Abstract

Wireless LANs, WLANs are now deployed at great pace both on company premises and in public hot spots. Having greatly improved the mobility of professional users, this advantage obviously has also created new threats to security. Accordingly, new needs for authentication and authorization and possibly accounting and charging ("AAA") of users did evolve. In this context, the IEEE 802.1x standard provides a framework for such AAA tasks. Originally designed for wire-based LANs, it is now extensively used for wireless LANs such as the IEEE 802.11 family of wireless LANs. 802.1x uses the IETF Extensible Authentication Protocol (EAP), the latter supporting multiple authentication methods. This paper describes a proposal to use 802.1x based AAA-functionalities to realise public WLAN usage scenarios such as "closed community membership" and "pre-paid, pay-per-use".

Introduction

Every user of computer is being affected due to rapid change in sizes of software. According to one survey the time is not so far when the computer memory will be sold in terabytes. But the question arises here, now we not need only large memories but also security and flexibility. Data sharing is one of the major issues of distributed computing. Actually distributed means branches of computers connected through wires or wireless mechanism. In distributed environment distribution of data can be done in such a way that while querying data, request can be sent on more then one computer from any data access point. Mobile computing is another hot issue in the field of computer sciences. From this not only our mind became free from spaghettis of wires but also it made our world a global village.

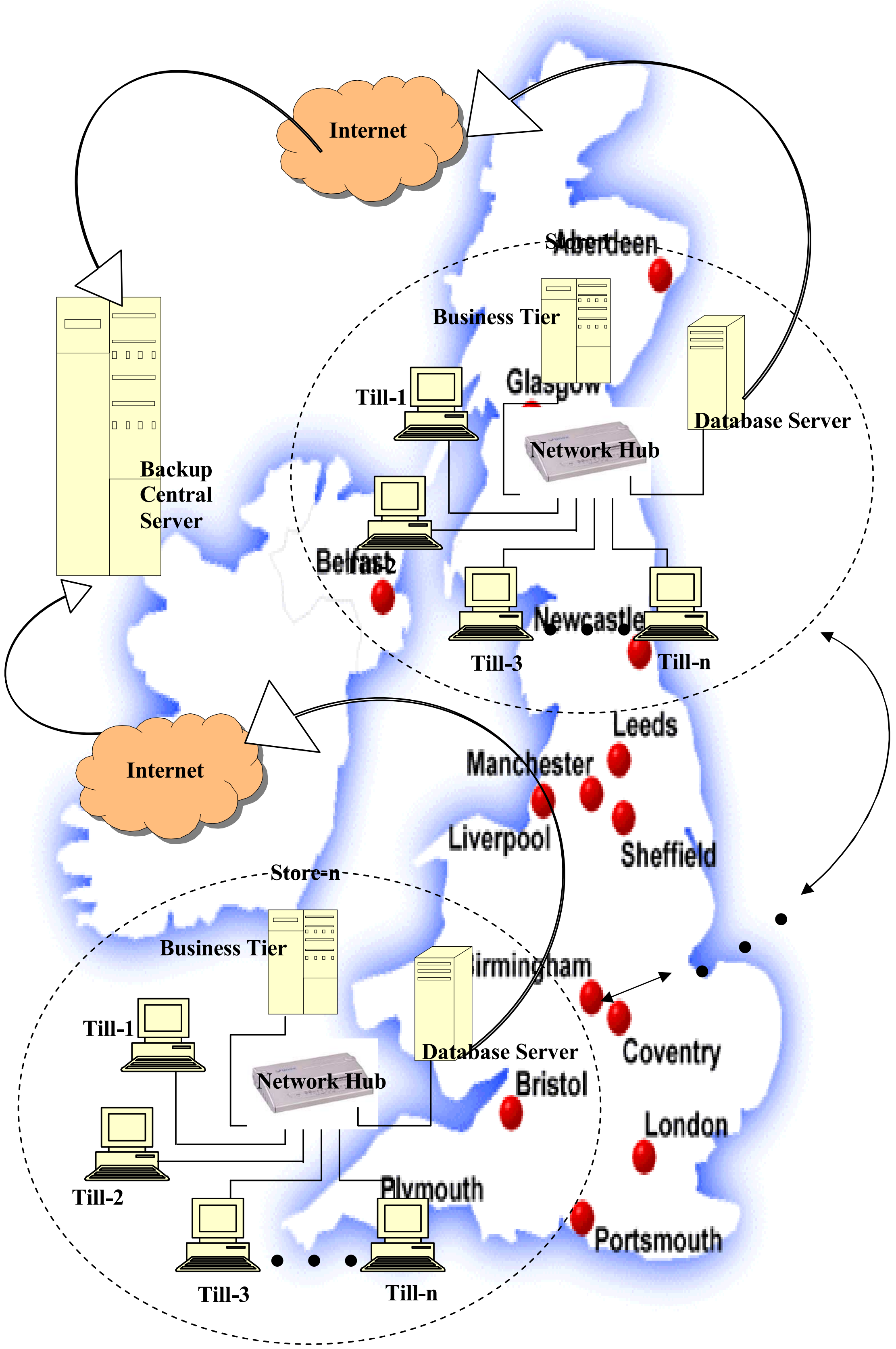
Security remained a main issue while creating any distributed system. Information leakage threat increases when computer send or receive any request or data from other computer. There are different techniques discussed later to make the environment secure.

The availability of wireless network connections to laptop computers and PDA's has created interest in the issues surrounding mobile computing. However, enabling users to be genuinely mobile in their work requires more than a wireless connection. Distributed system services are needed to support the locating of people, equipment and software objects, and, especially for mobile multimedia applications, network transport protocols which can adapt to a wide range of networking conditions must be developed.

The evolution of IEEE 802.11-based Wireless Local Area Networks (WLANs) and the development of mobile devices like laptops have changed the usage of Ethernet networks. Previously, computers were used to connect to an organization's network statically. Now, WLANs and mobile devices, providing for ubiquitous computing, have greatly improved the mobility of professional users. However, this mobility advantage has obviously also created new threats to security. The corresponding problem is how to decide which one of the possibly many users in an open radio cell is allowed to access the network, what resources he or she may use and how to possibly account and charge the use of resources. The IETF calls this important task "AAA" – Authentication – Authorization – Accounting, or "Triple A" for short. In the given context, one framework to establish an AAA-framework is the IEEE 802.1x standard. Initially targeting all of the many IEEE 802 LANs and MANs, 802.1 xs is now extensively used for AAA of WLANs.

Studying the proposal given in Authorisation and charging in public WLANs using FreeBSD and 802 [Pekka Nikander, 2002], this paper describes the use of 802.1x to set up interesting public WLAN usage scenarios such as "closed community membership" and "pre-paid pay- per-use accounts". Secondly this paper briefly describes concepts of IEEE 802.11, IEEE 802.1x, IETF EAP and IETF RADIUS. This also covers 802.11 based WLANs and different authentication methods, which leads how these technologies and authentication standards can support different public WLANs usage scenarios including pre-paid pay-per-usage accounts and community membership based accounts.

The idea of performing client-server computing by the transmission of executable programs between client and servers has been popularised in recent years. Different vendors had introduced many technologies to compete in the race of distributing software system, like CORBA, EJB, RMI, ODP, DCOM etc. In this paper the role of Microsoft Distributed Component Object Model, its advantages, disadvantages, its comparison with other distributed technologies has been discussed. Actually Microsoft Distributed Component Object Model is an object-oriented model designed to promote interoperability of software objects in distributed, heterogeneous environment. In this paper I try to avoid distinctions between COM and DCOM. But DCOM is enhancing form of COM, so at some places COM reference is necessary. Whenever distribution is concern, Microsoft relates DCOM with CORBA. CORBA provides the basic mechanism to use object on remote machines and it bases on object request broker architecture.


Internet
Store-1
Aberdeen
Business Tier
Database Server
Till-1
Glasgow
Network Hub
Backup
Central
Server
Belfast
Till-2
Newcastle
Till-3
Till-n
Leeds
Manchester
Internet
Liverpool
Sheffield
Store-n
Business Tier
Birmingham
Till-1
Database Server
Network Hub
Coventry
Bristol
London
Till-2
Plymouth
Portsmouth
Till-3
Till-n

Security, Flexibility and Scalability

It is not wrong in saying that technology is domineering current era. As computer sciences is rapidly rushing towards advancement, but its leaving a debate for us, "Is this innovation free from security threats, is system flexible and scalable to use in future"? It is very important to discuss these issues for every computer system and especially for distributed environments. Today there are many examples from which medium data is transferring from on system to another. GSM digital cellular telephone network is an example of system. Bluetooth is also another technique, as Frank Stajano (2002) described that "*Bluetooth is an embedded radio system for short-range communication between small devices*". There are some security services provided with Bluetooth which are authentication and encryption. In this chapter we can find that which problems distributed systems are facing with respect to security, flexibility and scalability and how it can be resolve? Before going in depth of these issues there is little about how computer works in distributed networks environment.

Let's start with wireless security. When we start using wireless you free your computer from the rat's nest of wiring that is your LAN. But by sending your network traffic through the air, instead of via CAT-5 cables, you are broadcasting data including passwords to anyone who is listening within range. The simple solution is use encryption for applications that send information in clear text. FTP and Telnet are two week examples for security point of view. SSH and SFTP are the safe equivalents, and SSH can even be used to tunnel other protocols, such as POP3, which also sends clear passwords.

First we can enable the WEP feature on your wireless access point. This encrypts all of the data that traverses the wireless section of network. It doesn't matter that WEP has been cracked, because few people in reality will be listening in and even fewer will be able to gather sufficient information to crack your key. If security is of utmost importance, place a firewall between the access point and rest of the LAN. This segments the relatively insecure systems away from main network. Next enable a VPN on this firewall and have it accept only traffic encrypted with the VPN client software. This will encrypt wireless traffic as far as firewall, which will pass the unencrypted traffic to the LAN using a cabled connection.

