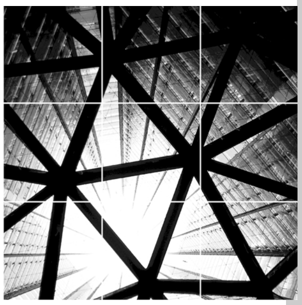
PART II FRAMEWORK



We are now past the preliminaries and ready to dive into the technical meat of *The Grid*. We lead off with Part II, which comprises just the single scene-setting Chapter 4, "Concepts and Architecture." This chapter introduces the central Grid concepts and architectural principles upon which the rest of the book is based. Its purpose is to orient the reader by presenting the concepts and terminology required to understand the material that follows. This is the chapter that should be read before any other.

4

CHAPTER

Concepts and Architecture

Ian Foster and Carl Kesselman

In this scene-setting chapter, we provide an overview of the purpose, evolution, architecture, and implementation of Grid systems—a picture that will then be filled out in the chapters that follow. This chapter thus both introduces Grids and provides a road map for the rest of the book.

The term "the Grid" was coined in the mid-1990s to denote a (then) proposed distributed computing infrastructure for advanced science and engineering. Much progress has since been made on the construction of such an infrastructure and on its extension and application to commercial computing problems. And while the term "Grid" has also been on occasion conflated to embrace everything from advanced networking and computing clusters to artificial intelligence, there has also emerged a good understanding of the problems that Grid technologies address, and at least a first set of applications for which they are suited.

Grid concepts and technologies were originally developed to enable resource sharing within scientific collaborations, first within early gigabit/sec testbeds (161, 163) and then on increasingly larger scales (108, 137, 394, 610). As discussed in Chapter 2, applications in this context include distributed computing for computationally demanding data analyses (pooling of compute power and storage; e.g., Chapter 10), the federation of diverse distributed datasets (e.g., Chapters 7–9), collaborative visualization of large scientific datasets (pooling of expertise), and coupling of scientific instruments with remote computers and archives (increasing functionality as well as availability, e.g., Chapters 1:4 and 1:6).

A common theme underlying these different usage modalities is a need for *coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations* (281). More recently, it has become clear (see Chapter 3) that similar requirements arise in commercial settings, not only for scientific and technical computing applications (where we can already point to success stories; e.g., Chapters 11 and 12) but also for commercial distributed computing applications (e.g., Chapter 13), including enterprise application integration (Chapter 14) and business-to-business partner collaboration over the Internet. Just as the Web began as a technology for scientific collaboration and was adopted for e-business, we see a similar trajectory for Grid technologies.

We thus argue that both science and industry can benefit from Grids. However, at the risk of stating the case too broadly, we make a more comprehensive statement. A primary purpose of information technology and infrastructure is to enable people to perform their daily tasks more efficiently or effectively. To the extent that these tasks are performed in collaboration with others, Grids are more than just a niche technology, but rather a direction in which our infrastructure must evolve if it is to support our social structures and the way work gets done in our society.

The success of the Grid to date owes much to the relatively early emergence of clean architectural principles, de facto standard software, aggressive early adopters with challenging application problems, and a vibrant international community of developers and users. This combination of factors led to a solid base of experience that has more recently driven the definition of the service-oriented Open Grid Services Architecture that today forms the basis for both open source and commercial Grid products. In the sections that follow, we expand upon these various aspects of the Grid story and, in so doing, introduce the principal issues to be addressed in the rest of the book.

4.1 VIRTUAL ORGANIZATIONS AND THE GRID

Consider the following scenarios:

- A company needing to reach a decision on the placement of a new factory invokes a sophisticated financial forecasting model from an application service provider (ASP), providing it with access to appropriate proprietary historical data from a corporate database on storage systems operated by a storage service provider. During the decision-making meeting, what-if scenarios are run collaboratively and interactively, even though the division heads participating in the decision are located in different cities. The ASP itself contracts with an on-demand cycle provider for additional "oomph" during particularly demanding scenarios, requiring of course that cycles meet desired security and performance requirements.

- An industrial consortium formed to develop a feasibility study for a next-generation supersonic aircraft undertakes a highly accurate multidisciplinary simulation of the entire aircraft. This simulation integrates proprietary software components developed by different participants, with each component operating on that participant's computers and having access to appropriate design databases and other data made available to the consortium by its members.
- A crisis management team responds to a chemical spill by using local weather and soil models to estimate the spread of the spill, determining the impact based on population location as well as geographic features such as rivers and water supplies, creating a short-term mitigation plan (perhaps based on chemical reaction models), and tasking emergency response personnel by planning and coordinating evacuation, notifying hospitals, and so forth.
- Thousands of physicists at hundreds of laboratories and universities worldwide come together to design, create, operate, and analyze the products of a major detector at CERN, the European high-energy physics laboratory. During the analysis phase, they pool their computing, storage, and networking resources to create a "data Grid" capable of analyzing petabytes of data (178, 368) (see Chapter 10).
- A large-scale Internet game consists of many virtual worlds, each with its own physical laws and consequences. Each world may have a large number of inhabitants that interact with one another and move from one world to another. Each virtual world may expand in an on-demand basis to accommodate population growth, new simulation technology to model the physical laws of the world will need to be added, and simulations need to be coupled to determine what happens "when worlds collide" (see Chapter 13).
- A biologist wants to understand how changes in neuron synapse response induced by a drug impact the performance of specific brain functions. To answer this question, he needs to perform low-level chemical simulations of the synapse and then map this information upward in the structural hierarchy of the brain. This analysis requires mapping simulation across many different databases, each containing information about different levels of the biological system.

These examples differ in many respects: the number and type of participants, the types of activities, the duration and scale of the interaction, and the resources being shared. However, they also have much in common. In each case, mutually distrustful participants with varying degrees of prior relationship (perhaps none at all) want to share resources in order to perform some task. Furthermore, sharing is about more than simply document exchange (as in “virtual enterprises” (148)): it can involve direct access to remote software, computers, data, sensors, and other resources. For example, members of a consortium may provide access to specialized software and data and/or pool their computational resources.

More abstractly, what these application domains have in common is a need for *coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a *virtual organization* (VO), a concept that is becoming fundamental to much of modern computing. VOs enable disparate groups of organizations and/or individuals to share resources in a controlled fashion, so that members may collaborate to achieve a shared goal.

As these examples show, VOs can vary greatly in their purpose, scope, size, duration, structure, community, and sociology. Nevertheless, we can identify a broad set of common concerns and technology requirements. In particular, we see a need for highly flexible sharing relationships, ranging from client-server to peer to peer; for sophisticated and precise levels of control over how shared resources are used, including fine-grained and multistakeholder access control, delegation, and application of local and global policies; for sharing of varied resources, ranging from programs, files, and data to computers, sensors, and networks; for virtualization of resources as services, so that diverse capabilities can be delivered in standard ways without regard to physical location and implementation; and for diverse usage modes, ranging from single-user to multiuser and from performance-sensitive to cost-sensitive and hence embracing issues of quality of service, scheduling, co-allocation, and accounting.

