1

Introduction

1.1 Background

In the 1960s, the great science-fiction writer Isaac Asimov [1] predicted a future full of robots, protecting and sometimes controlling human destiny. Fifty years later, a human-like and all-purpose robot still remains a dream of the robotics research community. However, technological progress in the last couple of decades have ensured that human lifestyle, human interactions and collaboration patterns have changed so dramatically that if anyone like Asimov had written about today's world 50 years back, it would have seemed like science fiction. If we compare the interaction and collaboration patterns of today with those of a decade back, we will find stark differences between the two. E-mails, blogs, messengers and so on are common tools used nowadays which were unknown ten years ago. People seldom stand in a queue in a bank; automated teller machines (ATMs) have become an essential commodity. Similarly, credit cards have taken over from cash and cheques as the new mode of transaction. Internets have become the de facto source of information for millions of people. The new technologies have redefined the ways in which interaction and collaboration between different individuals take place, which in turn are creating a new social-interaction methodology. For example, English is fast becoming a lingua franca for the technical community across the world and the interactions of that community are redefining the English language in a significant way. In addition, geographical and cultural borders are slowly disappearing as social networking sites like Orkut [2], Facebook [3] and so on change the ways people interact. Similar changes are also taking place in the enterprise-computing scenario. Until recently, application developers could safely assume that the target environment was homogeneous, secure, reliable and centrally-managed. However, with the advent of different collaborative and data-sharing technologies, new modes of interaction are evolving. These evolutionary pressures generate new requirements for distributed application development and deployment. Enterprises are now witnessing increasing collaboration and data sharing among the different participating entities, resulting in the need for and use of distributed resources and computing. Another important element that has increased the complexity of IT operations is the need for integration of different applications, with middleware developed in different platforms and by different vendors. We are also seeing a spurt of mergers and acquisitions which require integration of technologies across enterprises. Moreover, the enterprises are outsourcing the nonessential elements of the IT infrastructure to various forms of service provider. The technologies that have transformed the world so significantly fall under the bracket of *distributed computing* technologies.

Distributed computing technologies follow a similar pattern of interaction, where disparate and sometimes heterogeneous systems interact with one another over a common communication platform. Initiated by the academic and research community to fulfill the need to connect and collaborate, slowly this technology was adopted by enterprises. Finally, enterprises and user communities cannot live without some application of distributed computing. However, with the widespread adoption of distributed computing, experts are pointing out security issues that can hurt the enterprises and user communities in a huge way. Analyzing the security issues and solutions in distributed computing is not simple as there is a need to identify the interactions between different layers of the distributed computing environment. Different solutions exist and it is necessary to identify the different layers of the distributed computing environment and analyze the security issues in a holistic manner. This book is an effort in that direction.

1.2 Distributed Systems

Distributed systems involve the interaction between disparate independent entities, bounded by common language and protocols and working toward a common goal. Different types of distributed systems are found in real life. One of the biggest and perhaps the most complex distributed system is human society itself. In the digital world, the Internet has become a very important distributed environment for everybody.

1.2.1 Characteristics of Distributed Systems

If we look at any distributed system, for example the Internet, there are several mandatory characteristics, in addition to ‘good-to-have’ or desirable characteristics. Mandatory characteristics determine the basic nature of distributed systems, such as having multiple entities, heterogeneity, concurrency and resource sharing.

(1) *Multiple entities:* One of the key characteristics of a distributed system is the presence of multiple – in many cases a great many – entities participating

in the system. The entities can be users or subsystems which compose the distributed system.

(2) *Heterogeneity:* Another key characteristic is the heterogeneous nature of the entities involved. The heterogeneity may lie in the type of system or user, underlying policies and/or the data/resources that the underlying subsystems consume. The heterogeneity of distributed systems can be best observed in the Internet, where multitudes of systems, protocols, policies and environments interact to create a scalable infrastructure.
(3) *Concurrency:* Another important characteristic that distinguishes any distributed system from a centralized one is concurrency. Different components of distributed systems may run concurrently as the components may be loosely coupled. Therefore there is a need to understand the synchronization issues during the design of distributed systems.
(4) *Resource sharing:* Sharing of resources is another key characteristic of distributed systems.

In addition to the above mandatory characteristics, there are several desirable characteristics for a distributed system.

(1) *Openness:* A desirable characteristic for a distributed system is openness of the underlying architecture, protocols, resources and infrastructure, where they can be extended or replaced without affecting the system behavior. If we look at the Internet, this issue is nicely handled through the use of open standards: we can see the interplay between different protocols, standards, infrastructures and architectures without affecting the activities of the Internet as a whole.
(2) *Scalability:* One of the key motivations for going from a centralized system to a distributed one is to increase the overall scalability of the system. Hence to have a highly scalable system is desirable in any form of distributed system.
(3) *Transparency:* Another desirable characteristic is to have transparency in the operation. From the user's and the subsystem's point of view, the underlying systems should be transparent. Primarily, transparency can be of two types – *location transparency* and *system transparency*. The first type talks about the need to be transparent regarding the location disparity between different systems. The second talks about the need to be transparent about system issues like failure, concurrency, scaling, migration and so on.

