users modify them? Is the access control mechanism itself protected or done by the general operating system functions?

The *reference monitor* addresses these questions, with the aim of formally ensuring protection. Originally proposed in (Lampson, 1974), many works have been based on it, broadening the scope of the original definition. In the rest of this part we will use the notion of reference monitor. To help us with our explanations, let us identify a few useful properties of a generic reference monitor (RM), exemplified in Figure 18.16:

Completeness - the RM is invoked for any access to any object;

Obligation - the RM refers to an access control rule set;

Self-protection - the RM is immune to intrusion.

The completeness property is secured if the RM stands between all subjects and all objects. The obligation property is compatible with mandatory access control policies. One may of course implement an RM that does not exhibit the obligation property, following a discretionary policy[5]. The self-protection property is secured if the RM resides on a trusted computer base (TCB). As we discussed in Section 18.3, in certain applications one may implement an RM over an imperfect TCB and in that sense partially fulfill the self-protection property.

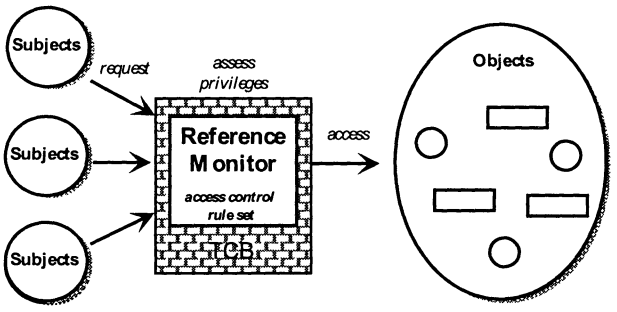


Figure 18.16. The Reference Monitor Model

18.8 ARCHITECTURAL PROTECTION: TOPOLOGY AND FIREWALLS

The way the architecture of the network is laid down helps implementing protection, and forms part of what we might call passive security measures. Devices like hubs, bridges and routers provide basic yet effective protection. Firewalls implement more sophisticated forms of architectural protection, both at physical and logical levels.

[5]The original work on the RM model did not follow a mandatory policy.

18.8.1 Topology

A naive form of internal network architecture is exemplified in Figure 18.17a. This flat approach has the consequence of exposing the whole infrastructure to an intruder that penetrates past the organization's entry router.

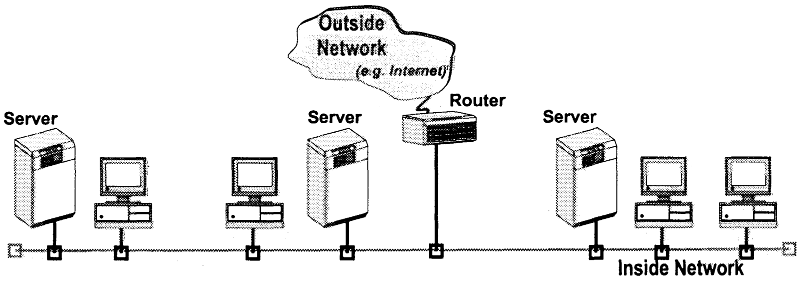


Figure 18.17. Network Architecture: (a) Flat

Figure 18.17b exemplifies the subnetting approach, that is, division of a network in two or more subnets, each with its own addressing mask, such that traffic is diverted right at the entry router to the appropriate subnet. This is valid both for traffic coming into the organization, and for traffic between different subnets of the organization. This division should be made according to an appropriate risk analysis: in the figure, the notion of a more exposed laboratory environment (Internal Network 2) versus the rest of the system (Internal Network 1) is patent. This is a primary form of error containment: intrusion in one subnet does not imply the immediate intrusion in the whole system. Subnetting also provides a convenient way for primary countermeasures: traffic may be easily blocked to/from specific networks, without affecting the operation of the whole organization.

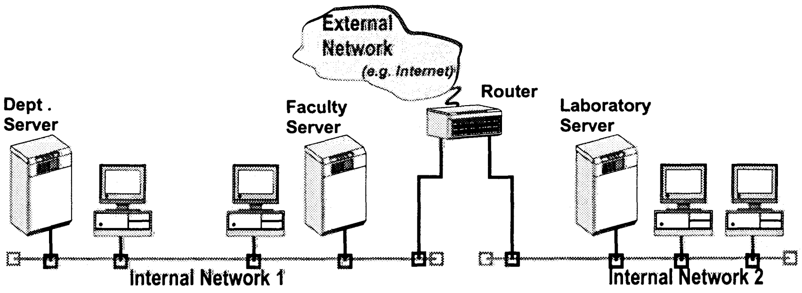


Figure 18.17 *(continued).* Network Architecture: (b) Subnets

The scenario depicted in Figure 18.17c exemplifies the utility of another device: the bridge. By placing the department services and the system administrator workstations in the Critical Network past the bridge, we neutralized sniffing attacks coming from Internal Network 1. Last but not least, structured cabling, such as hub-based Ethernet, eases reconfiguration and primary countermeasures. Switched Ethernet further renders sniffing ineffective, since it restricts the broadcasting ability of the medium.

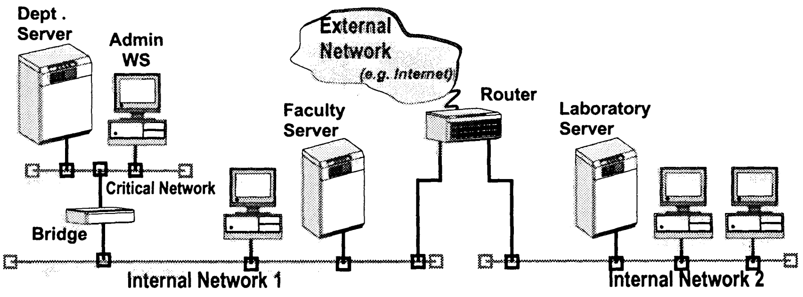


Figure 18.17 *(continued).* Network Architecture: (c) Bridges

18.8.2 Firewall Architecture

A basic firewall is like having a "doorman" at the (single) entrance of a facility: it allows or forbids information in and out, according to some criteria. More specifically, a firewall is a set of components placed between an external network and an internal network, with the following properties:

- all incoming and outgoing traffic must go through the firewall
- only authorized traffic must be able to get through
- the firewall hosts are trusted computing bases (TCBs)

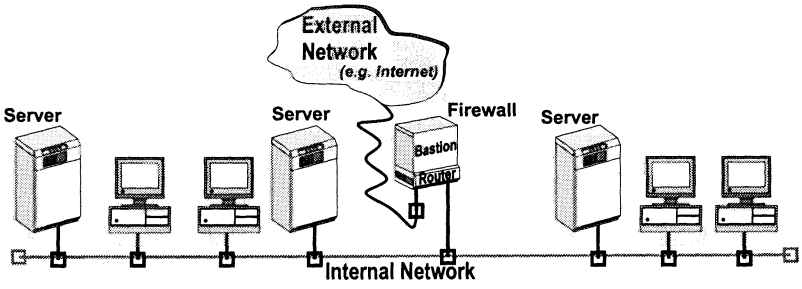


Figure 18.18. Single-level Firewall Architecture

The business of a firewall system designer is to approximate these properties, by putting the adequate firewall *functions* and *architecture* in place. Firewall architectures are built around routers, subnets and *bastions*, the trusted hosts that run firewall functions. These architectures take essentially two forms. The simplest, and most common, is a *single-level firewall* architecture, also known as *screened-host firewall*, as depicted in Figure 18.18. The firewall comprises a router and a bastion, a combination also known as a *dual-homed* host, that stands between the external network (e.g., Internet) and the organization's (internal) network, such that all traffic is inspected by the bastion. In a variation of this architecture, the bastion stands single-homed in the internal network, but all outgoing and incoming traffic goes through the bastion (e.g., Internet → router → bastion → internal host, and vice-versa). This architecture places all hosts in the internal network at the same level of threat. The problem is that any current organization is bound to have services that should operate under different exposure scenarios.

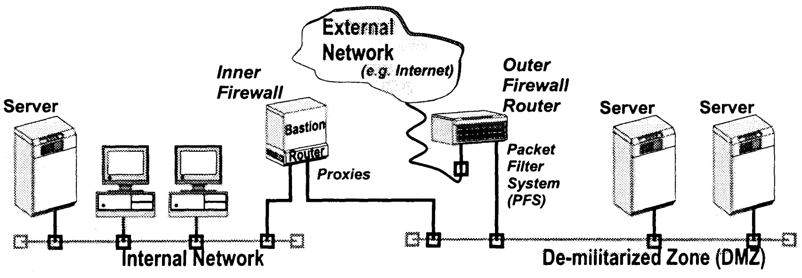


Figure 18.19. Two-level Firewall Architecture

Figure 18.19 presents the most used partial fix to that problem, a *two-level firewall* architecture, also known as *screened-subnet firewall.* The outermost firewall is normally also the outside router of the organization's network, and normally performs simple filtering functions. The inner firewall, a dual-homed host, performs more elaborate functions, such as representing internal protocols or applications. Between the inner and outer firewalls lies a subnet called the *de-militarized zone (DMZ)*. The DMZ is the place to locate hosts necessarily subjected to high levels of threat, such as anonymous public front-ends, e.g., to Web page, directory or commerce services. The reason for placing these servers in the DMZ instead of just letting them be freely accessible from the Internet is that they can still enjoy some protection from the firewall system. Besides, the outer firewall is also a useful device for countermeasures: it can be instructed in real-time by the inner firewall to disable certain flows considered suspicious, that attack either the inner network hosts, or the DMZ hosts. In a variation of this architecture, the bastion stands in the DMZ, with all traffic between external and internal networks going through it. Several of these bastions may exist in a DMZ, processing different flows and services.

Firewall functions are the logical complement to the physical separation achieved by the architecture. They are divided into two main groups, that we study next:

filters - the traffic flow *passes through* the firewall to end services in the internal network, its content being inspected by filters on a go-no-go basis

proxies - the traffic flow *ends or starts in* the firewall, being intercepted and processed by representatives of end services in the internal network

Table 18.2 illustrates the main differences between the two. Filters work by inspection of packet data that goes through the firewall, and as such they are essentially stateless. Working at communications packet level, the filtering process is necessarily semantically limited (addresses, ports, interfaces). Rules are content oriented, such as *"deny packets containing source address X"*. Proxies intercept calls to the genuine servers, blocking direct communication through the firewall, and act as representatives of those servers. As such, they are stateful, dealing as much with data as with state, since they must handle the progression of service requests such as *"connect to FTP server"*. More than packets, they reason in terms of protocols, users, and services, which provides room for richer and thus more accurate protection semantics. Rules are both content and action oriented, such as *"allow user X to access service Y"*. Since nothing comes for free, proxy systems are normally less efficient than packet filter systems.

Certain firewalls implement a variant of PFS, called *stateful packet filter (SPF)*. Filters, although acting right above layer 3, probe further into each packet looking for known high-level protocol headers. SPFs are a form of *dynamic packet filters*, in that they adapt the rules to a flow of packets. Firewall-1 (*see* Section 19.2) is one example. On the other hand, a variant of proxy exists called *adaptive proxy*, where the proxy is capable of interpreting varying degrees of threat for different instances of a same service. Gauntlet (*see* Section 19.2) is one example.

Table 18.2. Comparison between Firewall Functions

Filters	**Proxies**
inspection	interception
stateless	stateful
data based	data+state based
poor semantics	rich semantics
packet level	protocol/user/service level
content oriented rules	action oriented rules
faster	slower

18.8.3 Packet Filter Systems

The principle of packet filtering systems (PFS) is shown in Figure 18.20a. The filtering mechanism is defined by the following:

- there is a list of rules to allow or deny packets through the filter in either direction;

- each list element is a tuple ⟨*action,origin,destination,type,direction,subnet*⟩, where: *action* is either *allow* or *deny*; *origin* is a complete source *Id*, for example TCP/IP address/port, or subnet address/mask; *destination* is the same, for a destination *Id*; *type* when available is the type of packet, which often corresponds to a given protocol; *direction* is one of *inbound* or *outbound*; *interface* represents the subnet from/to which the packet comes/goes, through one of the interfaces to which the firewall is directly connected;

- the headers of all incoming and outgoing packets are scrutinized against the contents of the list.

- the rules are applied in order; a packet is accepted immediately an "allow" rule becomes true, or rejected if a "deny" rule becomes true;

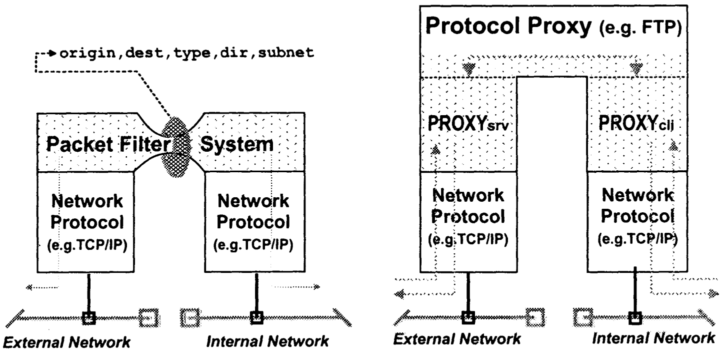

origin,dest,type,dir,subnet
Packet Filter System
Network Protocol (e.g.TCP/IP)
Network Protocol (e.g.TCP/IP)
External Network
Internal Network
Protocol Proxy (e.g. FTP)
PROXYsrv
PROXYcli
Network Protocol (e.g.TCP/IP)
Network Protocol (e.g.TCP/IP)
External Network
Internal Network

Figure 18.20. Types of Firewall functions: (a) Packet Filter, (b) Proxy

The PFS can be used for actions such as: blocking all packets coming from a suspicious host; disallowing telnet connections from the outside; enumerating the hosts allowed to access a given service. An example rule could be ⟨*deny from any to 194.117.21.00 typeTelnet inbound ie0*⟩, meaning that telnet packets coming from any machine in the subnet behind interface ie0, and addressed to any machine of protected subnet 194.117.21.00, are blocked. A prudent security policy can be implemented by using *allow* statements to specifically permit the desired traffic flows and denying everything by default as the last rule. Conversely, a permissive security policy will be implemented by using *deny* to block undesired flows, and allowing everything by default in the end. Firewall-1 (*see* Section 19.2) is a widely known example of PFS.

Network address translation (NAT) complements filtering to ensure logical separation. It consists of assigning invalid addresses to the internal network hosts, such that any traffic going in or out has to undergo a translation of the destination or source address, respectively, at the firewall router. This controls access of legitimate users to the external network, and hides the composition of the internal network to intruders.

18.8.4 Proxies

PFSs allow packet flows to internal hosts and are stateless: this means they cannot prevent direct probing and penetration attacks to internal hosts, as long as each individual packet looks innocent. If instead of inspecting passing traffic, it were intercepted and re-originated from the firewall, the chances of an intruder would be reduced. Figure 18.20b introduces such a firewall function, the proxy. Proxies reside in the firewall, and are representatives of application or protocol services (or daemons in UNIX language). A proxy stands for a genuine protocol server— such as HTTP, FTP, Telnet, or SMTP— which is normally in the internal network. The user wishing to access such a service is connected to the proxy server instead (left of the figure). The proxy client side performs the dialogue with the real server (right of the figure). Proxies relay all traffic back and forth between the user in the external network and the protocol server in the internal network. For that reason, proxies are also called *circuit gateways.* The proxy mechanism is defined by the following generic rules:

- there is a list of rules to allow, restrict or deny access to services;
- each element of the list is a service-specific tuple, that may comprise: *user Id* and *origin*; *service* and *server*; type of *access control* (e.g., ACM, ACL); type of *authentication* (e.g., password, signature); type of *restrictions* (e.g., from IDS); event *monitoring* and audit trail required; etc.;
- requests arriving at the proxy are tested against the rules, and serviced if "allowed" or blocked if "denied";
- state is maintained during service execution, to ensure it is carried on correctly.

Proxies significantly limit the freedom of action of an intruder in the internal network, and they are more precise than packet filters, since they act at a higher level of abstraction. A proxy rule may be something like ⟨*allow anyuser from domain cs.cornell.edu to ftp to FTP.di.fc.ul.pt requiring authentication X509_certificate requiring authorization local*⟩. It means that any user coming from cs.cornell.edu may attempt to log into the FTP server of di.fc.ul.pt, then pass an authentication process based on presenting a valid X.509 certificate, and then following the local ftp server ACL control.

18.8.5 Application Gateways

Application gateways are proxies working at a higher level of abstraction than circuit gateways. An application proxy server (a front-end of the application) is installed in the firewall, which for all purposes becomes the interface to the clients. The gateway, besides performing logging, validation and filtering functions, forwards the client's request to the real application server in the internal network, and receives the replies back. The overall picture is represented in Figure 18.20c.

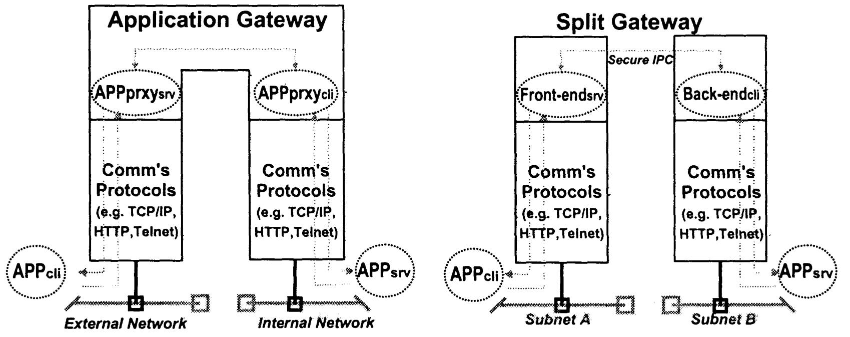

Application Gateway
APPprxysrv
APPprxycli
Comm's Protocols (e.g. TCP/IP, HTTP,Telnet)
Comm's Protocols (e.g. TCP/IP, HTTP,Telnet)
APPcli
APPsrv
External Network
Internal Network
Split Gateway
Secure IPC
Front-endsrv
Back-endcli
Comm's Protocols (e.g. TCP/IP, HTTP,Telnet)
Comm's Protocols (e.g. TCP/IP, HTTP,Telnet)
APPcli
APPsrv
Subnet A
Subnet B

Figure 18.20 *(continued)*
. Types of Firewall functions: (c) Application Gateway; (d) Split Gateway

If all access from the outside is done through protocol and application gateways, an internal network may be completely closed to the outside at the protocol level. An intruder's freedom of action is then reduced to trying to tamper with the gateway or with the firewall's O.S. In consequence, application gateways are the most powerful and precise protection devices in firewall architectures, because they exert control at the highest possible semantic level: that of the application itself. However, they are also the less versatile and most difficult to build. Whereas COTS packet filters and protocol proxies abound for the best known network environments and protocols, application gateways must normally be custom made, with very few exceptions. Gauntlet (*see* Section 19.2) is a widely known example of proxy and application gateway system.

Note that there is nothing that prevents combining filters and proxies in the same bastion, or distributing them by the components of a firewall. In two-level firewalls, a current configuration is as shown in Figure 18.19: the outer firewall performs packet filtering and routing; and the inner firewall hosts protocol and application proxies. As a concluding remark, keep in mind that the effectiveness of the firewall functions we have described is in the inverse proportion of their generality. It is up to the architect to select the configuration that best addresses the tradeoff between security and functionality, according to the security policy previously defined.

18.8.6 Split Gateway Architectures

Sophistication of Internet-based distributed computing models calls for modularity and splitting of functions, in support of multi-tiered client-server operation. Today's applications are becoming significantly more complex and performance-demanding than is achievable by CGI-based connection of Web servers to application servers. These applications are using more powerful, stateful middleware based on the connection of front-end Web servers to back-end application servers, through enabling technologies such as Object Request Brokers, Internet Inter-ORB protocols (IIOP), JDBC and ODBC database connectors, Web Request Brokers and Event Brokers. These constructs must be secure, and it makes sense that they operate across firewall architectures, for example, by running the back-end server in the internal network and then, both the front-end and the middleware in a firewall, or alternatively, the front-end server in a DMZ and the middleware in a firewall bastion.

To support the necessary modularity, address space separation, and location independence, we resort to what we may call a split gateway architecture, depicted in Figure 18.20d. It is an application gateway where the server-side and the client-side modules are stateful machines that operate independently, and communicate through a generic IPC. Both intra-host and inter-host IPC operation are supported transparently, such that the split gateway may either live on two separate hosts, as depicted in the figure, or in two disjoint realms of a protected O.S. in a single host. These hosts must be considered as bastions in what concerns their configuration and operation. IPC must be secure, in either configuration.

18.9 FORMAL SECURITY MODELS

In this section, we discuss formal ways of specifying and assessing security of computer systems. Namely, we discuss how, in a multi-level security mode, we specify which subject security classes can access which object access classes, and with what rights (e.g., *can a top-secret subject write to an unclassified object?*). Secondly, we describe standard security classification and evaluation criteria for computer systems (e.g., *how secure is computer system X?*).

18.9.1 Security Policy Models

It is obvious that for a verifiable goal to be attained, security policy rules must follow some formal specification. There exist several attempts to enforce mandatory access control policies in a formal way. The first such specification was the Bell-LaPadula model (Bell and LaPadula, 1973), which aimed at securing confidentiality. Other models followed, such as the Biba model, inspired by the Bell-LaPadula but addressing integrity instead (Biba, 1977), and the Clark-Wilson model, more adequate for enterprise systems (Clark and Wilson, 1987). Formal specification methods verifiable by model checking are discussed in (Ryan et al., 2000). There is not a generally accepted model addressing all needs, and this is a current research topic.

In generic terms, the relations between subjects and objects are established on the basis of their security classes and the need to know. We say s **dominates** o (or o is dominated by s) when the security class of s is at least as high as o's, and s needs to know o. We denote it as $s \geq o$.

Bell-LaPadula Model The Bell-LaPadula model, BeLa for short, describes the information flow in a system in terms of very primitive read/write operations. It aims at ensuring that the confidentiality property is respected in systems where data and computations of different security levels exist, and which can be accessed by subjects of different security classes. Consider subject s with security class $C(s)$, and object o, with security class $C(o)$. The access rights granted to s on o can be *read* or *write*. The properties of the BeLa model are then:

Simple Security Property - A subject s has *read* access to object o only if $C(s) \geq C(o)$

$*$-Property - A subject s has *write* access to object o only if $C(s) \leq C(o)$

Observe that a subject cannot read data from a level higher than its security class, that is, *read-up* is not possible. More counter-intuitively, the answer to the question in the beginning of the section— *can a top-secret subject write to an unclassified object?*— is "no!": a subject cannot write data to a level lower than its security class, that is, *write-down* is prevented. Also, if it writes data, it may not be able read it back. This causes a certain difficulty in retrieving, using and updating information for a repository. In fact, practical programming with this model entails significant complexity. However, note that by preventing read-up, confidentiality is protected in normal conditions. By making write-down impossible, information leakage attacks, directly or by means of Trojan horses, are blocked.

In the Biba model, which is the converse of the BeLa model, the security class interpretation concerns the sensitivity with regard to integrity. The model properties, conversely to the BeLa model, forbid *read-down* and *write-up*, as a way of preserving integrity of information. While no write-up is intuitive, note that the reason why read-down is prevented is to avoid corruption of the system with untrusted (low-integrity level) information.

18.9.2 Trusted Computer System Evaluation Criteria

Evaluating and grading the security of computer systems through objective evaluation criteria is important, for it allows us to compare systems of different models and makes. The Trusted Computer System Evaluation Criteria (**TCSEC**), or Orange Book (TCSEC, 1985) originating from the U.S.A., and the Information Technology Security Evaluation Criteria (**ITSEC**) (ITSEC, 1991), from Europe, were initial efforts in that direction. More recently, the bodies involved in both converged in a global standard, called Common Criteria for Information Technology Security Evaluation (**CC**) (CC-ITSE, 1998), bound to become an International Standard (ISO 15408) at the time of this writing.

The CC standard structures both the functionality requirements and the assurance requirements of a system i.e., what the product should do, and what trust can be placed in what it does. These requirements are expressed in terms of classes.

The standard is highly modular, making possible a huge number of combinations of classes of functionality and assurance requirements. However, it is expected that "typical" combinations emerge from the industry. The CC, like the preceding standards, provide several levels of trust. There are 7 Evaluation Assurance Levels (EAL) that measure the user trust on a system (EAL7 is

Table 18.3. Common Criteria (CC)- Security Levels

Levels	*Description*	*TCSEC equiv. Classes*
—		D
EAL1	functionally tested	-
EAL2	structurally tested	C1- Discretionary Security
EAL3	methodically tested and checked	C2- Controlled Access
EAL4	methodically designed, tested and reviewed	B1- Labeled Security
EAL5	semi-formally designed and tested	B2- Structured
EAL6	semi-formally designed, verified and tested	B3- Security Domains
EAL7	formally designed, verified and tested	A1- Verified Design

highest). Evaluation Assurance Levels are represented in Table 18.3, which also provides a mapping to the well-known TCSEC security classes. EAL1 applies when minimal protection, namely of personal information, is desired, but security is not a main concern. EAL2 and EAL3 are expected to have been tested against the functional criteria. In EAL4, it is expected that specific crucial subsets of the design have been methodically designed having security in mind, and that the whole has been thoroughly tested, and reviewed independently. Level EAL1-EAL4 assurance can generally be retrofitted into existing products and sub-systems (such as O.S.s). Levels above EAL4 require adequate design from the start. They provide maximum assurance, by application of specialized security engineering techniques. They also become more complex and expensive to implement. EAL5 would represent the top assurance still within the commercial systems area. EAL6 and EAL7 would apply to classified systems and military.

18.10 SECURE COMMUNICATION AND DISTRIBUTED PROCESSING

Secure channels and secure envelopes are basic paradigms of secure communication and the support for distributed processing models with security, such as remote sessions, RPC, and electronic mail. We study the above-mentioned models and mechanisms in this section.

18.10.1 Establishing Secure Channels

Secure channels, that we studied in Section 17.11, are one of the basic support primitives for distributed processing. They underlie file transfers, remote sessions, remote procedure calls, HTTP interactions, and so forth. They may be implemented in several ways. In what follows, we will make a general analysis of how to achieve each of the secure communication properties: authenticity, confidentiality, integrity. For simplicity, and without loss of generality, we consider the case of point-to-point communication.