2.1 Securities in networks

Today security is the biggest barrier faced by mobile computing on the way to their progress. It's not a new matter of discussion. It remained a threat from beginning of computer systems. Even since humans started communicating with each other, there has been a need to keep secrets. Same techniques of keeping secrets are required, when computers interact with one another. Actually technology is using in both positive and negative senses. Here is an example how it is using in negative sense. Whenever we hear the word hacker or cracker we get a negative image in our minds, everyday in newspapers, magazines, movies, TV etc we hear that some one got hacked due to which they had to suffer heavy looses. In general a hacker is considered to be a person who uses his skill with computers to try to gain unauthorised access to computer files or networks. Chris Brenton (2001) described that "*Active Defence states that a hacker is a person who has deep understanding of computers and networking*" further he says that hacker is some one who feels the need to go beyond the obvious. Means they are not the dumb person, it is different matter how they are using their capabilities. Following are the briefly explained some security techniques using in distributed network systems.

2.1.1 SSL Secure Socket Layer

SSL is a secure layer of protocols which runs between the layers of TCP/IP and higher-level protocols such as HTTP or IMAP. TCP/IP on behalf of the higher-level protocols helps to authenticate SSL-enabled client over SSL-enabled server. That also results to create an encrypted connection between client and server. SSL security technology not only helps to improve the safety of Internet communications but also on intranet. That's why SSL is a standard for encrypted client/server communication between network devices. It has ability to maintain secure data traffic over the network that has very importance in the distributed environment. SSL generates a key after each encrypted transaction, which comes in two strengths, 40-bit and 128-bit. This strength refers to length of key generated by transaction. As the key length will increase it will difficult to break the encryption security.

2.1.2 Encryption

Data encryption is generally considered the best technique securing data storage and transmission. According to techWeb (1999) statement "*Encryption is the transformation of data using an algorithm, from one form to another utilizing one or more encryption keys during the transformation process*". It is the art of storing information in a form that allows it to be revealed to those you wish to see it yet, hides it from all others. The original information to be hidden is called "plain text". The hidden information is called "cipher text". Encryption is any procedure to convert plain text into cipher text. Decryption is any procedure to convert cipher text into plain text. Public and private are key types of encryption. Encryption and decryption are two ways conversion of data from one form to another. While encryption, data changes to its original form depends on its algorithms. On the other hand if we run the same algorithm in reverse condition, then data again comes to its original form.

Supposing some wants to send credit card information from one computer to another over the Internet, what he will do is use public key to encrypt it and then the private key to decrypt it. Public key algorithms are very complex maths problems some are based on prime numbers, factoring, elliptic curves, logarithms etc. In the public key cryptosystem, the public can only determine private key if you solve the problem. There for the most important thing is that the problem should be so complex and time consuming that the attacker would think twice even before proceeding.

2.1.3 Digital Signatures

Digital signature is another way to make the system secure. "*Digital Signature is an electronic code that is attached to a message or file that gives it a unique identity and allows you to certify that a message or file that is sent by you actually came from you*". (Felix Weber, 2001) In order to truly understand the implications of digital signatures one must understand what a digital signature is and how it works. Simply, a digital signature is a way for you to identify yourself electronically. Just like a hand-written signature identifies a person separately, a little piece of computer data will do the same for a computer. But how do digital signatures work and what do they protect or sign. Well suppose you want to send someone an email and you want the recipient to know that the email did in fact originate from you. The way you would do that is by using a digital signature. But in order for the digital signature to ensure authenticity there must be some way to verify that this signature came from you. This is accomplished by using key pairs. When you get a digital signature of your own you are assigned a public key and a private key. The private key is used to sign the document, whichis then sent to the recipient. The recipient then uses your public key to verify the signature.

Here is an example of how digital signature works in distributed environment. Let's say computer A want to send a signed documents to computer B. A uses his private key to generate his digital signature or fingerprint. The private key creates a code series based on a complex mathematical algorithm that is embedded in the message that A sends. When B gets the electronic document he then uses A's public key which he got from a public key server to then verify the digital signature and ensure that the message or document that he has received is in fact from A. The whole communication process between both of them is secure communication. Different Internet security companies issue a PIN number that is unique to a computer.

2.2 How we can make a system secure?

While developing a security policy an organisation must identify those entities that are considered valuable enough to undergo security measures, with in these entities certain resources are more valuable than others and more focus should be given to them. Like in an electronic funds transfer system the exposure of the financial transactions likely will have more severe impact than the exposure of a personnel record of a customer. Following these principles would help in avoiding a lot of common security problems. That is true that this set of principles will not be able to cover every possible flaw that could show up, but it will minimise the chances of any kind of hacking or cracking and make the system more reliable. Here I am going to discuses a few important ones.

2.2.1 Securing the weakest link

One of the most common analogies in the security community is that security is a chain of links. A secure system is most likely secure as the weakest link. Hackers will always look for and attack weakest parts of the system. It's probably no surprise that the hackers will always tend to go after low hanging fruit. If they target your system for whatever reason, they're going to take the path of least resistance. That means they'll try to attack the parts of the system that look weakest, and not the parts that look strong. A similar kind of logic is widely applicable to the physical world. There is always more money in a bank than a general store, but which one is more likely to be robbed?, the general store, of course. Why? Because banks tend to have much stronger security precautions; general stores are much easier targets. This principle has is hundred percent applicable in the software world, but most people don't pay any attention. In particular, Kirk Job-Sluder (2002) concluded that *"Cryptography is always considered to be the weakest part of a software system".* Even if you use Secure Socket Layers1 with 512-bit RSA keys and 40-bit RC4 keys, which are considered incredibly weak cryptography, an attacker can probably find much easier ways to break in a system rather tan just attacking Cryptography. Sure, it is breakable, but doing so still requires a large computational effort. If the attacker wants access to the data that travels over the network, then they'll probably target one of the end points and try to find a flaw like a buffer overflow or memory leakage, and then they dig out the stage of data when it gets encrypted or decrypted. All the cryptography in the world cannot secure data if there is a buffer overflow.

2.2.2 Social engineering: A common weak link

Sometimes it is not the software that is the weakest link in your system; it could also be the general surrounding considers social engineering when an attacker uses social manipulation to break into a system. Typically, a service centre will get a call from a sincere sounding user, who will talk the service professional for a password that should not be given away. This sort of attack is generally quite easy to launch, because customer service representatives do not like to deal with stress. If they are dealing with someone is really mad about not being able to get into his account, most of them will give away the password and will try to calm the situation down.

One good strategy is to limit the capabilities of technical support as much as possible. Like the entire Customer service representatives wouldn't be allowed to peep or change the passwords of the users only a selected few persons should do it after a lot of questioning and enquiries. I remember when someone hacked my hotmail account I mailed hotmail's technical staff for help and they asked me a few but very complex questions which could only be answered by a real account user, the questions were like, name the last three passwords you used, what date the account was last accessed by me, what approx. date did I create my account etc.

2.2.3 Practice defence in depth

Here we can see the concepts behind in practicing defence in depth is to make the system risk free with many defensive techniques. So that if one layer of defence fails, another layer of defence will work, fixing the breach. Let us return to our example of providing security for a bank. Why is the typical bank more secure than the typical shopping store? Because there are a lot of security measures protecting the bank, there are a lot of security cameras security guards etc. If we look at the computer systems, always multiple defence systems should be installed, like if SSL doesn't work firewalls should and if firewalls don't encryption should.

2.2.4 Security failure

Software systems do have failure modes some get pretty unavoidable, what are avoidable are security problems related to failures. The problem is that when many systems fail in any way, they revert to insecure behaviour. In such systems, attackers only either wait that right kind of failure happen automatically or they try to create a right kind of cause for failure. Remote Method invocation (RMI) has a similar problem. When client and server wants to communicate over RMI and the server uses SSL or some other encryption protocol, the client does not support the protocol the server uses, the client downloads the implementation of proper socket from the server at runtime. This is a big security flow, because the server has not been authenticated at the time that the encryption interface was downloaded. An attacker could pretend to be the server, installing implementation of his own socket on each client, even when the already SSL installed on the client side. The problem is that if the client fails using default libraries in secure connection establishment, it will make a connection using any protocol an un-trusted entity gives it, thereby extending trust. The development teams and proper should solve these kinds of problems ensuring that system doesn't run in an abnormal state.

2.2.5 Be Reluctant to Trust

Programmers usually hide secretes in client code, just to minimise the server resources and assuming that their secret will be safe. But clever hackers always try to exploit these tricks and most of the time they are successful, I know that developers use a JavaScript code to validate credit card numbers on the client side, but if the hacker is clever enough he surly can generate a new credit card number out of the algorithm that verifies it. Trust is often easily extended in the area of social engineering that is the reason social engineering attacks are easily to launch.

2.3 How distributed system can be flexible?

Now distributed systems are running on mixture of mainframes servers and midrange platforms. Running different versions of UNIX, Novel NetWare and Windows NT, 2000, with workstations of different flavours of Microsoft Windows is not a big deal now. Because now systems became so complex and fast that it is possible that any company is running COBOL, PowerBuilder, or Java clients to talk to Sybase, Oracle, DB2, or IMS database at same time. It was old stage when development of new software and purchasing or replacing equipment was too crucial and costly. Because if any organisation is using same old information system from last twenty years, it's mean they are wasting their time and efforts by not using latest technology. As this part of chapter discusses how to make distributed system flexible, so for that each distributed object supports other objects with quality of services.