1.2.2 Types of Distributed System

Distributed systems can be divided into mainly three types: distributed computing systems, distributed information systems and distributed pervasive systems. The first type of system is mainly concerned with providing computations in a distributed manner. The second type of system is mainly concerned with providing

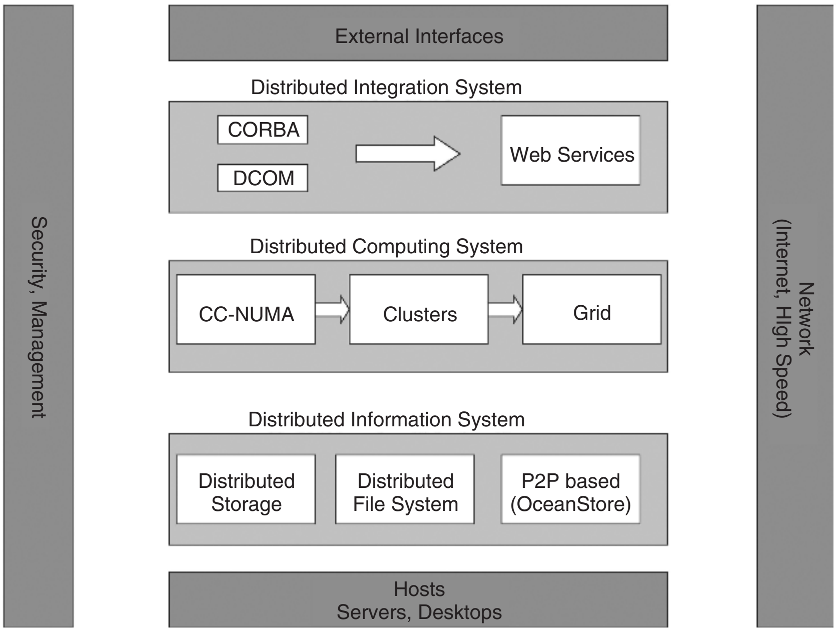

Figure 1.1 Distributed system landscape.

information in a distributed manner, while the third type is the next-generation distributed system, which is ubiquitous in nature.

1.2.2.1 Distributed Computing Systems

Distributed computing systems provide computations in a distributed manner. Computing power is needed in many different industries, including banking and finance, life sciences, manufacturing and so on. If we look at the computing resources available, we shall find that the laptops of today are perhaps as powerful as servers a decade ago. Moore's law, which states that computing power doubles every 18 months, is valid even today and will probably be true for the next 5–6 years. With the growth of the multicore technologies, Moore's law can be extended even further [4]. Computing power is increasing and so is demand. In this rat race, researchers have found an able ally in the form of networking. Between 2001 and 2010, while processing power is supposed to increase 60 times, networking capabilities are supposed to increase by 4000 times. This means that at the same cost, 4000 times the same bandwidth will be available in 2010 as compared to 2001 [5]. Therefore, the computing architectures developed a decade back will probably require a rethink based on the technological progress in the fields of computers and networks. Last decade saw the development of a field called *cluster computing* [6], where different computing resources are connected together using a very-high-speed network like Gigabit Ethernet or more recently Infiniband [7]. In addition to the technological progress and the huge requirement of computing power, the enterprises have also undergone a radical shift in IT operations in the last few years. Enterprises are now witnessing increasing collaboration and data sharing among the different participating entities, resulting in the need for and use of distributed resources and computing. Another important element that has increased the complexity of IT operations is the need for integration of different applications: middlewares developed on different platforms and by different vendors. We are also seeing a spurt of mergers and acquisitions that require integration of technologies across enterprises. Moreover, the enterprises are outsourcing the nonessential elements of the IT infrastructure. The dual pull of requiring more computing power and the integration of heterogeneous components into the IT infrastructure has led to the development of *grid technology*. This technology is seeing a classical evolution pattern. Initiated by the academic and research community to fulfill its needs, it is slowly being adopted by the enterprises, especially those who have high computing needs, such as the life sciences, finance and manufacturing industries. However, the promise of grid computing goes beyond that and the next few years should see a gradual adoption of grid as the natural choice among the other enterprises. But a widespread adoption of grid computing depends upon the ability of researchers and practitioners to reduce the pitfalls that lie along the way. One such pitfall is *security*, which is the focus of this book as a whole. In this chapter we will briefly talk about grid computing's evolution, benefits and concerns.