Authenticity In order to achieve authenticity in a secure channel, the principals should authenticate themselves, in one of the styles shown in Figure 17.17. Mutual authentication desirably guarantees that both principals know whom they are talking to, mandatory if the channel is bi-directional. Several protocols are discussed in Section 18.5. If only symmetric cryptography is being used, shared-secret authentication protocols are the ones to be used. If asymmetric cryptography is available, then one can take advantage from the power of signature-based authentication. Since the channel is being established for exchanging a possibly large number of messages, with desirably low latency, it is convenient to avoid having to do authentication on every future message exchanged. Some of the short-term key exchange protocols studied in Section 18.6 are embedded with authentication, finishing by leaving the principals with a *session key* K_{ss}, known both of them and no one else. By majority of reason, any future message exchanged that has a suitable cryptographic function of that key enjoys the authenticity property if: messages are always encrypted with K_{ss}; or messages are cryptographically checksummed with a function depending on K_{ss} (*see* MACs in Section 17.5, and Figure 17.8). Note that the validity of these two assumptions relies on subtle aspects: (a) that the recipient knows something about the partial content (e.g., headers) or about the structure of messages; (b) that forging or modifying the encrypted product, may not possibly yield something intelligible in terms of (a), after decryption; (c) as (b), for the cryptographic checksum and its verification.

Confidentiality When confidentiality is desired, the communication must be encrypted. A channel using exclusively asymmetric cryptography for both authentication and encryption would be extremely simple and secure. However, it is not recommended for immediate communication, since it is very slow. Practical secure channels resort to two alternatives: purely symmetric cryptography; hybrid cryptography. In a symmetric system, after shared secret authentication and key exchange have been performed, as we just saw, the channel has a session key. With hybrid cryptography, principals start with using their public/private key pairs to exchange, encrypted and/or signed, a session key as well (*see* the protocol described in Section 18.4.5). After that, in both approaches, they use that key for symmetric encryption/decryption.

Integrity In order to achieve integrity, messages must be cryptographically protected in a way that any modification whatsoever (accidental or intentional) is detected. This can be achieved in one of two ways: by a cryptographic checksum, such as suggested for authenticity (in this role, it is often called a message integrity check, or MIC); or by a digital signature. Note that the proviso that we made under *Authenticity* above, on the recipient having to know what it expects after decryption or verification, still applies here. Integrity can also be protected by encryption, but the method is not general, since it requires a modified message to decrypt to garbage, and this is not always achievable.

18.10.2 Secure Tunnels

The simplest form of building a secure channel through a network such as the Internet is shown in Figure 18.21a. Encrypted and/or integrity protected payload data blocks are encapsulated in protocol packets, i.e., IP datagrams. This does not always work. Take networks A and C, interconnected by network B, and a problem: we want a packet to go from A to C, but for some reason it cannot circulate through B (e.g., B does not understand the protocol). The first approach will not work, so we have to use another form of secure channel.

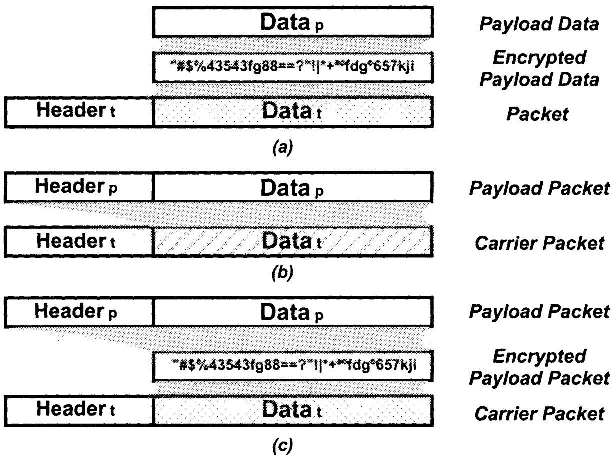


Figure 18.21. Tunneling: (a) Secure packet; (b) IP-over-IP; (c) Secure Tunnel

Let us start by understanding what a **tunnel** is: the encapsulation of a whole *payload packet* that circulates in network A, in a *carrier packet*, that circulates through network B, until network C, where the carrier packet is de-capsulated and the payload packet circulates again until the final destination. A classical tunnel is IP-over-IP, whose packet structure is depicted in Figure 18.21b. It consists of encapsulating a full IP datagram as if it were an upper layer service data unit, in another IP datagram. A **secure tunnel** is then a tunnel that guarantees the properties of a secure channel to the data carried inside it. Figure 18.21c suggests how it can work: the whole payload packet is treated as a block of data, and is encrypted and/or integrity protected. The result is then treated as data and encapsulated in the carrier packet. In the final destination, the operations are repeated in reverse order: the outer packet is de-capsulated, the data decrypted/verified, and the inner packet launched on the network for its final destination. IPsec (*see* Section 19.3) is the forthcoming standard for secure channels and tunnels on the Internet.

18.10.3 Distributed Authentication and Authorization

Among the several functions that assist secure computing, two are fundamental: authentication and authorization. We studied them independently, and we saw that one can be done without the other: secure communication requires authentication but not authorization; access to information requires authorization, but does often without authentication.

For example, message authentication can be achieved through MACs, message authentication codes, which are a *cryptographic checksum* technique, or through a *digital signature.* The disadvantage of MAC w.r.t. signatures is that it is a shared secret technique, and thus a participant cannot hand the message over, that is, persuade third parties who do not share the secret, of its authenticity.

Access to services and information often resorts to the assistance of a Security Server (SS), which designates a trusted third party that performs authentication and authorization. Let us see how these two functions work together in supporting client-server operation. In what follows, we use AT for the authentication service, and AC for the authorization or access control service, although they are often co-located in the same server:

1. *Client C and the AT run a protocol in order to authenticate C to the AT, in the course of which C receives an authentication certificate, Acert, for further use in the system*
2. *C wishing to use resource or service S, runs a protocol with AC, if necessary using Acert, requesting authorization to access S, in the course of which C receives a privilege certificate, Pcert, for S*
3. *Client C, using Acert and Pcert, presents itself to the server hosting S, and they run a protocol aimed at:*
    - *authenticating C to the server, based on Acert;*
    - *validating C's privileges to access S, based on Pcert and the access control list maintained by the server*
4. *If C is cleared, access is performed to S*

The content of *Acert* and the protocol steps run between C and SS, and C and S, depend on the type of protocol being used, either an arbitrated (KDC based) or a certified (CA-based) trusted-third-party protocol (*see* Section 18.4). The content of *Pcert* depends on the particular way access control is performed, i.e., whether it is capacity or ACL based, or both (ACM). Authentication may be mutual, generalizing the use of authentication certificates, that is, S may have its own *Acert* that authenticates it to C. Kerberos (*see* Section 19.4) is an example of a KDC-based security service.

18.10.4 Secure Remote Session

Remote sessions (*see* Chapter 1.3) were among the first paradigms of distribution ever used. Primitives `rlogin`, `telnet`, `rsh`, or `ftp`, are distinguished examples of long-lived distribution paradigms. Let us follow the steps of establishing a remote session, seen from a connecting site:

1. *bind to remote host socket*
2. *establish low level network connection between hosts*
3. *communication starts*
4. *perform cleartext remote login (e.g. telnet) authentication— authentication follows traditional mechanisms (password, address-based)*
5. *start session, also in cleartext*

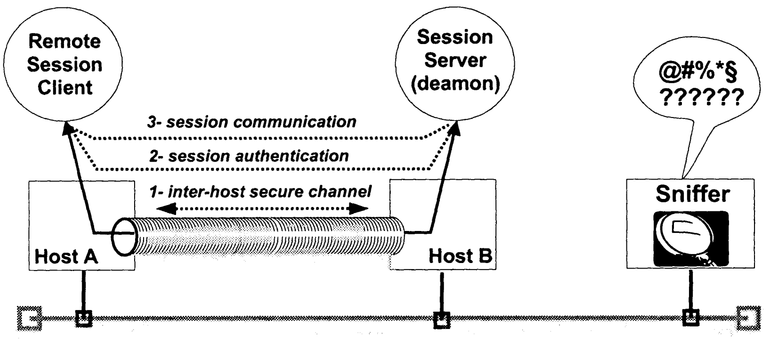


Figure 18.22. Secure Remote Session

However, remote session protocols were not designed to have security in mind, and became one of the most exploited vulnerabilities in distributed systems. Eavesdropping attacks easily yield not only conversation contents, but also, and more importantly, login/password pairs. A secure session should not be vulnerable to those attacks. To understand the principle of *secure* remote session, remember the desirable properties of the underlying secure channel: authenticity, integrity, confidentiality. The steps of establishing the secure session are the following (compare them with those of the cleartext session):

1. *bind and authenticate to remote socket*
2. *establish low level secure channel between hosts*
3. *encrypted communication starts*
4. *perform remote login (e.g. telnet) authentication*
5. *authentication can either follow traditional mechanisms (password, address-based) protected by secure channel, or be cryptographic (e.g., public key)*
6. *start session, either in cleartext or encrypted*

Observe Figure 18.22: notice that the first step is to establish a tamperproof channel between the hosts wishing to communicate, so that external attacks are not possible. However, at this point, the connecting host might be an attacker, or the connected host might be a spoofer, and either would act inside the secure channel. Then, the second step is to authenticate whoever is using the secure channel in the other extremity. For a moderate level of security, the authentication mechanism may be a classical login/password pair, which now goes encrypted on the network. However, for demanding levels of security, it may alternatively be any of the strong, cryptographic authentication mechanisms that we studied in Section 18.5, such as public key signature. Also depending on the security versus performance tradeoff, the third and final step, session communication on the channel, may subsequently be encrypted or not. SSH and SSL (*see* Section 19.1) are examples of secure remote session protocols.

18.10.5 Secure Client-Server with RPC

The security concerns address not only remote sessions, but client-server operations in general, such as RPCs. Many current services and applications are based on RPC: some of which carrying sensitive information, such as distributed file systems like NFS; others perform sensitive operations, such as transactional managers. This concern raises the need for secure RPC: a remote procedure call facility with strong authentication, encryption and protection. In order to prepare for a secure RPC, the client must execute the following steps:

1. *client binds to the desired service/server as usual*
2. *in that process, the client makes a call to the RPC runtime instructing it that this is a secure RPC and specifying the desired security options:*
    - *type of authentication;*
    - *level of security;*

On the server side:

1. *server exports name and registers security capabilities with RPC runtime*
2. *server initiates activity, normally with a login to the security service*
3. *server checks the security attributes of each call, and:*
    - *authenticates according to the chosen type;*
    - *performs access control based on client's authorization for the invoked service (e.g. ACL based)*
4. *if all is OK, the service is executed with the required level of security*

As an example, Table 18.4 lists the typical options of a secure RPC. Type of authentication specifies what kind of authentication model is followed. Level of security involves combinations of integrity and confidentiality assurance. The server acts as a reference monitor for the secure RPCs performed by clients, authenticating them and then checking their authorization for the invoked operation. SUN Open Network Computing (ONC) RPC and DCE RPC are examples of RPC packages with security facilities. In conclusion, note that using secure RPC instead of normal RPC does not involve much complexity, besides one or two additional calls to the runtime environment.

Table 18.4. RPC Security and Authentication Options

Parameter	*Options*
Authentication	none name-based (UNIX-like) shared-secret signature
Security	none integrity only integrity and confidentiality

18.10.6 Secure Envelopes and E-Mail

Secure envelopes complement secure channels as the basic communication support primitives for distributed processing. They are relevant in electronic e-mail, messaging systems in general, and transactional systems. Secure envelopes should resort to per-message security. There is no point in establishing a connection since message sending is sporadic. Again, let us see how each of the secure communication properties that we studied in Section 17.11 is achieved. We continue to consider the case of point-to-point communication, and consider the situation where there is no shared secret between principals, but public/private keys are in place, and known to the principals as appropriate.

Authenticity In order to achieve authenticity of a message in a secure envelope, the message should be signed with the sender's private key. The recipient can authenticate the sender by verifying it's signature. It is obviously wise to sign a digest of the message, as depicted in (with E_K as an asymmetric cipher, and K as the private key of the sender). See also Figure 17.10 in Section 17.6 for a description of such a protocol.

Confidentiality When confidentiality is desired, the message must be encrypted. Although the envelope is used for deferred communication, there is no point in being inefficient. In consequence, we will only use asymmetric encryption of the message for special cases of very small (control?) messages. Otherwise, hybrid cryptography with symmetric encryption of the message content seems more appropriate. The mechanism for generating a hybrid cryptographic envelope depicted in Figure 18.6 is perfectly appropriate.

Integrity After the steps to achieve either authenticity or confidentiality are performed, we have in place mechanisms to secure integrity. A digital signature as performed for authenticity guarantees message integrity. Encryption as performed for confidentiality also guarantees message integrity. Keep in mind the general remarks concerning integrity made throughout this section.

Secure E-Mail makes extensive use of the secure envelope concept, and in consequence secures the desirable properties of: *authenticity* of the sender of a message; *confidentiality* or privacy as used in this context, ensuring that only the recipient will read the message; *integrity* ensuring that the message is received as sent. Besides these obvious properties, electronic mail must have other security properties in emulation or improving those of paper mail:

- **non-repudiation of sending** - the recipient can prove that a message came from a given sender, who cannot deny
- **non-repudiation of delivery** - the sender can prove that a message was delivered to a given recipient, who cannot deny
- **anonymity** - the ability to deliver a message without revealing the identity of the sender
- **timestamping** - delivered messages can be totally ordered, even if a posteriori

PGP and PEM (*see* Section 19.1) implement the secure envelope concept and are used for secure e-mail.

18.11 ELECTRONIC TRANSACTION MODELS

What is an electronic transaction? It is a transaction involving assets, financial and other, made over computer and network systems. It assumes virtual payment instruments, which emulate conventional transaction protocols by informatic means. Electronic transactions (ET) are distributed in their nature, and assume several facets, which derive from the type of interaction of the players, the values involved, and the timing of payment that is, whether:

- ET values involved are average to low, typical of personal retail transactions, or are high, typical of wholesale inter-bank transactions;
- ETs take place on proprietary, or on open networks (e.g. Internet)
- buyer and merchant are introduced by a mediator, or just contact spontaneously;
- ETs need to contact the supporting infrastructure (e.g. PKI) on-line, or can perform off-line;

We are interested in personal retail transactions that take place on open networks, and in their several facets.

18.11.1 A Generic Model of Electronic Transaction

The generic model of ET is depicted in Figure 18.23. The main players are described in Table 18.5. The trusted third party materialized by the hierarchy of e-comm-specific certification authorities of the ET PKI is important to build trust between principals. When it is absent, such as in systems using only symmetric cryptography, the versatility of the system is limited, and the mediating role of the acquirer is bound to be more active in each transaction. When it exists, and whether it is concerned with credit card business, or digital cash or cheques, there is bound to be a Root CA which should be as independent as

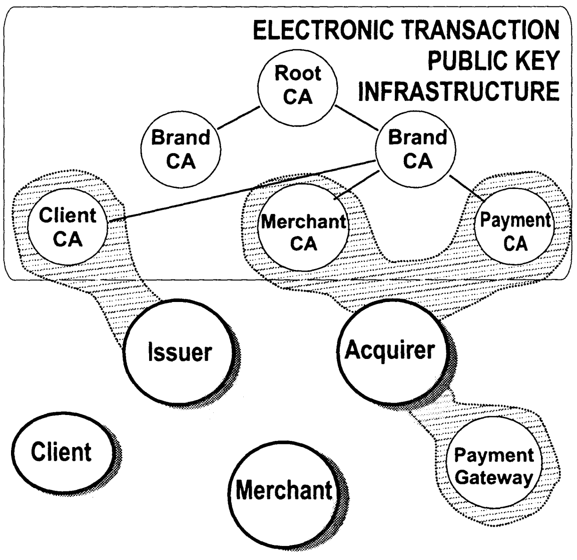


Figure 18.23. Generic Electronic Transaction Model

possible, so as to be above the Brand CAs (e.g. Mastercard). Brand operators break deals with Acquirers and Issuers, who set up their CAs under the Brand hierarchy. Real Acquirer and Issuer institutions may have deals with different Brands (e.g. Visa and Mastercard), but must maintain separate virtual trust chains. Acquirers also set up a payment gateway, the technical interface to the independent banking network, through which payments flow.

Table 18.5. Electronic Transaction Participants

Issuer	Normally a banking institution that issues the payment instruments: credit, debit or purse cards, digital cash, etc.
Client	The buyer in the transaction, he is normally a card holder and has an Id certificate, and is the transaction initiator
Merchant	The supplier of the good, he has an Id certificate
Acquirer	The mediator in the process, normally a financial institution that serves as a broker between the other players
ET PKI	The public key infrastructure, or a subset of it, concerned with facilitating electronic transactions, it issues all certificates

18.11.2 Classes of Electronic Transactions

According to our initial remarks, we may informally divide transactions in three classes, described in Table 18.6.

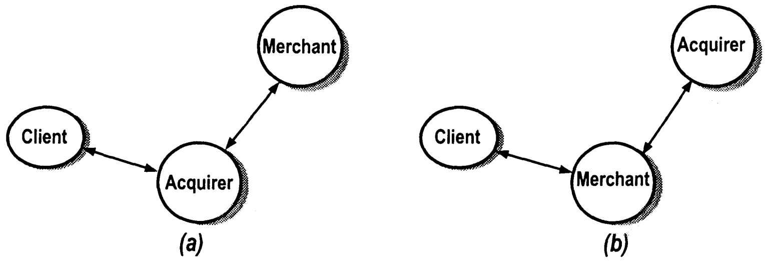


Figure 18.24. Electronic Transaction Classes: Mediated vs. Spontaneous

The mediated versus spontaneous classes are depicted in Figure 18.24. Observe that in the mediated class (Figure 18.24a), the mediator is in the way of the transaction, which must always be performed on-line. This is typical of earlier generation systems, where the rudimentary cryptography made it necessary that security and authenticity were ensured by the physical architecture, such as using proprietary networks and dedicated terminal devices. ATM networks are an example, where the transaction is "buying" money or paying for goods with the debit card. In these systems, the mediator soon becomes a bottleneck. The spontaneous class (Figure 18.24b) makes it possible for the client to produce credentials that authenticate it to the merchant, allowing the transaction to proceed as far as possible.

Table 18.6. Classes of Electronic Transactions

Mediated	The client must each time be introduced by a mediator that builds trust between the client and the merchant
Spontaneous	The client contacts the merchant spontaneously and presents stand-alone authentication credentials (e.g. certificates); the ET terminates either on-line or off-line:
Off-line	The credentials presented are enough to complete the transaction by the sole communication between client and merchant (e.g. digital cash)
On-line	The credentials presented are not enough to complete the transaction, requiring communication with the support infrastructure (e.g. checking credit card validity and ceiling)

The spontaneous off-line versus on-line classes are depicted in Figure 18.25. Continuing our discussion, observe that in the off-line case (Figure 18.25a), the client, prior to the transaction, acquires payment instruments to the issuer (1) that have stand-alone validity (e.g. she loads her smart card electronic wallet with digital cash at the card issuing bank). Then, she contacts the merchant, performs some electronic commerce protocol and finally gets to the stage of payment. The transaction can proceed off-line because the merchant not only believes in the client's Id certificate, but also on the payment credentials she produced (2), and gives her the goods after keeping her payment. The merchant can later contact the acquirer (3), possibly with a bunch of payments, and consolidate them. The acquirer collects the issuing bank (4) through the banking network.

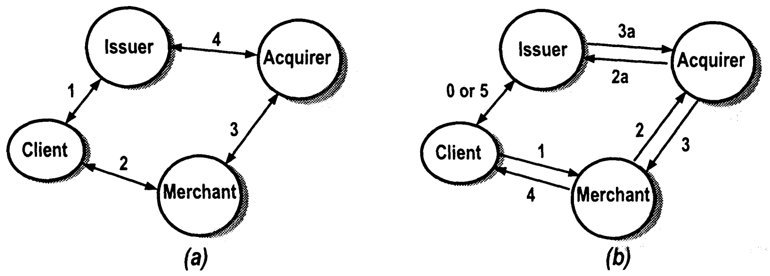


Figure 18.25. Spontaneous Electronic Transaction Classes: off-line vs. on-line

Sometimes, although the merchant accepts the client's Id certificate, the validity of the payment credentials must be checked, e.g. to prevent fraud (e.g. to check double spending of digital cheques, or the validity of credit cards). This requires the transaction to go on-line (Figure 18.25b): the client addresses the merchant (1) and when they get to the payment phase, the merchant goes on-line and contacts the acquirer (2). Depending on the specific type of transaction (e.g. credit or debit), the acquirer may return the payment authorization code (3) after performing local checks (e.g. credit card ceiling and validity), or may instead contact the issuing bank (2a-3a) for further checks (e.g. cheque double spending, account balance). When authorization comes, the merchant handles the goods to the client (4). It may capture payment later, because it has the irrevocable authorization code. Whether the client acquired payment instruments beforehand (0) or the issuer later collects from the client (5), depends on the specific business. An example of the former would be digital prepaid cheques, and an example of the later would be credit card operations.

18.11.3 An Analysis of Electronic Transaction Security

The cryptography used in ETs is relevant for the class to implement, and for the security properties observed by the client and by the operators, namely privacy and fraud protection. *Symmetric* cryptography does not allow mutual authentication in spontaneous transactions, and as such in absence of a arbiter or adjudicator, fraud cannot be effectively handled. Only mediated on-line transactions apply. Privacy is thus not great. The use of *asymmetric* cryptography is a pre-requisite, in order for certification authorities to be in place. In this case, spontaneous on-line transactions are immediately possible that achieve fraud prevention, since clients can present stand-alone identity certificates.

Off-line ETs are also possible, with protocols where fraud, although not prevented, can be detected, and the culprit identified. But this also depends on the business risk analysis and on how efficient is the court system. With *asymmetric* cryptography and *blind signatures*, one can build protocols that support non-traceable spontaneous ETs, with digital cheques or cash. That was shown in Section 17.7: on-line transactions prevent fraud, whereas with off-line transactions, fraud can at least be detected. The tradeoff to be made depends again on the risk analysis. Another aspect of the problem is that most current ET systems provide *one-sided security*. It is highly desirable to go towards *multi-party security* architectures, where the security of each participant does not depend on his a priori trust on the other participants, and where privacy and non-traceability can be ensured as much as possible. We are discussing these issues further in Section 19.5.

18.12 SUMMARY AND FURTHER READING

In this chapter, we discussed the main models of secure distributed computing. The first objective of the chapter was to provide insight to the system architect, about the main architectural options, strategies and frameworks that she has available. The second was to discuss the main models in a problem-oriented manner, establishing links, whenever possible, to the paradigms learned in the previous chapter.

As further reading, we advise the following works. Slade does a fairly complete practical study on computer viruses (Slade, 1995). Neumann gives an interesting account of several security related risks and hazards, some of them caused by a wrong evaluation of the severity of faults, or by the layout of inadequate strategies (Neumann, 1995). Further study on Lampson's model of distributed authentication can be found in (Lampson et al., 1992). In (Abrams et al., 1995) there are excellent studies on access control mechanisms and policies, and on security policies.

On attacks and countermeasures, there is an anti-eavesdropping mechanism described in (Rivest and Shamir, 1984), using an *interlock protocol*. Sophisticated spoofing attacks against Web pages or network downloadable software are reported in (Brewer et al., 1995; Felten et al., 1996). Attacks on using the same public key protocol for signing and encrypting are detailed in (Dolev and Yao, 1981; Kaufman et al., 1995), or (Schneier, 1996). Abadi and Needham do a study on attack-resilient design of protocols in (Abadi and Needham, 1994). Needham discusses attacks to a secure channel in (Needham, 1993).

A distributed TCB implementation is discussed in (Nicomette and Deswarte, 1997). There is an advanced discussion on authentication pitfalls in (Kaufman et al., 1995). Authentication and key distribution protocols can be further studied in the following publications. An attack on Needham and Schroeder (Needham and Schroeder, 1978) original protocol was reported by Denning in (Denning and Sacco, 1981), and corrected in (Needham and Schroeder, 1987).

Otway and Rees improved the latter protocol in (Otway and Rees, 1987). Denning and later the Kerberos protocol (Neuman and Ts'o, 1994) proposed to use timestamps as a form to foil replay attacks. However, Gong (Gong, 1992) showed that if an attacker succeeds in de-synchronizing the clocks, he can replay old messages that seem current to the slow clocks. This is called a *suppress-replay attack.* Neuman and Stubblebine corrected the problem in (Neuman and Stubblebine, 1993). See also (Gollmann, 2000) for a discussion on the pitfalls of verification of authentication protocols.

On protection, Cheswick and Bellovin wrote one of the most complete essays on firewalls (Cheswick and Bellovin, 1997). Formal access control models other than BeLa and BiBa exist, such as Denning's (Denning, 1976). Recent work on classification criteria taking fault tolerance and security both into account is the Squale Criteria (Corneillie et al., 1999). Security kernels are discussed with detail in (Ames et al., 1983; Schell, 1984). On the programming side, the Generic Security Service API (GSS-API) is an attempt to standardize an API for secure operations, independently from platform (Linn, 1996). The advantages are obvious, and for example, Kerberos V.5, among other products, is GSS-API compliant.

19 SECURE SYSTEMS AND PLATFORMS

This chapter gives examples of systems and platforms for secure computing. We are going to talk about remote operations and messaging, firewall systems, virtual private networks, authentication and authorization services, smart cards and payment systems, and secure electronic commerce. In each section, we will mention several examples in a summarized form, and then will describe one or two the most relevant in detail. Table 19.2 at the end of the chapter gives a few URL pointers to where information about most of these systems can be found. The table also points to the IETF Request for Comments site, where the RFCs cited can also be found.

19.1 REMOTE OPERATIONS AND MESSAGING

There exist a few remote secure session packages. The Secure Sockets Layer (SSL) is a basic secure channel plus a few ancillary protocols, which allows high-level remote session protocols to work securely, in a transparent way. Developed and used initially by Netscape, it ended-up as a de facto standard in its Version 3.0, and a variation of it is currently endorsed as a standard of the IETF, the *Transport Layer Security, (TLS V1.0 - RFC2246)*. There is a freeware version of SSL 3.0, SSLeay independently developed by Eric Young, that is currently incorporated in the Apache HTTP server. based on SSLeay, the OpenSSL is a collective initiative for developing and making available free SSL software. Also with relation to HTTP, there is an alternative protocol, Secure HTTP

(SHTTP) for achieving secure HTTP interactions, that has been around for years but has been overtaken in importance by SSL. When all that is needed is user authentication, secure, MAC based authentication of plain HTTP is specified in Basic and Digest Access Authentication (RFC2617). Secure Shell (SSH) is a suite of remote session protocols originally developed by Tatu Ylonen, commercialized by Secure Data Fellows, and currently endorsed as an IETF draft standard called SECSH. There are free versions of SSH for some systems, e.g. Linux. The SECUDE package is a freeware set of modules, libraries and APIs for developing remote session protocols, from GMD-Darmstadt in Germany. STEL is a freeware secure telnet developed at the University of Milano. S/Key is a one-time password system based on Lamport's hash, developed at Bellcore and now endorsed as an IETF RFC under the name of OTP, One-Time Password System (RFC2289). Privacy Enhancement for Internet Electronic Mail (PEM) and Pretty Good Privacy (PGP) are the two best known secure messaging packages, and can be used for secure e-mail amongst other things. PEM is a set of IETF RFCs (1421-1424) based on a somewhat complex structure, involving the PKI certification authority hierarchy. PGP is more lightweight in key management, and more versatile in functionality. PGP is currently undergoing a standardization effort, OpenPGP Message Format (RFC2440), to ensure interoperability of different implementations. Sun RPC and DCE RPC are examples of secure RPC packages. RSADSI supplies a few building modules for use in this kind of packages, such as RSAref, the main library of RSA cryptographic functions, and PKCS, the standard for formatting and encoding of cryptographic structures. Next, we analyze SSL, SSH, PGP and S/Key in detail.