While designing distributed systems it should keep in mind that each distributed object has to provide flexible services. Each resource of distributed system denotes as object. As Tetsuo Kanezuka and Makoto Takizawa (1998) discussed that "*An object is an encapsulation of data and operations for manipulating the data*". Due to this encapsulation system becomes easy to use and there are less chances of fault. There can be two ways to realise the fault-tolerant system; one is check pointing and other is replication. In check point protocol state concluded by Chandy, Misra, and Haas (1983) "*the state of the object is saved in the stable log at the checkpoint*". Its mean if the object has some fault, the object will roll backed to the checkpoint by restoring its state. On the other hand replication occurs on a specified time, depends on its schedule settings. First of it will check the connection between data servers. If there is something wrong either with host server or remote server. It will display an error message. But it will keep trying to ping the remote or local server. If something goes wrong while replicating data it roll backs all the information to point of start. These two approaches show how easy to trap fault if system is flexible. Main advantage of flexibility is that system can make decision by analysing system situation. For example if any organisation is using multicasting and broadcasting over the intranet then it can be quite effective but it will not so flexible while using on world wide web scale.

2.3.1 Object Naming

Providing proper name spacing is very necessary in distributed environment. Distributed objects can be named in a variety of ways. Objects name should be depend on object identifiers. Object identity can be measure from the purpose of that object. When an object is created it is assigned an identifier that uniquely identifies that object. This denotes its functionality as well. For example object name "fax" can be use to refer to the local fax machine. Many distributed system offers an intermediate solution: a user-chosen name is used to refer to a single object. One name refers to one object at the same time but can be bound to different objects at different times.

Sometime organisations prefer to use naming system, which maintains a mapping from names to network addresses. Philip H., Leendert D., Maarten S., Andrew S. Tanenbaum, and Wiebren de Jonge (1995) study shows that there can be a disadvantage of this approach that it supports for objects that migrate, or objects that are replicated is hard. Updating the network addresses is especially difficult if multiple names refer to the same object. We propose to separate naming objects from locating objects. A lookup of a name in the name service returns a location independent object handle. This object handle is passed to the location service which maps the object handle to multiple communication end-points This extra level of indirection allows multiple names to refer to the same object while at the same time the actual location of the object is maintained in one place only (logically that is, the location service might be replicated).

2.3.2 Object Location

We use a single distributed location service to keep track of where objects are. The location service as a whole stores the location information about all distributed objects. Since the number of distributed objects can be extremely large, the complete state of the location service needs to be partitioned. These partitions are stored in location objects. Each location object keeps the location information of some number of distributed objects. Location objects can also refer to other location objects.

A location object is just a regular distributed object. This means that the state of a location object can be replicated and that different location objects can use different replication strategies, different consistency protocols, etc. Objects can be registered with different location objects depending on the locality, replication, etc., of the object. In general, it makes sense if an object is registered at a location object at a short distance (from a network perspective). Similarly, an object with a high replication degree should be registered at a location object that is also widely replicated.

2.4 How distributed system can be scalable?

Distributed systems should be scalable. Typical present and future applications include web-based applications, e-commerce, multimedia news services, distance learning, remote medicine, enterprise management, and network management. Because they should be deployable in a wide range of scales, in terms of numbers of users and services, quantities of data stored and manipulated rates of processing, numbers of nodes, geographical coverage, and sizes of networks and storage devices. Small scales may be just as important as large scales. Scalability means not just the ability to operate, but to operate efficiently and with adequate quality of service, over the given range of configurations. Increased capacity should be in proportion to the cost, and quality of service should be maintained. CORBA ORBs must scale efficiently and predictably as the number of objects in end systems and distributed systems increases. Scalability is important for large-scale applications that handle large numbers of objects on each network node, as well as a large number of nodes throughout a distributed computing environment.

Networks like ATM, FDDI, and Fibre Channel now support QoS guarantees for bandwidth, latency, and reliability. However, my research on performance evaluation of conventional CORBA ORBs indicate that significant overhead is incurred when they are used for performance-sensitive and real-time applications over high-speed networks. Aniruddha Gokhale (1999) described about his experiments which empirically validated for poor ORB performance. He discussed about lack of integration with advanced OS and network features and inefficient server demultiplexing techniques. If not corrected, these sources of overhead will force developers to avoid CORBA middleware and continue to use lower-level tools like sockets or TLI. Unfortunately, lower-level tools fail to provide other key benefits of CORBA middleware such as robustness, flexibility, and reusability. These middleware benefits are crucial to the success of complex performance-sensitive distributed, real-time applications.

Technology is changing so rapidly that if organisations don't make system scalable then they have to pay a heavy amount in future. Different organisations are adopting different techniques to make the system scalable. Here is an example of Australian bank [ZDNet Australia (2003a)]. This example shows how they are making scalable their distributed banking system. This Australian bank is using a Java-based distributed system for making electronic payments switch. Switch is cross link between card holder, issuing bank, the bank that required transaction, local finance companies and foreign payment networks. One survey also mentioned by ZDNet Australia (2003b) in this example that Australian electronic payments are growing between eight to thirteen percent per year. That means Australian banks and financial institutes need to double their processing service after every three year. This is main cause of scalability for distributed systems.

In Australia, most switches are operated by banks and other financial institutions. Coles Myer operates its own switch, as do telcos, utilities, and other organisations in various geographical markets. Although the Australian market is relatively mature, electronic payments are growing by between eight and 13 percent per year, says Rod Dew, Distra's sales and marketing manager. Globally, the growth is between 20 and 30 percent, which means banks and other financial institutions may need to double their processing capacity every three years. This makes the scalability of distributed systems very attractive.

Reference:

- ✓ Brenton, Chris and Hunt, Cameron. (2001) Active Defence: A Comprehensive Guide to Network Security. San Francisco: Sybex, 2001. pp 723
- ✓ Chandy, K. M., Misra, J., and Haas, L.M., "Distributed Deadlock Detection", ACMTODS, Vol.1, No.2, 1983, pp144-156
- ✓ Felix Weber (2001). Digital signature, still waiting for bomb. Available from http://www.messe.ch/ca/y/wp/lang/e/csok/1/ Accessed on March 19, 2003
- ✓ Frank Stajano (2002). Security for ubiquitous computing. Wiley serious. pp 190 – 195
- ✓ Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.
- ✓ Kirk Job-Sluder (2002). Cryptpgraphy: A guide to protecting your files. Available on http://www.indiana.edu/~istb/conferences/2002/IST_Conf_2002_sluder.pdf Accessed on March 20, 2003
- ✓ Philip Homburg, Leendert van Doorn, Maarten van Steen, Andrew S. Tanenbaum, and Wiebren de Jonge (1995). An Object Model for Flexible Distributed Systems. Available on http://www.cs.vu.nl/pub/papers/globe/asci.95.pdf. Accessed on March 21, 2003
- ✓ Techweb (1999). Security in Distributed Computing: Did You Lock the Door? Available from http://www.networkcomputing.com/netdesign/1007part1a.html Accessed on March 19, 2003
- ✓ Tetsuo Kanezuka and Makoto Takizawa (1998). QoS-based Flexibility in Distributed Systems. Available on http://citeseer.nj.nec.com/cache/papers/cs/14012/http:zSzzSzwww.takilab.k.dendai.ac.jpzSzprofzSzpaperszSzicoinzSz1998zSzkanezSzqfds.pdf/qos-based-flexibility-in.pdf. Accessed on March 21, 2003
- ✓ Aniruddha Gokhale (1999) Scalability of end systems and distributed systems. . Available on http://www.cs.wustl.edu/~gokhale/WWW/personal/VITA/vita/node23.html. Accessed on March 21, 2003
- ✓ ZANet Australia (2003). New and Technology. Switch on payments, makes attractive scalability for distributed systems. Available at http://www.zdnet.com.au/newstech/enterprise/story/0,2000025001,20269390-3,00.htm. Accessed on Apr 07, 2003.

Distributed Object Design Models

Distributed object design model is not only study of relationship of whole and parts of components but also its architecture and process that how these components are connected to each other. Once there was a time when organisations follow single monolithic systems that evolved into larger and larger systems, until their individual parts began to have a complete sub-system. With the passage of time these subsystems had grown their selves into multiple layers of systems. This growth effected like that in fact each system was an assembly of many systems, which work all together. Afterward these layers system had given birth to distributed system. In this chapter author described that what the architecture is and how communication occurs between computers in distributed environment. As there are different techniques adopted by different organisations depend on their business requirement. Although there will be one object model which will more effective and efficient than any one other. But we can't give it the name of ever lasting best model. So it depends on scenario where we are going to fir that model. Different model's evaluation and their effects on small businesses can be found later in this and in next chapter.

3.1 Tier model

Fundamental reason behind the computer network existence is tier model. Growth of this model started from two tier architecture, followed by three tiers and then into n-tier architecture. These tiers are in the form of layers. Simple client/server model is one of example of two tier application. Upper tier shows application/presentation layer or front end layer and lower tier called back end/bottom layer or database/general layer. The difference between two and three tiers is difference of middle ware. If there is more then one middle layers then we will called it n-tier layers model. Three or n-tiers architectures is best model for business applications. These middle layers not only reside business logic but also they use for communication between presentation and bottom layers.

Business logic is actual core of the application. It takes input from user, interprets it, gets desire information from bottom layer and presents the results on presentation layer. On the other hand bottom layer provides the basic services of printing, files, and databases to the application logic. Means back end layer helps also in processing and after processing request, this layer hands over the result to business logic layer. P. Jalote (1994a) described a distributed system can be consisted on two layers: physical and logical layers. In physical layer computers connected by a communication network using any network topology. On the other hand in logical layer system consist of different set of processes and communication channel.

Tier model define the application's layers architecture. As P. Jalote (1994b) defined two types of layers for distributed environment, there implementation can be analyse by looking at following network models.

3.2 Peer-to-peer networks

In a peer-to-peer network, two or more computers share the same resources. Mean they share the printers, internet connections, and files etc. One computer can be responsible for one task and while other computer can be for another task. For example if one computer is responsible for handling printing sharing jobs, while the other computer may be responsible for internet connections sharing, file storing and retrieving. These computers can be connected through LAN or WLAN. Two computers can be connected to each other with one cross cable. But more then two computers will involve in network. Then there will be need of wiring hub. This hub will help for communication between computers.