1.2.2.2 Distributed Information Systems

Distributed information systems are responsible for storing and retrieving information in a distributed manner. There are many manifestations of this type of distributed system. The underlying storage system can be distributed in the form of storage area networks (SANs). SANs have become de facto storage infrastructures in most enterprises. SAN is a high-speed data storage network that connects different types of storage device. One of the most popular modes of storage communication is the Fibre Channel fabric. Another paradigm of the distributed information system is the distributed file system (DFS). The first secure DFS in common use was atheos file system (AFS) [8]. This file system was later followed by DFS [9]. AFS servers store sub-trees of the file system and use Kerberos [10] to provide authenticated access to the trees. Network file system (NFS) is another very popular DFS, which allows users distributed over the network to access distributed files. With the growth of peer-to-peer (P2P) technologies, highly-distributed storage is in vogue. Systems like OceanStore [11] are becoming popular. This uses a large number of untrusted storage devices to store redundant copies of encrypted files and directories in persistent objects. Objects are identified by globally unique identifiers (GUID), which are generated in a similar fashion to the unique identifiers in SAN file system (SFS). Each identifier is a hash of the owner's public key and a name. Objects can point to other objects to enable directories. All objects are encrypted by the client. By replicating the objects among servers, clients can even avoid malicious servers deleting their data. The extensive use of replication and public keys makes revocation of access and deletion of data difficult to achieve, but it does provide a nice model for a completely decentralized DFS.

1.2.2.3 Distributed Integration Systems

Distributed integration systems are responsible for integrating applications, policies and interfaces across diverse distributed systems. The last couple of decades have seen numerous implementations of distributed computing, such as CORBA [12], Java RMI [13], DCOM [14] and so on. None of these systems were taken up in a big way by the industries, mainly because of their tightly-coupled nature. Current trends in the application space suggest that enterprises are moving away from monolithic tightly-coupled systems toward loosely-coupled dynamically-bound components. With the growth of the Internet as a premier means of communication, a new paradigm called the Web Services [15] emerged, facilitating a new style of architecting systems, termed as service-oriented architecture (SOA). Web Services can be thought of as reusable, loosely-coupled software components that are deployed over the network, or specifically the World Wide Web. There are some advantages that the experts claim as the major reasons for the adoption of Web Services as a de facto standard for application integration. These are:

(1) *Simplicity:* Implementation of Web Services is very simple from the point of view of programmers and as a result, easy and fast deployments are possible. All the underlying technologies and protocols are based on Extended Markup Language (XML) [16], which is simple and intuitive.
(2) *Loosely coupled:* Since the very design of Web Services is based on loose coupling of its different components, they can be deployed on demand.
(3) *Platform independent:* Web Services architecture is platform- and language-independent since it is based on XML technologies. Therefore, one can write a client in C++ running on Windows, while the Web Service is written in Java running on Linux.
(4) *Transparent:* Since most of the deployed Web Services use Hypertext Transfer Protocol (HTTP) [17] for transmitting messages, they are transparent to firewalls, which generally allow HTTP to pass through. This may not always be the case for CORBA, RMI and so on.

According to many experts, CORBA and RMI provide a much better alternative to Web Services because of the flexibility and features that CORBA provide. Moreover, performance-wise the CORBA/RMI combination may be better than protocol designed over HTTP. However, because of its simplicity and the backing of the big commercial vendors, Web Services is steadily becoming a standard which none can ignore. There are many forums where debates are being pursued as we move on to the different components which constitute the Web Services. There are three main components of Web Services:

- *SOAP:* The Simple Object Access Protocol (SOAP) [18] is a lightweight protocol for exchange of information between diverse and distributed computing environments. It combines the extensibility and portability of XML with the ubiquitous Web technology of HTTP. It provides a framework for defining how an XML message is structured, using rich semantics for indicating encoding style, array structure and data types.
- *WSDL:* The Web Service Description Language (WSDL) [19] can be used to describe a Web Service, providing a standard interface. A WSDL document is written in XML and describes a service as a set of endpoints, each consisting of a collection of operations. XML input and output messages are defined for each operation and their structure and data types are described using an XML Schema in the WSDL document. The Web Description Services Language (WDSL) and XML Schema provide a complete definition for the service interface, allowing programmatic access to the Web Service in the manner of an API. Tasks like data requests or code executions can be performed by sending or receiving XML messages using, for example, SOAP.
- *UDDI:* The Universal Description, Discovery and Integration (UDDI) [20] specification defines a way to publish and discover information about Web Services. It is a collaboration between Ariba, IBM and Microsoft to speed interoperability and adoption of Web Services. The project includes a business registry (an XML document) and a set of operations on it. The registry can be used by programs to find and get information about Web Services and check compatibility with them, based on their descriptions. UDDI allows categorization of Web Services so that they can be located and discovered, and WSDL enables a programmatic interface to a service once it has been located.

1.2.3 Different Distributed Architectures

There are four different types of architecture that are used for designing distributed systems, namely client–server-based systems, Multinode systems, P2P systems and service-oriented systems. The first type of system is a client- and a server-based system, the second type of system distributes the data or the information across multiple nodes or systems, the third type of system is a P2P-based architecture where all components are peers or at the same level, and the last type of system is a federated model where interactions happen via standards-based messages.

1.2.3.1 Client–Server-Based Architecture

Client–server-based architecture is the most popular distributed system that has been used over the years. In this architecture, the server is responsible for providing service to the client or a set of clients. In a client–server kind of environment, a client requests a service from the server, which the server provides over the network in a remote environment. The main advantage of a client–server system is that the business services are abstracted from the set of clients accessing them. Security is implemented at the link and the end server, while fault tolerance is applied at the server end by replicating the functionality. Though extremely popular, there are some inherent limitations in this type of architecture, which led practitioners and researchers to other models.