19.1.1 Secure Sockets Layer (SSL)

SSL V3.0 has a basic secure channel layer, implemented by the *Record Protocol*, which uses a socket abstraction and runs on top of any transport protocol, such as, but not limited to, TCP/IP. This layer only knows about establishing a low-level secure channel and sending blocks of data back and forth, in a secure manner. SSL secure channels can securely encapsulate high-level session protocols, such as: HTTP, FTP, SMTP, or POP3. For example, to use HTTP with SSL, you just have to type URLs in the form `https://...`. SSL provides remote sessions with:

- anonymous, unilateral or mutual client/server authentication, with digital signature certificates whenever supported
- data compression
- communication encryption via symmetric cryptography
- message integrity via authentication codes (MAC)

A few ancillary SSL protocols recursively use the record layer to extend the capability of SSL to support secure remote session protocols. These are: the *Handshake Protocol*, the *Alert Protocol*, and the *ChangeCipherSpec Protocol*. The Record Protocol provides confidentiality and/or integrity of user message flows, encapsulating user data in `record` messages, which are either protected with a MAC or MAC-protected and encrypted. The Handshake protocol performs client and server authentication. The Alert protocol signals errors and exceptions through `alert` messages. The ChangeCipherSpec protocol is used whenever the cipher specifications change. This can be done in the middle of a session. SSL Version 3.0 supports RSA, X.509 certificates or Fortezza[1] for authentication. Encryption may be done by DES or RC4, with a 128-bit key limited to 40 bits for export versions. Integrity is secured by means of SHA and MD5 MACs.

Handshake Protocol The Handshake protocol (*see* Figure 19.1) initiates a session, performing negotiation, authentication and session key exchange. The protocol starts with the client and server exchanging nonces (*C-Random, S-Random*) negotiating the SSL version, session Id, and type of cryptography and compression (10,11). Next, authentication and key exchange takes place. The server normally sends its public key certificate (20a). Alternatively, if it does not have one, it sends a key exchange message (20b) with additional data to make an ad hoc key exchange. The server may also request a client certificate, if mutual authentication is desired (21-22), otherwise, only the server authenticates to the client. At this point, Hello is terminated (23). The client now sends a client key exchange message (30) to set up the initial cryptography. Both exchange now messages specifying the type of cryptography that will be used, and finish by sending one another *Finished* messages (32-33).

Authentication and Key Generation The goal of the key exchange process within the Handshake protocol is to create a *pre-master-secret*, which in turn will lead to a *master-secret*, from which all other keys will finally be derived. The initial key exchange depends on the authentication mode (anonymous, unilateral or mutual) and the cryptography suite, which may be RSA or Diffie-Helman, either plain or signed.

We will study *authenticated RSA*, the most relevant for secure Web transactions. Authentication is combined with key exchange. The client creates the *pre-master-secret*, a 46-byte random plus 2-byte version ID. The client then encrypts the *pre-master-secret* with the public key in the server's certificate, sent in the respective *S-Cert* message to the client, and sends it to the server, in an *EncryptedPremasterSecret* record of the *ClientKeyExchange* message. When the client receives the *Finished* message, it may deduce that the server has successfully decoded the *EncryptedPremasterSecret*, and is thus authentic.

At this point, both the client and the server have the *pre-master-secret*. The *master-secret* is computed with a few hashing operations having the *pre-master-secret*, *C-Random*, and *S-Random* as parameters. After a few more hashing operations, the cryptographic checksumming (MAC) and encryption keys are extracted. The *ChangeCipherSpec* message synchronizes the end

[1]Fortezza is a key escrow hardware assisted protocol that we will not address here.

	Action	Description
10	C → S	Client C opens connection with *ClientHello* message: ⟨*C-Random, SessionId, CipherSuites, CompressionMethods*⟩
11	S → C	Server S sends *ServerHello* message: ⟨*S-Random, SessionId, CipherSuite, CompressionMethod*⟩
20a	S → C	*S sends its certificate:* ⟨S-Cert⟩...
20b	S → C	*or S sends a* ServerKeyExchange *message*
21	S → C	*S sends the client a* CertificateRequest *message*
22	C → S	*C sends its certificate:* ⟨C-Cert⟩
23	S → C	S sends *HelloDone* message
30	C → S	C sends a *ClientKeyExchange* message
31	C → S	*C sends S* CertificateVerify *message*
32	C ↔ S	C and S both send *ChangeCipherSpec* messages
33	C ↔ S	C and S both send *Finished* messages
-	C,S	They are authenticated and have a secure channel set up

Figure 19.1. SSL Handshake Protocol (italicized steps are either optional or alternative)

of this process. The subsequent *Finished* messages go MAC-protected and encrypted with the recently negotiated keys, and are used to test if the process was successful.

This concludes our study on how to make secure sessions on the Web with SSL. Given that most extranet (and also intranet) access to applications is currently via Web protocols, the security of the architecture per se deserves some brief comments. Despite the cryptographic material available for secure web-based applications, these may fail on account of hidden vulnerabilities of browsers, servers, and languages themselves, from HTTP to Java. So, much attention should be given to configuration and operation of web-based systems in a way that their security is not jeopardized by those vulnerabilities.

19.1.2 S/Key

S/Key one-time password (OTP) system is a simple package aiming at protecting remote sessions from passive eavesdropping attacks. It does not store sensitive information, and works with personal terminals, from workstations and PCs, to CRT terminals. S/Key can also authenticate FTP, besides Telnet.

Principle of Operation The OTP mechanism is inspired by Lamport's hash (*see* Section 18.5, *Password-based Authentication*). A primordial secret password p_s is generated from a random number *seed* and a secret user passphrase

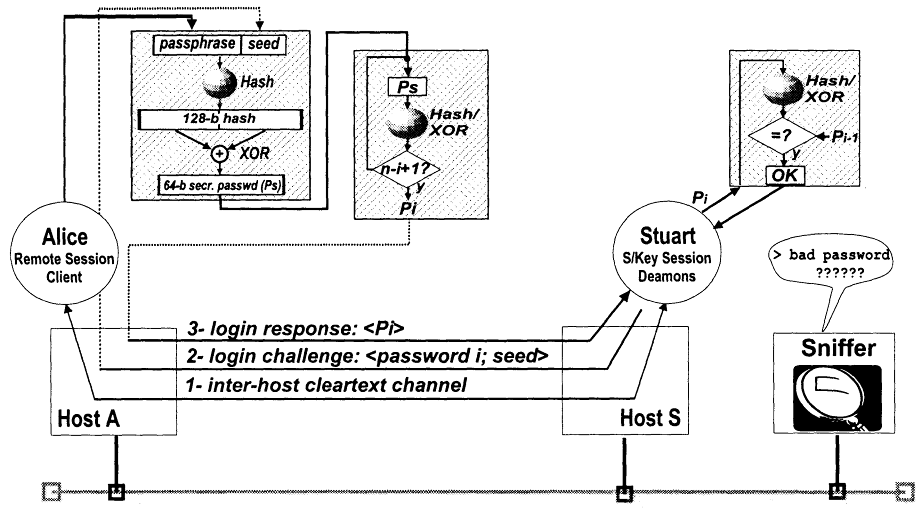


Figure 19.2. S/Key One-time Password System

P. A passphrase is an arbitrary length legible string. See Section 18.3.2 to recall why passphrases are good. P is concatenated to *seed*, and passes through a secure hash function. Several hash functions are currently supported, such as MD4, MD5 and SHA, so let us run our example with a generic H. The 128-bit output is halved, and both halves XOR'ed, yielding an 64-bit *secret password* p_s. Now, suppose we want to "buy" say a batch of $n = 16$ passwords. To generate the first 64-bit one-time password, $i = 1$, p_s goes through the hash/XOR function recursively n times. To get the second, it will go $n - 1$ times, until $i = n$, when it goes just once. The general expression to get password $p_i, 1 \leq i \leq n$, is: $p_i = H^{n-i+1}(p_s)$

The Real Thing S/Key operation is depicted in Figure 19.2. Alice gives her username. The server Stuart replies with the expected password *sequence number*, i, and the *seed*. The seed can be different from system to system, and thus allows Alice to use the same passphrase in all of them. Besides, it allows her to recycle the passphrase when passwords are exhausted. Now Alice has to have a local program installed on her computer to calculate the following: the program asks Alice to type in her passphrase P, the seed and the sequence number sent by the server; the output is p_i (*see* the expression in the last section), a highly random password; alternatively, she can request the system administrator to generate and print a list of passwords for her to take.

Alice sends the password down the line, in cleartext. How is it authenticated? The server database stores the last used password, p_{i-1}, so that it easily checks if p_i is a good password by hashing it once and confirming that $H(p_i) = p_{i-1}$. If the sniffer copies p_i, when he tries to use it, it will no longer work. Besides, since the algorithm works backwards, he cannot derive p_{i+1} from p_i either.

19.1.3 Secure Shell (SSH)

SSH is a secure session protocol suite that plugs classical holes in Internet/UNIX based protocols. It supports several encryption and authentication mechanisms. Besides protecting the login and authentication process, it also compresses and encrypts the session (DES,3DES,IDEA). MD5 is used for hashing. Authentication is modular: there exists the notion of server (host), service (application) and client authentication. SSH supports three authentication styles: traditional address-based (.rhosts, /etc/hosts.equiv) or UNIX password protected by the secure channel; traditional enhanced with RSA; pure RSA. Key distribution is also versatile: it can be manual, automatic or administrator based. There is a user authentication agent, in charge of keeping the RSA keys, in case RSA authentication is used. Several typical services are protected by this package:

- secure remote session (e.g., rlogin or telnet)
- secure remote execution (e.g., rsh)
- secure remote copy (e.g., rcp)

		Action	**Description**
1	C → S	$\langle C, service\rangle$	Client C requests service connection to SSH server S
2	S ↔ C	$\langle versId\rangle$	C and S exchange version info
3	S → C	$\langle K_h, K_a, ciphTyp, X_s\rangle$	S sends RSA keys of host server, K_h (TYP 1024-bit), and application service, K_a (TYP 768-bit), cipher suites, and a challenge (64-bit random), all in cleartext
4	C,S	$SID = H(K_h + K_a + X_s)$	S and C compute a 128-bit session Id SID (+ means concatenate)
5	C → S	$\langle ciphTyp, X_s, E_h(E_a(K_{cs}))\rangle$	C generates a random 256-bit session key K_{cs}, and sends it to S, along with the chosen cipher, and the server challenge. The session key is XORed with SID, encrypted with K_a and then with K_h
6	S → C	$\langle E_{cs}(cfm)\rangle$	S extracts key K_{cs} and sends a confirmation encrypted with it
-	S,C	-	*// low-level secure channel established*
7	C ↔ S	-	C now authenticates to the service in one of the methods available
8	C ↔ S	$\langle E_{cs}(msg)\rangle$	Session proceeds, encrypted with K_{cs}

Figure 19.3. SSH Secure Shell

The basic secure session operation can be understood by looking at Figure 18.22 back in Section 18.10. The session establishment protocol, depicted in Figure 19.3, underlies all the operation of SSH. Note that the first phase (1-6) is concerned with set-up of the low-level channel. Then, it is necessary to authenticate the remote session that will work on top of the channel (*see* the principles of *Secure Remote Session* in Section 18.10). This is done using one of the methods available: address-based with or without RSA, password, or RSA-only. Password authentication, the most used, is robust because it trusts nothing but the holder of the password, since it does not dialog with a login program, but with the SSH daemon.

SSH offers two useful additional functions: bi-directional TCP/IP port forwarding over the secure channel, implementing tunneling; X11 connection tunneling, to secure remote X terminal sessions, that usually go in the clear and are thus a security headache. The principle of tunneling is discussed in Section 19.3 (*see* Figure 19.7).

19.1.4 Pretty Good Privacy (PGP)

PGP is a freeware messaging and file encryption software, based on hybrid cryptography. Key management is based on public key cryptography (RSA), and payload encryption resorts to IDEA. It achieves the properties of secure envelopes.

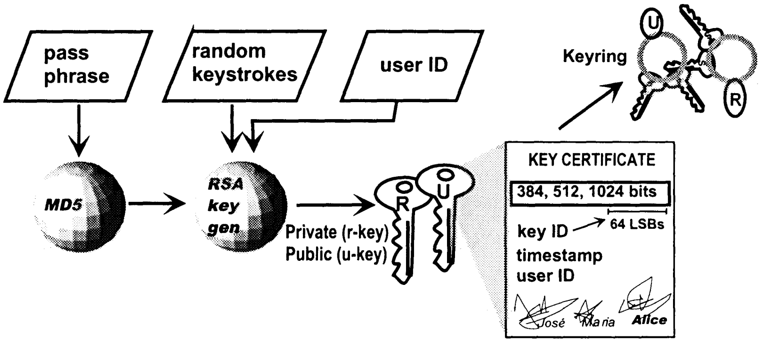


Figure 19.4. PGP - Key Generation

In PGP, all starts with key generation, shown in Figure 19.4. The user is prompted to supply some random data for the process (e.g., key strokes), and a passphrase. The passphrase is hashed and together with the random information and the user Id, they form the raw material to generate the RSA keys. Each key is then put in a certificate together with timestamp of generation and owner Id. Key certificates are kept in *keyrings* (public and private key rings). The public key certificate is then inserted on the *pubring*, whereas the private is inserted in the *secring*. Private keys are protected with the passphrase. Authentication is mutual, based on a ad-hoc chain of trust, instead of using a PKI: principals sign key certificates of other principals and so forth, creating a mutual chain of trust among clusters of people that are related.

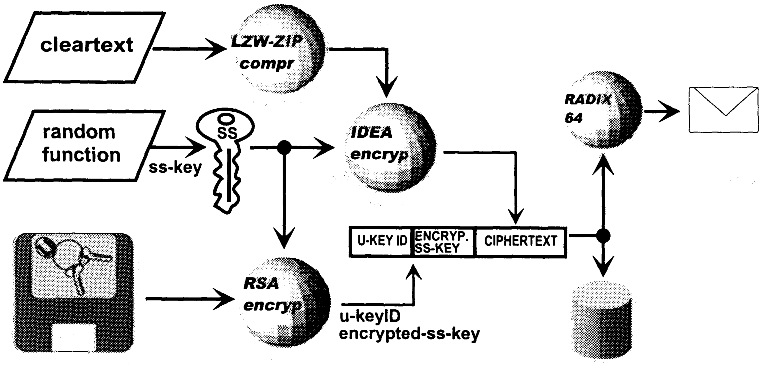


Figure 19.5. PGP Encryption

PGP can encrypt, sign, sign and encrypt. These operations follow the hybrid cryptographic envelope principle (*see* Figure 18.6) and are thus perfect for secure e-mail. Besides, PGP can encrypt local files with plain symmetric (IDEA) encryption, using a passphrase-derived key. The envelope encryption operation is shown in Figure 19.5. The cleartext is compressed first with the ZIP compression algorithm. As we pointed out earlier, this is a good idea, remember why? A symmetric encryption key (K_{ss}) is generated out of a random function. The cleartext is encrypted with IDEA using K_{ss}, and K_{ss} itself is RSA encrypted using public key K_u of the recipient as a key-encryption-key. Both the encrypted encryption key and the recipient Id go along with the ciphertext. Since PGP allows RSA key lengths in excess of 1024 bits, and IDEA itself does pretty well with 128-bit keys, this is bound to be very robust. Note that the result of encryption is a binary file. If the file goes to disk, this is OK. However, if it is an e-mail message, then RADIX-64 encoding converts it to an ASCII stream. Decryption is performed by reversing these operations at the other end: the recipient PGP extracts the key-encryption-key— the private recipient key K_r— from *secring*, decrypting K_{ss}, and then decrypting the payload with the latter.

Finally, signing is depicted in Figure 19.6. Signing follows the principle of digital signature with digests that we studied (*see* Figure 17.8). PGP makes sure that the cleartext has adequate format or control information to ensure it is verifiable at the other end. The text is hashed by MD5, and the result, concatenated with a random quantity to avoid replay attacks, is signed with the user's private key. A signature certificate is produced, by appending the key Id and the timestamp of generation. The cleartext, the certificate and the public key Id of the signer form the message, that is RADIX-64 coded if it should go by e-mail. The recipient PGP extracts the relevant public key from *pubring* and verifies the signature. Signed/encrypted messages combine both procedures.

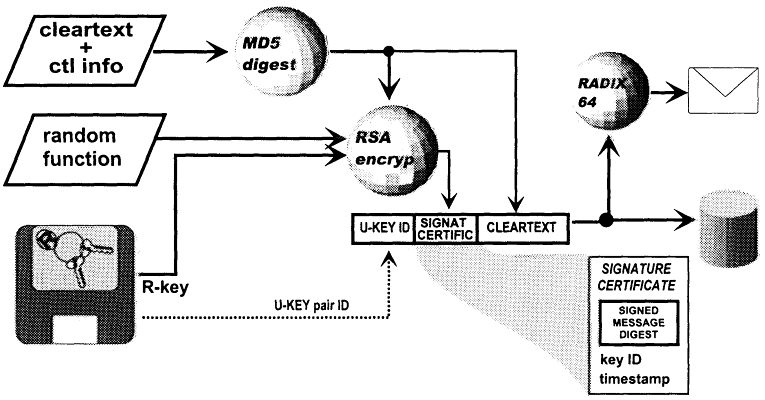


Figure 19.6. PGP Signatures

19.2 INTRANETS AND FIREWALL SYSTEMS

Intranets are the nickname for protected environments, normally organization networks closed to the outside or connected via protection devices, despite using Internet protocols. Since it hardly makes sense for an intranet to be physically disconnected, the most relevant devices for building intranet architectures are firewalls. There a number of commercial firewall systems, and perhaps the two best known, representing two competing classes, are the Checkpoint Firewall-1 and the Trusted Information Systems Gauntlet. Of the many free firewall packages around, a few are known to be effective and reasonably secure and bug free: the TIS firewall toolkit; the SOCKS proxy package (RFC1928-29,1961), and the LINUX packet filter. The TIS toolkit is a proxy package, supporting the most usual UNIX Internet daemons, such as telnet, rlogin, FTP, HTTP and mail. It has its own authentication server, supporting regular and one-time passwords. It also supports logging, and has a well-structured configuration and management interface, following a prudent policy. SOCKS is also a proxy package, but unlike TIS it bundles all servers in a single daemon, making it harder to fine-tune policies. It mainly supports telnet and ftp, plus a few ancillary services. Its security policy is less conservative than TIS. It supports authentication as well. The LINUX firewall package, IPchains, is a packet filter system that comes bundled with the distribution. Next, we study some of these systems with more detail.

19.2.1 Firewall-1

The Firewall-1 is filter oriented. However, it uses a form of stateful packet filtering (SPF), called Stateful Multi-Layer Inspection, which tries to understand the following high-level protocols: Telnet, FTP, SMTP, rlogin and rsh, NIS, NFS, HTTP, Gopher, Archie, WAIS, ICMP, RIP, SNMP. FW-1 supports several O.S.s, but it only supports two families of routers. FW-1 provides for NAT (Network Address Translation) and for secure cryptographic channels between modules, using symmetric and asymmetric encryption: DH key exchange; RSA for public key certificates; and DES or FWZ1 session keys. It is divided into two modules that may or may not be co-located: the control and the filtering modules. The control module hosts the GUI interface and the Management module. Configuration is done through a very powerful GUI. The filtering module hosts the inspection module and the daemons. Besides the firewall daemon (fwd), it provides authentication daemons for use with several known services: atelnetd, aftpd, ahttpd, aclientd. Users can be individually authenticated for the above-mentioned services, and the authentication methods supported are: UNIX regular and one-time passwords, MD5 MACs, Kerberos, Smart Card support, SSL and SHTTP. FW-1 supports event and audit trail, firewall status monitoring and alarm generation (Status Monitor and Log Viewer). Access control rules are defined through a special purpose script language, INSPECT (Rule Base Manager). Following the SPF philosophy, rules have higher-level semantics than normal packet filters. The Network Object Manager defines security labels for networks, servers, routers, etc. The User Manager defines access rights for users on objects. The Service Manager manages services. Extension of services is simply done by the addition of an additional set of expressions and macros. Performance is good, as usual with packet filter systems. Firewall-1's main assets are: excellent GUI interface; overall performance; enhanced application-aware packet filtering; modular structure supporting a number of firewall architectures.

The Inspection module lives between the network interface of the bastion and layer 3 (e.g., IP), and inspects every incoming or outgoing packet. The baseline policy is prudent (*see Packet Filter Systems*, Section 18.8). The dynamic filter only opens the ports involved in cleared transfers, and closes them when the transfer ends. Connectionless protocols are difficult to follow by PFSs. FW-1 simulates a connection for UDP and similar protocols, so that it can follow a flow and reject alien packets. Similarly, RPC does dynamic port allocation. FW-1 monitors the portmapper and checks further RPC traffic against its cache of mapped ports.

19.2.2 TIS Gauntlet

The Gauntlet firewall is proxy oriented. It supports proxies for the best known services, such as: Telnet, rlogin and rsh, FTP, SMTP, POP, HTTP, Gopher, X-Windows, lpr. It also supports some packet filtering activity. Authentication includes: UNIX passwords, MD5 MACs, Smart Card support, SSL and SHTTP. Configuration is menu-driven, and addresses: the firewall architecture (network interfaces, dual or single-homed, addresses and services); configuration of access rules and user authentication; system integrity check against write penetration attacks; log and event report manager. Extension of services is done by the addition of new proxies. Performance is fair, as with any proxy-based system. Gauntlet's main assets are: proxies are in essence *transparent*, not requiring any adaptation or change in users and client applications; it has a framework for developing custom application gateways, called *plug gateway proxy*, with which designers can support specific services and non-standard applications.

When a connection request comes in, the firewall analyzes the configuration rules (*see Proxies*, Section 18.8) and determines whether or not it should proceed. If so, the proxy contacts the end service, and from then on, the steps of this connection are performed by the proxy between the client and the end server, the proxy acting as server to the former, and client, to the latter (*see* Figure 18.20b in Section 18.8). However, everything happens at the proxy level, with the obvious performance implications. Later, TIS introduced the concept of *adaptive proxy*. An adaptive proxy uses a dynamic packet filter system (DPFS) at the internetwork layer. When a connection comes in, the DPFS notifies the proxy, providing information about the former. The proxy analyzes the connection parameters against the access control rules, as usual. However, when a connection is allowed through, it further decides if it proceeds at application level, or instead, because the connection is considered to be very low risk, it is forwarded directly at the internetwork layer. In that case, the dynamic packet filter manager inserts one or more rules for this connection. Subsequent packets of the connection are then automatically forwarded without consulting the proxy. Once a connection terminates, the connection rule is removed and the proxy is notified.

19.3 EXTRANETS AND VIRTUAL PRIVATE NETWORKS

In the measure that system architects became aware that leased lines are not secure links per se, and that using the Internet infrastructure provides significant financial gains, extranets emerged. Extranet technology aims at ensuring secure communication through the Internet, from the outside to an intranet, or between intranets, in essentially three situations: between distant facilities of the same organization; between facilities of different organizations; or between the facility and remote users belonging to the organization. This kind of architecture is thus relevant for: geographically distributed enterprises, in extension of their intranet; for virtual enterprises or enterprise networks, gathering suppliers, producers and clients, such as manufacturing clusters or business-to-business electronic commerce; or for mobile organization workers. The main attribute of extranet architectures is that security of external communication should approximate that achieved inside the intranet. Thus, important building blocks for these architectures are secure communication protocols (*see* Section 18.10), such as secure packets, tunnels and sessions, secure Internet and wireless communication protocols, and secure Web protocols.

19.3.1 Virtual Private Networks

The main architectural device for building an extranet is the *Virtual Private Network (VPN)*, an example of which is shown in Figure 19.7. Networks A and C, and the tunnel interconnecting them, constitute a very simple VPN over Network B. Networks A and C are intranets of the same organization, and it is desired that traffic goes from one to the other as if they were in the same facility. For example, all addresses in both networks might be of the same domain, but different subnets. Either intranet is isolated from direct access to/from the Internet by a *security gateway*, such that the tunnel is laid between the two security gateways. These can be implemented by firewalls. The source and destination addresses of the payload packet on the left of the figure are the actual source in network A (Alice) and the final destination in network C (Bob), whereas the source and destination addresses of the carrier packet are those of the security gateways in each extremity. Tunnels may be set up with the desired granularity. They may carry the whole data between two networks, or there may be separate encrypted tunnels for critical connections, even inside untrusted intranets. Whatever the selection condition, routing tables inside the source network must route packets scheduled to go through a tunnel, to the tunnel mouth, that is, the security gateway, and not through the usual outgoing router. The VPN concept also addresses host-to-gateway tunnels, to support extranet client-server access to the intranet by remote users (e.g. travelling employees). You can now generalize the examples given and imagine a real installation with say half a dozen intranets, each of them interconnected to every other by a tunnel, and several remote access tunnels or secure channels, either from remote client-only offices, or mobile salesmen or executives.

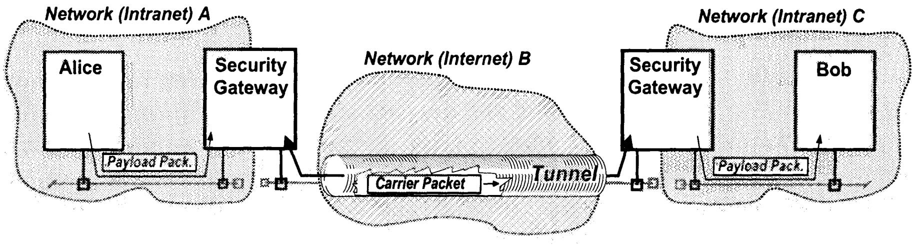


Figure 19.7. Virtual Private Network Architecture

The main technology behind extranets are tunnels. Most firewall manufacturers have extensions or separate packages implementing tunnels (e.g., Checkpoint, Gauntlet, Secure Data Fellows). Most of these implementations are not interoperable. In order to overcome this problem, the IETF is standardizing *IPsec* (Internet Protocol Security Architecture), a security architecture framework for IP.