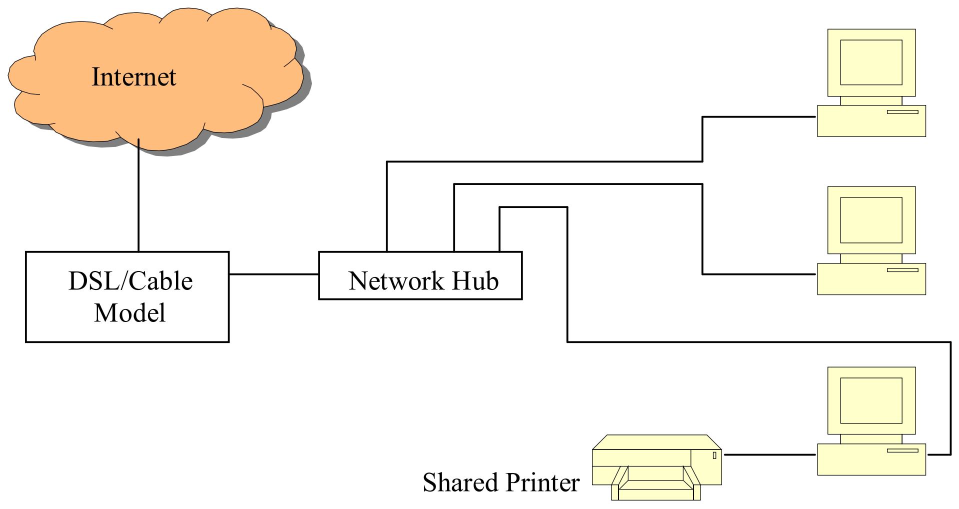


Figure 3.1. Peer-to-peer model

Above diagram is very common example of LAN based peer-to-peer model. In this model there is one printer and internet connection shared by whole network. Benefit of this model is saving the resources; more then one computer can share one resource. Internet is working through DSL or cable modem and accessible to network through hub. Peer-to-peer network can also implement on web also. In web base computing systems can be connect through internet. Thanos Zannetis and Panayiotis Karayiannis (2000) described web base computing like this.

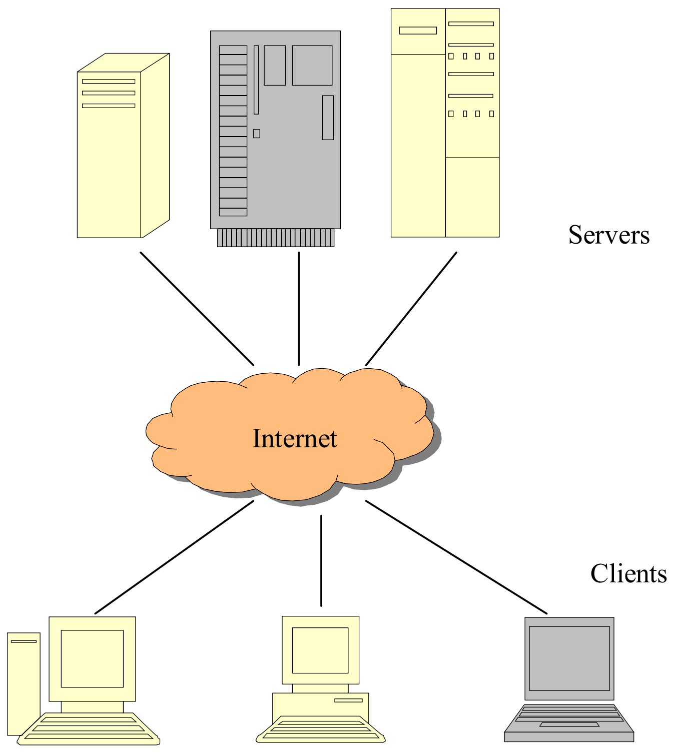


Figure 3.2. Peer-to-peer model (Web based computing)

The rapid growth and usage of internet is effecting on intensification of peer-to-peer applications. This technique of connecting computers in this way is not new. At start exchange system experts used it for telecommunication between dumb terminals through central exchange. Then later it adopted while modelling local area networks. The difference between web based and LAN based is that in web based peer-to-peer computer send and receive request to servers through internet connection. The data sending or receiving speed will bit slowly as compare to LAN. There need to consider more security issues while talking on web based computing as discussed in previous chapter.

Another example of peer-to-peer computing can be evaluated in figure configured below. In this type of computing each object has direct access to other. Every component can communicate to other by using any path. If there goes something wrong with one path, system can track alternative path.

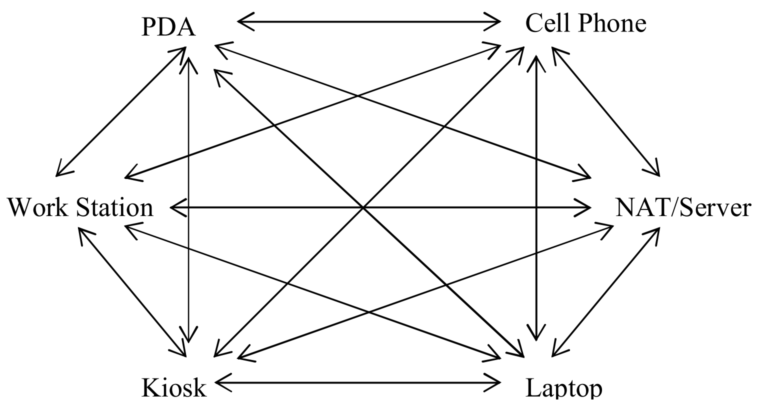


Figure 3.3. Peer-to-peer model

About above defined figures of peer-to-peer model divided in three categories by Robert Flenner, Michael Abbott, Toufic Boubez, Frank Cohen, Navaneeth Krishnan, Alan Moffet, Rajam Ramamurti, Bilal Siddiqui, and Frank Sommers (2003). These categories are simple peer, rendezvous peer and router peer. The difference between these three technologies is: simple peer have the least responsibility, they usually work possibly behind the general network and firewall. Rendezvous peer: as it is obvious from name is a dating service provider for other peers. This model also communicates over firewall. Router peers communicate over firewall and NAT routers. Router peer dig peer request across network and this information uses to replace the need of Dynamic Naming Service and IP addresses.

3.3 Client/Server networks

Client/server network model is most widely-used type architecture because of its efficiency. Although peer-to-peer network can perform multiple tasks like running applications, printing, faxing, documenting screens etc. but client/server architecture is more efficient to do these types of jobs. Basic client/server network consists on two computers connected to each other in a form like one is performing as server whiles other as client. Server is main computer who performs main jobs like printing, logging, record saving etc. On the other hand client is a computer who uses server resources and interacts with server for finding any resource. Then there can be such scenarios when client has stronger then server and vice versa called fat client thin server and fat server thin client respectively. Simple client/server architecture can understand from following illustration.

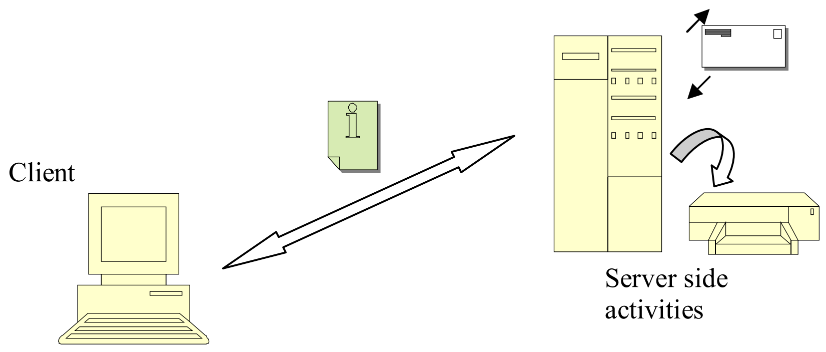


Figure 3.4. Strict Client/Server network

In network there can be more then one servers used for different type of jobs. For example a print server handles the print jobs of its network, file server use to store or retrieve the files, and an application server can be use for running or sending applications. So client/serer architecture have luxury of scalability for any type of organisation. As there are many computers and each computer has to perform a specific job so this type of network will much faster, due to resource sharing. When the same client/server model implements on mailing system, both the client and server side participates actively. Client software is responsible of creating, sending, reading, storing, forwarding mails and attachments. On the other side takes responsibility of security, connecting to mail server, placing messages in mail servers, notifying client to new email, and many other tasks. But actual message resides on server, client sends only request to server to perform any client-side jobs. Server receives client request, process it and send some results to client. As recent studies (Walter Glenn and James Chellis (2001)) describes that "*In this model, the software running on client machine is frequently referred to as the front-end program, while the software running on the server referred to as back-end program*".

The client/server computing has becoming a trademark for network distributing computing. Mostly small and large organisations are using this network technique. Difference between peer-to-peer and client/server architecture is that, in peer-to-peer every computer can act like a server and there can be more load on one computer at one time. On the other hand in client/server environment has one or more then one computers act like server for performing different jobs and due to this facility resource allocation load remain less on one computer. In usual client/server environment sometime one server called daemon. John Sullivan (2001) defined term daemon in this way that "*daemon is a program that runs continuously and exists for the purpose of handling periodic service requests that a computer system expects to receive*". Multiple clients share the resources of one common server. Web browser is good example of this resource sharing. Web browser is client program who send request to web server to another computer somewhere on the internet. So many web browsers can ping that web server at one time.