- *Scalability:* One of the primary limitations of this model is scalability. If the number of users increases significantly, the system fails to handle such a large load. There are two ways to handle this issue: scale up or scale out. Scaling up means moving to a higher end server to handle the same type of request. Though this may be an effective solution in some cases, it does not scale as there is a limitation to scaling up. The second approach, or scale-out approach, distributes the server into multiple servers, which improves scalability. We will talk about this approach later.
- *Flexibility:* Just having a client and a server reduces the overall flexibility of the system, the reason being that database management, application business logic and other processes are embedded in the server code and hence inflexible. Practitioners and designers slowly moved to a three-tier architecture mainly to tackle this problem.

1.2.3.2 Multinode

One variation of the client–server technology distributes the server into multiple nodes that can be used for parallel processing. There are several advantages of such a multinode configuration, namely *performance*, *fault tolerance* and *scalability*. Performance can be improved, since the different nodes involved in the process provide part of the service a single node was supposed to perform. Different components of the multinode system are: processing nodes, scheduler or load balancer and clients. Having different nodes perform similar actions can result in improvement of fault tolerance. Moreover, multinode systems improve scalability since they can scale out instead of scaling up. However, the advantages of multinode systems come at a cost, which is complexity. Managing synchronization, security, load balancing and so on in such an environment is extremely challenging.

1.2.3.3 Peer-to-Peer

The third type of architecture, which is becoming at the moment, is P2P. This type of system is different from client–server-based systems as, in P2P systems, all the nodes in the distributed system participate in the same hierarchy. This means that there is no concept of client and server, and each participant assumes the role of client and server based on need and requirement. Systems like Gnutella, Napster and so on are based on such principles. P2P systems have found significant applications in the area of file distribution and transfer. They are also being applied in the area of data and information storage. P2P systems have several advantages in terms of scalability and fault tolerance. Since the system is dependent on end systems and nodes, it scales infinitely. The scalability property is exhibited by all the P2P systems. Similarly, fault tolerance is also an important characteristic of such a system. However, several challenges exist, in the form of security and service level agreement (SLA). Since the end systems are responsible for performance, guaranteeing service is almost impossible. Management of security is also extremely difficult as maintaining security at the end systems is a challenge.

1.2.3.4 Service-Oriented Architecture

SOA is the latest in the evolution of distributed architectures, which builds upon the client–server and other such distributed architecture models. SOA implementations revolve around the basic idea of a service. A service refers to a modular, self-contained piece of software, which has a well-defined functionality expressed in abstract terms independent of the underlying implementation. Basically, any implementation of SOA has three fundamental roles: service provider, service requestor and service registry, and three fundamental operations: publish, find and bind. The service provider publishes details pertaining to service invocation with a services registry. The service requestor finds the details of a service from the service registry. The service requestor then invokes (binds) the service on the service provider. Web Services, described earlier, represent the most popular form of implementation of SOA.

1.2.4 Challenges in Designing Distributed Systems

The challenges in designing a distributed system lie in managing the different disparate entities responsible for providing the end service. Synchronization between the different entities needs to be handled. Similarly, security and fault tolerance are extremely important and need to be handled as well.

1.2.4.1 Synchronization

One of the most complex and well-studied problem in the area of distributed systems is synchronization. The problem of synchronizing concurrent events also occurs in nondistributed systems. However, in distributed systems, the problem gets amplified many times. Absence of a globally-shared clock, absence of global shared memory in most cases and the presence of partial failures makes synchronization a complex problem to deal with. There are several issues, like clock synchronization, leader election, collecting global states, mutual exclusion and distributed transactions, which are critical and have been studied in detail in literature.

- *Clock synchronization:* Time plays a crucial role as it is sometimes necessary to execute a given action at a given time, timestamping data/objects so that all machines or nodes see the same global state. Several algorithms for clock synchronization have been proposed, which include synchronization of all clocks with a central clock or through agreement. In the first case, the time server or external clock periodically sends the clock information to all the nodes, either through a broadcast or through multicast mechanisms, and the nodes adjust the clock based on the received information and the round-trip time calculation. In the second mechanism, the nodes exchange information so that the time clock can be calculated in a P2P fashion. It is to be noted that clock synchronization is a major issue in distributed systems and clock skew always needs to be considered when designing such a system.
- *Leader election:* This is another critical synchronization problem used in many distributed systems. Many varieties of solution are available, ranging from the old leader forcing the new leader on the group members based on certain selection criteria, to polls or votes where the node receiving the maximum number of votes gets elected as the leader.
- *Collection global state:* In some applications, especially when debugging a distributed system, knowledge of the global states is especially useful. Global state in a distributed system is defined as the sum of the local states and states in transit. One mechanism is to obtain a distributed snapshot which represents the consistent and global state in which the distributed system would have been. There are several challenges in moving a process to the consistent state.
- *Mutual exclusion:* In some cases, it is required that certain processes access critical sections or data in a mutually-exclusive manner. One way to tackle such a problem is to emulate the centralized system by having the server manage the process lock through the use of tokens. Tokens can also be managed in a distributed manner using a ring or a P2P system, which increases the complexity.