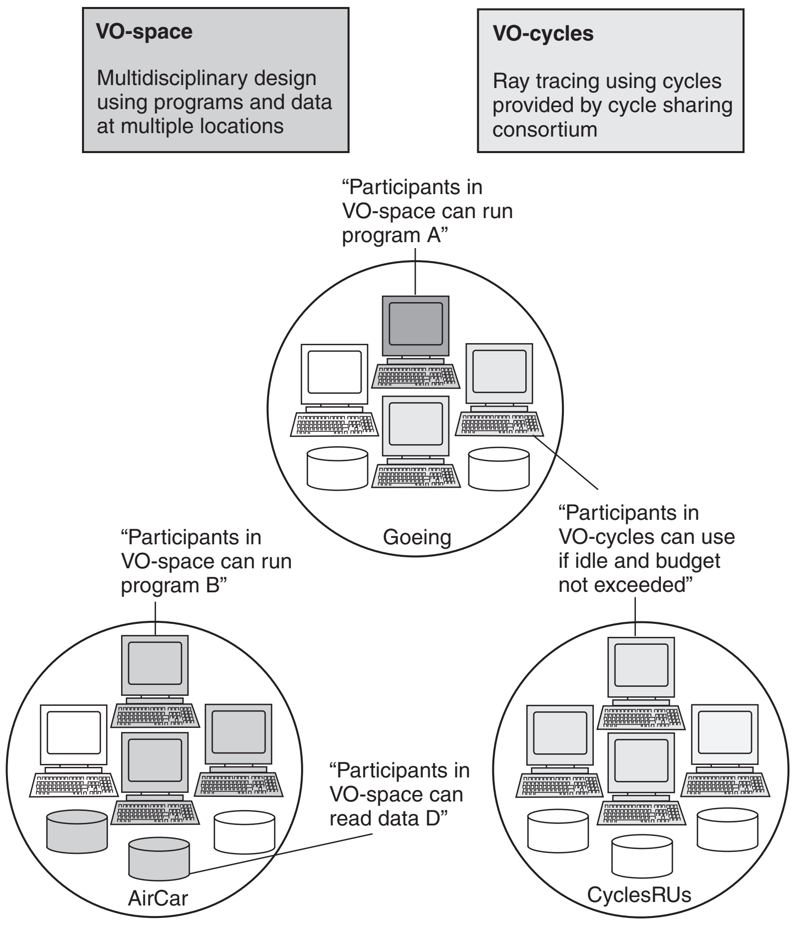
Resource sharing, virtual organization, and virtualization are not new concepts. For example, in 1965 the designers of the then-revolutionary Multics operating system wrote that “the time-sharing computer system can unite a group of investigators . . . one can conceive of such a facility as an . . . intellectual public utility” (665), and in 1969 Internet pioneer Len Kleinrock suggested, presciently if prematurely upon the installation of the first ARPANET node, that “we will probably see the spread of ‘computer utilities,’ which, like present electric and telephone utilities, will service individual homes and offices across the country” (415). And of course the distributed systems, networking, operating systems, collaborative work, and security communities have worked for more than 30 years on the principles and mechanisms required to support distributed resource sharing. What is new today is that, as a result of work by these and other pioneers, the Internet is by now quasi-ubiquitous, devices and networks are far more capable, and distributed computing technologies have advanced to the point where it has become practical to think about realizing resource sharing, virtual organization, and virtualization scenarios on a large scale.

4.1.1 Technical Challenges in Sharing Relationships

Depending on context, the virtual organizations with which we are concerned can be small or large, short- or long-lived, single or multi-institutional, and homogeneous or heterogeneous. Individual VOs may be structured hierarchically from smaller systems and may overlap in membership. Furthermore, regardless of these differences, developers of applications for VOs face common requirements as they seek to deliver QoS—whether measured in terms of common security semantics, distributed workflow and resource management, coordinated fail-over, problem determination services, or other metrics—across a collection of resources with heterogeneous and often dynamic characteristics.

We use the example in Figure 4.1 to illustrate some complexities that we face in addressing these issues. The figure depicts three physical organizations, AirCar, Goeing, and CyclesRUs, each of which participates in various virtual organizations that involve controlled sharing of its computational and data resources. In particular, AirCar and Goeing (fierce competitors in the aerospace industry) both collaborate within an international virtual organization, VO-Space, on the design of an advanced space vehicle. In addition, Goeing participates in a regional cycle-sharing consortium, VO-Cycles, in which it pools unused cycles with a local service provider, CyclesRUs, for computationally intensive rendering tasks.

Resource sharing is often conditional: each resource owner makes resources available, subject to constraints on when, where, and what can be done. (In the figure, the text in quotes denotes the policies that apply for each resource or service.) For example, AirCar might allow its VO-Space partners to invoke a simulation service only for "simple" problems (according to some agreed-upon definition). Resource consumers may also place constraints on properties of the resources they are prepared to work with. For example, a participant in VO-Cycles might accept only pooled computational resources certified as "secure." The implementation of such constraints requires mechanisms for expressing policies, for establishing the identity of a consumer or resource (authentication), and for determining whether an operation is consistent with applicable sharing relationships (authorization).


VO-space
Multidisciplinary design using programs and data at multiple locations
VO-cycles
Ray tracing using cycles provided by cycle sharing consortium
"Participants in VO-space can run program A"
Goeing
"Participants in VO-cycles can use if idle and budget not exceeded"
"Participants in VO-space can run program B"
"Participants in VO-space can read data D"
AirCar
CyclesRUs

4.1 FIGURE Sharing relationships within virtual organizations.

Sharing relationships can vary dynamically over time, in terms of the resources involved, the nature of the access permitted, and the participants to whom access is permitted. Also, these relationships do not necessarily involve an explicitly named set of individuals, but rather may be defined implicitly by the policies that govern access to resources. For example, CyclesRUs might allow access to anyone who can demonstrate that he is a "customer." Thus we require mechanisms for discovering and characterizing the nature of the relationships that exist at a particular point in time. For example, a new participant joining VO-

Cycles must be able to determine what resources it is able to access, the "quality" of these resources, and the policies that govern access.

Sharing relationships are often not simply client-server, but peer to peer: providers can be consumers, and sharing relationships can exist among any subset of participants. Sharing relationships may be combined to coordinate use across many resources, each owned by different organizations. For example, in VO-Cycles, a computation started on one pooled computational resource may subsequently access data or initiate subcomputations elsewhere. The ability to delegate authority in controlled ways becomes important in such situations, as do mechanisms for coordinating operations across multiple resources (e.g., coscheduling; see Chapter 18).