19.3.2 Secure Internet Communication: IPsec

IPsec (Internet Protocol Security Architecture) is the current initiative of the IETF (RFC2401) to provide cryptographically-strong security for the IP protocol (Kent and Atkinson, 1998). It addresses: access control, connectionless integrity, data origin authentication, protection against replays, confidentiality, and limited traffic flow confidentiality. IPsec is a protocol-independent framework, that guarantees negotiable security properties to IP flows between two nodes. Nodes are either hosts, or routers running IPsec, called *security gateways*. The properties are secured for armored data blocks, defined by a security header that encapsulates the attached data. The cryptographic operations on each module are specific to the several protocols that may be used. These security functions are implemented around two extension headers and respective processing protocols:

Authentication Header (AH) - provides connectionless integrity, data origin authentication, and protection against replays

Encapsulating Security Payload (ESP) header - provides confidentiality by encryption, and limited traffic flow confidentiality. It may also provide the functions of AH

IPsec headers can be combined with one another and with regular IP headers. The AH protects the integrity of a block of data, except for the fields that must be changed en-route. The AH includes security information for the receiver, the Security Parameter Index (SPI) field, and the authentication field, which has arbitrary length and depends on the algorithm being used. The ESP header includes again security information for the receiver (SPI), and the transformed data, according to the algorithm used. IPsec (either AH or ESP) can be used in two modes: transport-mode, which corresponds to the generic concept of secure channel depicted in Figure 17.18 back in Section 17.11; and tunnel-mode, which corresponds to the tunnel concept (a form of secure channel) illustrated in Figure 19.7 in this Section.

- **Transport-mode**– the protected data are upper layer service data units. This option encapsulates data from the layer above (TCP) with one of the IPsec headers, and then encapsulates it again in a normal IP datagram, achieving end-to-end security
- **Tunnel-mode**– protects full IP datagrams. This option builds a complete IPsec datagram, and then encapsulates it in a normal IP datagram. This IPsec-over-IP mode is useful for building tunnels, and for bypassing network areas that do not implement IPsec, achieving link security

Cryptographic checksums or signatures in AH, besides generally ensuring integrity, may provide reliable source address and sequencing information, to avoid spoofing and replay attacks. Data may be encrypted with ESP. Although IPsec is algorithm-independent, the default protocol is DES-CBC. Data flow confidentiality can be enforced to a certain extent with tunneling, since content of traffic, such as addresses, is hidden. Before payload transmission can start, IPsec must bootstrap through two crucial functions:

- **security association -** negotiation of protocols, ciphers and keys to be used
- **key distribution -** exchange of the keys needed for communication

The security association negotiation produces the above-mentioned SPI structure, which specifies things like: authentication and encryption algorithms; authentication and encryption keys; key and association lifetime. A security association is uniquely identified by a triple consisting of a Security Parameter Index (SPI), an IP Destination Address, and a security option (AH or ESP) identifier. Key distribution may be manual or automated, in which case it uses a protocol. Several of the key exchange models that we studied in Section 18.6 are foreseen, both in the public-key and symmetric shared-secret areas. Security association and key management are at the time of this writing very active topics in the IETF, with the Internet Security Association and Key Management Protocol, ISAKMP/Oakley, being a strong candidate (RFC2408,RFC2412).

Headers may be combined to achieve further protection. For example, in transport mode, by applying ESP encryption for confidentiality of upper layer data, and then encapsulating again with AH for MAC-based integrity and authentication of the final IP packet. This is called *transport adjacency. Iterated tunneling* concerns building tunnels inside tunnels. Several combinations are possible, but perhaps an obvious and useful one is when specific tunnels, say ESP protected, are built from host to host in different intranets of an organization, to serve different applications, and then all these tunnels go, AH protected, through an outer, main tunnel, carrying all the traffic from one intranet to the other intranet across the Internet.

19.4 AUTHENTICATION AND AUTHORIZATION SERVICES

Authentication services exist for a number of applications and systems. Modem dial-up access is many often authenticated with front-ends such as Radius (RFC2138), from Livingston Enterprise, or the Cisco TACACS (RFC1492). RADIUS (Remote Authentication Dial In User Service) is a package for remote network access authentication in an open systems environment. RADIUS is independent from the communications protocol, and has two modules: the authentication server and the client protocols. RADIUS servers authenticate users against a UNIX password file, the Network Information Service (NIS), and an internal database. Password information is sent encrypted over the line, by a shared secret key. TACACS is similar to Radius, but some differences exist. TACACS uses TCP instead of UDP used by Radius. TCP is more resilient to errors, and provides immediate indication of communication or server failure. Radius sends a lot of relevant information in cleartext. Sensitive parameters such as username, authorized services, and accounting, can be captured by an intruder. TACACS encrypts all user information. TACACS has modular authentication, authorization and access control. TACACS can bind to Kerberos for authentication. Kerberos (Neuman and Ts'o, 1994), is the most widely used general purpose authentication and authorization server, included in services such as the Andrew File System and DCE. Several firewalls include hooks to Kerberos. Unlike Kerberos, which is KDC-based, the Distributed Authentication Security Service or DASS (Kaufman et al., 1995) is a distributed, CA-based authentication service developed at Digital and endorsed by the IETF (RFC1507).

19.4.1 Kerberos

Kerberos is conceptually divided in two modules, the Key Distribution Center (KDC) and the Ticket Granting Service (TGS). However, they reside in the same host and share the same database. The KDC handles the primary login of a principal. The TGS is invoked each time a principal needs a credential, or ticket, to access a regular system service. The TGS also checks the privileges of the principal to access the request service. The unit of modularity of Kerberos is a *realm*, the set of resources under the control of a KDC.

From now on, it is important that you have in mind the Kerberos authentication protocol, presented in Figure 18.10 back in Section 18.5. Our description will be based on Kerberos version 5. Each principal shares a *master key* with Kerberos, which it stores in a database. User's keys are generated from the user password through a cryptographic hash. Currently, Kerberos only supports DES. The database is encrypted with Kerberos own master key, K_{kdc}. Kerberos requires clocks to be synchronized, since it uses timestamps as nonces in defense against replay attacks. The allowed de-synchronization is five minutes. The credentials produced by Kerberos, called *tickets*, have a specifiable validity, limited to 21 hours. In what follows, we denote K_a as A's master key, K_{ab} as a session key shared between A and B, and *tick(A,B)*= $E_B(A, K_{ab}, T_t, T_l)$, as the ticket given to A in order to access B. The ticket contains A's Id, the shared key, the timestamp of creation T_t, and its time-to-live T_l.

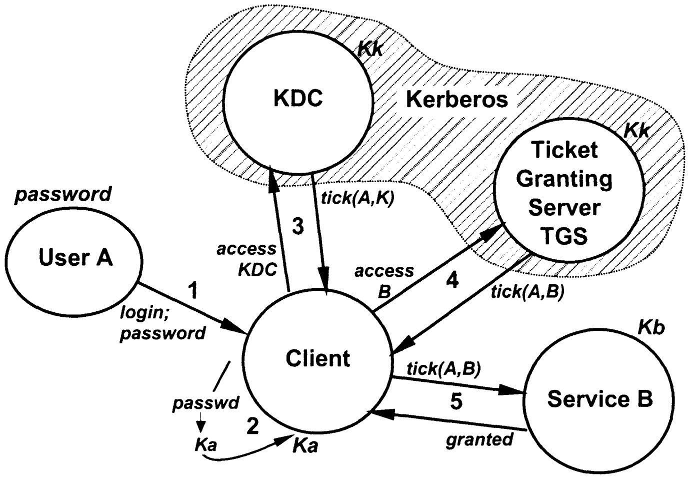


Figure 19.8. The Kerberos Security Service

The steps for a principal A to get access to a service B are depicted in Figure 19.8. The first step (1) is primary login. Principal A types in her $\langle login; password \rangle$ pair at the client host. The host hashes the password into A's master key K_a (2), and sends a login request to the KDC (3). Along with it, the client makes a proof of knowledge of K_a, by encrypting the current time with it. The KDC checks the time to see it is current (less than 5 minutes skew), which also proves that A knows K_a, and concludes the login process by handing A a *conversation* key K_{ak}, and a bf Ticket-Granting Ticket, TGT, all encrypted with K_a. The TGT is a credential to be used in any subsequent addresses to Kerberos *in this login session*, and the key protects these interactions when necessary. Client A is now ready to access actual services. When A wants to access service B, she requests so to the TGS (4), presenting her TGT. The protocol develops as described back in Figure 18.10, with A getting *tick(A,B)* and K_{ab}, and presenting *tick(A,B)* to B (5). The ticket also contains authorization data, produced by the TGS after checking A's privileges to access B. After authentication and authorization is cleared by B, A and B share session key K_{ab} and A can access B.

19.4.2 DASS

The Distributed Authentication Security Service (DASS) is a distributed, hybrid cryptography authentication service. Whereas Kerberos is based on the KDC model and uses symmetric cryptography, DASS is a good sample of a CA-based system. It relies on long-term asymmetric (RSA) keys served by Public Key Infrastructures (hierarchies of Certification Authorities), primary login asymmetric (RSA) keys, and symmetric (DES) session keys. Alice has a long-term asymmetric key pair $\langle Ku_a, Kr_a \rangle$, and so does Bob. Furthermore, client Alice has a login password P, and she can generate a password-derived encryption key K_P. As well as for any other user, certification authority CA with public key Ku_{CA}, stores a record for Alice comprising $\langle A, Ku_a, E_P(Kr_a), H(P) \rangle$, that is, her public key, her private key encrypted with the password-derived key, and a hash of the password itself. Any public key certificate can be obtained from a CA (*see* Figure 18.11 in Section 18.6).

In what follows, we show how Alice initiates a session with Bob, in order to illustrate the functionality of DASS. The protocol is presented in Figure 19.9, and is self-explanatory. The process starts with a *login* phase, during which Alice *pre-authenticates* to the CA, by proving that she knows the password relevant to her record in the CA. Next, Alice performs the *authentication* phase with Bob, at the end of which they both have a session key. Note that after login, authentication of specific accesses is performed in just one message (unilateral) or two messages (mutual). Other relevant hybrid distributed authentication mechanisms are EKE (Encrypted Key Exchange), described back in Figure 18.15, and the NetWare authentication service.

19.5 SECURE ELECTRONIC COMMERCE AND PAYMENT SYSTEMS

Electronic Commerce *(e-comm)* comprises business made through informatic means. The payment is electronic, which prefigures an *electronic transaction* (*see* Section 18.11). The procurement may also be electronic, i.e. made through the Web. The goods themselves may also be electronic (on-line books, MPEG-3 contents, software packages). An e-comm purchase has five phases: *procurement; negotiation and order; payment system selection; payment authorization and capture; delivery of goods*. Generically, a customer navigates through a

		Action	**Description**
-			*// pre-authentication*
1	A → CA	$\langle E_{CA}(K_{ss}, h_P, T_A)\rangle$	Alice invents symmetric key K_{ss}, sends it together with password hash and local timestamp, encrypted with the CA key
2	CA → A	$\langle E_{ss}(E_P(Kr_a))\rangle$	CA verifies that Alice knows the password hash, and checks the clock skew. If OK, CA sends Alice her doubly encrypted long-term private key
3	A	$LC_{al} = S_a(Ku_{al}, T_x)$	Alice decrypts with K_{ss} and the password-derived key, recovering Kr_a. Then, she generates a login key pair $\langle Ku_{al}, Kr_{al}\rangle$, and produces a public *login key certificate* LC_{al}, signed with Kr_a, containing the login key and the expiry time
-			*// authentication*
5	A	$SC_{ab} = S_{al}(E_b(K_{ab}))$	Alice creates a session key K_{ab}, and wraps it a *session key credential*, SC_{ab}, encrypted for Bob and signed with the login key
6	A → B	$\langle LC_{al}, SC_{ab}, X_a\rangle$	Alice logs into Bob by sending the login key certificate, the session key credential, and an authenticator based on local time and the session key
7	B	–	Bob checks the skew of the timestamp and Alice's signature on LC_{al}, and extracts the login key, using it to verify SC_{ab} and extract K_{ab}
-			*// Alice is authenticated to Bob*
8	B → A	$\langle X_b\rangle$	Bob sends back an authenticator based on local time and the session key
-			*// Alice and Bob are mutually authenticated*

Figure 19.9. DASS Authentication Protocol

portal or virtual shopping, browses a catalogue, this is the procurement phase. Next, she identifies the good and negotiates with the merchant, for price, conditions, and so forth, hopefully getting to order the goods, by means of a Web transaction, e-mail, phone, fax, mail, etc. This is the negotiation and order phase. The payment system is then selected, from a wealth of possibilities: electronic cash, electronic cheques, credit card, electronic bank transfer, even paper cheques by mail. The merchant acts in order to be sure that he will be paid by the client (this is the payment authorization and capture phase), and then delivers the goods, the final phase. *Electronic shopping* protocols are emerging, covering most of the phases enumerated. However, current protocols focus on the most delicate ones, payment system selection and payment authorization and capture, which pre-figure the so-called electronic transaction. In this section we are concerned with the security of payment devices, such as smart cards and digital wallets, and payment systems. The two most relevant frameworks for supporting electronic commerce are the SSL-related infrastructure and the Secure Electronic Transactions (SET).

19.5.1 Smart Cards

Smart cards are important pieces of e-comm gear. They provide a small, portable and secure means to carry value, and perform operations, some cryptographically secure. Smart cards occur in several types, described in Table 19.1.

Smart card technology is still recent (mid 70's) and is bound to evolve. Besides more powerful processors and larger memory for enhanced functionality, co-processors can be inserted for enhanced security e.g., to implement a guardian scheme (*see* Figure 17.16 in Section 17.7). A "distributed systems" philosophy has progressively emerged for the smart card area. The components of such an architecture are the smart card, the card terminal and the remote server. Standards have emerged, ISO-7816 is the basic one, specifying the mechanical and electronic structure, the I/O communication protocol and command definition for the card-terminal interface. The ETSI GSM standard specifies command messages for mobile phone SIM cards (*see* Section 19.3). The *EMV'96* standard (EMV'96: ICC Specifications for Payment Systems) has more recently been introduced by Europay, MasterCard and Visa, and endorsed by the industry in general.

Companies like Bull, GemPlus, Hitachi, Schlumberger, Motorola, IBM, currently support the ICC (Integrated Circuit Card or smart card) specification, which intends to be a standard for interoperability and secure operation of smart cards in electronic transactions. The specification consists of several parts, and addresses from the mechanical and electronic specification of cards and terminals and the minimum requirements of card and terminal functionalities, to business and applicational issues related with debit and credit on card. The *Java Card* standard, first appeared in 1996, is based on providing cards with a Java Card Runtime Environment, supporting a Java Card API on card, compatible with a subset of the Java language, in order to load and execute *Java Card Applets*. This allows cards to be programmed and loaded with programs, as normal distributed systems hosts. The Java Card API is bound to give a push to smart card based systems, namely in electronic commerce applications, because it is an *open system* specification: companies can develop their own products on the Java card.

19.5.2 Payment Systems

Electronic payment systems take several forms, as a matter of fact essentially emulating real payment systems:

- electronic cash
- electronic cheque

Table 19.1. Smart Card Types

MEMORY CARDS	Memory cards just have a block of non-volatile memory positions. They are very limited in functionality, and also in security, but are useful for small payments, such as phone cards. In this case, each position corresponds to a payment unit. Positions are cleared in the measure that they are "spent". Once cleared, the memory cannot be rewritten, so the card is discarded when finished.
LOGIC CARDS	Logic cards have limited processing capability, implemented by hardwired state machines, but represent an evolution with regard to memory cards.
PROCESSOR CARDS	Processor cards have increased processing capability, since they use microprocessors. They are capable of executing programs and protocols with outside devices. Some of these operations may be cryptographic. Processor (and logic) cards can either be *contact* or *contactless*:
Contact:	More usual, they have a range of contacts on the surface that connect to corresponding contacts on the card reader where they are inserted (instead of being swiped, as magnetic cards), to receive power and to dialogue.
Contactless:	They communicate with external devices by some wireless means. The s implest use short range electromagnetic fields, but the most elaborate resort to radio transponding, which is also used to energize the card. These cards have been used with success in highway toll systems in Europe.

- credit card
- electronic bank transfer

The last has been around for quite some time and is implemented through proprietary protocols in banking networks (e.g. SWIFT). The others are with the reach of the common user.

Ecash was developed by DigiCash, a company founded by David Chaum, the inventor of some of the protocols that we studied in Section 17.7. Ecash is a form of **digital cash** that allows fully anonymous spending. Clients and merchants must have accounts on an Ecash bank. The Ecash wallet, the cyberwallet, is loaded at the bank. Ecash enforces spontaneous on-line transactions: at the time of purchase, the merchant must be on-line with the bank, to ensure that the coins used for payment have not already been spent. Ecash has multi-party security. Ecash is easily integrated with the Web: the client runs the cyberwallet and the browser; the merchant runs a server and a CGI to run Ecash software. When payment of a purchase needs to be done, the merchant's Ecash server contacts the Ecash client, and then payment is performed.

Mondex was developed in the UK and has been in operation since mid'90. It is a prepayment smart-card based electronic cash system. The scheme uses a proprietary chip design from Hitachi, that creates end-to-end secure channels between chips. All devices with which a Mondex smart card should communicate must have such a chip. An optional PIN enhances personal security of the wallet, but if lost or damaged, the money inside cannot be recovered. The system supports multiple currencies and wallet-to-wallet transfer. There are no specific mechanisms for non-traceability, which is said to be ensured by a physical mechanism: the wallets, only way to reference a card holder, are distributed anonymously. Since the channel is end-to-end, Mondex cards can even be loaded by phone. More typically, they are loaded at a Mondex-enabled ATM, that talks to the bank, which has a sort of virtual vault, the Mondex Value Box, with a battery of Mondex chips to dialogue with remote client cards. Similarly, the merchant has a Value Transfer Terminal to receive payments.

CAFE Conditional Access for Europe, was a European project ended in 1996 that developed a secure electronic payment system using the blind signature principle. Unlike Ecash, CAFE used smartcards with guardians, which allowed completely non-traceable, fraud-free and off-line operation. CAFE has a few interesting characteristics. It is entirely public-key based, and achieves multi-party security. It can pay with cash but also sign cheques. It supports multiple currencies and multiple issuers of electronic money. CAFE uses high-quality infrared communication wallets, and provides recovery of lost, stolen, and damaged cards. An optional PIN enhances personal security of the wallet.

Millicent is a *micropayment* system developed at Digital, that allows payments down to USD0.001 to be made. It is still early to see whether this kind of systems will go anywhere, but perhaps they will find a use in Web navigation charging. Millicent uses a currency called *Scrip*, and believes in the principle that the cost of doing a fraud should be more than the value of the transaction. Users of the system aggregate payments until they have enough money to make a macropayment.

The two most relevant frameworks for supporting electronic commerce in general and *credit-card* based in particular, are the SSL-related infrastructure and the Secure Electronic Transactions (SET). We have already talked quite a lot about SSL. Electronic commerce around SSL involves a basic framework consisting of: SSL as a secure communication channel; a Public Key Infrastructure (PKI) in place which is recognized by the players; and SSL server and client authentication mechanisms. The players (client, merchant, acquirer, issuer) contact securely with each other using SSL-enabled servers and browsers. To perform mutual authentication of the principals and to authenticate the several instruments of the electronic transaction (e.g., a credit card credential), they resort to PKI certificate chains. To prevent fraud, they check certificate revocation lists routinely. In the next section, we will focus on an alternative framework: *SET*. The Secure Electronic Transactions protocol resulted from the convergence of early works by Visa and MasterCard on protocols for the secure presentation of credit cards, and is endorsed by several major players, including American Express, IBM, Microsoft, and Netscape. SET has evolved in the recent years as a fully-fledged electronic transaction framework and architecture, together with the relevant protocols. Expectations about its success must obviously be confronted with those about the SSL-based infrastructure.

19.5.3 Secure Electronic Transactions (SET)

SET is oriented to credit card payment, and the overall model obeys that of the on-line spontaneous transaction, depicted in Figure 18.25b, back in Section 18.11. In terms of the figure, the SET protocol is concerned with steps 1,2,3 and 4, between the client (card holder), the merchant and the acquirer, who implements the *payment gateway* between the card-holder issuer and the merchant. The interaction between the acquirer and the issuer is currently secured via a proprietary *banking network*. The SET trust model relies on the existence of a trusted third party, a PKI, that builds trust among the players, by certifying all the signatures involved in the transaction. SET uses X.509 certificates, produced by a chain whose main elements are a supra-national root CA, which in turn will certify each brand CA (e.g., Visa), and on a third level: card holder CAs; acquirers serving as payment and merchant CAs, and running payment gateways. Unlike all the others', the card holder certificate has its credit card number (primary account number, PAN) blinded by concatenation with a nonce and a fixed sequence, and then hashed. SET uses *selective end-to-end encryption*, such that content may be selectively revealed to parties. SET messages are fairly dense, so we first present the overall picture of a SET payment transaction, in Figure 19.10, comprised of request/response pairs: the buyer initializes the protocol (PInit); and then emits the purchase order (P); the merchant requests authorization (Auth) for the payment; once cleared, the payment is captured (Cap); the card holder may do an optional inquiry (Inq) at any time during the transaction, to find out about its state.

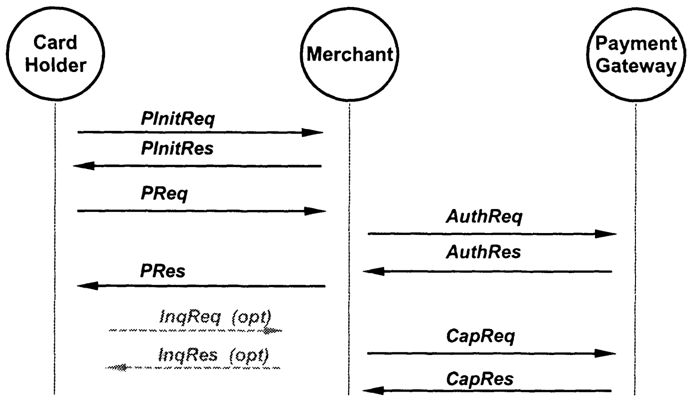


Figure 19.10. SET Payment Transaction

We now analyze the message structure, omitting unnecessary detail. The **PInitReq** message contains: the brand of card, an optional thumbprint or fingerprint (hash) of the certificates held by the card holder, a local transaction Id, and a nonce challenge. In response, the merchant generates a global transaction Id, timestamps it, includes the card holder's challenge and its own, signs everything with the merchant's key, and sends it to the card holder, together with any certificates that the latter might not have yet cached from former transactions, such as the merchant's and the acquirer's.

The card holder believes the merchant is OK when she receives a correct and fresh response (her challenge comes back), and in consequence she issues the Purchase Order (**PReq**). The purchase order comprises two parts: the Order Information (OI) and the Payment Instructions (PI). The OI contains the order description data (OIData) for the merchant, essentially data from the Init phase: global transaction Id, brand Id, the two challenges (to show freshness), and a nonce, to prevent dictionary attacks on the alphabetic contents of the OI, once hashed. The OI is validated with a *dual signature* field, which carries the hashes of both the OIData and PIData. This kind of signature relates the OI with the PI, and it has the property of being verifiable by only revealing one of OIData or PIData (to either the merchant or the acquirer). The card holder certificate goes along. The PI is a credential encrypted with the acquirer's key, containing payment data (PIData) for the acquirer. PI is forwarded by the merchant, who cannot read it. The PI contains the PIData: global transaction Id, amount, actual credit card data (CardData) encrypted with *extra-strong* plain 1024-bit RSA, and a hash of the order description OD. This is validated with the same kind of dual signature as the OI, and the whole (PIData plus signature) is finally encrypted. Sending PI is equivalent to signing the credit card ticket in a conventional transaction.

The merchant now verifies the signature on OI by following the certificate chain in the PKI. The merchant will request authorization and initiate capture. The **PRes** response may be issued at any time after this check. The authorization request (**AuthReq**) carries elements of the transaction (namely a hash of OD and of OIData) and the PI credential, all signed with the merchant's private key and encrypted with the acquirer's public key. If the transaction elements and those inside PI match, the acquirer knows that both the card holder and the merchant agree on the transaction (goods and amount): the dual signature in PI proves the connection of that order to the card holder. The acquirer obtains authorization from the banking network and if all is OK, sends the relevant code in an **AuthRes** message, together with a **Capture Token**, a credential for the merchant to get paid. The merchant will capture payment, eventually merging Capture Tokens of several transactions.

Remember that SET is the payment part of the whole purchase. SET can and should be integrated in broader Web-based electronic commerce applications. The SET consortium provides a SET reference API and implementation to guide implementors. It is free for non-commercial use.

19.6 MANAGING SECURITY ON THE INTERNET

Internet protocols have their *design vulnerabilities*. They improve with time, but will hardly disappear. On the other hand, configuration of a large facility is hard to do without any mistake, leaving *configuration vulnerabilities*. Attacks and intrusions may go un-noticed, if there are many new events arriving. This introduces *detection latency*, that may amplify the effects of an intrusion. These reasons are more than enough to justify an investment in *security management* of any facility. System Management strategies and tactics are discussed with more detail in the Management Part (Part V) of this book, namely how to insert these technologies in a coherent management framework. Amongst the relevant functions, we are concerned with: security enhancement tools; fault diagnosis tools; intrusion detection tools; auditing tools. These functions, as well as tools to perform them, will be detailed in Section 24.7 of that part.

19.7 SUMMARY AND FURTHER READING

This chapter gave examples of systems and platforms for secure computing. The objective of the chapter was to provide the reader with some knowledge about existing products and systems, but above all to relate these systems with the notions learned throughout this Part. We reviewed remote operations and messaging packages, firewall and virtual private network systems, authentication and authorization services, devices and frameworks for secure electronic commerce.

Further reading on Java and Web Security can be found in (McGraw and Felten, 1997; Garfinkel and Spafford, 1997). A thorough discussion on electronic payment systems is done in (Mahony et al., 1997). Cheswick gives a detailed account of firewall and Internet-related security tools (Cheswick and Bellovin, 1997). In (Quinn, 1996), an up-to-date survey is given of tools for UNIX host and networking security. Table 19.2 gives a few pointers to information about some of the systems described in this chapter. Some of the sites are extremely complete repositories of security-related software.