1.2.4.2 Fault Tolerance

If we look at the issue of fault tolerance from the distributed systems perspective, it is both an opportunity and a threat. It is an opportunity as distributed systems bring with them natural redundancy, which can be used to provide fault tolerance. However, it is a threat as the issue of fault tolerance is complex, and extensive research has been carried out in this area to tackle the problem effectively. One of the issues that haunts distributed systems designers is the source of many failures. Failures can happen in processing nodes and transmission media, and due to distributed agreement.

- *Processing sites:* The fact that the processing sites of a distributed system are independent of each other means that they are independent points of failure. While this is an advantage from the viewpoint of the user of the system, it presents a complex problem for developers. In a centralized system, the failure of a processing site implies the failure of all the software as well. In contrast, in a fault-tolerant distributed system, a processing site failure means that the software on the remaining sites needs to detect and handle that failure in some way. This may involve redistributing the functionality from the failed site to other, operational, sites, or it may mean switching to some emergency mode of operation.
- *Communication media:* Another kind of failure that is inherent in most distributed systems comes from the communication medium. The most obvious, of course, is a permanent hard failure of the entire medium, which makes communication between processing sites impossible. In the most severe cases, this type of failure can lead to partitioning of the system into multiple parts that are completely isolated from each other. The danger here is that the different parts will undertake activities that conflict with each other. Intermittent failures are more difficult to detect and correct, especially if the media is wireless in nature.
- *Errors due to transmission delays:* There are two different types of problem caused by message delays. One type results from variable delays (jitter). That is, the time it takes for a message to reach its destination may vary significantly. The delays depend on a number of factors, such as the route taken through the communication medium, congestion in the medium, congestion at the processing sites (e.g. a busy receiver), intermittent hardware failures and so on. If the transmission delay is constant then we can much more easily assess when a message has been lost. For this reason, some communication networks are designed as synchronous networks, so that delay values are fixed and known in advance. However, even if the transmission delay is constant, there is still the problem of out-of-date information. Since messages are used to convey information about state changes between components of the distributed system, if

the delays experienced are greater than the time required to change from one state to the next, the information in these messages will be out of date. This can have major repercussions that can lead to unstable systems. Just imagine trying to drive a car if visual input to the driver were delayed by several seconds.
- *Distributed agreement:* The problem of distributed agreement has been briefly touched upon in the previous subsection. There are many variations of this problem, including time synchronization, consistent distributed state, distributed mutual exclusion, distributed transaction commit, distributed termination, distributed election and so on. However, all of these reduce to the common problem of reaching agreement in a distributed environment in the presence of failures.

1.2.4.3 Security

Perhaps the most compelling challenge associated with distributed systems is the issue of security. The complexity of the issue arises from the different points of vulnerability that exist in a distributed system. The processing nodes, transmission media and clients are the obvious points that need to be secured. With the growth of heterogeneity in different layers of enterprise infrastructure, the complexity increases enormously. This whole book is devoted to this subject. In the next section, we will provide a brief motivation for different layers of distributed systems in an enterprise scenario and touch upon the security issues to be delved into in this book.

1.3 Distributed Systems Security

As mentioned earlier, security in distributed systems is critical and absolutely essential. However, it is also extremely challenging. Distributed security in the digital world is akin to security in the real world. As the last few years would suggest, protecting physical infrastructure is turning out to be a nightmare for security professionals. The reason is that malicious adversaries can reside anywhere, and everything is their potential target. In the digital world as well, protecting the infrastructure is turning out to be a catching game. The main reason for this is that the IT infrastructure in all enterprises is distributed in nature. Before understanding the security in distributed systems in relation to enterprise IT, we need to understand the enterprise IT landscape. In this section, we will discuss the enterprise IT scenario in a layered perspective. The whole book will then be aligned to this layered view with respect to distributed IT security.

1.3.1 Enterprise IT – A Layered View

Figure 1.2 shows a high-level view of the layered enterprise. The view consists of four main layers: hosts, infrastructure, applications and services. While the