The same resource may be used in different ways in different contexts. For example, a computer might be used only to run a specific piece of software in one sharing arrangement, but provide generic compute cycles in another. This lack of a priori knowledge about how a resource may be used means that performance metrics, expectations, and limitations may be part of the conditions placed on resource sharing or usage.

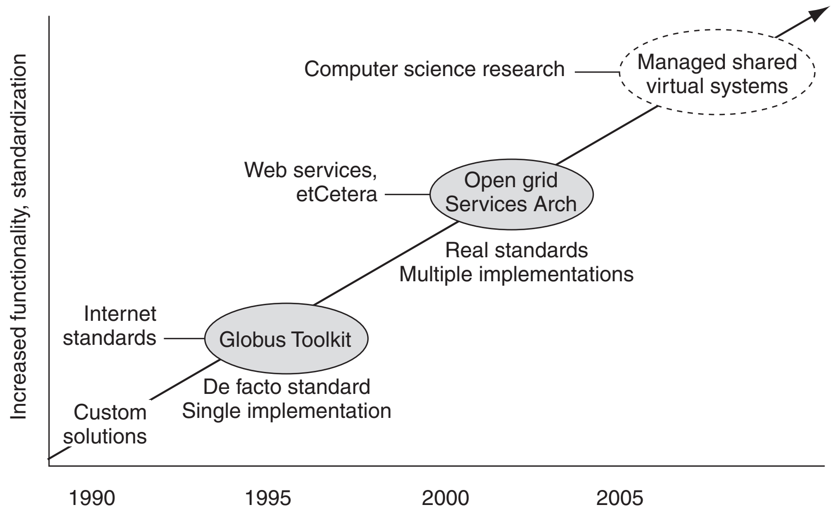
In addition to such issues of security and policy, Grid users are often vitally concerned with achieving various *qualities of service (QoS)* in the virtual systems formed by integrating distributed components. This concern is fundamental not only within the distributed virtual organizations discussed previously but also, increasingly, within a single enterprise. In the past, computing typically was performed within highly integrated host-centric enterprise computing centers. The rise of the Internet and the emergence of e-business have, however, led to a growing awareness that an enterprise's IT infrastructure is becoming increasingly decomposed, both externally (as it extends to encompass external networks, resources, and services) and internally (as enterprise IT facilities become more heterogeneous and distributed). The overall result is a decomposition of highly integrated internal IT infrastructure into a collection of heterogeneous and fragmented systems.

Enterprises must then reintegrate (with QoS) these distributed servers and data resources, addressing issues of navigation, distributed security, and content distribution inside the enterprise, much as on external networks. Enterprises are also now expanding the scope and scale of their enterprise resource planning projects as they try to provide better integration with customer relationship management, integrated supply chain, and existing core systems. The aggregate effect is that *qualities of service traditionally associated with mainframe host-centric computing* (503) *are now essential to the effective conduct of e-business across distributed compute resources, inside as well as outside the enterprise*. In many ways, this requirement is simply a restatement of the need for infrastructure that facilitates controlled sharing of resources across organizational boundaries: that is, the Grid.

4.1.2 Evolution of Grid Technologies

Grid technologies provide mechanisms for sharing and coordinating the use of diverse resources and thus enable the creation, from geographically and organizationally distributed components, of virtual computing systems that are sufficiently integrated to deliver desired qualities of service (281). These technologies include security solutions that support management of credentials and policies when computations span multiple institutions; resource management protocols and services that support secure remote access to computing and data resources and the co-allocation of multiple resources; information query protocols and services that provide configuration and status information about resources, organizations, and services; and data management services that locate and transport datasets between storage systems and applications.

Grid technologies have emerged from some 10 years of research and development in both academia and industry, which furthermore continues today. As illustrated in Figure 4.2, we can distinguish four distinct phases in this evolution.


Increased functionality, standardization
Computer science research
Managed shared virtual systems
Web services, etCetera
Open grid Services Arch
Real standards Multiple implementations
Internet standards
Globus Toolkit
De facto standard Single implementation
Custom solutions
1990
1995
2000
2005

4.2 FIGURE The evolution of Grid technologies.

Custom solutions. Starting in the early 1990s, work in "metacomputing" and related fields involved the development of custom solutions to Grid computing problems (161, 163). The focus of these often heroic efforts was on making things work and exploring what was possible. Applications were built directly on Internet protocols with typically only limited functionality in terms of security, scalability, and robustness. Interoperability was not a significant concern.

Globus Toolkit. From 1997 onward, the open source Globus Toolkit version 2 (GT2) (276) emerged as the de facto standard for Grid computing. Focusing on usability and interoperability, GT2 defined and implemented protocols, APIs, and services used in thousands of Grid deployments worldwide. By providing solutions to common problems such as authentication, resource discovery, and resource access, GT2 accelerated the construction of real Grid applications. Also by defining and implementing "standard" protocols and services, GT2 pioneered the creation of interoperable Grid systems and enabled significant progress on Grid programming tools. The GT2 protocol suite leveraged existing Internet standards for transport, resource discovery, and security. Some elements of the GT2 protocol suite were codified in formal technical specifications, reviewed within standards bodies, and instantiated in multiple implementations: notably, the GridFTP data transfer protocol (65) and elements of the Grid Security Infrastructure (660). However, in general, GT2 "standards" were neither formal nor subject to public review. Similar comments apply to other important Grid technologies that emerged during this period, such as the Condor high-throughput computing system.

Open Grid Services Architecture. The year 2002 saw the emergence of the Open Grid Services Architecture (279) (OGSA; Chapter 17), a true community standard with multiple implementations, including, in particular, the OGSA-based GT 3.0, released in 2003. Building on and significantly extending GT2 concepts and technologies, OGSA firmly aligns Grid computing with broad industry initiatives in service-oriented architecture and Web services. In addition to defining a core set of standard interfaces and behaviors that address many of the technical challenges introduced previously, OGSA provides a framework within which one can define a wide range of interoperable, portable services. OGSA provides a foundation on which can be constructed a rich Grid technology ecosystem comprising multiple technology providers.

Managed, Shared Virtual Systems. The definition of the initial OGSA technical specifications is an important step forward, but much more remains to be done before the full Grid vision is realized. Building on OGSA's service-oriented infrastructure, we will see an expanding set of interoperable services and systems that address scaling to both larger numbers of entities and smaller device footprints, increasing degrees of virtualization, richer forms of sharing, and increased qualities of service via a variety of forms of active management. This work will draw increasingly heavily on the results of advanced computer science research in such areas as peer-to-peer (Chapter 29), knowledge-based (115) (Chapter 23), and autonomic (365) (Chapter 26) systems.