Table 19.2. Pointers to Information about Secure Systems and Platforms

Class of System	*System*	*Pointers*
CERT		www.cert.org
ITU	(ex-CCITT)	www.itu.int
IETF RFCs		www.rfc-editor.org
Remote Operations and Messaging	**SSL**	home.netscape.com/eng/ssl3
	SSLeay	ftp.psy.uq.oz.au/pub/Crypto/SSL
		www.ssleay.org
		www.openssl.org
	SHTTP	www.homeport.org/~adam/shttp.html
	TLS	www.ietf.org/html.charters/tls-charter.html
	SSH	www.ssh.org
		www.uni-karlsruhe.de/~ig25/ssh-faq
	PGP	www.pgpi.com/
	S/Key	ftp.bellcore.com/pub/nmh/skey
		ftp.cerias.purdue.edu/pub/tools/unix/netutils/skey
		www.ietf.org/html.charters/otp-charter.html
	OPIE	ftp.inner.net/pub/opie/opie-2.32.tar.gz
	RSADSI	www.rsa.com
	SECUDE	www.darmstadt.gmd.de/secude
	PEM	ripem.msu.edu
	DCE-RPC	www.opengroup.org/dce
	SUN-ONC	www.sun.com
Firewall Systems	**FW-1**	www.checkpoint.com/products/firewall-1/index.html
	Gauntlet	www.nai.com
	TIS toolkit	ftp.tis.com/pub/firewalls/toolkit
	SOCKS	ftp.cerias.purdue.edu/pub/tools/unix/firewalls/socks
	SSLproxy	www.obdev.at/Products/sslproxy.html
	squid	squid.nlanr.net/Squid
Virtual Private Networks	**VPN-1**	www.checkpoint.com/products/vpn1/index.html
	VPN+	www.datafellows.com
	IPsec	www.ietf.org/html.charters/ipsec-charter.html
		www.antd.nist.gov/antd/html/security.html
	ISAKMP	www.antd.nist.gov/antd/html/security.html
	Stunnel	www.stunnel.org
	Web sec.	www.cs.princeton.edu/sip
Authentication and Authorization Services	**Kerberos**	athena-dist.mit.edu/pub/kerberos
	Radius	www.livingston.com/tech/technotes/500
		ftp.cerias.purdue.edu/pub/tools/unix/netutils/radius
	TACACS	www.cisco.com/warp/public/707/index.shtml
Secure Electronic Commerce and Payment Systems	**Java Card**	www.gemplus.com
	EMV'96	www.mastercard.com/emv
	DigiCash	www.digicash.com
	Mondex	www.mondex.com
	Millicent	www.millicent.digital.com
	SIBS	www.sibs.pt/en/multibanco.html
	SET	www.setco.org/set_specifications.html

20 CASE STUDY: MAKING VP'63 SECURE

This chapter brings our case study one step further: making the VP'63 (VintagePort'63) Large-Scale Information System secure. Increased distribution of the infrastructure through the Internet, combined with remote access of company salespersons dictated this step in the project, in order to address concerns with privacy and integrity of the company's information system. As selling on the Internet becomes attractive, plans are also made for setting-up an electronic commerce server, a major step for a company that did not even have a passive Web presence.

20.1 FIRST STEPS TOWARDS SECURITY

The reader should recall that this is the next step of a project implementing a strategic plan for the modernization of VP'63, started in Chapter 5, and continued in the Case-Study *chapters of each part of this book. The reader may wish to review the previous parts, in order to get in context with the project.*

The team identified the following problem areas with regard to security, deriving from the corporate strategic plan:

- point-to-multipoint payload interconnection flows between the enterprise units, now made through open networks (e.g. Internet);
- point-to-point remote session interconnections between employees and enterprise units, now made through open networks (e.g. POTS, GSM, Internet);

- multipoint-to-point anonymous connections from anywhere on the Internet to the commercial Web site, not only to acquire information, but also to perform electronic transactions.

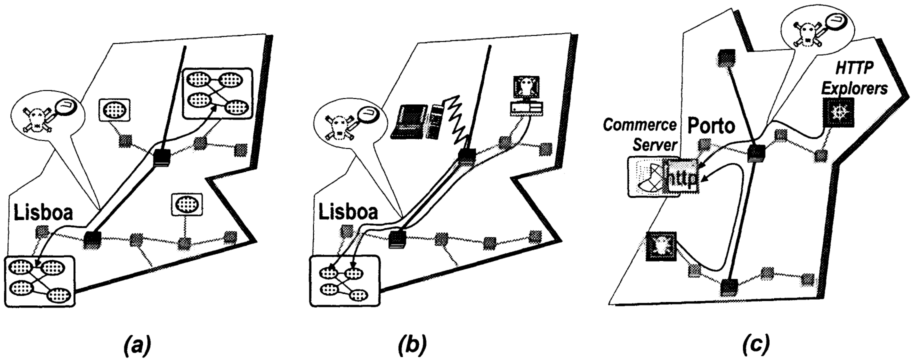


Figure 20.1. Security Problem Areas: (a) Site Interconnection; (b) External Remote Access; (c) Anonymous Transactions

The risk of operation was evaluated for these problem areas. One of the premises of the project is the use of COTS components, with their known vulnerabilities, which can reduced by configuration and/or function elimination. A preliminary abstract analysis of the degree of vulnerability suggested that this presents disadvantages (vulnerabilities do exist) and advantages (they are well-known and fixes exist), but yields a high cost-effectiveness ratio. On the other hand, a preliminary abstract analysis of the level of threat revealed the following:

- Site interconnection (Figure 20.1a)– the payload flow may be subjected to attacks on confidentiality and integrity.
- External remote access (Figure 20.1b)– individual access sessions may fall to intrusion campaigns that compromise the authenticity property, and from then on, the confidentiality and integrity of the internal state of the system.
- Anonymous transactions (Figure 20.1c)– attacks on the commerce server protocols with the attempt of fraud may assume several facets; general and perhaps distributed denial-of-service attacks are also to be feared.

The architectural approach for security will be laid out around: the extranet and the virtual private network of the company over the Internet; the intranet and its firewall gateways to the outside. The team decided that the set-up for business-to-business (B2B) transactions will be deferred to a later phase when the technologies to be installed now are mature. This is because B2B depends both on commerce server and on VPN technologies, which will be developed with separate purposes in this phase.

20.2 GLOBAL SECURITY: EXTRANET AND VPN

Recalling the infrastructure laid out in the first phase of the project, the infrastructure should now evolve to a strict WAN-of-LANs organization, where every facility has a single logical connection point to the Internet, the Facility Gateway. All internet addresses behind the Gateway are invalid, which brings a certain degree of protection to probing (e.g. port scanning) attacks, and on the other hand allows creating a seamless virtual domain that spans all facilities, so that all nodes anywhere in the company's installations are seen as being in a single network. For this to be possible, IP-over-IP tunnels are created between every Gateway and all the others.

Figure 20.2 depicts the big picture of the VP'63 Virtual Private Network (VPN) design, interconnecting all VP'63 facilities over the Internet following this model. In order for the payload traffic to be protected against attacks, the tunnels are secured using link encryption between Gateways.

This set-up can be generalized in several ways under an extranet perspective. To begin with, it solves the problem of remote fixed client-only offices, i.e., the small installations that once used to have a single remote terminal hooked by leased line or dial-up. These offices will establish secure payload tunnels to main facilities in the same way. The other problem are nomadic salesmen or executive notebooks, bound to access the VP'63 network through the Internet or modem dial-up. Functionally, they should desirably have the same kind of direct access *into* the VP'63 intranet as provided by the site interconnection tunnels. However, given the mobility and sporadic character of access, building trust on these connections is more difficult. In consequence, they had better be provided through an external remote access service that establishes a more powerful filtering point at the Gateway, to be addressed upon the detailed Intranet and Gateway design.

20.3 LOCAL SECURITY: INTRANET AND FACILITY GATEWAY

The architecture of the intranet of a facility is shown in Figure 20.3, focusing on the Facility Gateway architecture. Under a security viewpoint, the Gateway must protect the intranet and provide services hosted by internal servers in a secure way. The team has studied the design of the following services: portal passive services (rendering, etc.); portal active services (messaging, search, transactions, etc.); remote access (from the outside); internet navigation and messaging (from the inside).

The Gateway is normally laid out around a two-level or screened-subnet firewall architecture. The minimal functionality a Facility Gateway should provide is the insertion in the VPN infrastructure, and for simple installations (small client offices) that can be ensured by a single host acting as a bastion router. Figure 20.3 depicts the maximal Gateway architecture, in fact a set of hosts, providing all the services foreseen. The outer firewall is the router that provides access to the Internet and implements the secure IP-over-IP protocols (e.g. IPSec). It also implements the NAT (network address translation) that

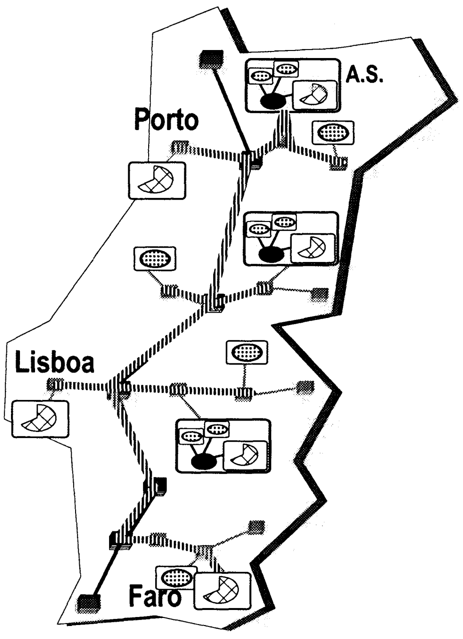


Figure 20.2. Extranet with view of the VPN

hides VP'63 internal addresses. As a firewall, it performs some form of packet filtering, necessarily limited since some of the services on the DMZ are for anonymous access. Still, on the intrusion detection side it is capable of some counter-reaction. The inner firewall is a bastion router, acting as a multi-port firewall to the intranet subnets. Between the outer and inner firewalls lies the DMZ (de-militarized zone), where extranet services are installed.

The passive services of the portal are ensured by a Web (HTTP) server with local storage of static pages for immediate rendering. The Web server is placed on the DMZ, since it serves anonymous accesses. It also acts as the overall portal for all other public access services from the outside. As such, it also provides hooks for active services of the portal: email to the enterprise, but more importantly, it connects to a lightweight transactional server on the intranet. The lightweight server is so called due to the underlying philosophy: hosting fragments or replicas of the global database so that they may have a better performance serving the basic on-line commerce (e-comm) applications, search queries and electronic transactions. Once the architecture in place, new contents and new commerce offers can be readily offered in a scalable way. The performance of this lightweight solution may be fairly high with an adequate configuration and a correct balance of the fragment semantics, between read-only, weak consistency (caching) and strong consistency (active replication). It also provides an indirect way of reconciling operations with the business information system, through the global database, without burdening its own transactional front-end with e-commerce transactions, which have a highly unpredictable behavior and evolution.

The remote access service from the outside is given a low level of trust, as discussed before. As such, a dial-up RAS service is hosted in a DMZ node, with dial-up line/caller authentication. From then on, remote requests, both via dial-up and via the Internet through the outer firewall, are treated equally and directed to a proxy remote session (telnet) RAS server on the inner firewall. Plaintext telnet connections are not allowed under any circumstance, employees will be instructed to have a secure telnet package installed on their portable machines. Roaming access from alien machines will not be allowed. Requests for all the above-mentioned services are authenticated on a need basis on a strong authentication server, placed on the intranet but accessible from any firewall and DMZ services. This offers incremental levels of authentication, e.g., operation-dependent authentication for e-comm transactions. The authentication server hosts private (employee) and semi-private (regular client) credentials, but may also be hooked to existing PKI-CA systems (anonymous users).

The tunnels merely serve to reach another facility, and nowhere else on the Internet, so direct Internet connection from the inside has to be specifically addressed. Internet navigation and email sending are the only outgoing services to be supported at this stage, provided through central outgoing HTTP and email servers located on the main facility. This may have some impact on performance, namely on the Web side, but is otherwise transparent from applications and provides a necessary control point.

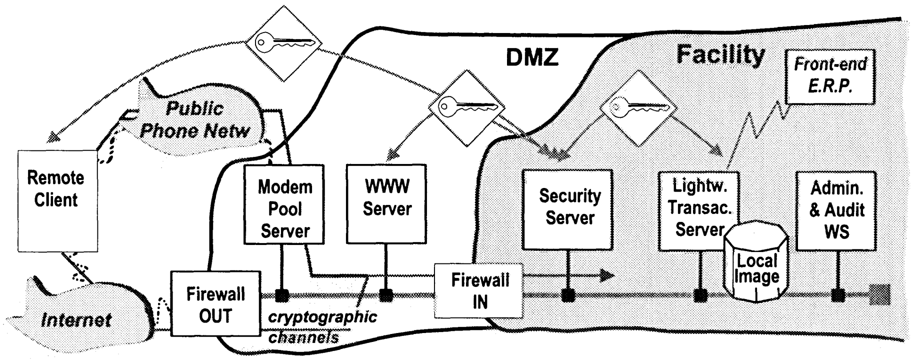


Figure 20.3. Intranet with view of the Portal

Further Issues

These issues need some refinement now, and the reader was assigned the study of a few questions that were still left to be solved:

Q.4. 1 Sketch the routing hops for an IP packet going between nodes in two facilities separated by a secure tunnel.

Q.4. 2 *Propose the detail of the Gateway cryptographic set-up underlined in Figure 20.3 (key types, distribution, and integration in protocols).*

Q.4. 3 *Is there a secure way for employees to access email and news via alien Web browsers while roaming without access to a company machine?*

Q.4. 4 *How should outgoing direct Internet access be decentralized on a per-facility basis, if increased use starts creating a bottleneck on the central HTTP and email servers?*

Q.4. 5 *How can the Web server designed for the portal be improved w.r.t. fault tolerance and load balancing?*

Q.4. 6 *How can the transactional and search engine front-end designed for the portal be improved w.r.t. fault tolerance?*

Q.4. 7 *Discuss other alternatives for remote access and e-commerce based on different balances between threat and vulnerability than those assumed in the present design.*

Q.4. 8 *Denial-of-service attacks may indeed become a concern. Discuss the possible design of some form of availability measures facing such attacks.*

Q.4. 9 *The architecture proposed is essentially an attack prevention one. Discuss the possible design of some form of attack tolerance.*

Q.4. 10 *Delineate a strategy for networking and data security assuming potentially harmful insider users, which have been precluded from the current model.*

V Management

The direct forces fight the enemy on the ground, but the indirect forces ensure victory. Their combinations are infinite.
— Sun Tzu, The Art of War, circa 500 B.C.

Contents

Overview

Part V, Management, addresses the management of distributed systems, that is, the issue of ensuring that distributed systems are configured correctly in order to provide adequate service, and that they remain correctly configured and providing adequate service throughout their life. In the measure that distributed systems technologies achieve maturity and widespread use, management will become one of the most important disciplines in the area. This part introduces the Fundamental Concepts of Management in Chapter 21, and continues in Chapter 22 with the main Paradigms for Distributed Systems Management, such as: managers, managed objects and MIBs, domains, main management functions (e.g., configuration, fault, accounting), and monitoring. Chapter 23 addresses Models of Network and Distributed Systems Management, and Chapter 24 discusses Management Systems and Platforms. In these two chapters the notions of previous chapters are consolidated. Frameworks and strategies for management are discussed, and the relevant models presented: centralized, decentralized and integrated management, domains. Chapter 25 finalizes the case study, this time: managing VP'63.

21 FUNDAMENTAL CONCEPTS OF MANAGEMENT

This chapter discusses the problem of management. The fundamental concepts are presented, and the rationale for configuring and managing systems is discussed. The main architectures for systems management are introduced, in order to be further developed in the following chapters.

21.1 A DEFINITION OF MANAGEMENT

What is management? Systems management is the set of *planning, supervision* and *control* functions of a system, such that it provides the adequate service, as expected by its users. The service is defined upon the *configuration* of the system.

Systems management includes strategic as well as tactical factors. **Strategic** factors are concerned with establishing *management policies*, and planning the system architecture and functionality so that these policies are fulfilled. **Tactical** factors address the measures and mechanisms put in practice to actually fulfill the strategic objectives, and the timely reaction to the varying operating conditions such that the system maintains its functionality.

Why is management necessary? Whereas all previous parts of this book were concerned with conferring architectural, functional or non-functional properties to a system, management is concerned with the global measures that assist in maintaining the whole of these properties through the system's useful life. Systems evolve, and need *planning* and *configuration* in order to adapt to change.

Systems today are interconnected, and in consequence isolated and uncoordinated efforts may have undesirable or even harmful effects. This requires *integrated* approaches, and adequate *tools*. Finally, systems became so complex that manual approaches are obviously insufficient, requiring as much *automation* of functions as possible.

As the quote with which we opened this part metaphorically implies, the combinations between strategy and tactics, organization and technology, planning and reacting, are infinite. But the successful combinations are only a few.

21.1.1 The Management Life Cycle

A management support system works pretty much as a process control system. The "controller" is the *manager* and the "process" is the *managed system*. The "control cycle" is depicted in Figure 21.1. The manager monitors the state of the system by receiving *events* from it and interpreting and processing them under the light of the management policy. The manager controls the system by issuing *control operations* on it. These operations are either dictated by strategic management or issued as a result of the processing of events from the system. That is, the manager either initiates some action, such as installing new routing tables or configuring a new printer, or reacts to an environment change, such as repairing a partitioned network, or performing load balancing on a pool of servers upon detection of overload.

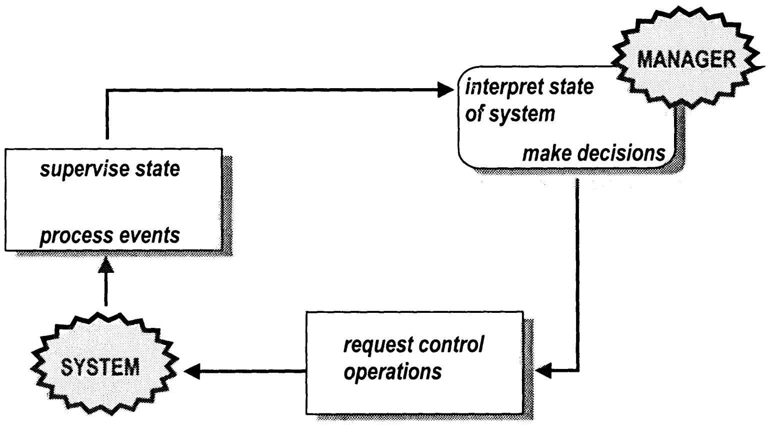


Figure 21.1. Management Life Cycle

A common representation of the flow of information concerned with managing a system (Sloman, 1994) is given in Figure 21.2. The flow starts at strategic level, with long term directives (strategic management policies) and immediate action directives (strategic management decisions) issued to the tactical managers, who *interpret* them. The mission of the managers is to implement strategic decisions in the best possible manner. For that, they issue *control* commands to the system being managed, that act on the *resources.* Some of these commands may consist of polling or sampling the state of the resources. Resources respond to these solicited actions, and may also trigger unsolicited events, or *notifications*, back up to the managers, informing them of changes of state. Managers *monitor* the system through these solicited and unsolicited actions. Some of them require feedback in the form of new commands that close the control loop.

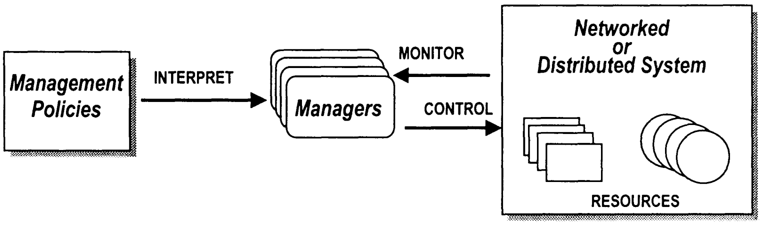


Figure 21.2. Management Information Flow

21.1.2 Organizational vs. Technical Management

In face of what was just said, we should understand that management exists at two levels of abstraction in an organization:

- **organization-level** – dictated by the strategic executives of the organization, not necessarily the field executives
- **technical-level** – performed by the technical executives, or *systems administrators*

In most organizations, the CTO (Chief Technology Officer) is the liaison between the two, since she discusses the strategic issues with her fellow executive managers, and coordinates the systems administration teams. The CTO should enforce the above-mentioned separation of duties. Failure to do so may lead to abnormal and undesired situations, whose extreme examples would be: letting the technical staff acquire knowledge and decision power that belong to the managerial area; putting the technical staff under the direct orders of executives who do not master the technologies.

A key factor of success in management is the adequacy of the system, and of its information and management models, to the models of human thinking and of the organization it serves. Inappropriate models may have been at the root of many a failure of the introduction of informatics[1] in businesses. Two orders of reasons arise when debating this problem. The first is concerned with the mapping between computer-level information and human-level perception.

[1] "Informatics" is a word of european origin getting increased acceptance in the community of computer users and developers. It is used to denote in general terms all that is related with use of computers and networks in information processing, access and manipulation.

Quoting Mintzberg: "Many management information systems (MIS) seem not to be for management at all. They are computer information systems and proceed on the assumption that managers care that the information has been processed by a machine" (Mintzberg, 1989). Computer-generated information is often too limited, too general, too late and too imprecise for human managers to handle adequately. The second is concerned with the mapping of roles in the organization onto the representations allowed by the actual computing model. Think of the following example: why should a senior system administrator— who is a technician and not an executive, let alone the CEO (Chief Executive Officer)— have read and write (or delete...) access to all the information of the company where she works? She normally has it indeed, because of the way most commercial systems work (she has `root access`), but is this a faithful metaphor of that company's business model? Most probably not, that is, she as a middle officer should have neither the power (e.g., to block or destroy the system), nor the knowledge (e.g., of the whole salary policy) indirectly given to her by the way the system is set up.

Can we do something about it? We are persuaded that the answers lie in *systems architecture*, and we hope this book may give a few contributions. From the earlier parts: the mapping of the functional characteristics of businesses onto the functional attributes of technologies; the provision of non-functional attributes to overcome the shortcomings of technologies (dependability, timeliness, security). From this part: the notion that the dichotomy between organizational and technical management levels must be cast into the system architecture; and that this can only be made through the adequate models and tools to handle the information flow between both levels.

21.1.3 Management Support Services and Functions

So what is management in practical terms? Imagine a large enterprise, with facilities in several locations, and thousands of interconnected machines. It is necessary to control how the system is laid-down and configured, draw a map of the existing links, keep directories of registered users and service names. It is also necessary to diagnose, circumvent or repair faults and recover from errors. These faults must include malicious faults that affect security. Performance and quality-of-service guarantees must be preserved for the various services. The utilization of resources by the several users must be accounted for, and so forth. The main management functions arise from these needs:

- configuration management
- fault management
- accounting management
- performance management
- quality-of-service (QoS) management
- security management
- name and directory management

These functions are supported by a few classes of services, such as:

- remote operation execution
- management information storage
- event reporting
- log control

Operations related with the functions above are triggered on remote devices. These operations read and modify the state of management information, whose storage is performed both at the devices and at the managing hosts. The paradigm that supports this concept of management data repository is called **Management Information Base (MIB)**. The MIB holds the data structures concerning the managed resources, and their format is standardized for most architectures. It is through the MIB that most management operations are performed: reading status of resources, writing state variables. The *structure of management information (SMI)* is the collection of specifications of these structures and variables, normally organized in standards.

Event reporting is concerned with the unsolicited notifications (also called up-calls) of resources to the management entities, by which they report unprogrammed events, such as: changes in the environment configuration (new hosts discovered), errors (printer out of paper), failures (a link that is down), and so forth. Events need some processing, sometimes at the source, in order that the destination is not showered with irrelevant or redundant event notifications. *Event discriminators* are special programs that filter and select events according to pre-defined rules (e.g., thresholds), and pre-process them in order that the information that arrives at the upper layers has more elaborate semantic contents than the raw event (counters, rates). For example, a number of error events may be produced as a consequence of a failing network link: several garbled frame transmissions; frames that are completely lost; excess conflict or collisions on the network access. Instead of producing n event reports, a discriminator may send a `failed()` report up as a consequence of integrating those several low-level events in the same window of time (*see Arrival Distributions* in Chapter 12).

It is a normal procedure that events are logged for ulterior analysis. The *log control* is concerned with the conditions in which this log is performed: if the log record is issued after a threshold on a level or on the number of consecutive occurrences is exceeded; if the log itself has a water mark and generates an alert after it is reached; if the log is done at the resource where events were produced, or at a central point (a manager), etc.

21.1.4 Distributed Systems Management

Until now, we have discussed management in general terms. It should be clear however, both from these introductory remarks and from previous parts of the book, that the interesting management is distributed systems management, since current systems have that nature. Distributed systems management (DSM) is concerned with ensuring that distributed applications execute correctly, over a distributed infrastructure that also remains correct. This is important, since it indirectly explains the difference between DSM and NM (network management). In fact, Network Management has been a well-known and established discipline before distributed systems have gained their current momentum. It is concerned with the infrastructure— the computer and telecommunication networks— and with how the information goes back and forth, whereas DSM is concerned with how applications use that information in order to provide services to the users.

Table 21.1. Comparison between NM and DSM Functions

Network Mgt.	Distr. Systems Mgt.
node connectivity	information flow
reaction to partitions	information storage
load/congestion control	system SW integrity
performance tuning	service availability
routing	load balancing

Of course, they are complementary, but it is important to establish a difference, since given the complexity of today's systems, there is room for specialization in either field. For the sake of example, Table 21.1 presents a listing of typical functions of network and distributed systems management.

21.2 SYSTEMS MANAGEMENT ARCHITECTURES

Systems management architectures have evolved during the recent years. There have been several factors behind that evolution, amongst which we stress: the expansion of internetworking brought the need to cope with heterogeneity and domain independence; the expansion of distributed systems created the opportunity to use distributed systems techniques for decentralized but coordinated management. Evolution took place in several stages, expressed by classes of management architectures, which we describe next.

21.2.1 Homogeneous with Centralized Management

The homogeneous with centralized management architectures, depicted in Figure 21.3a, prefigure the situation of the early networks, mostly proprietary, and where little interconnection existed. These systems were small-scale and had by nature a centralized management, implemented by a single management system, and a single console, the station from which management is performed. Management was fairly straightforward, since systems were homogeneous and small.

