business computing only with commercially-supported software, help lines, customer support, services and deployment teams. In 1999, Applied MetaComputing was founded to carry out the technology transition of Legion. In 2001, Applied MetaComputing raised $16M in venture capital and changed its name to AVAKI. The company acquired legal rights to Legion from the University of Virginia and renamed Legion to “Avaki”. Avaki was released commercially in September, 2001. Avaki is an extremely hardened, trimmed-down, focused-on-commercial-requirements version of Legion. While the name has changed, the core architecture and the principles on which it operates remain the same.

Many of the technological challenges faced by companies today can be viewed as variants of the requirements of grid infrastructures. The components of a project or product – data, applications, processing power and users – may be in distinct locations from one another. Administrative controls set up by organizations to prevent unauthorized accesses to resources hinder authorized accesses as well. Differences in platforms, operating systems, tools, mechanisms for running jobs, data organizations, etc. impose a heavy cognitive burden on users. Changes in resource usage policies and security policies affect the day-to-day actions of users. Finally, large distances act as barriers to the quick communication necessary for collaboration. Consequently, users spend too much time on the procedures for accessing a resource and too little time using the resource itself. These challenges lower productivity and hinder collaboration.

A successful technology is one that can transition smoothly from the comfort and confines of academia to the demanding commercial environment. Several academic projects are testament to the benefits of such transitions; often, the transition benefits not just the user community but the quality of the product as well. We believe grid infrastructures are ready to make such a transition. Legion had been tested in non-industry environments from 1997 onwards during which time we had the opportunity to test the basic model, scalability, security features, tools and development environment rigorously. Further improvement required input from a more demanding community with a vested interest in using the technology for their own benefit. The decision to commercialize grids, in the form of the Avaki 2.x product and beyond was inevitable.

Despite changes to the technology enforced by the push to commercialization, the basic technology in Avaki 2.x remains the same as Legion. All of the principles and architectural features discussed earlier continue to form the basis of the commercial product. As a result, the commercial product continues to meet the requirements outlined in the introduction. These requirements follow naturally from the challenges faced by commercial clients who attempt to access distributed, heterogeneous resources in a secure manner.

Beyond commercialization of the Legion technology in Avaki, the Legion team is focusing on issues of fault-tolerance (autonomic computing) and security policy negotiation across organizations. This work is being done in the context of the Global Grid Forum (GGF) open standards OGSI and OGSA.

5.2 Future Directions in Globus

The earlier days of Globus were, like Legion, largely characterized as being focused on interconnecting supercomputer centers across geographic boundaries more easily. As such, the community being courted by Globus was largely DoE sites and NSF sites (i.e., academic settings).

This situation began to change, at least by early 2001, when Globus began to seize upon opportunities outside the national labs and academia. On August 2, 2001, IBM made an announcement regarding their support the UK E-science effort, in which IBM's first open support for the Globus project was announced. Later that year, at Supercomputing 2001, the Globus project announced the equivalent of "strategic partnerships" with 12 vendors (Compaq, Cray, SGI, Sun, Veridian, Fujitsu, Hitachi, NEC, Entropia, IBM, Microsoft and Platform Computing).

On April 12, 2002, Globus 2.0 was officially announced (a "public beta" of Globus 2.0 was available as of Nov 15, 2001) Globus 2.0 is the basis for a number of Grid activities, including the NSF Middleware Initiative (NMI), the EU DataGrid, and GriPhyN VDT.

The major development with regard to the future of the Globus toolkit occurred on February 20, 2002, when the Globus Project and IBM announced the intention create a new set of standards that would more closely integrate Grid computing with Web services. This set of standards is called the Open Grid Services Architecture (OGSA). This effort has since seen significant success, and is now community-based and homed in the Global Grid Forum (see the next section for a more complete discussion). In January 2003, at GlobusWorld 2003, the Globus project announced the beta of the first version of the Globus toolkit, GT3, to be OGSI-compliant.