We define a Grid as a system that coordinates distributed resources using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service. We examine the key elements of this definition:

- *Coordinates distributed resources.* A Grid integrates and coordinates resources and users that live within different control domains—for example, the user's desktop versus central computing, different administrative units of the same company, and/or different companies—and addresses the issues of security, policy, payment, membership, and so forth that arise in these settings. Otherwise, we are dealing with a local management system.
- *Using standard, open, general-purpose protocols and interfaces.* A Grid is built from multipurpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. As we discuss is material to follow, it is important that these protocols and interfaces be *standard* and *open*. Otherwise, we are dealing with an application-specific system.
- *To deliver nontrivial qualities of service.* A Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service—relating, for example, to response time, throughput, availability, and security—and/or coallocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.

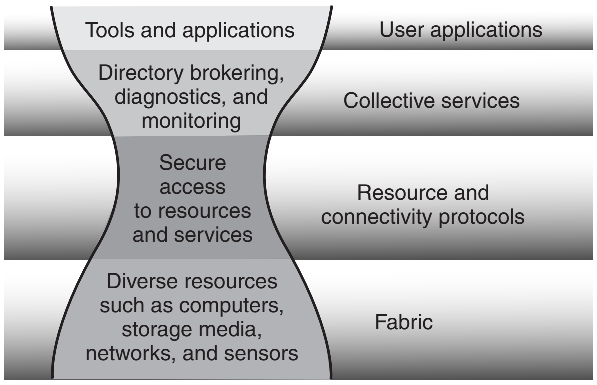
The second point is of particular importance. Standard protocols (and interfaces and policies) allow us to establish resource-sharing arrangements dynamically with *any* interested party and thus to create something more than a plethora of balkanized, incompatible, noninteroperable distributed systems. As we discuss at greater length in the following, relevant standards are being developed rapidly within the Global Grid Forum and other bodies. For an entity to be part of *the* Grid it must implement these "inter-Grid" protocols, just as to be part of the Internet an entity must speak IP (among other things). Both open source and commercial products can interoperate effectively in this heterogeneous, multivendor Grid world, thus providing the pervasive infrastructure that will enable successful Grid applications.

In the Internet, it is not uncommon that a specific set of hosts is disconnected from other hosts within an intranet. However, this partitioning occurs as a result of policy and not because of implementation. In general, all networked computers use TCP/IP and its associated protocols; and despite these policy restrictions, we still talk about a single Internet.

Similarly, we speak about *the Grid* as a single entity, even though different organizations and communities use Grid protocols to create disconnected Grids for specific purposes. As with the Internet, it is policy issues (e.g., security, cost, operational mode), not implementation issues, that prevent a service or resource from being accessible.

4.2 GRID ARCHITECTURE

We have argued that the establishment, management, and exploitation of dynamic, cross-organizational VO sharing relationships require new technology. We present a *Grid architecture* that identifies fundamental system components, specifies the purpose and function of these components, and indicates how these components interact with one another. Our goal is not to provide a complete enumeration of all required components but to identify requirements for general component classes. The result is an extensible, open architectural structure within which can be placed solutions to key VO requirements. Our architecture and the subsequent discussion organize components into layers, as shown in Figure 4.3. Components


Tools and applications
User applications
Directory brokering, diagnostics, and monitoring
Collective services
Secure access to resources and services
Resource and connectivity protocols
Diverse resources such as computers, storage media, networks, and sensors
Fabric

4.3 FIGURE The layered Grid architecture.

within each layer share common characteristics but can build on capabilities and behaviors provided by any lower layer.

Our Grid architecture is based on the principles of the "hourglass model" (34). The narrow neck of the hourglass defines a small set of core abstractions and protocols (e.g., TCP and HTTP), onto which many different high-level behaviors can be mapped (the top of the hourglass), and which themselves can be mapped onto many different underlying technologies (the base of the hourglass). By definition, the number of protocols defined at the neck must be small. In our architecture, the neck of the hourglass consists of *resource* and *connectivity* protocols, which facilitate the sharing of individual resources. Protocols at these layers are designed so that they can be implemented on top of a diverse range of resource types, defined at the *fabric* layer, and can in turn be used to construct a wide range of global services and application-specific behaviors at the *collective* layer—so called because they involve the coordinated ("collective") use of multiple resources.

4.2.1 Fabric: Interfaces to Local Control

The Grid fabric layer provides the resources to which shared access is mediated by Grid protocols, for example, computational resources, storage systems, catalogs, network resources, and sensors. A "resource" may be a logical entity, such as a distributed file system, computer cluster, or distributed computer pool; in such cases, a resource implementation may involve internal protocols (e.g., the NFS storage access protocol or a cluster resource management system's process management protocol), but these are not the concern of Grid architecture.

Fabric components implement the local, resource-specific operations that occur on specific resources (whether physical or logical) as a result of sharing operations at higher levels. There is thus a tight and subtle interdependence between the functions implemented at the fabric level, on the one hand, and the sharing operations supported, on the other. Richer fabric functionality enables more sophisticated sharing operations; at the same time, if we place few demands on fabric elements, then deployment of Grid infrastructure is simplified. For example, resource-level support for advance reservations (i.e., the ability to request and obtain a commitment for access at a future time; see Chapter 18) makes it possible for higher-level services to aggregate (coschedule) resources in interesting ways that would otherwise be impossible to achieve. However, as many resources do not support advance reservation "out of the box," a requirement for advance reservation would increase the cost of incorporating new resources into a Grid.

Experience suggests that, at a minimum, resources should implement *introspection* mechanisms that permit discovery of their structure, state, and capabilities (e.g., whether they support advance reservation), on the one hand, and *resource management* mechanisms that provide some control of delivered quality of service, on the other, as in the following examples:

- *Computational resources*. Mechanisms are required for starting programs and for monitoring and controlling the execution of the resulting processes. Management mechanisms that allow control over the resources allocated to processes are useful, as are advance reservation mechanisms (see Chapter 18). Introspection functions are needed for determining hardware and software characteristics as well as relevant state information such as current load and queue state in the case of scheduler-managed resources.
- *Storage resources*. Mechanisms are required for putting and getting files. Third-party and high-performance (e.g., striped) transfers are useful (652). So are mechanisms for reading and writing subsets of a file and/or executing remote data selection or reduction functions (118). Management mechanisms that allow control over the resources allocated to data transfers (space, disk bandwidth, network bandwidth, CPU) are useful (587), as are advance reservation mechanisms (see Chapters 18 and 22). Introspection functions are needed for determining hardware and software characteristics as well as relevant load information such as available space and bandwidth utilization.
- *Network resources*. Management mechanisms that provide control over the resources allocated to network transfers (e.g., prioritization, reservation) can be useful (see Chapter 1:19). Introspection functions should be provided to determine network characteristics and load (682).

Other important classes of resources include database systems used to store structured data (see Chapter 22) and sensors of various kinds.