21.2.2 Heterogeneous with Uncoordinated Management

Figure 21.3b shows how these architectures evolved, when networked systems started to proliferate. Whether homogeneous or heterogeneous, they consisted of gluing together several of the above-mentioned smaller networks, managed locally. In this new scenario, individual networks continued to be managed independently, in an uncoordinated way, despite the fact that they already had some interconnection. The uncoordinated management architectures represent the state of affairs where several network architectures exist, but where loose management is adequate, since they do not require close interaction. Namely, there exist several management systems, and several uncoordinated or very loosely coordinated consoles. This was the status quo just before the advent of large-scale distributed systems.

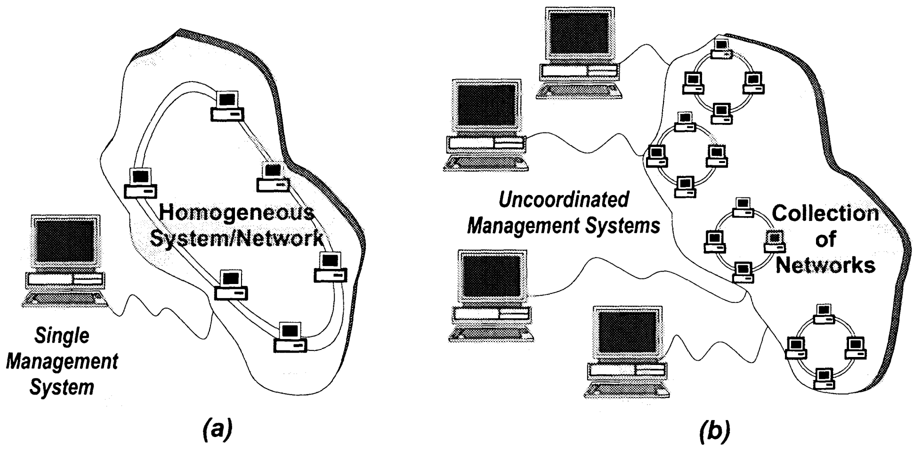


Figure 21.3. Management Architectures: (a) Homogeneous with Centralized Management; (b) Heterogeneous with Uncoordinated Management

21.2.3 Heterogeneous with Coordinated Management

The heterogeneous with coordinated management architectures, illustrated in Figure 21.4a, apply in situations with the same degree of technical evolution as above. However, the several network domains must act together, because the system has a moderate or large scale, and its components have significant interaction— e.g., they belong to or are controlled by a same organization and share significant amounts of data per time unit. In consequence, the management is physically centralized in a control room where all the several management systems and all the consoles are located. Although management is technically heterogeneous (each console runs a proprietary subsystem) it is coordinated at organizational level.

21.2.4 Heterogeneous with Centralized Management

In the meantime large-scale distributed systems made their appearance and the Internet technologies contributed to this evolution. The scale of systems under the control of an organization becomes such that several heterogeneous systems have to coexist under the same managerial umbrella. Besides, those systems have to be managed in a closely coupled way, since they support distributed applications that run across them.

This situation paved the way for the appearance of heterogeneous architectures with centralized management: whilst multiple management systems exist, they are ran from a single console. That is, the organizational-level coordination is mapped onto the technical-level coordination of all the consoles, from a centralized station. We also observe the utilization of distributed systems technologies to help manage the distributed system itself. In this kind of architectures the console centralization, as suggested in Figure 21.4b, is achieved through remote session protocols, such as Telnet, RPC and X-Windows.

The centralized console corresponds to the high-level notion of a physical control or admin center. Nevertheless, the console is location independent: the center can change location and be instantiated elsewhere. Technically, this works by having the physical management console, wherever it is, open windows over each of the management subsystems depicted in the figure. Since each system has its own interface, the heterogeneity of applications is still visible, at least for the more subtle semantic details. However, this is a major step towards integration of management functions, which we discuss next.

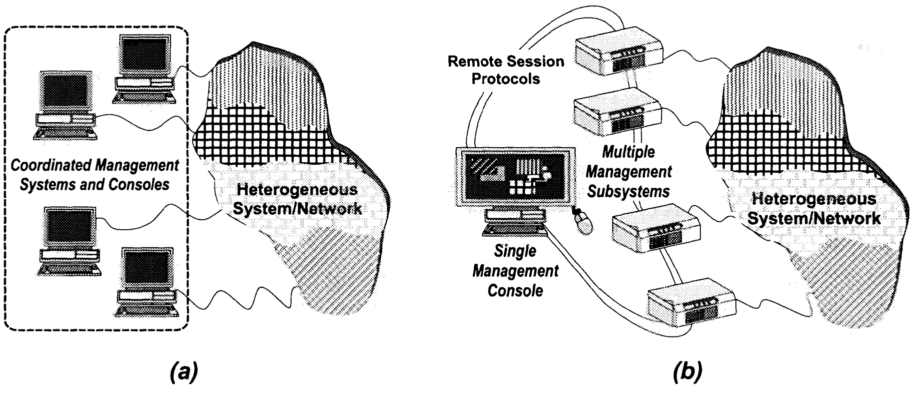

Coordinated Management Systems and Consoles
Heterogeneous System/Network
(a)
Remote Session Protocols
Multiple Management Subsystems
Single Management Console
Heterogeneous System/Network
(b)

Figure 21.4. Management Architectures: (a) Heterogeneous with Coordinated Management; (b) Heterogeneous with Centralized Management

21.2.5 Heterogeneous with Integrated Management

The heterogeneous architecture with integrated management is an advanced distributed systems management architecture, and the state-of-the-art approach, in what concerns available commercial systems. As depicted in Figure 21.5a, the advance with regard to the centralized management architecture is that the multiple management systems have a local scope (they could have local consoles), they can run local management programs by delegation of high-level management.

This architecture is more sophisticated than the previous centralized one, in that it assumes a degree of delegation and in consequence of hierarchy. Organizational-level management coordination is mapped onto a single *integrated management system and console.* The above-mentioned hierarchy also hides the heterogeneity of the local management systems: the applications running on the console address the local subsystems in a homogeneous manner, even if they have different makes. This is done through the utilization of distributed systems technologies, including but going beyond the level of abstraction of remote sessions: remote management protocols and APIs, common GUI interfaces, and sometimes a common database (MIB) representation.

The several systems have thus an apparent homogeneity, since at least their graphical representation and user interface at the central management system are uniform. Standardized protocols establish the dialogue between the integrated management system, and the local— and to a certain extent autonomous— management subsystems. Such as with centralized management architectures, there is location independence: the integrated management system and its console can change location and be instantiated anywhere.

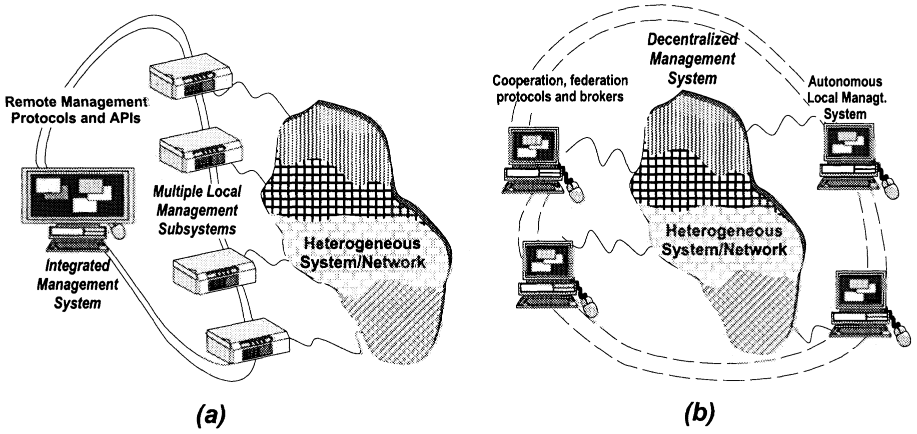

Remote Management Protocols and APIs
Multiple Local Management Subsystems
Integrated Management System
Heterogeneous System/Network
(a)
Decentralized Management System
Cooperation, federation protocols and brokers
Autonomous Local Managt. System
Heterogeneous System/Network
(b)

Figure 21.5. Management Architectures: (a) Heterogeneous with Integrated Management; (b) Heterogeneous with Decentralized Management

21.2.6 *Heterogeneous with Decentralized Management*

Distributed applications are attaining a scope, both in complexity and scale, that renders centralized or integrated management ineffective or even impossible. Observe enterprise networks, business-to-business *e-comm*, or other interactions that are done between realms of distributed systems that belong to different organizations, or even countries. Within the scope of network management, the Internet has been managed under such a perspective: sophisticated routing protocols can make the difference between routing inside what are called *autonomous systems (AS)*, and between different AS's, which are essentially independent management realms.

However, large-scale distributed systems present more complex problems than just communication. This evolution requires a sort of federated management, in essence a heterogeneous architecture with decentralized management, as exemplified in Figure 21.5b. Management is fully decentralized. The organizational-level management is supposedly cooperative, and technical management is performed within the scope of each autonomous local management system, which may itself follow an integrated management approach, with its console and local applications. The effective construction of these architectures relies on distributed system paradigms such as: request brokers, message buses and enabling protocols and algorithms for federation and cooperation.

21.3 CONFIGURATION OF DISTRIBUTED SYSTEMS

A great deal of the success in operating and managing a distributed system depends on whether it is configured adequately. The use of structured configuration methods, besides allowing to build better systems, makes them more manageable. Configuring distributed systems architectures has two major steps: the hardware architecture; and the software architecture.

Configuring the *hardware architecture* begins with selecting the components: routers, hubs, links, and so forth, for the network infrastructure; servers, to install the services; and workstations, for the users to access the system. Then, placing and interconnecting these components, through physical links. That forms the infrastructure, and is concerned with defining the network layout and placing the hosts or nodes. These processes are iterative, and it is normal to come back to hardware configuration after having a first pass at software configuration.

Configuring the *software architecture* consists of selecting the software components: drivers and protocols; service modules; client modules when applicable. Then, placing and interconnecting these modules. This latter part is concerned with activities such as: placing services with servers, following rationale such as load balancing, proximity, or criticality; interconnecting protocols with higher-layer services, such as binding application modules to communication protocols; interconnecting multi-tier services, such as binding a web server to a web request broker and finally to a database engine. Part of the binding is dynamic, during runtime, making use of previously configured services such as name or directory services, and brokers or traders (ANSA, 1990).

Components should be modular, encapsulated, and have a well-defined interface, where both the services required by the component, and the services supplied by it are represented (*see* Figure 21.6a, for a representation based on the model of (Magee et al., 1993)). Note that this is concerned with a generic definition of systems architecture, and not necessarily related with a *client-*

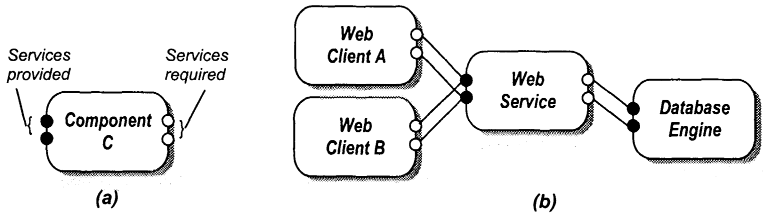


Figure 21.6. Configuration of Distributed Systems

server relation. The interface definition should be precise and non-ambiguous. This suggests the use of an object oriented approach, and of interface specification languages **(ISL)** or interface definition languages **(IDL)**. These languages may be further augmented by graphical configuration methods, where software architectures can be defined by interconnecting software components graphically, sometimes in more than one level. Figure 21.6b shows one such example of configuration, where two web clients (browsers) "require", and are thus connected to (by instantiating the adequate protocols), a web service (HTTP server), which in turn "requires" the provision of the database engine service (possibly through a CGI) to supply the material with which to build the dynamic pages needed for the web server to provide the service requested by the web clients.

System configuration may be static or dynamic. *Static configuration* assumes that whatever the initial configuration is, it will remain stable during the system lifetime, such as in Conic or Durra (Magee et al., 1989; Barbacci et al., 1993). It is through *dynamic configuration* that the desirable relationship with distributed systems management is established. In fact, the idea is that systems change, in the course of operational events such as faults or of mere evolution, and it should be possible to accommodate that change without pain, that is, without stopping the system and going back to the design desk and testing laboratory. Incorporating the ability to support operational changes in the system configuration requires foreseeing those changes, for example by defining several operational modes, and instantiating the relevant components when needed. Incorporating evolution can be achieved by expressing the system configuration in the form of a configuration database, and defining the initial system configuration in a non-declarative language, such that the configuration may change during the system lifetime in a non-programmed way, such as done in Darwin (Magee et al., 1993) or Olan (Bellissard et al., 1996). Clipper (Agnew et al., 1994) and Evolution (Radestock and Eisenbach, 1996) are other examples of dynamically reconfigurable configuration languages.

21.4 SUMMARY AND FURTHER READING

In this chapter we debated the introductory notions concerning configuration and management of distributed systems. After defining management and discussing its lifecycle, we presented management in practical terms, by introducing the main functions and support services. We presented the several architectures for distributed systems management, ranging between homogeneity and heterogeneity, and between centralization, decentralization, integration and autonomy. We finalized by making an introductory discussion of the problem of distributed systems configuration. In (Sloman, 1994), a good introduction to systems management, and distributed systems management in particular, can be found, where some of the discussed issues are further detailed. A distributed programming environment based on 'configuration programming' is presented in (Magee et al., 1994).

22 PARADIGMS FOR DISTRIBUTED SYSTEMS MANAGEMENT

This chapter addresses the main paradigms for distributed systems management. Management models have been developed in the past few years, mainly in the course of standardization activities, such as OSI Systems Management, the Internet Engineering Task Force, or the Open Distributed Processing initiative, but also under significant research effort. As these models have matured, a number of significant paradigms have been retained, and made it possible to define the generic body of research and technology of today's systems management. We will make a non-exhaustive effort to study the main paradigms, and in consequence, we will address: managers and managed objects, domains, management information bases, and the several management functions— configuration, faults, performance and QoS, accounting, security, names and directories.

22.1 MANAGERS AND MANAGED OBJECTS

Management is about *managers*, the performers of the act of managing, and the targets of the act, the *managed objects*. The manager executes management functions, by requesting operations on the objects managed by it, and by receiving notifications from those objects.

Managed objects are a form of uniformly modeling whatever is manageable or needs to be managed. We talk of objects in a broad sense, which obviously intends to capture some of the interesting properties of genuine objects: the

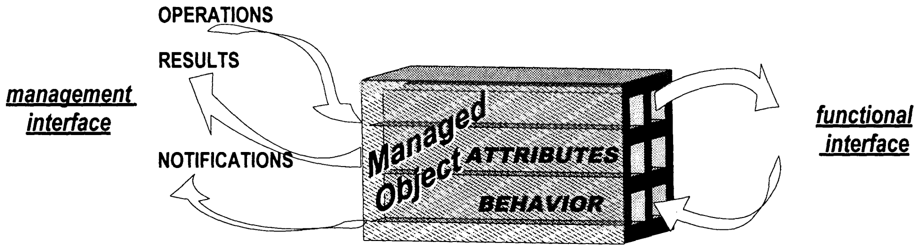


Figure 22.1. Modeling a Managed Object

encapsulation, the well-defined interface. Managed objects have a certain functional encapsulation. Sometimes, a natural one, around a hardware module such as an ethernet adapter board, sometimes one that has to be extracted from a bundled subsystem, such as a disk inside a computer. Besides their functional interface, they also have a **management interface**. Likewise, this interface is not always a natural one, since some objects are hardware units, and the interface has to be implemented by software structures outside them, in some controlling station that represents their state in the best manner possible.

This model of a managed object is represented in Figure 22.1, where we can see the separation between interfaces. The object itself is characterized by its behavior, represented by actions invoked at the interfaces, and its attributes. The management interface allows three kinds of **actions**:

operations:	requests invoked on the object, either to read its state, or to modify it
action req.	to perform an action on an object, such as writing a variable (attribute), open a port, reset a connection
information req.	to read the state of an attribute, e.g., the state of error counters, or of a network printer
results	issued back to the caller by the managed object, in response to invoked operations
notifications	unsolicited information sent by the managed object to the manager, about events observed by the object

The **attributes** are the object properties visible from the outside, and have a value that depends on their nature. Some attributes are writable, others are read-only (e.g. counters externally available, but updated internally by the object), others are constants (e.g., identifications, network board addresses). Part of the operations on objects are primitive operations on attributes: `get()` or `read()`, to obtain attribute values; `replace()` or `write()` to change the attribute value. Complex actions, such as combined actions on object attributes, are represented by a template of the form `application_specific()`. Attributes are *never* accessed directly, but through the management interface, so that the appropriate validations can be made (e.g., bounds checking, ac-

```
LaserPrinter MANAGED OBJECT CLASS
DERIVED FROM ...  (existing top class)
CHARACTERIZED BY:
  BEHAVIOR
  ATTRIBUTES
      PrinterID   GET,
      MACaddress  GET,
      IPaddress   GET,
      OnLine      GET,
      TonerLevel  GET,
      CurrentTray GET,
      DoubleSide  GET,
      TimeSincePrinterReset GET,
      .........
    ACTIONS
      OnOffLinePrinterToggleAction,
      ResetPrinterAction,
      .........
     NOTIFICATIONS
      MaintenanceRequired,
      OutOfPaper,
      OutOfToner,
      .........
NAME BINDING
SUBORDINATE OBJECT CLASS  LaserPrinter;
NAMED BY ...    (existing superior class)
WITH ATTRIBUTE            PrinterID;
.........
```

Figure 22.2. Example of Managed Object: a Network Laser Printer

cess control, etc.). Other general actions on objects are: `create()`, which creates an instance of the object; `delete()`, which destroys an instance; and `action()`, which encapsulates one of several management functions (e.g., configuration, fault, performance/QoS, accounting, or security). The object can in turn invoke a `notification()`, or up-call, to the manager, to inform it of some relevant event (e.g., buffer overflow, printer out of paper). An example description of a networked laser printer as a managed object is given in Figure 22.2. The attributes, actions and notifications are defined in the description. For example, if `get(OnLine)` returns `true`, and it is followed by `action(OnOffLinePrinterToggleAction)`, the printer will go off-line.

22.2 DOMAINS

Management domains are groupings of objects for the purpose of establishing management policies. They are an important paradigm, firstly as a vehicle for mapping management policies to mechanisms, secondly as a means for easily executing management operations on groups of objects. The following are examples of domains: workstations on a same LAN, managed locally; the set of servers that form a distributed file system; the group of internal routers of an organization facility.

Domains allow a set of operations, such as: `include()` and `remove()`, which add or remove an object ID to a domain; `list()`, which lists all objects in a domain; `move()` is built by doing `include()` in new domain and `remove()` from old domain; `create()` and `delete()` operations, which we saw earlier, are in fact done under the scope of domains, if the system supports them. Why? If an object existed outside any domain, there would be no management policy for it, and in consequence it could not be managed.

Domains have another advantage: by policy, restrictions can be established to the operations on a domain, and certain guarantees on the correctness of management operations can be automated. For example: a domain of Intel machines can be set-up so that only iAPX-compatible binaries may be installed on the objects of that domain; a domain whose objects are compatible with a given management protocol can disallow the insertion of objects that are not. Inside a domain, sub-sets of objects may be addressed by regular expressions, for example ⟨*all workstations of domain LAB1 with Linux O.S.*⟩.

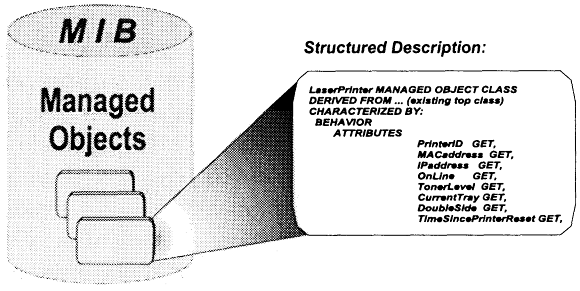
22.3 MANAGEMENT INFORMATION BASE

The management information base *(MIB)* paradigm is the conceptual repository of the management information. It should contain the descriptions of all the resources relevant to the system management, such as:

- network components: hosts, gateways, routers,...
- protocol entities: TCP, XNS, IP, X.25,...
- dynamic objects: TCP connections, secure tunnels,...
- administrative objects: trouble tickets, user records, installation procedures,...
- auxiliary objects: schedules, event records, filters,...
- application objects: services, algorithms,...
- object attributes: error counters, flags, water marks,...

After having introduced the managed object paradigm, it is obvious that the MIB must be constructed in terms of the former, as depicted in Figure 22.3. The description of the objects should be done in a common, standardized, and structured way, for example as shown back in Figure 22.2. Note that it is desirable that access to the MIB is automated, as much as possible. MIBs may have a very large number of objects, and management code elements may be reused, or applied to groups of objects, belonging to devices that are possibly of different brands and makes. This is only possible if the description of the objects is precise and has a well-determined content. Structured languages exist to achieve these objectives, such as ASN.1 (Abstract Syntax Notation) (ASN.1, 1990).

Technically, the MIB may assume several forms. It is normally distributed, scattered through several devices. Management platforms may have more complex MIBs, possibly under the control of a database manager, that hold caches or copies of managed objects residing elsewhere. In terms of behavior, a MIB captures much of the notions we studied in the Real-Time Part of this book, of active and real-time databases (*see* Chapter 13): the information stored has temporal validity; changes in the database may cause triggers to be produced.



```
LaserPrinter MANAGED OBJECT CLASS
DERIVED FROM ... (existing top class)
CHARACTERIZED BY:
 BEHAVIOR
    ATTRIBUTES
                 PrinterID   GET,
                 MACaddress  GET,
                 IPaddress   GET,
                 OnLine      GET,
                 TonerLevel  GET,
                 CurrentTray GET,
                 DoubleSide  GET,
                 TimeSincePrinterReset GET,
```

Figure 22.3. The Management Information Base (MIB)

22.4 MANAGEMENT FUNCTIONS

In this section we will introduce the main management functions, some of which will be detailed in the subsequent sections: configuration management, fault management, performance and QoS management, accounting management, security management, name and directory management. Some of these will deserve a detailed discussion in the sections to follow.

Configuration management is concerned with analyzing and maintaining the configuration of a network or system. It is normal that this information is graphically displayed. Typical functions of configuration management are: automated discovery of network topology; automated update of configuration; remote configuration and reconfiguration.

Fault management is concerned with detecting and solving error situations. We have studied the fundamental concepts and mechanisms relevant to fault tolerance in Part II of this book. Let us put the subject in context with management. Faults give origin to errors, which are reported in alarms. Alarms may be divided in a few broad classes: communication, software, hardware, environment, and non-functional attributes such as QoS or security. The alarm may further carry a few useful parameters, such as: probable cause (e.g., the physical fault causing the error); severity (the potential for causing damage); origin (location of the object causing or suffering the error); and optional information about a possible exceed threshold (e.g., omission degree, that is, the number of allowed omission errors during an interval). After the alarm, comes the solution: reconfiguring the system under operator intervention, repairing the fault that caused the error, or tolerating the fault with redundancy. Fault diagnosis, either by routine or after service is re-established, is desirable: connectivity testing, data and protocol engine integrity, link integrity (loop-back and echo), self-test. Typical functions of fault management are: monitoring the system to detect alarms and perform preventive fault diagnosis; alarm processing; error confinement and recovery; interfacing with operator and user assistant tools, such as trouble ticket and help desk facilities; audit trail, i.e. logging alarms for ulterior analysis and fault diagnosis.

Two related functions, **Performance** and **QoS** management, address the problem of maintaining the performance of the system within specified limits. QoS management is a more precise measure of the non-functional attributes of a distributed system. Performance management is more concerned with network metrics: speed, throughput and latency. It may include performance as well as fault parameters, such as the allowed omission degree or error rates. Typical functions of performance/QoS management are: general QoS parameter measurement; monitoring of the system for detection of QoS violations, such as network congestion, host overload, unusual delays; execution of specific measurements, calculation of statistics and production of reports; medium-term capacity planning, such as latency and throughput.

Accounting management has some analogies to performance management, but its main functions focus on user metrics, such as: collection of utilization data; construction of statistics; resource usage accounting; allocation of resource usage quotas and costing. Such as with performance management, there are special objects devoted to accounting management. The *meter control* and *meter data* objects respectively control the acquisition and storage of accounting data, such as kilobytes of bandwidth, megabytes of disk storage, CPU seconds, etc. One meter control object may control several meter data objects. The *usage record* objects hold the relevant records of utilization per user, that will be used to produce for example per user accounting reports.

We have already dedicated quite a few pages to security in this book. Talking about **Security** management is talking about applying those notions in the context of management. Part of the strategic management policy concerns security, and tactical management with this regard concerns the necessary measures for the implementation of the security policy. Typical functions of security management are thus: intrusion detection; authentication; protection measures; contingency plans for security hazards, such as intrusion countermeasures. Security management is mostly related with how to do the mapping or establish the correspondence between *security policy*, and *security measures.*

Name and **Directory management** are two sides of a same coin. We have already discussed the need for name and directory services in Part I of this book (*see Name and Directory Services* in Chapter 4). Name and directory services bear an obvious relationship with management, since in order to manage objects, we need information about them, such as how to name them and their whereabouts.

22.5 CONFIGURATION MANAGEMENT

The basis of configuration management is that objects make available configuration information, in the form of attributes. The description of managed objects relevant to configuration management is made in terms of three classes of attributes:

- **state** - referring to the global state of the object, such as if it is up or down;

- **status** - referring to detailed state variables internal to the object, such as whether a certain functionality is available, or a certain alarm was issued;
- **relation** - referring to the relations among objects, such as if the object is a replica or back-up of another.

The object state variables may for example be (Langsford, 1994): *operational*– whether the object is `disabled` or `enabled`; *utilization*– whether the object is normally loaded (`active`), heavily loaded (`busy`), or free (`idle`); *administrative*– under operator intervention (`locked`), or in normal operation (`unlocked`). All these variables but the *administrative* are read-only. The status attributes denote the object state variables visible from the outside, for example the `OnLine` variable, or the `OutOfPaper` alarm. The relation attributes may denote that the object belongs to a `Group`, or that it is the spare copy `StandByOf` another object.

Planning is related with configuration management, in the sense that it precedes configuration. As a matter of fact, they alternate, as the system evolves and needs to respond to new challenges. Planning is a strategic action, and it is very important for network or system management. In fact, we have already mentioned that the system must have the capacity of adapting to changing situations, of having a certain dynamics, of complying with QoS specifications. This takes place on-line, while the system is running, it is part of the tactical abilities of the system to respond to new situations. However, none of this can be achieved if the system has not been configured adequately. Strategic planning concerns the phase *before* the system is configured, or intermediate phases where it is necessary to reconfigure the system in order that it can respond to significant modifications of the operational conditions. It consists of taking into account all the requirements that the system will be faced with and study configuration alternatives, playing with hardware and software architecture, after which a configuration plan is prepared and executed, sometimes step-by-step, to ensure painless commissioning.