6 Relationship to Upcoming Standards

OGSI (Open Grid Services Infrastructure) Version 1.0 [83] and OGSA (Open Grid Services Architecture) [51] are standards emerging from the GGF (Global Grid Forum). These are the prevailing standards initiatives in grid computing, and will in our opinion, define the future of Grid Computing.

OGSI extends web services via the definition of Grid Services. In OGSI, a grid service is a web service that conforms to a particular set of conventions [76]. For example, grid services are defined in terms of standard WSDL (Web Services Description Language) [77] [78] [79] with some extensions, and exploits standard web service binding technologies such as SOAP (Simple Object Access Protocol) [80] [81] [82], WS-Security, etc. However, this set of conventions fundamentally sets grid services apart from web services. Grid services introduce three fundamental differences:

- named service instances, and a two-level naming scheme that facilitates traditional distributed systems transparencies,
- services have a minimum set of capabilities including discovery (reflection) services; and
- explicitly-stateful services with lifetime management.

OGSI has been developed in the OGSI-WG (working group) in the GGF. The specification emerged from the standards process in the second quarter of 2003. It was submitted to the GGF Editor as a recommendation track document and is now undergoing public comment. OGSI will be the basic interoperability layer (in terms of RPC, discovery, etc.) for a rich set of higher-level services and capabilities that are collectively known as OGSA, the Open Grid Services Architecture.

OGSA is being developed in the OGSA-WG of the GGF. The OGSA-WG is an umbrella working group within the GGF. The OGSA-WG will ”spin off” working groups to develop specialized service standards that, together, will realize a “meta-operating system" environment. The first of these working groups formed is the OGSA-Security working group. Working groups on topics such as "resources" (hosts, storage, etc.), scheduling, replication, logging, management interfaces, fault-tolerance, etc. are anticipated.

The Legion authors enthusiastically support the OGSI effort. We support the OGSI/A standards efforts of the GGF for two primary reasons, the congruence of the OGSA architecture with the Legion architecture, and the importance of standards to users. The OGSA architecture is highly congruent with both the existing Legion architecture and our architectural vision for the future. This congruence is not surprising given that both OGSA and Legion have the same objective: to create a "meta-operating system for the grid. Our objective building and designing Legion was:

> "to provide a solid, integrated, conceptual foundation on which to build applications that unleash the potential of so many diverse resources. The foundation must at least hide the underlying physical infrastructure from users and from the vast majority of programmers, support access, location, and fault transparency, enable inter-operability of components, support construction of larger integrated components using existing components, provide a secure environment for both resource owners and users, and it must scale to millions of autonomous hosts." [7]

This vision is the mantra of OGSA as well. The means to the end are similar in both cases, and include the definition of base class services for basic building blocks of grids, hosts, storage, security, usage polices, failure detection mechanism, and failure recovery mechanisms and policies, e.g., replication services, and so on. The major architectural difference is in the RPC mechanism. OGSA and OGSI are based on Web Services standards (SOAP/XML, WSDL, etc.), standards that did not exist when Legion was begun.

7 Summary

Legion and Globus are pioneering grid technologies. Both technologies share a common vision of the scope and utility of grids. To an outsider, i.e., a person interested in grids but not intimately familiar with either project, the differences between the two projects is unclear. Indeed, over the years, the authors of this paper have had to explain the differences several times in several settings ranging from conference talks to informal dinner-table discussions. Doubtless, the architects of Globus have had similar experiences. This paper is an attempt at explaining those differences clearly.

The objective of this paper is not to disparage one or the other project, rather to contrast the architectural and philosophical differences between the two projects, and thus educate the grid community about the choices available and the reasons they were made along every step of the design. What is especially satisfying is the fact that both these projects are now converging towards a common, best-of-breed architecture that is certain to benefit designers and users of grid systems. Such a convergence would not have been possible without the rivalry of earlier days.