4.2.2 Connectivity: Communicating Easily and Securely

The connectivity layer defines core communication and authentication protocols required for Grid-specific network transactions. Communication protocols enable the exchange of data between fabric layer resources. Authentication protocols build on communication services to provide cryptographically secure mechanisms for verifying the identity of users and resources.

Communication requirements include transport, routing, and naming. Although alternatives certainly exist, it is common to assume that these protocols are drawn from the TCP/IP protocol stack: specifically, the Internet (IP and ICMP), transport (TCP, UDP), and application (DNS, OSPF, RSVP, etc.) layers of the Internet-layered protocol architecture (96). This is not to say that in the future, Grid communications will not demand new protocols that take into account particular types of network dynamics—for example, on high-performance optical networks or wireless networks.

The complexity of the security problem makes it important that any connectivity layer security solutions be based on existing standards whenever possible. As with communication, many security standards developed within the context of the Internet protocol suite are applicable. Chapter 21 provides a comprehensive discussion of the security demands of VO environments (146), so we note here just some of the more important requirements:

- ✦ *Single sign-on*. As Grid users frequently want to initiate computations that access multiple remote resources, a user should be able to "sign on" (authenticate) just once, rather than once per resource or administrative domain accessed.
- ✦ *Delegation* (280, 303, 371). A user must be able to endow a program with the ability to run on the user's behalf, so that the program is able to access the resources on which the user is authorized. The program should (optionally) also be able to delegate a subset of its rights to another program: what is sometimes referred to as restricted delegation.
- ✦ *Integration with local security solutions*. In a heterogeneous Grid, each site or resource provider may employ any of a variety of local security solutions. Grid security solutions must be able to interoperate with these various local solutions. They cannot, realistically, require wholesale replacement of local security solutions but rather must allow mapping into the local environment.
- ✦ *User-based trust relationships*. For a user to use resources from multiple providers together, the security system must not require each of the resource providers to cooperate or interact with each other in configuring the security environment. For example, if a user has the right to use sites A and B, the user should be able to use sites A and B together without requiring that A's and B's security administrators interact.

Grid security solutions should also provide flexible support for communication protection (e.g., control over the degree of protection, independent data unit protection for unreliable protocols, support for reliable transport protocols other than TCP) and enable stakeholder control over authorization decisions, including the ability to restrict the delegation of rights in various ways.

We discuss briefly below and in far more detail in Chapter 21, in the following, how these and related requirements can be addressed using a relatively small set of standard connectivity protocols.

4.2.3 Resource: Sharing Single Resources

Having established identity, the Grid user needs to be able to interact with remote resources and services. This is the role of the resource layer, which builds on connectivity layer communication and authentication protocols to define protocols for the secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. Resource layer implementations of these protocols call on fabric layer functions to access and control local resources. Resource layer protocols are concerned entirely with individual resources and hence ignore issues of global state and atomic actions across distributed collections; such issues are the concern of the collective layer discussed next.

Two primary classes of resource layer protocols can be distinguished:

- *Information protocols* are used to obtain information about the structure and state of a resource, for example, its configuration, current load, and usage policy (e.g., cost).
- *Management protocols* are used to negotiate access to a shared resource, specifying, for example, resource requirements (including advanced reservation and quality of service) and the operation(s) to be performed, such as process creation or data access. Since management protocols are responsible for instantiating sharing relationships, they must serve as a "policy application point," ensuring that the requested protocol operations are consistent with the policy under which the resource is to be shared. Issues that must be considered include accounting and payment. A protocol may also support monitoring the status of an operation and controlling (for example, terminating) the operation.

Although many such protocols can be imagined, the resource (and connectivity) protocol layers form the neck of our hourglass model and as such should be limited to a small and focused set. These protocols must be chosen so as to capture the fundamental mechanisms of sharing across many different resource types (for example, different local resource management systems), while not overly constraining the types or performance of higher-level protocols that may be developed.

The list of desirable fabric functionality provided in Section 4.2.1 summarizes the major features required in resource layer protocols. To this list we add the need for "exactly once" semantics for many operations, with reliable error reporting indicating when operations fail.

4.2.4 Collective: Coordinating Multiple Resources

The collective layer contains protocols and services not associated with any one specific resource but instead capturing interactions across collections of resources. Because collective components build on the narrow resource and connectivity layer "neck" in the protocol hourglass, they can implement a wide variety of sharing behaviors without placing new requirements on the resources being shared. For example:

- *Directory services* allow VO participants to discover the existence and/or properties of VO resources. A directory service may allow its users to query for resources by name and/or by attributes such as type, availability, or load (204). (See, in particular, Chapters 22 and 23—and the discussion of MDS-2 in Chapter 27.)
- *Coallocation, scheduling, and brokering services* allow VO participants to request the allocation of one or more resources for a specific purpose and the scheduling of tasks on the appropriate resources. Examples include AppLeS (114), Condor-G (292), Nimrod-G (57), and the DRM broker (108). (See Chapters 18, 24, and 1:12.)
- *Monitoring and diagnostics services* support the monitoring of VO resources for failure, adversarial attack ("intrusion detection"), overload, and so forth. (See Chapter 20).
- *Data replication services* support the management of VO storage (and perhaps also network and computing) resources to maximize data access performance with respect to metrics such as response time, reliability, and cost (66, 368). (See Chapter 22.)

Programming models and tools (discussed at length in Chapter 24) often define and/or invoke collective layer functions.

- *Grid-enabled programming systems* enable familiar programming models to be used in Grid environments, using various Grid services to address resource discovery, security, resource allocation, and other concerns. Examples include Grid-enabled implementations of the Message Passing Interface (300, 403) (Chapter 24) and manager–worker frameworks (156, 443) (Chapters 19 and 24).

- *Workflow systems* provide for the description, use, and management of multistep, asynchronous, multicomponent workflows (see Chapter 24).
- *Software discovery services* discover and select the best software implementation and execution platform based on problem parameters; for example, see NetSolve (153) and Ninf (497), described in Chapter 24.
- *Collaboratory services* support the coordinated exchange of information within potentially large user communities, whether synchronously or asynchronously; for example, see CAVERNsoft (220, 437), Access Grid (182) (Chapter 15), Butterfly.net (Chapter 13), and commodity groupware systems.

Collective layer services must also address security, policy, and accounting issues:

- *Community authorization servers* enforce community policies governing resource access, generating capabilities that community members can use to access community resources (519). These servers provide a global policy enforcement service by building on resource-layer information and management protocols and security protocols in the connectivity layer. Akenti (648) addresses some of these issues. See Chapter 21 for further discussion.
- *Community accounting and payment services* gather resource usage information for the purpose of accounting, payment, and/or limiting of resource usage by community members.

These examples illustrate the wide variety of collective layer protocols and services that are encountered in practice. Note that whereas resource layer protocols must be general in nature and are widely deployed, collective layer protocols span the spectrum from general purpose to highly application- or domain-specific, with the latter existing perhaps only within specific VOs.