Sometimes, a network or distributed system is already in place when a management system or platform is initialized. In this particular situation, the management platform must *learn* the configuration of the system being managed. It can be loaded manually, or the platform can find it by itself, through **automated discovery**, an interesting paradigm of configuration management. Automated discovery consists of capturing information from the environment, in order to discover, locate, and collect information about devices, and their managed objects. Automated discovery is straightforward when all system devices respond to the protocols used by the platform to perform its discovery campaign. When that does not happen, an effort must be made to complement it (Norton, 1994; Siamwalla et al., 1999). Protocols at different layers may contribute to discovery. For example, a router may discover what is at the end of each link it is connected to, and share this information with the other routers, creating a topology map of the network. Any host (e.g., a bridge) can learn about the hosts located in the LAN segment(s) it is connected to, by logging the MAC source addresses it sees passing. These are examples of *passive dis-*

covery. In complement, there may be *active discovery* actions, where devices purposely scan for new resources. In practical systems, what happens is that the platform discovers a very large percentage of devices passively, complements that through active discovery, and the information about configuration ends-up being completed and tuned manually.

22.6 PERFORMANCE AND QOS MANAGEMENT

There are special objects devoted to performance/QoS management. These *metric objects* serve to collect statistical information about the system evolution, and have methods that compute statistical functions about the utilization of resources, quality of the infrastructure, and so forth. These methods can be simple averaging functions such as computing the mean of the last n samples of an attribute, or more sophisticated ones that avoid storing $n-1$ samples, such as an exponentially weighted mean (Sloman, 1994).

QoS management has become increasingly important as QoS architectures proliferate, for example in areas such as distributed multimedia. As a matter of fact, it should be seen as a generalization of performance management, that is, we could talk only about QoS management. Its goal is to ensure that a *QoS specification* for a service remains within the specified parameters for the duration of service provision. Recall that a **QoS specification** consists of a list of dimensions, and values and intervals for them (*see Quality-of-Service Models* in Chapter 13). For example:

$$\langle \texttt{throughput} > 1Mb/s;\ 100\mu s < \texttt{latency} < 50ms;\ \texttt{BER} < 10^{-5} \rangle$$

Once contracted, a QoS specification may be violated both by the provider or by the contractor. In fact, ensuring a given QoS at the start of an application is not a guarantee that the QoS will hold for the application lifetime. The problem is to *maintain* the QoS of *individual* activities, supplied by resources *shared* by other activities, in the presence of changes in the *operating conditions* of the infrastructure. One part of the solution to the problem is *monitoring* the infrastructure, to ensure that the QoS remains within the parameters, and detect deviations early enough. This management function is assisted by QoS failure detectors. Monitoring may indicate the systematic (statistically meaningful) failure of a QoS parameter. This is reported up to the support middleware or to the application, with an event notification. There are several possible responses to this event: termination; renegotiation; adaptation. Termination is the outcome when the application is not capable of operating with a lesser QoS. Renegotiation takes place as an attempt of the application to contract a similar QoS, sometimes with a different combination of parameters, or a different network support. Adaptation occurs when the application is elastic enough to withstand the reduction in QoS, an adapt to the new operating characteristics (*see* again *Quality-of-Service Models* in Chapter 13). The other facet of maintaining QoS is *policing* the user activity to ensure that the QoS requested at every moment is within the contracted QoS. For example, the user might be grabbing more bandwidth than it had contracted, possibly jeopardizing the QoS of other applications.

22.7 NAME AND DIRECTORY MANAGEMENT

Recall that a name service allows the identification and location of users and services without requiring previous knowledge of their whereabouts. Directory services have a broader scope than their name service counterparts. They hold more information about subjects than just the location.

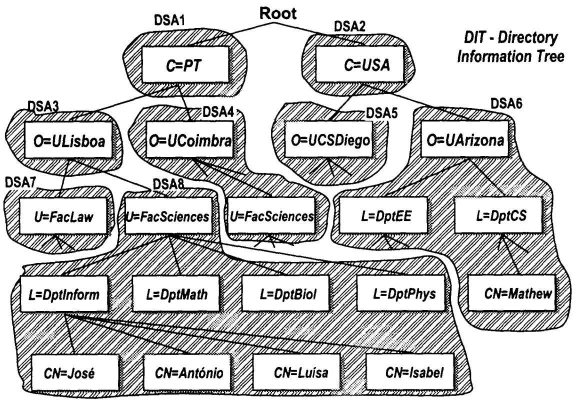


Figure 22.4. Managing X.500 Information

From the management viewpoint, a directory service, more than a name service, offers crucial support. We refer to *Name and Directory Services* in Chapter 4) to show how object names and other information can be managed in a large-scale infrastructure under X.500, as depicted in Figure 22.4. In order to manage an object, we need to have some information about it. This information is stored in the *Directory Information Tree (DIT)*, managed by several *Directory System Agents (DSA)* hierarchically. DSAs are the managers of the entities of the subsystem they subtend (shaded areas). A name can be looked up by traversing the DIT following the distinguished name fields down to the final leaf, the common name of the object or subject. Once the name obtained, all the relevant information about the corresponding object can also be retrieved.

22.8 MONITORING

Monitoring is a paradigm representing the set of activities aiming at knowing the state and evolution of the system during its operation. It is the basis of most tactical management functions (the reader will perhaps remember that we mentioned the word 'monitoring' several times during our discussion of other paradigms in this chapter). As a matter of fact, tactical management relies on the reactive system principle: the loop of monitoring (state acquisition) and control (state modification).

Acquisition is performed through *sensors*, which may be implemented either in hardware (e.g., power supply voltage drop sensor), or in software (e.g.,

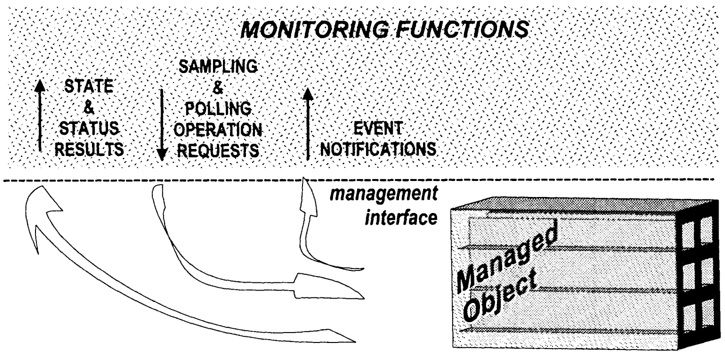


Figure 22.5. Monitoring

boolean state transition detector). Sensors are incorporated in the managed object concept, and monitoring is characterized by interactions that extract state snapshots and collect events from managed objects, as presented in Figure 22.5. The monitoring subsystem must acquire information both by polling or sampling the managed system state (solicited operations to the managed object) and by capturing all the events produced by the managed system (unsolicited notifications from the managed object). In this matter, monitoring incorporates notions on input-output sensing and actuating, studied in the Real-Time Part of this book (*see* Chapter 12).

22.9 SUMMARY AND FURTHER READING

This chapter addressed the main paradigms available to the systems architect in order to implement the management models that we discuss next. Namely, we discussed managers and managed objects as the fundamental actors, the role of domains in structuring management operations, the MIB (management information base) as the conceptual central repository and representative of the structure and state of the resources, and the main management functions, of which we followed with detail: configuration, performance and QoS, name and directories, and monitoring. For a deeper study on the paradigms debated in this chapter, see (Hayes, 1993; Sloman, 1994). Namely, Hutchison et al. do a treatment on quality of service management, Zatti addresses name and directory management. See also a CORBA-based QoS management framework in (Hong et al., 1999). Further material on fault detection and alarm correlation can be found in (Ricciulli and Shacham, 1997; Lewis and Dreo, 1993; Hood and Ji, 1996). An application of the domains paradigm to role-based enterprise management can be found in (Lupu and Sloman, 1997). A study on information push and pull paradigms applied to web-based management is done in (Martin-Flatin, 1999). In (Crane et al., 1995; Fosså, 1997), object-oriented and graphical configuration management environments based on the Darwin (Magee et al., 1993) language are presented.

23 MODELS OF NETWORK AND DISTRIBUTED SYSTEMS MANAGEMENT

This chapter aims at giving the reader a global view of the problem of management. After studying several paradigms, we see how they fit in several models for management of networks and distributed systems. We start by presenting management frameworks: functional, such as configuration, management, monitoring; and structural, such as the tool and platform levels. Then we discuss the strategic alternatives the architect is faced with, clarifying the difference between strategy and tactics, and the subtleties between distribution, centralization and decentralization. Finally, specific models for distributed systems management are presented.

23.1 MANAGEMENT FRAMEWORKS

Network or Distributed Systems Management? It is important to emphasize the difference between network and distributed systems management again in this context. The framework of network management is focused on communication, and is the more mature branch of management. Network management (NM), or the telecommunications management network (TMN) are of course adequate frameworks for telecom or computer network operators. Distributed systems management (DSM) has the broader scope of system support and application processing, in addition to networking support and communication. In our opinion, it is very advisable for an enterprise to talk about DSM whenever possible, despite eventually having a considerable networking infrastructure. The end purpose of most networking facilities we see today is the support of distributed applications for organizations and for public use. DSM is the adequate framework for that level of abstraction, whereas NM is the framework to be considered by whoever is providing the infrastructure, be it network operators, or the organization itself, or both.

Configuration Together with planning and development, configuration, that we addressed in Section 21.3, materializes the framework of preparation of the hardware and software architecture of the system. This involves selecting the hardware and software components, and then placing and interconnecting them. We also discussed structured configuration methods in order to improve manageability of the system. In essence, successful configuration depends on how well a few desirable characteristics are mastered:

- graphical programming— easy placement and interconnection
- well-defined components— modularity and encapsulation
- modeling of behavior and interactions— precise interface specification

Management Understood as the set of activities concerned with ensuring the correct operation of the system while at work, management is concerned with tactical issues, also called short-term management, as opposed to strategic or long-term management. In Section 22.4 we saw that the main management functions are: configuration management, fault management, performance and QoS management, accounting management, security management, name and directory management. These management functions are performed by managers, as we saw in Section 22.1. They operate on managed objects, either directly, or through agents that establish a hierarchy, or through proxies that implement gateway functions, as we will see in Section 23.3. The management information model is materialized by the Management Information Base, or MIB. The MIB, introduced in Section 22.3, is the conceptual repository of all the information relevant to management. We will see that there are several standard MIB formats in Sections 24.1 and 24.2.

Monitoring Introduced in Section 22.8, monitoring is the framework of real-time supervision of the state and of the evolution of the system. It assists most of the management functions. Note that management functions are structured around the MIB, and a good part of the content of the latter refers to time-sensitive information that resides on the managed objects (e.g., error counters, number of open connections, used memory). Besides, a great deal of on-line or tactical management operation is event-triggered, that is, managed objects send information up (notifications) that need attention, processing and timely reaction. In consequence, we need a monitoring subsystem that follows these events and extracts state information in real-time: periodically for state samples, and upon their occurrence for events. We are going to present a complete monitoring model in Section 23.9 that foresees data processing capabilities, dissemination to peer managers, and graphical presentation facilities. The reader will note a few keywords relevant to monitoring that have been introduced in the Real-Time part of this book: timeliness; sensing and actuating; event-triggered.

The Tool Level Structurally, management and monitoring functions are performed by special applications that we call management tools. Tools specialize in one or a few functions and are normally installed at the manager location. For example, we can have a configuration management tool, or a security management tool. The console from where the tool is run is the operator's console for those functions. We will discuss tools in Section 24.4.

The Platform Level Obviously, a fully-fledged management facility requires a comprehensive set of management and monitoring functions. A platform is a set of tools integrated in a single package, such that the different tools share common information acquired from the managed objects, and have similar, if not common, interfaces. Platforms can be distributed, but they can be controlled from a single location, where the management console is instantiated. We will discuss platforms in Section 24.5.

23.2 STRATEGIES FOR DISTRIBUTED SYSTEMS MANAGEMENT

Strategic Management Strategic management has a long-term view of the system. It starts with system planning and configuration. Strategic planning aims at replying to requirements and casting them into the configuration of the system. Once the system configured, strategic management is about establishing management policies, that is, preparing for the tactical management of the system. Systems should not be managed ad hoc. Worse than not having a management platform is installing such a platform without knowing precisely what to do with it. Strategic management decisions are things such as deciding about restrictions of access and space quota to the public FTP directory per class of outside and inside user, or defining the backup policy with regard to periodicity and completeness, per class of service and user.

Tactical Management Tactical management is the real-time, reactive, and short-term part of systems management. Supposedly educated by a management policy, tactical management concerns the execution of several management functions, processing information obtained by monitoring. Tactical management is a good part of the management activity, and most of the actions are concerned with responding to unforeseen situations, raised by events and changes in state variables. Tactical management decisions are for example establishing disk quotas for the FTP directory given an actual disk capacity, increasing the disk quota upon the organization of an event by the institution, or implementing an automated hierarchical and periodic backup procedure.

Distributed Management or Distributed Systems Management? Distributed management is about managing systems in a distributed way. Distributed systems management is about managing distributed systems. Obviously, distributed managing of distributed systems seems a good idea, but we are really talking about different things. Note that management is classically centralized, in the person of the *manager*, who runs a *management console*, from where she controls the system. This characteristic *does not need* to change because a system is distributed. In fact, networks have a great geographical dispersion, and they have been managed in a non-distributed way for years. On the other hand, a great deal of the existing distributed platforms follow paradigms that aim at achieving distribution transparency, that is, hiding distribution from the users, and making the system look as a huge single virtual machine. It may even be a good strategy for small-scale homogeneous systems.

But that characteristic *should* change when the system is not only distributed but is also heterogeneous, has geographical dispersion and is possibly large-scale. As we noted in Section 21.2, the fact is that the proliferation of heterogeneous and large-scale distributed systems and networks made it impossible to do centralized management of these complex infrastructures in a non-distributed way. Distributed protocols and algorithms are necessary to: handle the correct execution of remote operations as if they were executed from the central console, perform manager-initiated group operations and information dissemination reliably, manage replicated information, and so forth.

Centralizing or Decentralizing? The last paragraph prefigures a strategy that favors the use of distributed *techniques* in support of centralized management, as an artifact of the solution to problems such as heterogeneity, scale, unreliability, inconsistency, etc. What is called integrated management takes the use of distributed techniques even further, in order to run the management protocols and applications themselves as distributed protocols and applications, albeit from a central locus of control. It is a centralized management strategy where transparency is achieved as much as possible through the use of sometimes sophisticated distribution techniques. However, note that this attitude maintains the classical *manager-centric* view of management. Alternatively, we may follow a strategy in favor of *decentralized* management of a distributed system. In this case, distribution appears as a natural consequence of the autonomy of the local systems. The same techniques that have recently been used to support autonomous and cooperating distributed applications may be used for these management systems, such as request brokers, message buses, and so forth (*see* Chapter 4). The several models for centralized, decentralized and integrated management will be addressed in Sections 23.4, 23.5, and 23.6.

23.3 A GENERIC MANAGEMENT MODEL

A generic management model is presented in Figure 23.1, in terms of which all specific models that we encounter can be described. The manager is the entity that executes management functions on, and collects information from, the managed objects, either directly or through assisting components. In fact, managers may access managed objects in one of three ways:

- directly;
- through agents in which they delegate direct access to objects;
- through proxies, to access alien resources.

Managers typically reside in the management console host, whereas agents and proxies normally reside in managed hosts, and control a set of devices, for example those of the host itself, or those of a LAN segment where the host resides. The interactions shown in dashed lines between manager and agent are in fact performed by **management communication protocols**. The same happens with agents and managed objects when residing in different sites, or with managers and managed objects when accessed directly. These protocols are either proprietary or standardized, though the trend is for a standardization of all management access, for example through the ISO CMIP or the Internet Simple Network Management Protocol (SNMP). Managed objects are shown as possessing two interfaces, the *functional interface*, from where it is possible to make the object perform its functions, and the *management interface*, from where it is possible to manage the object.

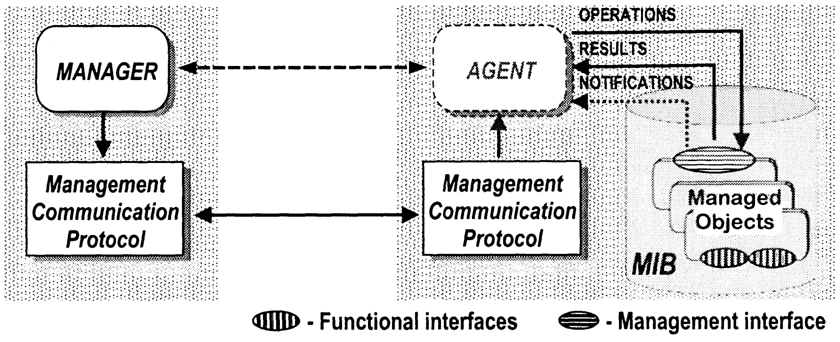


Figure 23.1. Generic Management Model

Agents As sub-managers, so to speak, agents perform operations on behalf of the high-level managers. As show in Figure 23.1, the agent accesses the objects through operation requests, in result of high-level commands issued to it by the manager. It gets results and notifications from the objects it subtends, which it forwards to the manager. The manager may access managed objects directly, a situation appropriate to small systems, corresponding to the configuration where the dotted 'agent' box disappears in Figure 23.1, and the manager uses the management communication protocols to issue requests, get results, and receive notifications.

The agent concept is inspired by a "hardware" view of the system, the view partaken by the ISO OSI model. In an abstract view of the system, for example that conveyed by the ODP model, instead of being special components, the manager and the agent are both objects. The manager object accesses managed objects directly or delegates in an agent, which becomes in fact a *composite*

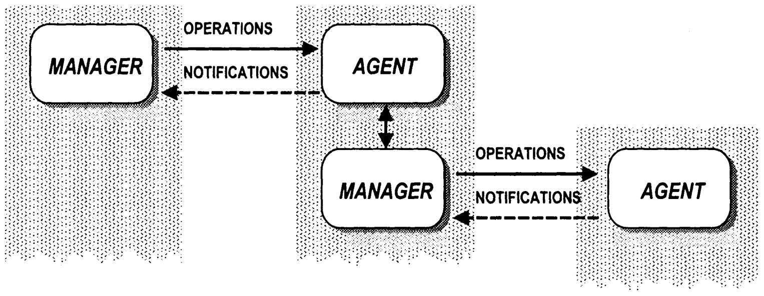


Figure 23.2. Management Hierarchy

managed object. That is, the agent presents the manager with a managed object interface, so that all the manager sees are objects. However, the agent behaves as a manager to the objects below on behalf of the actual manager. This renders the model homogeneous, since the manager accesses composite objects as normal managed objects, through their management operation-result-notification interface, and these objects access in turn a cluster of actual managed objects, through the same kind of management interface.

Hierarchy By recursing the manager-agent relation, one may introduce hierarchy in the model. This is extremely useful when managing large-scale architectures, and/or when the system architecture is itself hierarchical. As shown in Figure 23.2, the agent offers a "slave" interface to the manager, while acting to the layer immediately below— another agent— as a *mid-level manager*, and so forth. Note that the 'composite object' model also fits perfectly in a hierarchical structure: an object is a manager for a collection of managed objects below, but is a managed object to the upper layer manager object, and so forth. *Delegation* is the mechanism for introducing decentralization down the hierarchy (Goldszmidt and Yemini, 1995). Managers may delegate an increasingly higher level of functionality on the agents, conferring them the sort of properties (e.g., autonomy, reactivity, pro-activeness) that ultimately lead to highly decentralized structures based on intelligent and/or mobile agents (Sahai and Morin, 1998; Zhang and Covaci, 1997).

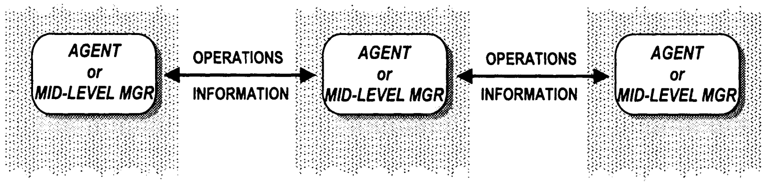


Figure 23.3. Management Cooperation

Cooperation Agents may cooperate to fulfill a management activity. Rather than hierarchical, this establishes a peer-to-peer relation between agents or mid-level managers. As suggested in Figure 23.3, agents cooperate in order to perform a complex management task, exchanging information and operation requests. Hierarchy and cooperation may be combined: cooperation relations may be established between the agents of a given level of the hierarchy. These agents may have a one-to-one relationship with a manager, or a one-to-many relationship, understood as a collective delegation from the manager to a group of agents, in order to perform some task. Inside the group, agents establish a many-to-many, or cooperative relationship. For example, the manager-agency paradigm (Post et al., 1996) defines a specialization of the manager-agent hierarchy, where a cluster of agents can be grouped in an *agency*, which has a common management policy, and consistency requirements among its members.

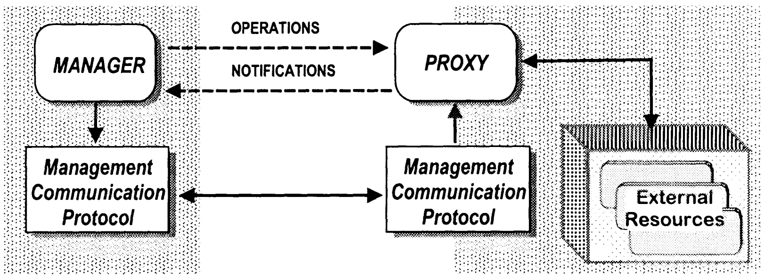

OPERATIONS
MANAGER
NOTIFICATIONS
PROXY
Management Communication Protocol
Management Communication Protocol
External Resources

Figure 23.4. Proxy Management

Proxies There must be a way of accessing alien devices, that is, devices that implement a different management communication protocol, proprietary or not, or do not implement any protocol at all (for example because they are too simple, or just passive). In this case, a proxy agent or object acts for the manager as a normal agent or a managed object, and they both communicate through the management communication protocol of the architecture, as depicted in Figure 23.4. On the other end it dialogues with the alien resources in whatever manner necessary, sometimes through direct wired connections. In essence, it plays the role of a gateway in a communication stack or of a transformer in an object model.

23.4 CENTRALIZED MANAGEMENT MODEL

Centralized local management is the classical model, inspired by the traditional mainframe-oriented "computing center" concept. Homogeneity and geographical concentration made integration happen naturally. The structure was also compact in terms of personnel, who resided at a single central facility. With the advent of networking, a simple management tool would extend the existing functionality to manage the network infrastructure. This model was challenged when the use of networks expanded, and it became necessary to coordinate several clusters of resources, either because they lived in different locations or had heterogeneous functionality.

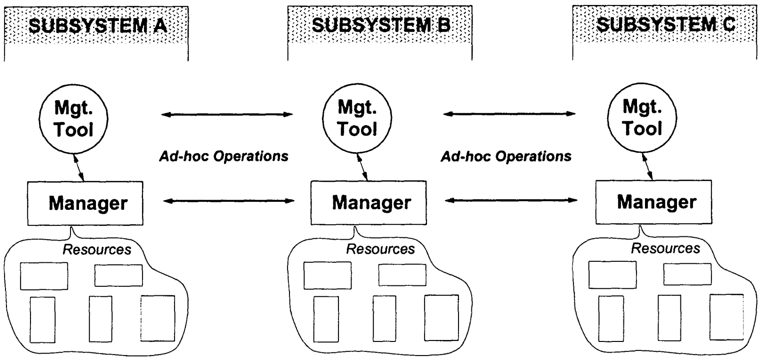


Figure 23.5. Islands of Management in the Centralized Model

This first attempt at the transposition from local to distributed management originated what we may call *islands of management.* This primitive model relied on simple tool and protocol level scripts and file transfers in order to loosely coordinate operations, but it retained all the autonomy of each island, as depicted in Figure 23.5. The islands preserved the computing-center model of management, originating a syndrome of competing power by the local managers. This situation conflicted with the increasing integration of information in each enterprise, requiring tighter coordination.

23.5 INTEGRATED MANAGEMENT MODEL

Management had inevitably to attain the desired level of integration, which was achieved in this evolution interim through mobile management teams, to execute the same actions at all sites, or through a greater cooperation between the autonomous management teams, to a large extent manual.

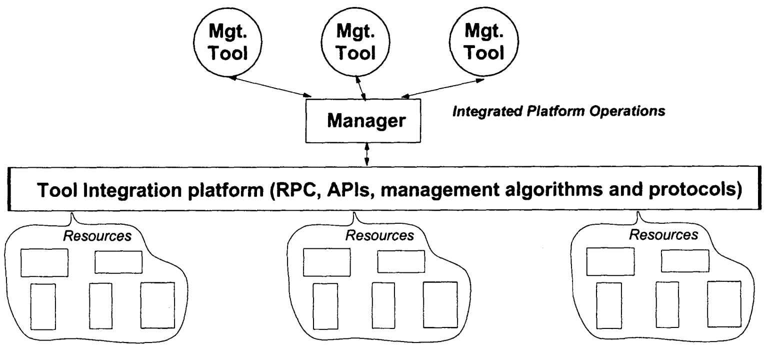

Mgt. Tool
Mgt. Tool
Mgt. Tool
Manager
Integrated Platform Operations
Tool Integration platform (RPC, APIs, management algorithms and protocols)
Resources
Resources
Resources

Figure 23.6. Integrated Management Model

The integrated management model solves the conflict, by allowing management to be organizationally centralized, both strategically and tactically, without incurring the shortcomings of physically centralized management. Major strategic decisions impact all subsystems in the same manner, and tactical management actions are performed in a centrally coordinated way. However, by resorting to the adequate distributed systems techniques, this model achieves a separation of concerns between the organizational and technical levels. Technically, management actions can be performed remotely, in a distributed manner, from a management platform. This platform is location independent. The model is shown in Figure 23.6. Integration has two facets: running several tools in a coordinated way; and managing the whole of the system resources in the several subsystems. Amongst the necessary techniques are: remote operation support (remote session, RPC); APIs common to several tools; distributed algorithms/protocols, such as management communication protocols; and common information formats (e.g. MIB). More recently, a form of integrated management, *web-based management*, emerged and became quite fashionable. It takes advantage of web technologies to implement lightweight client-server models (*see 3-tier Client-server Architectures* in Chapter 1). The form-based remote access mechanisms improve tool interface integration around the HTTP-HTML panoply, and allow lightweight, highly location-independent access.