3.4 DCOM (Distributed Component Object Model) vs. CORBA (Common Object Request Broker Architecture)

Complications are increasing rapidly in the field of software development. Multiple platforms support requirements of client/server applications for Business requirements have forced developers to adopt new approaches. Now it needs to develop and adopt component-based and distributed technology. More recent studies (Raman Khanna 1998) show about distributed systems that '*The term distributed system has been used to describe loosely coupled multiprocessor system, clusters of computers, and co-operating multiple computer systems*". That's why all major users of information technology are moving from centralised to distributing computing. Distributed software system technology provides greater openness, functionality, scalability, heterogeneity, and resource sharing, fault-tolerance, and user productivity as a result. In the field of distributed technology, Microsoft introduced a distributed technology named DCOM. DCOM is an ex-tension of Microsoft's Component Object Model.

Whenever distribution is mentioned, CORBA will appear at some point. CORBA and DCOM are the two most widespread middleware technologies. CORBA is inherited from Object Management Group while DCOM comes from Microsoft COM frameworks. On the other side, OMG provides CORBA as specifications for ORBs. OMG provides the specifications regarding the mappings between COM and CORBA. Davide Marcato (2000) described that '*Microsoft expanded DCOM by including transaction services, easier programming and improved support for UNIX and other platforms*'.

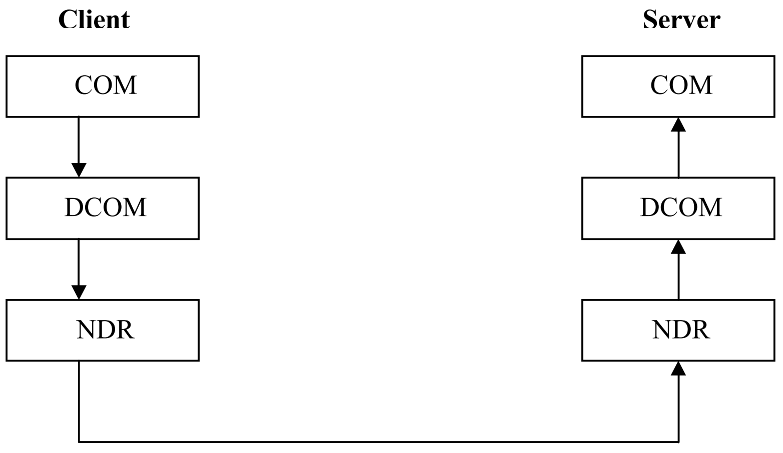


Figure 3.5. DCOM Client/Server Mechanism

In above figure client of DCOM has its own address space, either they are on a local machine or a remote machine. The client uses COM objects through interfaces. The client requests that an object is activated on the server and is passed back an interface on that object. On the other hand CORBA defines an environment where clients can make requests upon an object and object can send response back to client. It's a distributed system, so client and objects are not necessarily in the same address space, or on the same machine. In CORBA, ORB uses to communicate between client and server. Richard Grimes (2000) stated about CORBA architecture in this way that

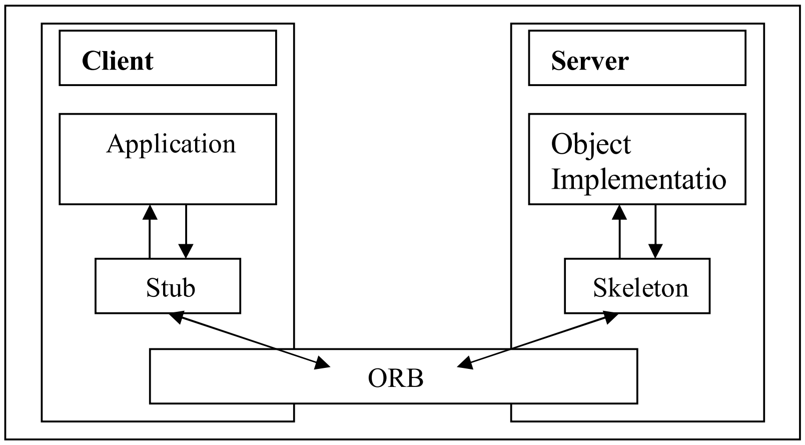


Figure 3.6. CORBA Architecture

It architecture addresses three main points

- ✓ Interface Mapping.
- ✓ Interface Composition Mapping.
- ✓ Identity Mapping.

DCOM also supports marshalling. In a popular study Herbert S. (1996) argued that "*Marshalling is the mechanism for transmitting data across process and network boundaries*". So that's mean the purpose of marshalling is to allow data pass from one process to another, even across the network and marshalling can also use to pass interface from one COM to another. In the framework of DCOM, marshalling involves placing the data into data packet that can be sent across to server computer. Ralph Droms (2002) discussed the subject of DHCP resources that "*The Dynamic Host Configuration Protocol (DHCP) is an Internet protocol for automating the configuration of computers that use TCP/IPs*". Today there are some configuration protocols like BOOTP and DHCP that can be used to dynamically configure TCP/IP. DCOM allows objects to be accessed and activated remotely. Before DCOM, COM had no specific security features. This meant that a client process, running under a particular security context and could access in a local server. But now DCOM extends the way that COM objects would not work as remote objects. Microsoft could not allow this, so mechanism has been implemented to allow pre-DCOM clients to call remote objects and clients to remote call pre-DCOM server.

DCOM has many key strength points. First of all its ease of use Creating COM objects for Windows using ATL, MFC, or VB is fairly easy using the Visual Studio tools. Enabling those objects to be called remotely through DCOM requires additional, difficult administration tasks. DCOM allows for the transmission of binary data and handles the ordering of the bytes for correctness using standard marshalling. Custom marshalling (discussed later) adds an additional mechanism by which the DCOM object and its proxy can exchange data in a custom, proprietary format. Custom marshalling is advantageous when the object designer wishes to expose an interface that would be easy to use but would suffer performance penalties when works over remote network. DCOM inherently employs object identity through the use of CLSIDs and server names using CoCreateInstanceEx. Object the client using AddRef also directly controls lifetime and Release calls on the object, and indirectly by the server with a pinging and timeout mechanism.

Interface methods for a DCOM object can return pointers to other interfaces. This pointer passing is fully supported by DCOM as long as the interface pointer being passed can also be remote. Emery (1997) describes that "*The logical boundary for component applications is no longer on a single machine, so the benefits of component development across a broader set of multi-user applications*". DCOM provides a transparent mechanism for extending component applications across a network, including the Internet. A critical aspect for a distributed component is that ability to grow with the amount of equipment, data, and functionality. DCOM features can enhance your application's scalability. Due to this applications can be referred to as "three-tier" or "n-tier".

On the other hand DCOM has some drawbacks. Although DCOM has been ported to other platforms and has been submitted to the W3C for standards approval, it has been used only for Windows applications. Cross-platform communication using DCOM is not a common sight. But somehow it is also available on MacOS and various UNIX platforms. Lastly from the advantages of DCOM and some understanding of distributed computing it can conclude on that scalability, openness, heterogeneity and resource sharing is requirement for any distributed system. Secondly it can't easy to say that which one is best either CORBA or DCOM, so it depend on scenario. No doubt CORBA has much power to support multiple platforms, which DCOM have not. But DCOM is more manageable then CORBA according to its technology architecture. DCOM is very simple to use and right choice for any client/server application on Microsoft platform.

3.5 REST (Representational State Transfer)

REST is distributed design model specially design to use web services for passing data between computer systems. Even it can be called founder of modern web architecture. Internet which is biggest example of distributed computing is using this model. Microsoft also introduced use of web services in its latest programming dot net environment. It is very useful for programmers, they can access any component deployed anywhere on web for their programs. REST use same technique as web browser follows to request any web server. It consists on XML, URLs and HTTP. REST is basically design for use under the architecture of World Wide Web and its two main contents HTTP and URLs. The purpose of designing this architecture is rather then developing a new form from scratch programmers can use it over the web by using its web services. A question arises here, before concluding on this argument that new technology is leading towards progress firstly there should be a brief evaluation of what web have already?

Microsoft also introduced SOAP (Simple Object Access Protocol) technology few months ago for such developers who find difficulty while deploying distributed computing with CORBA or DCOM. SOAP builds a layer over HTTP and XML called remote procedure call (RPC). It also uses the XML syntax to send request or response to other computer over internet. It is one of the negative points of SOAP. Clients have to wait a lot while XML parsing, no system level network programming available for this purpose. SOAP has also lack of interoperability, there is a lot of problems while message passing over SOAP including security issues. SOAP didn't use any firewall during communication, it uses commonly port 80 for communication. But other programs got problem if they want to talk over this port while communication of SOAP. Joel (2002) briefly concluded that "*The real problem with SOAP is that it's a completely inadequate remoting system. It doesn't have events (which makes it useless for large classes of applications)*". In short as mostly research studies conclude on SOAP that still its untested and new technology.

But REST model is overcoming the problems of SOAP. Most recent studies (Roy T. Fielding and R. N. Taylor, 2003a) tells that REST has one advantage over previous attempts at RPC architecture such as DCOM, CORBA, RMI is that REST has separate server implementation. Separate server implementation means it doesn't take client resources to transfer data of any size, it has ability to use gateway and proxy server and cache components for data transmission. It also increases the network performance and helps to execute client's request more quickly. But in future work it should be consider that data should be not leak while bringing from different locations. Switching of data packets should be so secure and reliable that there should be minimum chances of information leakage. It can be made so if we include a key with data and only the person who knows the key can access the desired data.