Collective functions can be implemented as standalone services or as libraries designed to be linked with applications. In both cases, their implementation can build on resource layer (or other collective layer) protocols and APIs. For example, given a collective coallocation API that uses a resource layer management protocol to manipulate underlying resources, we can define a co-reservation service protocol and implement a co-reservation service that speaks this protocol, calling the coallocation API to implement coallocation operations and perhaps providing additional functionality, such as authorization, fault tolerance, and logging. An application might then use the co-reservation service protocol to request end-to-end network reservations.

Collective components may be tailored to the requirements of a specific user community, VO, or application domain, for example, a library that implements an application-specific coherency protocol, or a co-reservation service for a specific set of network resources. Other collective components can be more general-purpose, for example, a replication service that manages an international collection of storage systems for multiple communities or a directory service designed to enable the discovery of VOs. In general, the larger the target user community, the more important it is that a collective component's protocol(s) and API(s) be standards based.

4.2.5 Applications

The final layer in our Grid architecture comprises the user applications that operate within a VO environment. Applications are constructed in terms of, and by calling upon, services defined at any layer. At each layer, we have well-defined protocols and APIs that provide access to some useful service: resource management, data access, resource discovery, and so forth.

We emphasize that—as is discussed in more detail in Chapter 24—what we label "applications" and show in a single layer in Figure 4.3 may in practice call upon sophisticated frameworks and libraries (e.g., the Common Component Architecture (88), SciRun (517), Cactus (110), workflow systems (130)) and feature much internal structure that would, if captured in our figure, expand it out to many times its current size. These frameworks may themselves define protocols, services, and/or APIs, for example, Web service orchestration frameworks.

4.3 IMPLEMENTING GRID ARCHITECTURE

As discussed in the introduction, the technologies used to implement Grid architecture concepts have evolved over time, from a de facto standard in the form of the Globus Toolkit version 2 to the more formal standard Open Grid Services Architecture (OGSA), implemented by the Globus Toolkit version 3 (GT3) as well as other open source and commercial systems.

We briefly review here the principal features of GT2 and explain both how these features address the Grid technology requirements introduced previously and how they fit into our Grid architecture. We then introduce OGSA. We start with some general remarks concerning the utility of a service-oriented Grid architecture, the importance of being able to virtualize Grid services, and essential service characteristics. Then we define what we mean by a *Grid service*. Technical details on OGSA are provided in Chapter 17 and subsequent chapters.

GT2 has proven to be effective and influential not only because it provides technical solutions to challenging problems encountered when building Grids, such as authentication and secure resource access, but also because it does so by defining standard protocols that enable interoperability. Both advantages have proven important in practice, with many developers writing tools and applications that assume the basic functions introduced in the following (GSI security, GRAM resource access, GridFTP data access, etc.) and large numbers of sites deploying GT2 services in support of these tools and applications (see Chapter 27).

4.3.1 Globus Toolkit Version 2

GT2 (276, 281) is a community-based, open-architecture, open source set of services and software libraries that support Grids and Grid applications. GT2 addresses issues of security, information discovery, resource management, data management, communication, fault detection, and portability. GT2 is the foundation for thousands of major Grid projects worldwide in both academia and industry. Several of these projects are discussed in Chapter 27, which discusses practical issues that arise when deploying GT2 services.

4.3.1.1 Fabric

GT2 is primarily concerned with implementing Grid protocols, not fabric-level behaviors, and as such assumes the existence of suitable software on fabric elements for such purposes as local CPU scheduling, file system management, and system monitoring. However, the toolkit does include some components designed to facilitate interfacing to resource-level protocols. For example, software is provided for discovering structure and state information for various common resource types, such as computers (e.g., OS version, hardware configuration, load (230), scheduler queue status), storage systems (e.g., available space), and networks (e.g., current and predicted future load (451, 681)), and for packaging this information in a form that facilitates the implementation of higher-level resource-layer protocols. Resource management, on the other hand, is generally assumed to be the domain of local resource managers, although the General-Purpose Architecture for Reservation and Allocation (GARA) (283) prototyped a "slot manager" that can be used to implement advance reservation for resources that do not support this capability. Others have developed enhancements to the Portable Batch System (PBS) (31) and Condor (446) that support advance reservation capabilities; see Chapter 27.

4.3.1.2 Connectivity

GT2's connectivity layer is defined by the public-key-infrastructure (PKI)-based Grid Security Infrastructure (GSI) (280) protocols, which provide for single sign-on authentication, communication protection, and some support for restricted delegation. In addition, standard Internet protocols are assumed. In brief:

- *Single sign-on* allows a user to authenticate once and then create a proxy credential that a program can use to authenticate with any remote service on the user's behalf. A *proxy credential* is a digitally signed certificate that grants the holder the right to perform operations on behalf of the signer, typically only for a limited period of time. Proxy credentials are critical to Grid computing because they allow a user to initiate a computation that accesses multiple remote resources—without having to hand over sensitive credentials (such as a private key in a PKI-based system) to that computation.
- *Delegation* allows for the creation and communication to a remote service of a delegated proxy credential that the remote service can use to act on the user's behalf, perhaps with various restrictions; this capability is important for nested operations.

GSI addresses interoperability by defining a credential format and an authentication and remote delegation protocol for transmitting those credentials to remote services. The credential format is an extended form of the X.509 certificate, a widely employed standard for PKI certificates. The remote delegation protocol is based on the transport layer security (TLS) protocol (the follow-on to the popular secure socket layer, SSL), although other public key-based authentication protocols can also be used by GSI. In addition, GSI addresses application programmability and portability by defining extensions to the generic security services application programming interface (GSS-API) so that applications can invoke authentication operations conveniently using a high-level API, rather than performing protocol operations directly. Technical specifications define the credential (660), protocol, and GSS-API extensions (470). Further details on GSI design and implementation, including the techniques used to map to local security mechanisms, are provided in Chapter 21.

4.3.1.3 Resource

We now discuss those GT2 resource layer protocols used for remote invocation of computation, resource discovery and monitoring, and data transport.

The Grid Resource Allocation and Management (GRAM) protocol provides for the secure, reliable creation and management of remote computations (205). A two-phase commit protocol is used for reliable invocation, based on techniques used in the Condor system (446) (see Chapter 19 for details). GSI mechanisms are used for authentication, authorization, and credential delegation to remote computations. Multiple interoperable implementations of the GRAM protocol have been constructed (205, 386, 429), but there is no formal protocol specification.