Glossary

Term	Definition
GGF, Global Grid Forum	GGF is the standards organization for Grid computing. Members are drawn from academia, the private sector, and government in the Americas, Europe, and Asia-Pacific. www.ggf.org.
OGSI	Open Grid Services Interface. A standard developed by GGF that uses

	web services (SOAP, WSDL, etc.) as its base.
OGSA	Open Grid Services Architecture. OGSA is a general grid architecture under development in GGF. Its first component, OGSI, has already been approved.
PBS	Portable Batch System. A commercial load management system available for free.
LSF	Load Sharing Facility. A commercial load management system available from Platform Computing.
LoadLeveler	A load management system available from IBM.
SGE	Sun Grid Engine. A load management system available from Sun.
Codine	The load management system that became SGE,
DCE	Distributed Computing Environment. A standard developed by OMG for distributed computing.
OMG	Object Management Group – a standards organization.
MPI	Message Passing Interface. The standard (supported by most vendors) for inter-process communication on parallel computers. Often used in distributed systems as well.
NOW	Networks of Workstations. A UC Berkeley project.
CORBA	Common Object Request Broker Architecture. A standard for distributed object systems developed by OMG.
GRAM	Globus Resource Access and Management
MPL	Mentat Programming Language
DNS	Domain Name Service. A means of mapping domain names, e.g., cs.virginia.edu, to IP addresses.
LDAP	Light-weight Directory Access Protocol.
URL	Uniform Resource Locator
FTP	File Transfer Protocol
GSSAPI	Generic Security Services Application Programming Interface. The GSSAPI is a generic API for doing client-server authentication.
PKI	Public Key Infrastructure
SSL	Secure Socket Layer
OpenSSL	An open implementation of SSL
NFS	Network File System. The standard for Unix developed by Sun.
J2EE	Java 2, Enterprise Edition. The standard Java hosting environment, developed by Sun.
Kerberos	A centralized authentication system developed at MIT.
IPC	Inter-process communication
TCP/UDP	Internet protocol transport layer protocols for reliable streams and unreliable datagram's respectively.
MDS	MetaComputing Directory Service. A Globus service that builds on LDAP.
MPICH	An open implementation of the MPI standard developed at Argonne National Lab.
SOAP/XML	Simple Object Access Protocol. The heart of the emerging web services standards specifies how services invoke one another.
WSDL	Web Services Description Language. A standard IDL (interface description language) for specifying SOAP interfaces.

8 References and Bibliography

8.1 Legion

[1] S.J. Chapin, D. Katramatos, J.F. Karpovich and A.S. Grimshaw, “Resource Management in Legion”, *Journal of Future Generation Computing Systems*, 15: 583-594, 1999.

[2] S.J. Chapin, C. Wang, W.A. Wulf, F.C. Knabe and A.S. Grimshaw, “A New Model of Security for Metasystems”, *Journal of Future Generation Computing Systems*, 15: 713-722, 1999.

[3] A.J. Ferrari, F.C. Knabe, M.A. Humphrey, S.J. Chapin and A.S. Grimshaw, “A Flexible Security System for Metacomputing Environments”, *7th International Conference on High-Performance Computing and Networking Europe (HPCN’99),* 370-380, Amsterdam, April 1999.

[4] A.J. Ferrari and A.S. Grimshaw, “Basic Fortran Support in Legion”, *Technical Report CS-98-11*, Department of Computer Science, University of Virginia, March 1998.

[5] A.S. Grimshaw, “Easy to Use Object-Oriented Parallel Programming with Mentat”, *IEEE Computer*, 26(5): 39-51, May 1993.

[6] A.S. Grimshaw, W.A. Wulf, J.C. French, A.C. Weaver and P.F. Reynolds Jr., “Legion: The Next Logical Step Toward a Nationwide Virtual Computer”, *Technical Report CS-94-21*, Department of Computer Science, University of Virginia, June 1994.