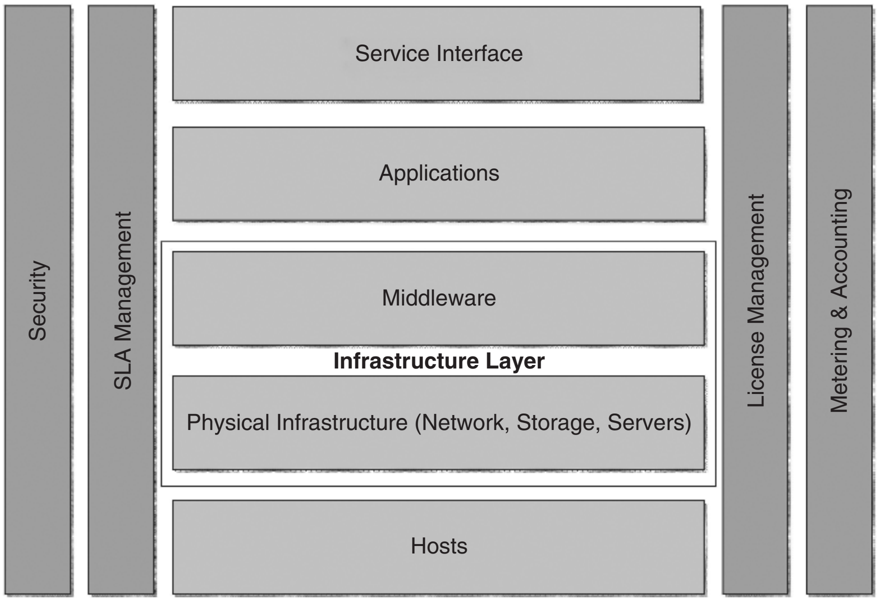

Figure 1.2 Layered enterprise view.

host layer consists of client desktops and low-end servers, forming the lowest stratum of the enterprise IT, the infrastructure layer consists of network, storage and middleware functionalities, which are used by the application, host and service layers. The applications are custom and component of the shelf (COTS) applications that are used in the enterprises in a day-to-day manner. Finally, we have the service layer, which provides standards-based services to the external world as well as to the enterprise itself. We will take this view into account for all our subsequent discussions.

1.3.1.1 Hosts

The lowest layer in the enterprise consists of hosts, which are mainly composed of client desktops and low-end servers. Even a few years back, hosts were meant only to submit requests to servers and perform some low-end user-level tasks like editing Word files and so on. However, with the growth of grid computing, concepts like cycle stealing, scavenging and so on are coming to the fore. Middleware technologies are able to take advantages of idle desktops or low-end servers like the blades for distributed processing. With the growth of P2P technologies, hosts are also managing distributed files in a much more scalable manner. Technologies like Torrent [21] are redefining the way storage is carried out today. With the growth of dimensions of hosts, several issues need to be tackled which were not a problem before.

- *Manageability:* The issue of managing heterogeneous systems is becoming increasingly complex. With hundreds and thousands of hosts a part of the computing and storage infrastructure, this is an administrator's nightmare. Several management and monitoring tools are needed to address this problem.
- *Metering:* With hosts becoming more and more important in the overall business scenario, metering assumes a very important role. How to meter and what to meter are serious questions.
- *Security:* Perhaps the most challenging issue in the host-based storage and computation is security. Not only do hosts need to be protected from malicious outside agents, infrastructure needs to be protected from malicious hosts as well.

1.3.1.2 Infrastructure

The second layer in the IT enterprise is the infrastructure. It is diverse and complex because of the sheer heterogeneity of products, technologies and protocols used in enterprises today. The infrastructure is basically composed of two main components: the physical infrastructure, consisting of high-end servers, storage and networks, and the middleware, consisting of the cluster and grid middlewares.

- *Physical infrastructure:* Physical infrastructure consists of the server infrastructure, network infrastructure and storage infrastructure. One of the key characteristics of the physical infrastructure is heterogeneity. From the size and type of servers used, through the networking speed to the storage devices, heterogeneity remains a key ingredient. Another key characteristic is that each component of the physical infrastructure is distributed in nature, making security a major concern.
- *Middleware:* Cluster and grid middleware dominates in a high-performance environment. Integration of grid technologies with mainstream IT is a challenge.

1.3.1.3 Applications

Applications address the diverse enterprise needs of an enterprise, be they business applications or horizontal technological applications. Of special importance is the emergence of Web-based applications as a crucial component of enterprise landscape, essential for delivering the right functions to consumers via the Web. From a security perspective, application security has assumed major importance in the recent past. Security issues may crop up due to either weakness in the design of an application, or an insecure coding practice, which can compromise some of the security requirements. In the case of the Web, the openness of the medium and the protocols is responsible for further security complexity.

1.3.1.4 Services

Services represent a higher level of interaction in distributed systems, building over the underlying applications and data. Hence the typical underlying distributed system security issues (including confidentiality, integrity and so on) are applicable; additionally, the specific concerns arising out of the loose coupling and interaction via XML messages introduce extra complexities. The higher level of loose coupling required for SOA mandates more flexible ways of handling security for SOA. Additionally, standards will be key as there is a need to interoperate across heterogeneous implementations of underlying systems. Finally, the openness and plain-text nature of XML-based distributed invocations is a cause of further complexity and higher vulnerability. Likewise, typical distributed-system attacks like DOS, cross-site scripting attacks and so on manifest at service level too, albeit with variations.

1.3.2 Trends in IT Security

As we move toward a distributed IT infrastructure, security issues become more and more critical. The pervasive growth of the IT infrastructure, along with its heterogeneous nature, makes security a really complex issue to look at. If we look at a typical IT infrastructure, there are hundreds of different applications that are interacting with one another. The applications are either custom built, vendor products or even open-source systems. Each of the products interacts with complex sets of infrastructure components, including servers, desktops, middlewares and so on. Added to this complexity is that of the heterogeneous networking infrastructure, including wired, wireless and so on, and devices like BlackBerrys, personal digital assistants (PDAs) and others. With the growth of sensor networks, integration of IT infrastructure and small sensor motes will make the problems exceedingly challenging. With the heterogeneity and pervasive nature of enterprises set to grow, several security trends have been identified in this section, which are slowly being adopted by enterprises around the world. The key security trends that can be observed are: movement of security to higher layers, protection of the periphery, protection of identities, standardization and integration of heterogeneous policies and infrastructure.