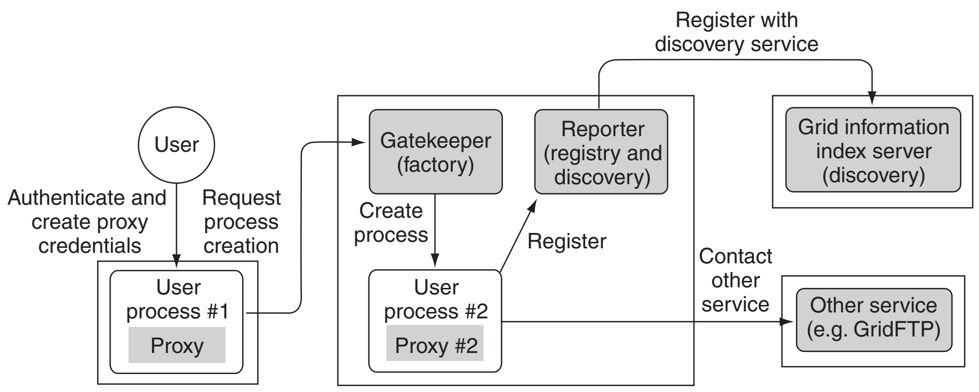
GT2's implementation of the GRAM protocol uses a small, trusted "gatekeeper" process to initiate remote computations, a "job manager" to manage the remote computation, and a "GRAM reporter" to monitor and publish information about the identity and state of the local computation.

The Monitoring and Discovery Service (MDS-2) (204) provides a uniform framework for discovering and accessing configuration and status information such as compute server configuration, network status, and the capabilities and policies of services. The framework defines both a *data model* for representing information about resources and services, and resource-level *protocols* for disseminating and accessing this information. The data model and protocols are described in Chapter 27.

The GT2 MDS-2 implementation provides two main components: a configurable *local registry* or data publisher used to manage the collection and dissemination of information at a particular location, and an index node or *collective registry* used to maintain and support queries against information from multiple locations. Both the local and collective registries perform caching so as to reduce the frequency with which requests for information must be communicated to the original source. Experimental studies show that this caching can be important in high-load situations (698).

The third GT2 component that we describe is GridFTP. This extended version of the file transfer protocol (FTP) is used as a management protocol for data access; extensions include use of connectivity layer security protocols, partial file access, and management of parallelism for high-speed transfers (64). FTP is adopted as a base data transfer protocol because of its support for third-party transfers and because its separate control and data channels facilitate the implementation of sophisticated servers. See Chapter 22 for more details.

Figure 4.4 shows these various GT2 resource layer protocols in action. We first see the user authenticating and generating a proxy credential for User


Register with discovery service
User
Authenticate and create proxy credentials
Request process creation
User process #1
Proxy
Gatekeeper (factory)
Reporter (registry and discovery)
Create process
Register
User process #2
Proxy #2
Grid information index server (discovery)
Contact other service
Other service (e.g. GridFTP)

4.4 FIGURE Selected Globus Toolkit mechanisms.

process #1, thus allowing that process to act on the user's behalf. That process then uses the GSI-authenticated GRAM protocol to request the creation of a new process at a remote location. This request is processed by the Gatekeeper process at the remote location, which creates the new User process #2 along with a new set of proxy credentials, which the new process can use to issue requests to other remote services. The existence and other properties of the new process are registered with the MDS-2 information infrastructure.

4.3.1.4 Collective and Tools

The GT2 software distribution provides only a limited number of collective layer capabilities: the DUROC resource coallocation library (206) is one example. Many GT2-compatible collective layer services and libraries, as well as programming and system tools that depend on those libraries, have been developed by others. Some of these systems have been mentioned in previous sections.

4.3.2 Open Grid Services Architecture

By 2001, the rapidly increasing uptake of Grid technologies and the emergence of the Global Grid Forum as a standards body made it timely and feasible to undertake a standardization (and, in the process, a significant redesign) of the core GT protocols. The following design goals drove this activity, which ultimately produced the Open Grid Services Architecture:

- *Factoring of component behaviors*. GT2 protocols such as GRAM combined several functions (e.g., reliable messaging for at-most-once invocation of remote services, and notification for monitoring) that ended up being either reimplemented or unavailable to other functions such as GridFTP or application programs. Thus, a goal in OGSA was to identify essential Grid functions and then to express them in a way that would allow their use in different settings.
- *Service orientation*. A *service* is a network-enabled entity with a well-defined interface that provides some capability. While GT2 defined service interfaces to specific resource types (e.g., GRAM for compute resources), service orientation was not consistently applied, and it did nothing to facilitate the definition of arbitrary services or service composition. A goal in OGSA was to enable a uniform treatment of all network entities so that, for example, collective layer behaviors could be expressed as virtualizations of underlying resource layer protocols.
- *Align with Web services*. GT2 builds on a mix of low-level protocols and did not provide any standard interface definition language. With the emergence of Web services as a viable Internet-based distributed computing platform, a design goal for OGSA was to leverage the Web services standards (e.g., the Web Services Definition Language, or WSDL), application platforms, and development tools.

The result of this design activity is a service-oriented framework defined by a set of standards being developed within the Global Grid Forum (GGF). A fundamental OGSA concept is that of the *Grid service:* a Web service (see Chapter 17) that implements standard interfaces, behaviors, and conventions that collectively allow for services that can be transient (i.e., can be created and destroyed) and stateful (i.e., we can distinguish one service instance from another). The foundational Open Grid Services Infrastructure (OGSI) specification defines the interfaces, behaviors, and conventions that control how Grid services can be created, destroyed, named, monitored, and so forth. OGSI defines a set of building blocks that can then be used to implement a variety of resource layer and collective layer interfaces and behaviors.

OGSA then builds on this OGSI foundation by defining the services and additional interfaces and behaviors required in a functional Grid environment. For example, interfaces are required for discovery, data management, resource provisioning, and service virtualization. Other services are required for security, policy, accounting, and billing. Service orchestration provides for the coordination of service workflow. GGF working groups are engaged in identifying required functions, rendering these functions as OGSI-compliant interfaces, and defining relationships among the resulting service definitions.

From an organizational perspective, these OGSA services can be viewed within the context of the layered architecture of Figure 4.3. At the connectivity layer, we have a small number of service definitions that address critical issues of authentication, credential mapping, and policy verification, while at the resource level we can identify another small set of service definitions, for data access, job submission, bandwidth allocation, and so on. In a virtualized environment, we find that interfaces defined at the resource layer can reappear at the collective layer as interfaces to virtualized services that from their observable behavior are indistinguishable from resource layer services. Our Grid architecture is thus recursive, with services being composed of services.

OGSA has been broadly adopted as a unifying framework for Grid computing with backing from major scientific and academic Grid communities as well as significant acceptance in the commercial section, from vendors to end users. There are multiple interoperable implementations of this common standards-based, protocol-oriented approach to infrastructure, including GT3, an open source reference implementation of the OGSI and basic OGSA services. A more detailed discussion of OGSA and OGSI can be found in Chapter 17 and in subsequent chapters that discuss specific functionalities and services.