[7] A.S. Grimshaw, “Enterprise-Wide Computing”, *Science*, 256: 892-894, August 1994.

[8] A.S. Grimshaw, J.B. Weissman and W.T. Strayer, “Portable Run-Time Support for Dynamic Object-Oriented Parallel Processing”, ACM *Transactions on Computer Systems*, 14(2): 139-170, May 1996.

[9] A.S. Grimshaw and W.A. Wulf, “Legion – A View from 50,000 Feet”, *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing*, August 1996.

[10] A.S. Grimshaw, A.J. Ferrari and E. West, “Mentat”, *Parallel Programming using C++*, eds. Gregory V. Wilson and Paul Lu, The MIT Press, 383-427, 1996.

[11] A.S. Grimshaw and W.A. Wulf, “The Legion Vision of a Worldwide Virtual Computer”, *Communications of the ACM*, 40(1): 39-45, January 1997.

[12] A.S. Grimshaw, A.J. Ferrari, G. Lindahl and K. Holcomb, “Metasystems”, *Communications of the ACM*, 41(11): 486-55, November 1998.

[13] A.S. Grimshaw, A.J. Ferrari, F.C. Knabe and M.A. Humphrey, “Wide-Area Computing: Resource Sharing on a Large Scale”, *IEEE Computer*, 32(5): 29-37, May 1999.

[14] A.S. Grimshaw, A. Natrajan, M.A. Humphrey and M.J. Lewis, A. Nguyen-Tuong, J.F. Karpovich, M.M. Morgan and A.J. Ferrari, "From Legion to Avaki: The Persistence of Vision", *Grid Computing: Making the Global Infrastructure a Reality*, eds. Fran Berman, Geoffrey Fox and Tony Hey, November 2002.

[15] M.A. Humphrey, F.C. Knabe, A.J. Ferrari and A.S. Grimshaw, "Accountability and Control of Process Creation in Metasystems", *Proceedings of the 2000 Network and Distributed Systems Security Conference (NDSS'00)*, San Diego, California, February 2000.

[16] L.J. Jin and A.S. Grimshaw, "From MetaComputing to Metabusiness Processing", In *Proceedings IEEE International Conference on Cluster Computing – Cluster 2000*, Saxony, Germany, December 2000.

[17] M.J. Lewis and A.S. Grimshaw, "The Core Legion Object Model", *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing*, August 1996.

[18] M.J. Lewis, A.J. Ferrari, M.A. Humphrey, J.F. Karpovich, M.M. Morgan, A. Natrajan, A. Nguyen-Tuong, G.S. Wasson and A.S. Grimshaw, "Support for Extensibility and Site Autonomy in the Legion Grid System Object Model", to appear *Journal of Parallel Distributed Computing*, 63: 525-538, 2003.

[19] A. Natrajan, M.A. Humphrey and A.S. Grimshaw, "Capacity and Capability Computing using Legion", *Proceedings of the 2001 International Conference on Computational Science*, 273-283, San Francisco, California, May 2001.

[20] A. Natrajan, M.A. Humphrey and A.S. Grimshaw, "Grids: Harnessing Geographically-Separated Resources in a Multi-Organisational Context", *High Performance Computing Systems (HPCS)*, Windsor, Canada, June 2001.

[21] A. Natrajan, M. Crowley, N. Wilkins-Diehr, M.A. Humphrey, A.J. Fox, A.S. Grimshaw and C.L. Brooks III, "Studying Protein Folding on the Grid: Experiences using CHARMM on NPACI Resources under Legion", *10th International Symposium on High Performance Distributed Computing (HPDC)*, San Francisco, California, August 2001.

[22] A. Natrajan, A. Nguyen-Tuong, M.A. Humphrey, M. Herrick, B.P. Clarke and A.S. Grimshaw, "The Legion Grid Portal", *Grid Computing Environments, Concurrency and Computation: Practice and Experience*, 14(13-15): 1365-1394, 2001.