1.3.2.1 Security in Higher Layers

One of the security trends that is observed currently is the movement of security implementation to higher layers. If we look at the different layers of enterprise systems, security protocols and systems are available at each and every one. For example, most of the enterprises conform to SOA. Different security protocols are available at the infrastructure layer, the middleware layer and so on. Enterprises are slowly exploring the ideas of having security at the Web Services layer, which has led to the standardization and development of WS-Security standards. Similarly, enterprises are looking at securing the higher layers so that more flexibility can be obtained. However, one of the issues in moving security up the layers is performance versus scalability. The higher the security implementation the more the security overhead, and hence the more the performance overhead. Therefore, the decision to have security at a particular layer depends on the amount of flexibility that the system requires and the performance requirement of the system. Taking the above example, instead of WS-Security, one can implement Transport Layer Security (TLS). The performance of a TLS-based system will be more than that of a WS-Security-based system; however WS-Security provides an end-to-end secure environment which is not provided by TLS.

1.3.2.2 Protection of the Periphery

Another important trend that can be observed in enterprise-security cenarios is that the security is provided at the periphery to protect the network by filtering the traffic flowing to and from it. Different types of filtering technique are employed to protect the data flowing into the network. Filtering can be as simple as going through the packets and preventing data coming from certain ports. Similarly, requests going to a particular port can be prevented. However, enterprises are also moving toward more sophisticated methods of filtering, like application-level filtering and XML-based filtering techniques. In these techniques, the filters or firewalls actually look into the XML or application payload and identify whether the packet is of a malicious nature.

1.3.2.3 Protection of Identities

When I reflect upon my activities today, I find that I have used multiple credentials to access resources of different forms. I used my company identity card to enter the office premises, entered the password to get into the office network, used my smart card to access the high-security lab, used my personal identification number (PIN) to access my ATM account, and used my passport to get a US visa – and this was just one day. Different identity checks were required by different systems, and my identities were in different forms, which I either carried in my head or as a card or a paper. I am surely not an exception; every one of us is doing the same, maintaining multiple credentials to access different forms of resource. This has really become pronounced with the growth of Information Technologies, where there are multitudes of system interfaces which require some sort of user authentication. As a result, individuals possess multiple digital identities and credentials, many of which are short-lived. At this point, one may be concerned about the relationship between identities and credentials. The identity of an individual user is unique; however, it may be manifested in different ways to disparate systems through user credentials. For example, my identity credential to the United States consulate is my passport, while to the company network it is the combination of the network's user ID and password. Therefore, when we talk of managing different user identities, it is actually the user-identification credentials we are talking about. However, credentials go beyond just identifying the user: they may authorize a user to access a certain resource or be used as a proof of authentication. Credentials can be short-lived, for example identity cards or passwords which expire when the individual leaves a company, or after a fixed amount of time. Other examples of short-time credentials are the tickets issued in busses for a short ride. Individuals manage their credentials by a combination of papers, cards and their own memory, as I did today. Secure management of user credentials is a very important challenge. Identity theft topped the list of complaints to the United States Federal Trade Commission in 2002, accounting for 43% of all complaints [22]. Therefore, identity and user-credential management is surely a very important problem, and several research and development efforts are being undertaken in this direction.

1.3.2.4 Standardization

With the growth of heterogeneity in the enterprises, standardization is fast becoming a key for any enterprise security system. If we look at any enterprise, the number of heterogeneous elements available is mind-boggling. Several enterprises over the years have custom-built applications, even middlewares, to interact with vendor products. When architects look at the security issues in such enterprises, they find the need to integrate security across these products, middlewares and applications. The way to solve the problem is through standardized interfaces and protocols. There are a couple of advantages to taking the standardized route, especially in designing the security systems in enterprises. Firstly, rather than designing a custom protocol, standards are based on well-established theoretical bases and principles. Hence, through standards, one can be sure that vulnerabilities are not introduced in those layers. Secondly, standard interfaces make integration a slightly less cumbersome problem.

1.3.2.5 Integration

Perhaps the most complex and challenging problem in any enterprise is the integration of different protocols, standards, applications, middlewares and so on. This becomes especially complex for new and evolving technologies, like grid computing for example. Though technically grid computing is a powerful technology which provides flexibility and performance in the current infrastructure setup, when enterprises move into the grid environment, the challenges of integration just hide all the benefits that exist. Enterprises which have application servers, data-base tiers, different business intelligence products and monitoring tools, and management systems, would integrate with an open-source Globus toolkit that is based on standards like Security Assertion Markup Language (SAML) and WS-Security. However, the enterprises do not support those standards and either the enterprise applications have to move to the newer standards, which may involve a lot of work and customization, or the grid systems must be customized to work with the existing standards and protocols. In most cases, enterprises prefer the second route as they generally do not want to touch the systems which 'work'. As a result, the vicious circle of newer technologies and integration difficulties persists.