4.4 THE GRID COMMUNITY

In the five years that have passed since the first edition of this book, the development of Grid technologies and applications has grown to become a worldwide effort. Participation in the Global Grid Forum (GGF) has multiplied more than tenfold since its inception as the U.S. Grid Forum in 1998. Today, delegates from over 400 organizations in over 50 countries attend the thrice-yearly GGF meetings to participate in working groups defining technical specifications or in research groups discussing future directions in Grid technologies and applications, or simply to learn about the technology.

A major impetus for this surge of interest is certainly the needs of the scientific community, which, as discussed in Chapter 2, has urgent demands across a broad range of fronts for more effective and large-scale integration of resources and services. Both grass-roots efforts and major government-funded Grid activities in Japan, Australia, Singapore, China, Europe, North America, and other places have spawned literally thousands of projects on Grid technologies and applications. Given the complexity of the overall problem and the international dimension of many scientific collaborations, the development of common technical standards and collaborative development of key software is vital to success. Thus, GGF fills an important role in the coordination of international Grid efforts.

Industrial involvement in GGF also continues to grow, with around one-third of GGF participants now coming from industry. Both established technology companies such as Fujitsu, HP, Hitachi, IBM, NEC, Microsoft, Oracle, Platform, and Sun and startups such as Avaki, DataSynapse, and United Devices participate in working groups. In addition, we see an increasing number of commercial end-users at GGF meetings.

Another major forum for discussion of Grid technologies is the annual GlobusWorld event, a combined user and technical training meeting for users of the Globus Toolkit.

4.5 FUTURE DIRECTIONS

The tools and experience established over the past five years provide a solid foundation for future developments in Grid applications and technologies. However, success in the ultimate goal of a global infrastructure for distributed system integration and resource sharing will require significant progress in standards, social and business models, and basic research. The sustained collaborative engagement of scientific application groups, industry, and the computer science community is vital.

OGSA stands poised to become the dominant infrastructure for Grids, with significant consolidation around both the general approach and specific specifications such as OGSI. The next major hurdle is to continue refinement of the OGSA model and specifically to define and develop the additional building block services (e.g., end-to-end provisioning, global service discovery, service virtualization) that will raise the level of abstraction for all Grid applications. If we take the Internet as an analogy, then OGSI provides us with TCP/IP; now we need to create the domain name service, routing protocols, and ultimately HTTP analogs. Later chapters discuss specific capabilities and services that, collectively, define a good part of what we expect to be the research, development, product, and application agendas for the next five years.

The development of new application paradigms is a major focus of current work and can be expected to expand in the future. Predictions by industry analysts have focused primarily on commercial intranet deployments with a particular emphasis on increasing resource utilization. Yet this perspective only scratches the surface of the Grid's capabilities. The flexible formation of dynamic collaborations can have a profound effect on organizational structure. However, to achieve this potential, we need advances in the way applications are structured, standardization of associated service definitions, and in some cases new business models as well. For example, dynamic federation requires not only services for dynamically evaluating and enforcing trust relationships but also a dynamic provisioning and payment model to produce a federated (or virtualized) service set that maps to the requirements of the collaboration.

There is an urgent need for program development and execution tools for Grid environments: compilers, debuggers, performance monitors, and libraries. High-level languages and programming paradigms suitable for dynamic environments are required. Grid-enabled libraries (Chapters 16 and 24) and workflow systems are important, but more sophisticated autonomic techniques should also be pursued (Chapters 25 and 26). We require an improved understanding of correctness, performance, troubleshooting, and optimization in dynamic environments. Users need to be able to assemble multiple services to create both Grid infrastructure and Grid applications that meet requirements. The cost of assembling these services must be sufficiently low that users can focus on their goals rather than on building customized infrastructure. Research is required on theories and techniques to describe and reason about the semantics and behavior of services and the compositional effects of putting services together. New tools to support the discovery, composition, and use of services based on high-level descriptions of requirements must also be developed.

We need to understand how to scale the Grid both to larger numbers of entities and to smaller devices. The future pervasive digital infrastructure will seamlessly combine reliable high-performance computing systems, communication networks, and variable low-performance embedded or portable devices with integrated wireless facilities. Resources will vary in their availability, quality, and reliability. Fundamental computing research is needed to enable the realization of trusted ubiquitous systems formed from the coalition of these potentially uncertain components. Effective solutions to these scaling problems must inevitably involve the development of infrastructure elements capable of adapting to changes in application or user needs without undue human intervention: what IBM has termed “autonomic computing” (365), a term that encompasses automated management, configuration, optimization, healing, and protection.

The social, economic, and political aspects of Grids are going to become increasingly important. (The third edition of this book will surely feature authors from the social sciences). We envision a wide variety of different Grids ranging from highly controlled “intra-Grids” using secure private networks to spontaneous community Grids using the global Internet. The large number of users, cultures, and usage modalities will demand not only new policy specification, monitoring, and enforcement mechanisms but also an improved understanding of the social and economic issues that influence stability and productivity. Social scientists also have much to contribute to our understanding of issues of trust (301, 321, 416) and usability (511, 676].

Another area in which fundamental research is required relates to the role of knowledge systems and services, not only for future Grid applications, but also for the effective functioning of the Grid itself. As discussed in Chapter 23, we need to be able to manage the traceability and integrity of information, and to trace provenance, all the way from initial data through information to knowledge structures. New theories and techniques are required to allow tolerant, safe, and scalable reasoning over uncertain and incomplete knowledge that embraces data, metadata, and knowledge activities.

SUMMARY

We have introduced the principal topics to be discussed at greater length in the chapters that follow. We provided a concise statement of the "Grid problem," which we define as controlled and coordinated resource sharing and resource use in dynamic, scalable virtual organizations. We have also both motivated and defined a Grid architecture, in which are identified the principal functions required to enable sharing within VOs and the key relationships that exist among these different functions. Finally, we have introduced the open source Globus Toolkit that has enabled the rapid adoption of Grid technologies and the Open Grid Services Architecture specifications now supporting the continued expansion of Grid technologies and applications.

FURTHER READING

For more information on the topics covered in this chapter, see *www.mkp.com/grid2* and the following references:

- ✦ Much of the material in Chapter 1:2 is not repeated here, and remains highly relevant.
- ✦ Two recent articles in *Physics Today* (269) and *Scientific American* (270) provide good high-level introductions to Grid computing and its applications.
- ✦ The 2003 report of the National Science Foundation's Blue Ribbon Panel on Cyberinfrastructure summarizes the scientific motivation for Grids (37).
- ✦ A recent book edited by Berman, Fox, and Hey provides good coverage of research in Grid computing (113). See also the proceedings of the annual IEEE Symposium on High Performance Distributed Computing (HPDC).