[23] A. Natrajan, M.A. Humphrey and A.S. Grimshaw, "The Legion Support for Advanced Parameter-Space Studies on a Grid", *Future Generation Computer Systems*, 18(8): 1033-1052, October 2002.

[24] A. Nguyen-Tuong, A.S. Grimshaw and M. Hyett, "Exploiting Data-Flow for Fault-Tolerance in a Wide-Area Parallel System", *Proceedings of the 15th International Symposium on Reliable and Distributed Systems (SRDS-15)*, 2-11, 1996.

[25] A. Nguyen-Tuong, S.J. Chapin, A.S. Grimshaw and C. Viles, "Using Reflection for Flexibility and Extensibility in a Metacomputing Environment", *Technical Report 98-33*, Dept. of Computer Science, University of Virginia, November 1998.

[26] A. Nguyen-Tuong and A.S. Grimshaw, "Using Reflection for Incorporating Fault-Tolerance Techniques into Distributed Applications", *Parallel Processing Letters,* 9(2): 291-301, 1999.

[27] G. Stoker, B.S. White, E.L. Stackpole, T.J. Highley and M.A. Humphrey, "Toward Realizable Restricted Delegation in Computational Grids", *European High Performance Computing and Networking*, June 2001.

[28] B.S. White, A.S. Grimshaw and A. Nguyen-Tuong, "Grid-Based File Access: The Legion I/O Model", *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, August 2000.

[29] B.S. White, M.P. Walker, M.A. Humphrey and A.S. Grimshaw, "LegionFS: A Secure and Scalable File System Supporting Cross-Domain High-Performance Applications", *Proceedings of SuperComputing 2001*, Denver, Colorado, November 2001.

[30] C.L. Viles, M.J. Lewis, A.J. Ferrari, A. Nguyen-Tuong and A.S. Grimshaw, "Enabling Flexibility in the Legion Run-Time Library", *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '97)*, 265-274, Las Vegas, Nevada, June 1997.

[31] A.S. Grimshaw, J. B. Weissman, E. A. West and E. Loyot, "Metasystems: An Approach Combining Parallel Processing And Heterogeneous Distributed Computing Systems", *Journal of Parallel and Distributed Computing*, 21(3): 257-270, June 1994.

[32] J.F. Karpovich, A.S. Grimshaw, and J. C. French, "Extensible File Systems (ELFS): An Object-Oriented Approach to High Performance File I/O", *Proceedings of OOPSLA '94*, 191-204, Portland, OR, Oct 1994.

[33] W.A. Wulf, C. Wang, D. Kienzle, "A New Model of Security for Distributed Systems", *Technical Report 95-34*, Dept. of Computer Science, University of Virginia, August 1995. CHUCK

8.2 Globus

[34] G. Allen, D. Angulo, I. Foster, G. Lanfermann, Chuang Liu, T. Radke, E. Seidel and J. Shalf, "The Cactus Worm: Experiments with Dynamic Resource Selection and Allocation in a Grid Environment", *International Journal of High-Performance Computing Applications*, 15(4), 2001.

[35] J. Bester, I. Foster, C. Kesselman, J. Tedesco and S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems", *Sixth Workshop on I/O in Parallel and Distributed Systems*, May 5, 1999.

[36] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer and V. Welch, "A National-Scale Authentication Infrastructure", *IEEE Computer*, 33(12): 60-66, 2000.

[37] A. Chervenak, I. Foster, C Kesselman, C. Salisbury and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets", *Journal of Network and Computer Applications*, 23(3): 187-200, Jul. 2001.

[38] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems", *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, 62-82, 1998.

[39] K. Czajkowski, I. Foster and C. Kesselman, "Resource Co-Allocation in Computational Grids", *Proc. of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, 219-228, 1999.