Reference:

- ✓ P. Jalote (1994). Fault Tolerance in Distributed Systems. Prentice-Hall, 1994.
- ✓ Thanos Zannetis and Panayiotis Karayiannis (2000). Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs.
- ✓ Robert Flenner, Michael Abbott, Toufic Boubez, Frank Cohen, Navaneeth Krishnan, Alan Moffet, Rajam Ramamurti, Bilal Siddiqui, and Frank Sommers (2003). Java P2P Unleashed with JXTA, Web Services, XML, Jini, JavaSpaces, and J2EE. Available on http://www.developer.com/java/ent/article.php/1496861. Accessed on April 01, 2003.
- ✓ Walter Glenn and James Chellis (2001). MCSE: Exchange 2000 Server Administration, study guide. Sybex. pp 13 – 15
- ✓ John Sullivan (2001). Client/server architecture and evaluation. Available at http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci211796,00.html. Accessed on Apr 07, 2003
- ✓ Richard Grimes (2000). DCOM Programming. A guide to creating practical applications with Microsoft Distributed Component Object Model. Published by Wrox Press. pp 25-37
- ✓ Dave Emery (1997). DCOM Advantages. Available on htttp://dominoext.wonderware.com/PublicWWR5/Studios.nsf/34ae3e7bc4f43c1e88256b04006b79e7/E57B629E774C911E88256B3000000CA5/$file/dcom.pdf Accessed on Apr 08, 2003
- ✓ Davide Marcato (2000). Distributing Computing with SOAP. Available on http://www.devx.com/upload/free/features/vcdj/2000/04apr00/dm0400/dm0400.asp. Accessed on Apr 07, 2003
- ✓ Mark Michaelis and Herbert Schildt (1996). COM + programming. Osborne/McGraw-Hill. pp 456-458
- ✓ Ralph Droms (2002). DHCP Resources. Available on. http://www.dhcp.org. Accessed on Apr 02, 2002
- ✓ Raman Khanna. (1998). Journal of Parallel and Distributed Computing. Volume 60. ISBN 0745-7515
- ✓ Joel (2002). SOAP vs. REST [dive into mark]. Available on http://diveintomark.org/archives/2002/04/25/soap_vs_rest.html. Accessed on Apr 09, 2003
- ✓ Roy T. Fielding (2003). Principled Design of the Modern Web Architecture Available at http://www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf. Accessed on Apr 09, 2003

Issues and Future Directions of Distributed Models in

Software industry is playing an important role in promotion of computers in our daily life. But such general usages of computer lead to a high competition between software development companies. Now the software development companies are stressing on importance of software quality and performance. For the promotion of their software they are adopting many software development distributed standards. These distributed standards guide them towards the development of effective, reliable, cheap, secure and efficient software's. In last chapter there was a brief description of different object models and how computer sends request and receives response to and from another computer. Here are understandings for an object model that can be most suitable for small business organisations. It is necessary for getting the maximum output from computer systems that they interconnected to each other with suitable model depending on organisation requirement. Some information can be found about software portion of this project. How small organisation developed software is influencing the distributed world? And which distributing technique it is following? Before further discussing object model more deeply there is concise discussion on what mean by small business organisations and concept of small in term of software engineering?

4.1. Software Engineering in the Small

There are two main issues that have to be addressed in IEEE software September/October 2000 journals to a framework for managing software product development in small companies. First they focused on software product development, and secondly on small companies. Focusing on software product development introduces the concepts of software engineering processes and practices. The concept of Software Engineering in the Small came when companies felt the problems of large companies. In a short manner we can say actually these problems are the inventors of small industries. As in IEEE software Laitinen M., Fayad M. and Ward R (2000), they highlighted the NATO Software Engineering Conference (Germany) and told about its discussion on identification of problems with large companies. After that in 1975, De Remer introduced the term of "programming in small".

4.1.1. SMALL

Term "Software Engineering in the Small" means the role of software engineering in small prospective. Here the word of "SMALL" is mentioning small company size, development range, pace and mode. Small term can vary by business-to-business and country resources to resources. For example if we compare building software to building a house. Building a house requires some specialists, but most of the people other then those specialists would just play their well-defined role without a lot of thinking on their own. This is because the techniques are well known, understood and not questioned. If a Software Engineer would build a wall, he or she would think about every little stone: Should it go to this or that position? How do I stick the stones together? At this stage there can also argue that these both domains are not comparable, but maybe they are in fact comparable but the software development domain is just not yet developed that so far we might not have well enough developed techniques. In that time when the first houses have been made out of stone, people had to think more about basic things, but now specific techniques have been established. Maybe we're trying to build pyramids with techniques suitable for building small mud houses. However, we are currently working on a project with more than 100 people and it seems like it is managed so that it will be successful. The effort is huge. So 200 to 300 people are working for development of a building, can be small. But on the other hand if same strength of staff working in a software house then we can say it's a huge.

4.1.2. Current practices

According to Laitinen M., Fayad M. and Ward R (2000) observation that "*according to United States bureau of the Census 1995 Country Business patterns, almost 90% of software and data-processing companies have fewer then 50 employees.*" That means mostly investors are favouring small organisations rather then large, because they understand the problems of large companies. I would like to discuss later in my chapter why companies should be small and their effect on software quality management. Here I would like to mention that what current practices are running in market for the improvement of quality in order to develop quality software.

Here I would like to give some references of TickIT guide. It is old standard, but still main ingredients like complexity, reusability, reliability, documentation and testability still use in major companies. This guide uses to develop quality software system under the construction and certification of ISO 9001 [The TickIT guide (1998)]. I think it is best way to satisfy to customer, because when he/she receives certificate of ISO 9001 with desired product. The certification is guarantee, that the purchased product is fulfilling software quality requirements.

Finally in current practices debate I would like to bend attention towards Capability maturity model (CMM). Many organizations have had difficulty obtaining their deserved level in software process maturity appraisals. Without an accurate identification of appraisal findings, the software improvement plan cannot be effective. So companies are using levels, because it provides a way to build organisational capability for performing software engineering. But there is a little problem according to the Hawaii's International Conference on System Sciences (2002) "*CMM was written to address the process for large, complex software efforts, something a small company with 3-10 developers probably would not undertake*". Means CMM can't be use in small companies, but I am not fully agreed with this statement, if we see wide scope of CMM. The CMM does not dictate which development process model one should use, it only tells us to use the one that suits you best and tailor it for different needs. So there actually is flexibility in the CMM, it only gives criteria for mature processes, specifically a process must be: defined, documented, trained, practised, supported, maintained, controlled, verified, validated, measured, and improvable. CMM also recommends that maturity and effectiveness of processes should be interpreted in the context of the business environment of the company and the specific circumstances of the projects.

4.1.3. Why Small?

Now I would like to pay attention towards that why software companies should be small? According to M. Voelter, J. Eckstein and N. Josuttis (2001) observation that most of the projects fail due to a lack of communication among project members, towards the customer, with management, etc. This raises the question whether communication can ever be established in a successful way for teams with 100 people or more. I think that communication can establish in small teams more in strong manner rather then compare to large teams. If there are less number of people, than there are much chances of understanding each other minds. They can easily communicate to each other and to customer as well. In small industries people can easily fulfil the requirement of system. They can easily satisfy the requirements of customer. They can make system requirement more flexible and extendible. Because in small environment people have to do more then one task by himself. Then he/she take cares every aspect of development. This step is very useful in software reusability. As in small teams there are not so complex hierarchy within employees of company, so they can work easily multiple heterogeneous execution environments. Due to less hierarchy people can work in very friendly and challenging environment.

We should not misunderstand the word of SMALL. It doesn't mean small can't do anything. Software engineering in the small is much powerful then large. For example, in software development, remove the system complexity and increasing its performance is very important task. If Software Company is large, they will spend more time on software management, making long lifecycles, hiring many developers etc. in these efforts they can forget their primary objective. On the other hand, no doubt small companies are small in every manner of development, but they have the ability to develop highly complex systems and their primary objective is performance of system, which is most important for them.

4.2. Replication Model

Coping database from one place to another is called database replication. Database management systems provide built in utility for this process. Data server sends a copy of database to another data server on timely bases. This distributed database model is very useful for those organisations, who wants that their data come in central server from different locations. In this model many servers can replicate their data from different locations on one server. In this scenario this one server can act like a central server. Local data servers can access the replication server over the internet or intranet. It is not necessary that there should be big network of client computers and server. Then this network accesses the central server. Replication is possible on small under one roof. There can be a chance when local data server accesses the replication server on local area network. In a small scenario replication can be occur in between two to three server. If there are two servers then which server will send the data called subscriber. On the other hand the server which receives the data from subscriber called publisher. If there is another server in between these two it called distributor. If there are only two computers performing replication then subscriber can be act as distributor. But in some case distributor can be a separate server. Actually distributor is a server who makes image of data which is ready for replication then it establishes connection between publisher and subscriber and transfer the image of data from subscriber to distributor. There are two techniques of data sending on replication server.

4.2.1. Push

In push replication technique, data server push data on replication server. Firstly it checks network connection to replicated server, if it finds successful then it makes the copy of data and start transferring to resulted server. In whole process because local server sends request, check connection, make copy in its own memory space to copy the data so it called push technique. There are few advantages of push techniques described by Open Universal Software (2000a). I am partially agreed with their statements. They described that server once send data to replicated server then there will not so much work load remain on local server. But replication occurs if on specific schedule, means after every five minutes. Then on rush hours local server resources will consume so much. Push technique is useful if server send the data at night to replication server. Then rest of day server will remain free, it only executes local request. This will help also in increasing the security because only local clients are accessing server so there are less number of chances that anyone enter in network.

4.2.2. Pull

Pull technique is opposite process of push. As it is obvious from name in this process publisher extracts data from subscriber. Main advantage of pull replication is all work load during process remain on publisher. Subscribers can do their task easily and there is not so much resource consumption for it. But on the other side publisher take copies of data from many data servers. Then in this scenario pull replication can be harmful. Because when there are hundreds of databases on publishers and publisher extracts data from hundreds of subscribers then there is need to consider some performance issues. In performance issues publisher should be fat server. Connection between publisher and subscriber should be secure. Maintaining data acuracy is key point in this scenario. Open Universal Software (2000b) also discussed a weak point of pull replication. In this technique administrator can't control all subscribers centrally. Administrator need to configure each subscriber properly.

4.2.3. Disadvantages of Replication Model

Replication is good technique for taking the backup of the data. But still there are some loop holes in this procedure. Every database management system has its own architecture and its own method of performing replication. While talking about Microsoft SQL Server, which is also a backend for the software part of this project. MS SQL server provides three types of replication named Snapshot, transactional and merge replication.