1.4 About the Book

In this book we look at the global picture of the distributed computing systems and their security vulnerabilities, and the issues and current solutions therein. We divide the distributed systems into four layers: infrastructure, host, application and service. The reason for this layering lies in the fact that enterprises have systems built in this manner, and integration issues come to the fore when analyzing them as we have done. The host issues look at the issues pertaining to a host in a distributed computing system and the vulnerabilities that it will be subjected to; vulnerabilities include mobile codes coming into the system and tasks being executed. The infrastructure level issues concern the infrastructure as a whole, that is the networking and the mobile infrastructure on which the distributed systems are perched. The application layer is concerned with applications that are being developed on top of the infrastructure. Lastly, the service layer looks at building distributed services. As we can see, each of the layers presents unique challenges in terms of security. Moreover, the book looks at the orchestration of applications and services across the different layers in order to look at the global picture.

1.4.1 Target Audience

The book does not assume that the reader is an expert in security or distributed systems technologies. However, some prior knowledge about general security principles and/or distributed computing technologies will be required to understand the chapters covering advanced security issues. The book is primarily targeted at architects who design and build secure distributed systems. It would also benefit managers who make business decisions and researchers who can find research gaps in existing systems.

- *Professionals and architects:* Through this book, professionals and architects working on distributed systems will be made aware of the security requirements. It will also enlighten them about the security features of some existing open-source as well as proprietary products. The book also aims at identifying processes and models which could help architects design more secure systems.

- *Managers and CIOs:* Though the book has significant technical depth, managers and CIOs will gain significantly from it by understanding the processes, gaps and solutions which exist. The book therefore will be able to provide them with information which will be useful for making important business decisions.
- *Researchers and students:* Experienced researchers and students in the field of distributed computing will be able to get a comprehensive overview of all the security issues in distributed computing. This will help them make important research decisions by analyzing the existing gaps.

References

[1] Isaac Asimov Online (2008) http://www.asimovonline.com, accessed on June 13th, 2008.
[2] Orkut (2008) http://www.orkut.com, accessed on June 13th, 2008.
[3] Facebook (2008) http://www.facebook.com, accessed on June 13th, 2008.
[4] Koch, G. (2005) *Discovering Multi-Core: Extending the Benefits of Moore's Law*, Technology Intel® Magazine.
[5] Vilett, C. (2001) *Moore's Law vs. Storage Improvements vs. Optical Improvements*, Scientific American.
[6] Rajkumar B. (eds) (2003) *High Performance Cluster Computing: Architectures and Systems*, Kluwer Academic Publishers.
[7] InfiniBand Trade Association (2008) *InfiniBand Architecture Specification*, Vol. **1**, Release 1.1, November 2002. Available from http://www.infinibandta.org, accessed on June 13th, 2008.
[8] Howard, J.H., Kazar, M.L., Menees, S.G. *et al.* (1988) *Scale and Performance in a Distributed File System*, Vol. **6.1**, ACM Transactions on Computer Systems, pp. 51–81.
[9] Kazar, M.L., Leverett, B.W., Anderson, O.T. *et al.* (1990) Decorum file system architectural overview. *Proceedings of the Usenix Summer 1990 Technical Conference*, USENIX, pp. 151–64.
[10] Neumann, C. and Ts'o, T. (1994) Kerberos: an authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, September 1994.
[11] Kubiatowicz, J., Bindel, D., Chen, Y. *et al.* (1999) *Oceanstore: An Architecture for Global-Scale Persistent Storage*, ASPLOS.
[12] Vinoski, S. (1997) CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14, 2 February 1997.
[13] Grosso, W. (2001) *Java RMI*, O'Reiley.
[14] Ruben, W. and Brain, M. (1999) *Understanding DCOM*, Prentice Hall. ISBN 0-13-095966-9.
[15] Cerami, E. *Web Services Essentials*, O'Reilley, ISBN: 0596002246, 2002.
[16] Ray, E. *Learning XML*, O'Reilley, ISBN: 0596004206, 2003.
[17] Fielding, R., Gettys, J., Mogul, J. *et al.* (1999) Hypertext Transfer Protocol – HTTP 1.1, IETF RFC 2616, June, 1999.
[18] W3C Team (2003) SOAP Version 1.2, Part 0: Primer, W3C Recommendations, June 2003.
[19] W3C Team (2001) Web Services Description Language (WSDL) 1.1, W3C Note, March 2001.
[20] Computer Associates (2005) IBM, Microsoft, Oracle, SAP, SeeBeyond Technologies, Systinet, and Others, UDDI v.3.0, *OASIS Standard*, Feb 2005.
[21] BitTorrent (2008) http://www.bittorrent.org, accessed on 13th June 2008.
[22] Basney, J., Yurcik, W., Bonilla, R. and Slagell, A. (2006) Credential Wallets: A Classification of Credential Repositories, Highlighting MyProxy. 31st Annual TPRC, Research Conference on Communication, Information, and Internet Policy, Sep. 2006.