[40] K. Czajkowski, S. Fitzgerald, I. Foster and C. Kesselman, "Grid Information Services for Distributed Resource Sharing", *Proc. of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, IEEE Press, August 2001.

[41] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations", *Proc. 6th IEEE Symposium on High-Performance Distributed Computing*, 365-375, 1997.

[42] I. Foster, C. Kesselman and S. Tuecke, "The Nexus Task-Parallel Runtime System", *Proc. 1st Intl. Work. on Par. Processing*, 457-462, 1994.

[43] I. Foster, J. Geisler, W. Nickless, W. Smith and S. Tuecke, "Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment", *Proc. 5th IEEE Symp. on High Perf. Dist. Computing*, 562-571, 1997.

[44] I. Foster, J. Geisler, C. Kesselman and S. Tuecke, "Managing Multiple Communication Methods in High-Performance Networked Computing Systems", *Jour. Parallel and Distributed Computing*, 40: 35-48, 1997.

[45] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", *International Journal of Supercomputing Applications and High Performance Computing*, 11(2): 115-128, 1997.

[46] I. Foster, C. Kesselman, G. Tsudik and S. Tuecke, "A Security Architecture for Computational Grids", *Proc. 5th ACM Conference on Computer and Communications Security Conference*, San Francisco, 83-92, 1998.

[47] I. Foster, N. Karonis, C. Kesselman, G. Koenig and S. Tuecke, "A Secure Communications Infrastructure for High-Performance Distributed Computing", *6th IEEE Symp. on High-Performance Distributed Computing*, 125-136, 1998.

[48] I. Foster and N. Karonis, "A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems", *Proc. 1998 Supercomputing Conference*, November 1998.

[49] I. Foster and C. Kesselman, "Computational Grids, The Grid: Blueprint for a New Computing Infrastructure", *Morgan-Kaufman*, 1999.

[50] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *Intl. Jour. Supercomputer App.*, 15(3): 200-222, 2001.

[51] I. Foster, C. Kesselman, J.M. Nick and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", *Open Grid Services Infrastructure WG*, Global Grid Forum, June 22, 2002, also appears as "Grid Services for Distributed System Integration", *IEEE Computer*, 35(6), 2002.

[52] I. Foster, "What is the Grid? A Three Point Checklist", *GridToday*, no. 100136, Jul. 2002.

[53] J. Frey, T. Tannenbaum, I. Foster, M. Livny and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", *Cluster Computing*, 5(3): 237-246, 2002.

[54] K. Keahey, T. Fredian, Q. Peng, D.P. Schissel, M. Thompson, I. Foster, M. Greenwald and D. McCune, "Computational Grids in Action: The National Fusion Collaboratory", *Future Generation Computer Systems*, 18(8): 1005-1015, 2002.

[55] K. Keahey and V. Welch, "Fine-Grain Authorization for Resource Management in the Grid Environment", *Proceedings of Grid2002 Workshop*, Baltimore, Nov. 2002.

[56] C. Lee, R. Wolski, I. Foster, C. Kesselman and J. Stepanek, "A Network Performance Tool for Grid Computations", *Supercomputing '99*, Portland, Nov. 1999.

[57] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman and B. Tierney , "File and Object Replication in Data Grids", *Journal of Cluster Computing*, 5(3): 305-314, 2002.

8.3 Other

[58] –, "IBM LoadLeveler: User's Guide", Intl. Bus. Mach. Corp., Sept. 1993.

[59] D. Abramson, R. Sosic, J. Giddy and B. Hall, "Nimrod: A Tool for Performing Parameterised Simulations using Distributed Workstations", *Proc. 4th IEEE Intl. Symp. on High Performance Dist. Computing (HPDC)*, Washington, Aug. 1995.