References

✓ TickIT (1998). The TickIT Guide, A guide to Software Quality Management System Construction and Certification to ISO 9001. 12 January 1998. Issue 4.0. DISC TickIT Office. pp A9 – A14

✓ M. Laitinen, M. Fayad and R. Ward (2000). Software Engineering in the Small, guest editor's introduction. IEEE Software, September/October 2000, pp 75 – 77.

✓ Hawaii's International Conference on System Sciences (2002). A Tentative Framework for Managing Software Product Development in Small Companies. IEEE Software 2002.

✓ M. Voelter, J. Eckstein and N. Josuttis (2001). Software Engineering in the large – does it work at all? Available from http://www.voelter.de/conferences/ot2002b.html Accessed on April 15, 2002

✓ Open Universal Software (2000). Data Distributor. Available from http://www.universal.com/DataReplication/DataDistributor/DataDistributor.htm#The%20Push. Accessed on April 17, 2003

Computational Grids

5.1 Abstract

"Grid" computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation. In this chapter, I have defined this new field. First, I reviewed the "Grid problem," which I defined as a flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources—what I refer to as *virtual organizations*. In such settings, I encountered unique authentication, authorization, resource access, resource discovery, and other challenges. It is this class of problem that is addressed by Grid technologies. Next, I presented an extensible and open *Grid architecture*, in which protocols, services, application programming interfaces, and software development kits are categorized according to their roles in enabling resource sharing. I describe requirements that I believe any such mechanisms must satisfy and discussed the importance of defining a compact set of *intergrid protocols* to enable interoperability among different Grid systems. Finally, I discussed how Grid technologies relate to other contemporary technologies, including enterprise integration, application service provider, storage service provider, and peer-to-peer computing. I maintain that Grid concepts and technologies complement and have much to contribute to these other approaches.

5.2 Introduction

The term "the Grid" was coined in the mid 1990s to denote a proposed distributed computing Infrastructure for advanced science and engineering. Considerable progress has since been made on the construction of such an infrastructure, but the term "Grid" has also been conflated, at least in popular perception, to embrace everything from advanced networking to artificial intelligence. One might wonder whether the term has any real substance and meaning. Is there really a distinct "Grid problem" and hence a need for new "Grid technologies"? If so, what is the nature of these technologies, and what is their domain of applicability? While numerous groups have interest in Grid concepts and share, to a significant extent, a common vision of Grid architecture, I do not see consensus on the answers to these questions.

My purpose in this chapter is to argue that the Grid concept is indeed motivated by a real and specific problem and that there is an emerging, well-defined Grid technology base that addresses significant aspects of this problem. In the process, I have developed a detailed architecture and roadmap for current and future Grid technologies. Furthermore, I have asserted that while Grid technologies are currently distinct from other major technology trends, such as Internet, enterprise, distributed, and peer-to-peer computing, these other trends can benefit significantly from growing into the problem space addressed by Grid technologies.

The real and specific problem that underlies the Grid concept is ***coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*.** The sharing that I am 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 I call a ***virtual organization*** (VO).

The following are examples of VOs: the application service providers, storage service providers, cycle providers, and consultants engaged by a car manufacturer to perform scenario evaluation during planning for a new factory; members of an industrial consortium bidding on a new aircraft; a crisis management team and the databases and simulation systems that they use to plan a response to an emergency situation; and members of a large, international, multiyear high energy physics collaboration. Each of these examples represents an approach to computing and problem solving based on collaboration in computation- and data-rich environments.

As these examples show, VOs vary tremendously in their purpose, scope, size, duration, structure, community, and sociology. Nevertheless, careful study of underlying technology requirements leads us to identify a broad set of common concerns and 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 multi-stakeholder 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; and for diverse usage modes, ranging from single user to multi-user and from performance sensitive to cost-sensitive and hence embracing issues of quality of service, scheduling, co-allocation, and accounting.

Current distributed computing technologies do not address the concerns and requirements just listed. For example, current Internet technologies address communication and information exchange among computers but do not provide integrated approaches to the coordinated use of resources at multiple sites for computation. Business-to-business exchanges focus on information sharing (often via centralized servers). So do virtual enterprise technologies, although here sharing may eventually extend to applications and physical devices. Enterprise distributed computing technologies such as CORBA and Enterprise Java enable resource sharing within a single organization. The Open Group's Distributed Computing Environment (DCE) supports secure resource sharing across sites, but most VOs would find it too burdensome and inflexible. Storage service providers (SSPs) and application service providers (ASPs) allow organizations to outsource storage and computing requirements to other parties, but only in constrained ways: for example, SSP resources are typically linked to a customer via a virtual private network (VPN). Emerging "Distributed computing" companies seek to harness idle computers on an international scale but, to date; support only highly centralized access to those resources. In summary, current technology either do not accommodate the range of resource types or do not provide the flexibility and control on sharing relationships needed to establish VOs.

It is here that Grid technologies enter the picture. Over the past few years, research and development efforts within the Grid community have produced protocols, services, and tools that address precisely the challenges that arise when I seek to build scalable VOs. 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.

Because of their focus on dynamic, cross-organizational sharing, Grid technologies complement rather than compete with existing distributed computing technologies. For example, enterprise distributed computing systems can use Grid technologies to achieve resource sharing across institutional boundaries; in the ASP/SSP space, Grid technologies can be used to establish dynamic markets for computing and storage resources, hence overcoming the limitations of current static configurations. I have discussed the relationship between Grids and these technologies in more detail below.

In the rest of this chapter, I have expanded upon each of these points in turn. My objectives are to (1) clarify the nature of VOs and Grid computing for those unfamiliar with the area; (2) contribute to the emergence of Grid computing as a discipline by establishing a standard vocabulary and defining an overall architectural framework; and (3) define clearly how Grid technologies relate to other technologies, explaining both why emerging technologies do not yet solve the Grid computing problem and how these technologies can benefit from Grid technologies.

It is my belief that VOs have the potential to change dramatically the way we use computers to solve problems, much as the web has changed how we exchange information. As the examples presented here illustrate, the need to engage in collaborative processes is fundamental to many diverse disciplines and activities: it is not limited to science, engineering and business activities. It is because of this broad applicability of VO concepts that Grid technology is important.

5.3 The Emergence of Virtual Organizations

Consider the following four scenarios:

1. A company needing to reach a decision on the placement of a new factory invokes a sophisticated financial forecasting model from an ASP, providing it with access to appropriate proprietary historical data from a corporate database on storage systems operated by an SSP. 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 a cycle provider for additional "oomph" during particularly demanding scenarios, requiring of course that cycles meet desired security and performance requirements.

2. 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.

3. 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.

4. 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.

These four 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. But they also have much in common. In each case, a number of 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") 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. An actual organization can participate in one or more VOs by sharing some or all of its resources.

5.4 Grid Architecture Descriptions

My goal in describing Grid architecture is not to provide a complete enumeration of all required protocols (and services, APIs, and SDKs) but rather to identify requirements for general classes of component. The result is an extensible, open architectural structure within which can be placed solutions to key VO requirements. The architecture and the subsequent discussion organize components into layers. Components within each layer share common characteristics but can build on capabilities and behaviours provided by any lower layer. In specifying the various layers of the Grid architecture, I have followed the principles of the "hourglass model". The narrow neck of the hourglass defines a small set of core abstractions and protocols (e.g., TCP and HTTP in the Internet), onto which many different high-level behaviours 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 my 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 behaviours at the *Collective* layer—so called because they involve the coordinated ("collective") use of multiple resources.

The layered Grid architecture and its relationship to the Internet protocol architecture. Because the Internet protocol architecture extends from network to application, there is a mapping from Grid layers into Internet layers.

5.4.1 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, catalogues, 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 I place few demands on Fabric elements, then deployment of Grid infrastructure is simplified.

For example, resource level support for advance reservations makes it possible for higher-level services to aggregate (co schedule) resources in interesting ways that would otherwise be impossible to achieve. However, as in practice few resources support advance reservation "out of the box," a requirement for advance reservation increases the cost of incorporating new resources into a Grid.

Issue / significance of building large, integrated systems, just-in-time by aggregation (co scheduling and co-management) is a significant new capability provided by these Grid services.

Experience suggests that at a minimum, resources should implement *enquiry* 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. The following brief and partial list provides a resource specific characterization of capabilities.

5.4.2 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. Enquiry 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.

5.4.3 Storage resources

Mechanisms are required for putting and getting files. Third-party and high-performance (e.g., striped) transfers are useful. So are mechanisms for reading and writing subsets of a file and/or executing remote data selection or reduction functions. Management mechanisms that allow control over the resources allocated to data transfers (space, disk bandwidth, network bandwidth, CPU) are useful, as are advance reservation mechanisms. Enquiry functions are needed for determining hardware and software characteristics as well as relevant load information such as available space and bandwidth utilization.

5.4.4 Network resources

Management mechanisms that provide control over the resources allocated to network transfers (e.g., prioritization, reservation) can be useful. Enquiry functions should be provided to determine network characteristics and load.

5.4.5 Code repositories

This specialized form of storage resource requires mechanisms for managing versioned source and object code: for example, a control system such as CVS.

5.4.6 Catalogues

This specialized form of storage resource requires mechanisms for implementing catalogue query and update operations: for example, a relational database.

5.4.7 Globus Toolkit

The Globus Toolkit has been designed to use (primarily) existing fabric components, including vendor-supplied protocols and interfaces. However, if a vendor does not provide the necessary Fabric-level behaviour, the Globus Toolkit includes the missing functionality. For example, enquiry software is provided for discovering structure and state information for various common resource types, such as computers (e.g., OS version, hardware configuration, load, scheduler queue status), storage systems (e.g., available space), and networks (e.g., current and predicted future load), and for packaging this information in a form that facilitates the implementation of higher-level protocols, specifically at the Resource layer. Resource management, on the other hand, is generally assumed to be the domain of local resource managers. One exception is the General-purpose Architecture for Reservation and Allocation (GARA), which provides 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) and Condor that support advance reservation capabilities.

5.5 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. While alternatives certainly exist, I have assumed here 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. 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.

With respect to security aspects of the Connectivity layer, I have observed that the complexity of the security problem makes it important that any solutions be based on existing standards whenever possible. As with communication, many of the security standards developed within the context of the Internet protocol suite are applicable.

Authentication solutions for VO environments should have the following characteristics:

5.5.1 Single sign on

Users must be able to "log on" (authenticate) just once and then have access to multiple Grid resources defined in the Fabric layer, without further user intervention.

•

5.5.2 Delegation

A user must be able to endow a program with the ability to run on that 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 conditionally delegate a subset of its rights to another program (sometimes referred to as restricted delegation).

5.5.3 Integration with various local security solutions

Each site or resource provider may employ any of a variety of local security solutions, including Kerberos and UNIX security. 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.

•

5.5.4 User-based trust relationships

In order 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, and 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.

5.5.5 Globus Toolkit

The Internet protocols listed above are used for communication. The public-key based Grid Security Infrastructure (GSI) protocols re used for authentication, communication protection, and authorization. GSI builds on and extends the Transport Layer Security (TLS) protocols to address most of the issues listed above: in particular, single sign on, delegation, integration with various local security solutions (including Kerberos), and user-based trust relationships. X.509-format identity certificates are used. Stakeholder control of authorization is supported via an authorization toolkit that allows resource owners to integrate local policies via a Generic Authorization and Access (GAA) control interface. Rich support for restricted delegation is not provided in the current toolkit release (v1.1.4) but has been demonstrated in prototypes.

5.6 Resource: Sharing Single Resources

The Resource layer builds on Connectivity layer communication and authentication protocols to define protocols (and APIs and SDKs) for the secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. Resource layer implementations of these protocols call 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:

•

5.6.1 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).

5.6.2 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. While many such protocols can be imagined, the Resource (and Connectivity) protocol layers form the neck of my 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 3.1 summarizes the major features required in Resource layer protocols. To this list I have added the need for “exactly once” semantics for many operations, with reliable error reporting indicating when operations fail. *Globus Toolkit*: A small and mostly standards-based set of protocols is adopted.

A Grid Resource Information Protocol (GRIP, currently based on the Lightweight Directory Access Protocol: LDAP) is used to define a standard resource information protocol and associated information model. An associated soft-state resource registration protocol, the Grid Resource Registration Protocol (GRRP), is used to register resources with Grid Index Information Servers, discussed in the next section.

. The HTTP-based Grid Resource Access and Management (GRAM) protocol is used for allocation of computational resources and for monitoring and control of computation on those resources.

. An extended version of the File Transfer Protocol, Grid FTP, is 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. FTP is adopted as a base data transfer protocol because of its support for third-party transfers and because it’s separate control and data channels facilitate the implementation of sophisticated servers.

. LDAP is also used as a catalogue access protocol.

The Globus Toolkit defines client-side C and Java APIs and SDKs for each of these protocols.

Server-side SDKs and servers are also provided for each protocol, to facilitate the integration of various resources (computational, storage, network) into the Grid. For example, the Grid Resource Information Service (GRIS) implements server-side LDAP functionality, with callouts allowing for publication of arbitrary resource information. An important server-side element of the overall Toolkit is the "gatekeeper," which provides what is in essence a GSI-authenticated "intend" that speaks the GRAM protocol and can be used to dispatch various local operations. The Generic Security Services (GSS) API is used to acquire, forward, and verify authentication credentials and to provide transport layer integrity and privacy within these SDKs and servers, enabling substitution of alternative security services at the Connectivity layer.

5.7 Collective: Coordinating Multiple Resources

While the Resource layer is focused on interactions with a single resource, the next layer in the architecture contains protocols and services (and APIs and SDKs) that are not associated with any one specific resource but rather are global in nature and capture interactions across collections of resources. For this reason, I have refered to the next layer of the architecture as the *Collective* layer. Because Collective components build on the narrow Resource and Connectivity layer "neck" in the protocol hourglass, they can implement a wide variety of sharing behaviours without placing new requirements on the resources being shared. For example:

5.7.1 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. Resource-level GRRP and GRIP protocols are used to construct directories.

5.7.2 Co-allocation, 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, Condor-G, Nimrod-G, and the DRM broker.

5.7.3 Monitoring and diagnostics services

Support the monitoring of VO resources for failure, adversarial attack ("intrusion detection"), overload, and so forth.

5.7.4 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.

5.7.5 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 and manager-worker frameworks.

5.7.6 Workload management systems and collaboration framework

Also known as problem solving environments ("PSEs")—provide for the description, use, and management of multi-step, asynchronous, multi-component workflows.

5.7.7 Software discovery services

Discover and select the best software implementation and execution platform based on the parameters of the problem being solved. Examples include NetSolve and Ninf.

5.7.8 Community authorization servers

Enforce community policies governing resource access, generating capabilities that community members can use to access community resources.
These servers provide a global policy enforcement service by building on resource information, and resource management protocols (in the Resource layer) and security protocols in the Connectivity layer.

Reference:

- ✓ *Realizing the Information Future: The Internet and Beyond*. National Academy Press, 1994. http://www.nap.edu/readingroom/books/rtif/.
- ✓ Abramson, D., Sosic, R., Giddy, J. and Hall, B. Nimrod: A Tool for Performing Parameterized Simulations Using Distributed Workstations. In *Proc. 4th IEEE Symp. On High Performance Distributed Computing*, 1995.
- ✓ Aiken, R., Carey, M., Carpenter, B., Foster, I., Lynch, C., Mambretti, J., Moore, R., Strasnner, J. and Teitelbaum, B. Network Policy and Services: A Report of a Workshop on Middleware, IETF, RFC 2768, 2000. http://www.ietf.org/rfc/rfc2768.txt.
- ✓ Allcock, B., Bester, J., Bresnahan, J., Chervenak, A.L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D. and Tuecke, S., Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. In *Mass Storage Conference*, 2001.
- ✓ Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S.,McInnes, L. and Parker, S. Toward a Common Component Architecture for High Performance Scientific Computing. In *Proc. 8th IEEE Symp. on High Performance Distributed Computing*, 1999.
- ✓ Arnold, K., O'Sullivan, B., Scheifler, R.W., Waldo, J. and Wollrath, A. *The Jini Specification*. Addison-Wesley, 1999. See also www.sun.com/jini.
- ✓ Baker, F. Requirements for IP Version 4 Routers, IETF, RFC 1812, 1995. http://www.ietf.org/rfc/rfc1812.txt.
- ✓ Barry, J., Aparicio,M., Durniak, T., Herman, P., Karuturi, J.,Woods, C., Gilman, C., Ramnath, R. and Lam, H., NIIIP-SMART: An Investigation of Distributed Object Approaches to Support MES Development and Deployment in a Virtual Enterprise. In *2nd Intl Enterprise Distributed Computing Workshop*, 1998, IEEE Press.
- ✓ Baru, C., Moore, R., Rajasekar, A. and Wan, M., The SDSC Storage Resource Broker. In *Proc. CASCON'98 Conference*, 1998.
- ✓ Beiriger, J., Johnson, W., Bivens, H., Humphreys, S. and Rhea, R., Constructing the ASCI Grid. In *Proc. 9th IEEE Symposium on High Performance Distributed Computing*, 2000, IEEE Press.

- ✓ Benger, W., Foster, I., Novotny, J., Seidel, E., Shalf, J., Smith, W. and Walker, P., Numerical Relativity in a Distributed Environment. In *Proc. 9th SIAM Conference on Parallel Processing for Scientific Computing*, 1999.
- ✓ Berman, F. High-Performance Schedulers. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 279-309.
- ✓ Berman, F., Wolski, R., Figueira, S., Schopf, J. and Shao, G. Application-Level Scheduling on Distributed Heterogeneous Networks. In *Proc. Supercomputing '96*, 1996.
- ✓ Beynon, M., Ferreira, R., Kurc, T., Sussman, A. and Saltz, J., DataCutter: Middleware for Filtering Very Large Scientific Datasets on Archival Storage Systems. In *Proc. 8th Goddard Conference on Mass Storage Systems and Technologies/17th IEEE Symposium on Mass Storage Systems*, 2000, 119-133.
- ✓ Bolcer, G.A. and Kaiser, G. SWAP: Leveraging theWeb To ManageWorkflow. *IEEE Internet Computing,*:85-88. 1999.
- ✓ Brunett, S., Czajkowski, K., Fitzgerald, S., Foster, I., Johnson, A., Kesselman, C., Leigh, J. and Tuecke, S., Application Experiences with the Globus Toolkit. In *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, 1998, IEEE Press, 81-89.
- ✓ Butler, R., Engert, D., Foster, I., Kesselman, C., Tuecke, S., Volmer, J. and Welch, V. Design and Deployment of a National-Scale Authentication Infrastructure. *IEEE Computer, 33*(12):60-66. 2000.
- ✓ Camarinha-Matos, L.M., Afsarmanesh, H., Garita, C. and Lima, C. Towards an Architecture for Virtual Enterprises. *J. Intelligent Manufacturing*.
- ✓ Casanova, H. and Dongarra, J. NetSolve: A Network Server for Solving Computational Science Problems. *International Journal of Supercomputer Applications and High Performance Computing, 11*(3):212-223. 1997.
- ✓ Casanova, H., Dongarra, J., Johnson, C. and Miller, M. Application-Specific Tools. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 159-180.
- ✓ Casanova, H., Obertelli, G., Berman, F. and Wolski, R., The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proc. SC'2000*, 2000.

✓ Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets. *J. Network and Computer Applications,* 2001.