[60] T. Anderson, D. Culler, D. Patterson and the NOW team, "A Case for NOW (Networks of Workstations)", *IEEE Micro*, 15(1): 54-64, 1995

[61] A. Bayucan, R.L. Henderson, C. Lesiak, N. Mann, T. Proett and D. Tweten, "Portable BatchSystem: External Reference Specification", *Tech. Rep.*, MRJ Technology Solutions, Nov. 1999.

[62] A. Chien, "An Interview with Entropia's Andrew Chien", by Alan Beck, *GridToday*, no. 100110, Jul. 2002.

[63] F. Ferstl, "CODINE Technical Overview", Genias Software, Apr. 1993.

[64] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, "PVM: Parallel Virtual Machine", *MIT Press*, Cambridge (MA), 1994.

[65] W. Gentzsch, "Response to Ian Foster's "What is the Grid?"", *GridToday*, no. 100191, Aug. 2002.

[66] W. Gropp, E. Lusk and A. Skjellum, "Using MPI: Portable Parallel Programming with the Message-Passing Interface", *MIT Press*, 1994.

[67] N.H. Kapadia, R.J. Figueiredo and J.A.B. Fortes, "PUNCH: Web Portal For Running Tools", *IEEE Micro*, 38-47 Jun. 2000.

[68] B.A. Kingsbury, "The Network Queueing System (NQS)", *Tech. Rep.*, Sterling Software, 1992.

[69] M.J. Litzkow, M. Livny and M.W. Mutka, "Condor - A Hunter of Idle Workstations", *Intl. Conf. on Dist. Computing Syst.*, 104-111, 1988.

[70] H. Lockhart Jr., "OSF DCE Guide to Developing Distributed Applications", *McGraw-Hill Inc.*, New York, 1994.

[71] Object Management Group, "The Common Object Request Broker: Architecture and Specification", Rev. 2.0, Jul. 1995 (updated Jul. 1996).

[72] L. Smarr and C.E. Catlett, "Metacomputing", *Communications of the ACM*, 35(6): 44-52, June 1992.

[73] M. van Steen, P. Homburg and A. Tanenbaum, "The architectural design of Globe: a wide-area distributed system", *Internal Rep. IR-422*, Vrije Universiteit, 1997.

[74] S. Zhou, "LSF: Load Sharing in Large-scale Heterogeneous Distributed Systems", *Work. on Cluster Computing*, Dec. 1992.

[75] S. Zhou, J. Wang, X. Zheng and P. Delisle, "Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems", *Software Prac. and Experience*, 23(2), 1993.

[76] A.S. Grimshaw and S. Tuecke, "Grid Services extend Web Services", *Web Services Journal*, 3(8), Aug. 2003.

[77] R. Chinnici, M. Gudgin J.-J., Moreau and S. Weerawarana, "Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language", Technical Report, *W3 Consortium*, Jun. 2003.

[78] M. Gudgin, A. Lewis and J. Schlimmer, "Web Services Description Language (WSDL) Version 1.2 Part 2: Message Patterns", Technical Report, *W3 Consortium*, June 2003.

[79] J.-J. Moreau and J. Schlimmer, "Web Services Description Language (WSDL) Version 1.2 Part 3: Bindings", Technical Report, *W3 Consortium*, June 2003.

[80] N. Mitra, "SOAP Version 1.2 Part 0: Primer", Technical Report, *W3 Consortium*, June 2003.

[81] M. Hadley, N. Mendelsohn, J.-J. Moreau, H.F. Nielsen and M. Gudgin, "SOAP Version 1.2 Part 1: Messaging Framework", Technical Report, *W3 Consortium*, June 2003.

[82] J.-J. Moreau, H.F. Nielsen, M. Gudgin, M. Hadley and N. Mendelsohn, "SOAP Version 1.2 Part 0: Adjuncts", Technical Report, *W3 Consortium*, June 2003.

[83] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling and P. Vanderbilt, "Open Grid Services Infrastructure (OGSI) Version 1.0", Draft, *Global Grid Forum*, April 2003.