23.6 DECENTRALIZED MANAGEMENT MODEL

The integrated management model is technically distributed, but still organizationally centralized— in philosophy and operation. This situation conflicts with the increasing integration of distributed applications in geographically distributed enterprises, and across enterprises, in enterprise networks. In these scenarios, chances are that organizations wish to retain a certain autonomy in managing their facilities, while still contributing to the operation of the whole large scale distributed system, in sort of a federated way.

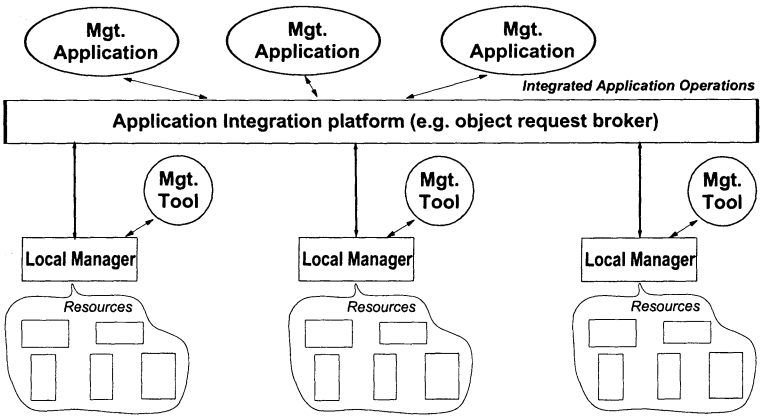


Figure 23.7. Decentralized Management Model

The decentralized management model is based on two premises: distributed and decentralized tactical management; more or less decentralized strategical management. The model is depicted in Figure 23.7. Local subsystems have local tactical management, with the necessary tools and/or platforms. Coordination is achieved through an integration platform. Given the structure of this model, the level of abstraction of the integration platform is necessarily higher than that of the integrated management model. For example, it could be at the level of an object request broker through which high-level applications dialogue with the local platforms, gathering information and providing high-level directives. Such as with political models, strategic management decentralization assumes several degrees, from confederated models to federated models. That is, the degree of freedom of local management, or in opposite terms, the degree of control exerted by the applications on local management platforms, may vary.

The **agent** paradigm has assumed great importance in the development of decentralized management models. Mobile agents have been used for building decentralized management architectures, exemplified in several works: the Java-based mobile intelligent agent framework discussed in (Zhang and Covaci, 1997), or by the 'mobile network manager' concept implemented by the MAGENTA environment (Sahai and Morin, 1998). Using the principle of delegation already discussed in Section 23.3, managers commit specific functions to mobile agents that execute them in remote parts of the system. Agents may even itinerate through the system to perform their function. Mobile agents partake the same security concerns that have already been pointed out to Java (McGraw and Felten, 1997; Garfinkel and Spafford, 1997). However, if these problems are adequately addressed, mobile agents may become the main paradigm for decentralized systems management.

Note that we have described the several generic models in an evolving way. Arriving at decentralized management through integrated management, even if only conceptually, considerably helps the understanding of the problems of distribution and decentralization, and the important difference between the two that we have always emphasized throughout this part of the book. Next we describe some models in particular, such as OSI, ODP, monitoring, and domains.

23.7 OSI MANAGEMENT MODEL

In relation to our generic model, the OSI Systems Management model is largely based on the manager/agent duality. This duality is exclusive, that is, a manager cannot interact directly with the objects subtended by an agent. Managed objects are external representations of the devices (hardware or software) they manage, and as such the conceptual functional and management interfaces do not always co-reside.

The management information representation (MIB formats, etc.) is specified in the standard Structure of Management Information (SMI) (ISO10165, 1992). OSI foresees several of the management functions we have discussed in the last chapter, namely, configuration, faults and alarms, monitoring and event management, log control, performance, accounting, and security. These are specified under standard Systems Management Functions (SMF) (ISO10164, 1992). The architecture is depicted in Figure 23.8, and relies on the full OSI communication stack up to Layer 7, including accessory Layer 7 services such as ACSE (Association Control Service Element - ISO 8649/8650) and ROSE (Remote Operations Service Element - ISO 9072-1/2). There are essentially three categories of management entities: layer management, common management information; system management applications.

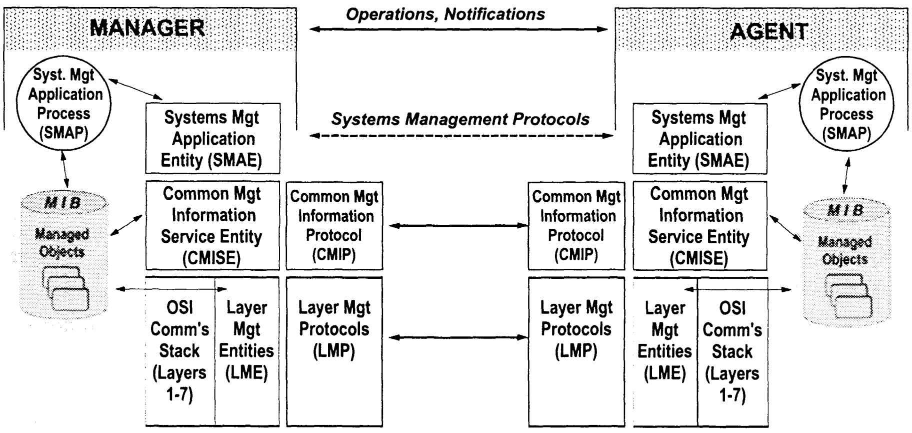

MANAGER
Operations, Notifications
AGENT
Syst. Mgt Application Process (SMAP)
Systems Mgt Application Entity (SMAE)
Systems Management Protocols
Systems Mgt Application Entity (SMAE)
Syst. Mgt Application Process (SMAP)
MIB
Managed Objects
Common Mgt Information Service Entity (CMISE)
Common Mgt Information Protocol (CMIP)
Common Mgt Information Protocol (CMIP)
Common Mgt Information Service Entity (CMISE)
MIB
Managed Objects
OSI Comm's Stack (Layers 1-7)
Layer Mgt Entities (LME)
Layer Mgt Protocols (LMP)
Layer Mgt Protocols (LMP)
Layer Mgt Entities (LME)
OSI Comm's Stack (Layers 1-7)

Figure 23.8. OSI Management Model

The OSI model intends the communication layers to have tactical management capability. These low-level functions are simple and autonomous, and normally do not require high-level intervention. They are typical of real-time management functions that can be automated and need be performed responsively. For example, reconfiguring the medium after a ring fault in a token ring network, recovering the token after a loss in a token bus, or rerouting in a wide-area network on account of a link failure. These functions are performed by the Layer Management Entities *(LME)*, through appropriate Layer Management Protocols at each of the lower OSI layers.

Next, the Common Management Information Service Entities *(CMISE)* define the interaction types between managers and agents. They are based on the connection-oriented remote operation paradigm (request-response), although certain responses are optional. These interactions are performed by the Common Management Information Protocol *(CMIP)*, which interprets for OSI the management communication protocol foreseen in the generic model. CMIP resorts to ACSE to establish an association (an application-level connection in OSI), and then to ROSE for the request/reply interactions. The main primitive classes supported by CMISE are: *association*, *operation*, *notification*. All begins with establishing an association with a remote managed host. Then several operations can be issued, such as `create` and `delete`, respectively to create a new managed object, or delete it, or to access attributes, such as `get` to request the value of an attribute, or `set`, to set its value. Managed hosts may issue unsolicied notifications with `event-reports`.

The CMISE primitives are essentially a template for primitive operations on the MIB objects, which may be combined into more complex sequences. These complex operations are requested by Systems Management Applications, supported on Systems Management Application Entities (SMAE). The management applications run cooperatively in Systems Management Application Processes *(SMAP)*, that is, between a *Manager SMAP* and *Agent SMAPs*, as depicted in Figure 23.8.

23.8 ODP MANAGEMENT MODEL

We saw that under the OSI philosophy, the managed objects are entities external to the components they represent. If the component is hardware, this is normal. However, if it is for example a protocol process, it would not be necessary, and introduces coherence problems that might be avoided if the managed object state resided with the component itself. This is the approach of the ODP management model, which relies on the fact that all components are objects.

In terms of our generic management model, the manager and the agents— if they exist— are fully-fledged objects. Managed objects comprise both the functional and the management interfaces, such that the management functionality is part of the object, unlike the OSI model. Objects can be arranged in a manager/managed-object hierarchy according to the structure and scale of the system.

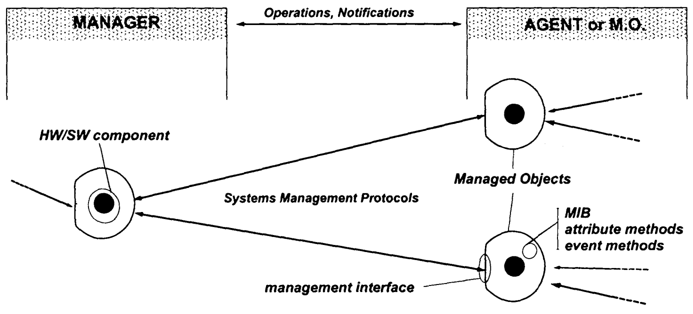


Figure 23.9. ODP Management Model

A simplified view of the ODP management model is presented in Figure 23.9. Note that components are wrapped with all the necessary functionality to manage and be managed, including the management interface, the methods pertaining to the management functions that we studied, and the state in the form of MIB elements. This configuration accounts for the use of the term *self-managed object* in ODP.

23.9 MONITORING MODEL

Monitoring is horizontal to many management functions, and it may be complex enough to be structured as a subsystem in a management architecture. A fairly complete model of monitoring (Mansouri-Samani and Sloman, 1994) is presented in Figure 23.10:

- **acquisition**– the monitor acquires information from the system being monitored: state reports; event reports
- **processing**– the raw information is treated
- **dissemination**– treated information is sent to other units
- **presentation**– information is presented at the management consoles

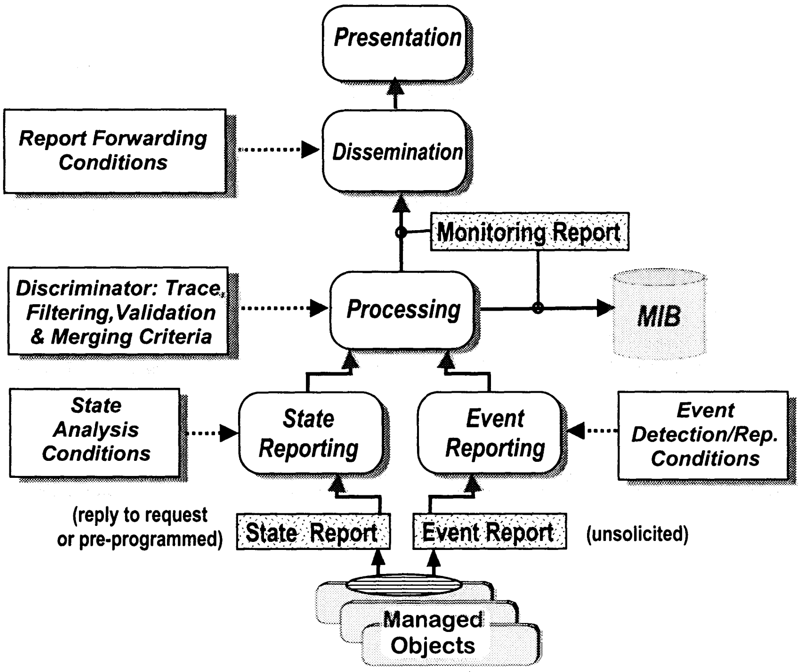

Presentation
Report Forwarding Conditions
Dissemination
Monitoring Report
Discriminator: Trace, Filtering, Validation & Merging Criteria
Processing
MIB
State Analysis Conditions
State Reporting
Event Reporting
Event Detection/Rep. Conditions
(reply to request or pre-programmed)
State Report
Event Report
(unsolicited)
Managed Objects

Figure 23.10. A Model of Monitoring

Sampling and polling are parameterized by pre-defined conditions. Event detection and reporting conditions are also parameterized. Several conditions influence event detection, such as: intrusiveness, the measure in which the sensor disturbs the managed object; location, whether detection is internal to the managed object, or external, e.g., by sampling; promptness, whether the event is detected immediately it takes place, or in a deferred manner, by record analysis. This raw information is often post-processed, since it is sometimes necessary to obtain the information in other forms. This involves several kinds of functions, such as: generating composite variables, such as means, rates or counters; building histograms; filtering glitches out; validating, such as checking bounds; merging data from different sources. Processing takes place under several classes of procedures, such as: construction of attribute traces during pre-defined intervals, merging of different traces, validation and filtering of results, updating of the MIB.

One of the functions of monitoring is to generate *reports*:

- **event reports** – such as the depassing of thresholds, or occurrence of errors
- **state reports** – amount of memory allocated, current network connections
- **solicited (or on-request) reports** – observation and report generation of state variable samples triggered by the manager at pre-defined instants
- **unsolicited (or on-demand) reports** – event report generated by the managed object, triggers the observation at the adequate moment, under several event detection conditions

In a distributed system, the treated information is disseminated between managers, and presented, normally in a graphical way (histogram bars, graphics, charts, meters and counters, and so forth), to the operator.

23.10 DOMAINS MODEL

The concept of domain has been introduced earlier. Let us look at domains as a model for structuring managed objects (Sloman and Twidle, 1994). Domains help coping not only with scale, but also with heterogeneity, both of resources and of their management policies. Domains can establish *indirection*, *grouping*, and *hierarchy*. Domains *do not* establish encapsulation. Indirection derives from the way objects are referenced. Each member object has a unique identifier, which provides an indirection to it. Objects are grouped in a domain. The *object set* (or policy set) of a domain is the identification of the group of objects belonging to that domain. The object set can be referred to collectively, through a group identifier. An object can belong to more than one domain. Finally, domains accept hierarchy: a domain may be member of another domain.

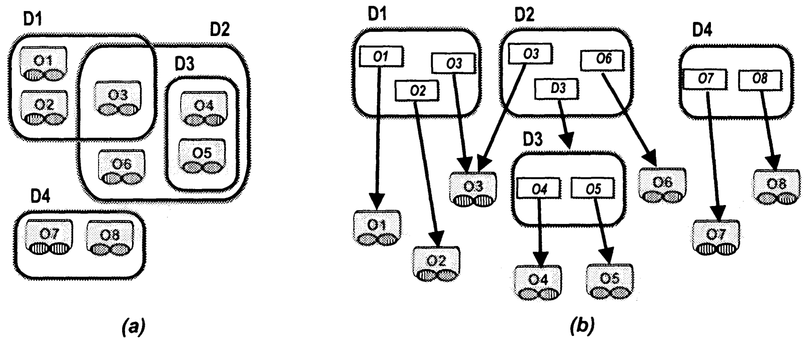


Figure 23.11. Domains Model: (a) user view; (b) implementation view

What was said above suggests the three possible relations between domains: A *disjoint* from B (distinct domains); A *overlapping* with B (objects in more than one domain); A *contained* in B (hierarchical domains). An example of a collection of objects defined in terms of domains is given in Figure 23.11. Figure 23.11a provides the user view. It shows that there exist four domains $D1$ through $D4$. The structure was chosen to emphasize the relations we have just mentioned. Note that $D4$ is disjoint from any other. $O3$ belongs simultaneously to $D1$ and $D2$, so the latter overlap. $D3$ is contained in $D2$. Figure 23.11b provides the implementation view. Observe that objects preserve whatever status they have in the system: there is no encapsulation, the domains data structure "points" to objets instead.

23.10.1 Policy and Role Based Management

Domains offer an adequate basis for establishing management policies. Policy-based management has been gaining importance in the measure that systems grow bigger and more complex. Policies establish relations between subjects and objects, or between managers and managed objects, with a view of specifying implementation-independent behaviors, such as authorization, and obligation. Policies have been used recently to specify security, and QoS, to name a couple.

Policies enable the application of role theory to management. Role-based management consists of the definition of roles in an organization, and of obligation and authorization policies to specify role relationships and interactions (e.g., client-server, producer-consumer, peer-to-peer). An application of the domains paradigm to role-based enterprise management can be found in (Lupu and Sloman, 1997).

23.11 SUMMARY AND FURTHER READING

In this chapter, we discussed the main models of distributed systems management. The first objective of the chapter was to provide insight to the systems architect on the main strategies and frameworks available. The second objective was to discuss the main management models in a problem-oriented manner, establishing links, whenever possible, to the paradigms learned in the previous chapter. The models were purposely described in an evolving way, trying to clarify in the mind of the reader the sometimes subtle issues at stake in the checkboard of management: distribution and decentralization, policy and politics, scale and heterogeneity, interconnectivity and interdependency, etc. For further reading, please see (Tschichholz et al., 1996). A recent survey on distributed systems management can be found in (Martin-Flatin et al., 1999). Management policies are discussed in (Wies, 1994; Koch and Kramer, 1995; Lupu and Sloman, 1997). More recently, mobile and intelligent agent technologies have been proposed for distributed systems management (Magedanz and Eckardt, 1996; Zhang and Covaci, 1997). An excellent survey on, and evaluation of, mobile code approaches in management can be found in (Baldi et al., 1997). Fault-tolerant systems management in the scope of ODP is discussed in (Powell, 1991).

In (Hegering and Abeck, 1994) a detailed discussion is found on integrated management, as well as interesting notions on strategical management issues. A detailed study of the monitoring model is given by Mansouri-Samani and Sloman in (Mansouri-Samani and Sloman, 1994). The OSI Management Model is addressed with detail in both of the above-mentioned works. For the ODP model in general, see (ODP, 1987). Fusion between the ISO stack model and the ODP object model has been tried by several research groups (Deri and Ban, 1997; Znaty et al., 1995; Banker and Mellquist, 1995; Mazumdar, 1996). ISO has endorsed the 'domains' concept in management, and it is used in several documents, e.g., in (ISO10040, 1992). A detailed specification of a domains model was produced in the scope of project DOMINO, a revised version of which can be found in (Sloman and Twidle, 1994).

The Telecommunications Management Network (TMN) model addresses the management of telecommunication networks (Aidarous and Plevyak, 1998). These networks have evolved from essentially dumb copper infrastructures to fully-fledged distributed systems (Znaty and Hubaux, 1997). This evolution is related with the Telecommunications Intelligent Network Architecture (TINA) framework. Material on this subject can be found in (Sloman, 1994; Dupuy et al., 1995).

Enterprise management has a connection with systems management, which involves managing systems and networks from the perspective of the company organization and of human cooperation. It is a relevant subject for a deeper approach into managing large-scale corporate systems, and some notes on the subject can be found in (Daneshgar and Ray, 1997).

24 MANAGEMENT SYSTEMS AND PLATFORMS

This chapter gives examples of management systems and platforms. Namely, we discuss ISO (CMISE/CMIP) and Internet (SNMP) management services and protocols, standard MIBs, management tools and platforms, and the Distributed Management Environment (DME). We finish the chapter with a presentation of several tools specifically addressing security management. In each section of the chapter, we will mention several examples in a summarized form, and then will describe one or two of the most relevant in detail. Table 24.6 at the end of the chapter gives a few URL pointers to where information about most of these systems can be found. The table also points to the IETF Request for Comments, ISO and ITU sites, where any cited standards can also be found.

24.1 CMISE/CMIP: ISO MANAGEMENT

Recall, as introduced in Section 23.7, that ISO management is based on a set of services, CMISE, Common Management Information Services (CMISE, 1988), and that those services are implemented by a management communication protocol, CMIP, Common Management Information Protocol (CMIP, 1988). CMISE is organized around three classes of services: association, operation, and notification. The main primitive services of CMISE are:

common management association	◦ `m-initialize`, `m-terminate`, and `m-abort`, respectively to start, terminate or abort a management connection
management operation	◦ `m-get` and `m-cancel-get`, respectively to request the value of an attribute, and cancel that request while pending; ◦ `set`, to set the value of an attribute; ◦ `action`, to invoke a specific action on a managed object, for example, request head cleaning of a printer; ◦ `create` and `delete`, respectively to create a new managed object, or delete it
management notification	◦ `m-event-report`, to issue agent-to-manager notifications

According to the specific management needs, managed hosts may implement several types of management *associations*: event; event/monitor; monitor/control; full manager/agent. The *event* association is used for example when we want a simple device to send nothing but notifications to a manager host, using only management notification services. The *event/monitor* association allows the manager to read (monitor) the status of devices, in addition to receiving events. The *monitor/control* association is typically used for configuration of a system. The *full manager/agent* association is obviously used when we wish to impose no restrictions on the operations between two hosts.

event association	hosts only send `m-event-report` messages
event/monitor association	additionally uses operation service `m-get`
monitor/control association	uses all management operation services, but does not use management notification services
full manager/agent association	implements all CMISE services enumerated above

As we explained in Section 23.7, CMIP interprets requests from CMISE, and uses the ACSE and ROSE protocols to reach remote hosts. There is a fairly straighforward mapping between the management *operation* and *notification* CMISE services and CMIP primitives or *protocol data unit* types (PDUs). This relationship is shown in Table 24.1. For a notification, CMIP entities exchange `m-Event-Report`, and then `m-Event-Report-Cfm` as a confirmation. Getting data is done by `m-Get` primitives. In response, one or more `m-Linked-Reply` PDUs are returned with the requested data. The interaction may be canceled while pending by an `m-Cancel-Get-Cfm` PDU. Setting attributes is done with `m-Set`, and confirmed with `m-Set-Cfm`. Actions are triggered by `m-Action` PDUs. They may be confirmed with `m-Action-cfm`, or with `m-Linked-Reply`, when a result must be returned.

Table 24.1. CMISE services and CMIP PDUs

Interaction	Service	Protocol (CMIP)
notification	M-EVENT-REPORT	m-Event-Report, m-Event-Report-Cfm
get data	M-GET	m-Get, m-Linked-Reply
cancel get	M-CANCEL-GET	m-Cancel-Get-Cfm
set data	M-SET	m-Set, m-Set-Cfm, m-Linked-Reply
action	M-ACTION	m-Action, m-Action-cfm, m-Linked-Reply
create	M-CREATE	m-Create
delete	M-DELETE	m-Delete

24.2 SNMP: INTERNET MANAGEMENT

Simpler, but less powerful and versatile than CMISE/CMIP, the implementation of the Simple Network Management Protocol or SNMP (RFC1157), is on the other hand more straightforward, and has spawned its success in the Internet world and not only (*see* also RFC1155, the Internet Structure of Management Information – SMI). SNMP has evolved since its inception, through SNMPv2 (*see* mainly RFC1901, RFC1902) and is currently SNMPv3 (*see* mainly RFC2271, RFC2274). SNMPv2 introduces a more elaborate structure for the SMI. Whereas SNMPv1 only supported TCP/IP, SNMPv2 is multiprotocol: IP, Appletalk, Novell IPX, ISO Connectionless Network Protocol (CLNP). Whereas in the CMISE framework managed hosts or agents may be asked to perform complex actions, in SNMP agent-side interactions are mostly reduced to reading and writing attributes. The SNMPv2 workplan addressed security aspects, but only with SNMPv3 were these finally addressed, in 1998 (Stallings, 1998). The main evolution brought by SNMPv3 was to establish a common framework for incorporating security in all SNMP versions. More recently, agent technology was introduced in SNMP, through the Agent Extensibility (AgentX) Protocol (RFC2257), which allows communication between master agents and subagents.

With relation to the generic model presented in Section 23.3, the SNMP management model is based on the manager/agent relation, the *SNMP station*, and the *SNMP agent*. It also supports proxies. SNMPv2 supports hierarchy, in the form of intermediate manager/agents, or *mid-level managers*. Typically, agents reside in managed nodes (PCs, WSs, network printers, routers, and so forth), from where they control the devices they subtend. SNMP services and PDUs are shown in Table 24.2. Note that there are no actions defined, as we already had mentioned. SNMP only allows simple reading and writing of attributes (`Get-req` and `Set-req`), and notifications from agents (`Trap`). The response to any request comes in the form of a single response PDU, `Resp`. When getting a structure, such as a table, `Get-next-req` PDUs are used after the first `Get-req` PDU, returning the next element of the structure in order. `Inform-req` and `Get-bulk-req` were introduced in SNMPv2. `Inform-req` supports unsolicited

Table 24.2. SNMP services and PDUs

Interaction	Protocol (SNMP)
notification	Trap
get data	Get-req, Get-next-req, Resp
set data	Set-req, Resp
information	Inform-req, Resp
get bulk data	Get-bulk-req, Resp

manager-to-manager communication, to exchange and/or disseminate information about management operations. `Get-bulk-req` optimizes the reading of large amounts of data, which was awkwardly done in SNMPv1 with a stream of `Get-next-req`. `Get-bulk-req` asks for the next n values, instead of just one.

While SNMPv1 lack of security made it possible for anyone to act as a manager and change managed objects at will in any system to which he had physical access, SNMPv3 provides an authentication and encryption framework. Authentication uses the HMAC protocol (RFC2104), with a choice of MD5 or SHA as the hash function (RFC2202). Encryption is performed by the DES protocol in CBC mode. SNMPv3 establishes a security policy by defining an attack model (akin to a fault model in fault tolerance). SNMPv3 should secure against: modification of information– changing in-transit message parameters; masquerading– impersonating an authorized manager in requesting operations; message stream modification– reordering or replaying messages; disclosure– unauthorized read of exchanged management information. It *does not* secure against: denial of service; traffic analysis. For notes on the mentioned protocols, the reader is advised to *see Using Cryptographic Protocols* in Chapter 18.

24.3 STANDARD MIBS

We have seen that standardization of the information representation is crucial for the interoperability of applications and protocols in different hosts and devices in a system. For example, for the Internet we have the Structure of Management Information, SMI (RFC1155 or RFC1442 for SNMP version 2), and detailed standards for the several relevant MIBs. As examples, we have the Internet MIB-II (RFC1213, RFC1450 for SNMP v.2), the Bridge MIB (RFC1493), or the Remote Network Monitoring RMON MIB, versions 1 and 2 (RFC1757, RFC2021).

Table 24.3 presents a few of the data types defined in the Internet SMI standard. MIB standards further define the specific contents of the data structures associated with the entities relevant to management. For example, there will be MIB definitions for TCP, IP, routers, bridges, network adapters, etc. Manufacturers will have "space" asigned in the MIB conceptual structure to insert data

Table 24.3. Example Abstract Data Types defined in SMI

Type	Description
IpAddress	an IP address
Counter	nonnegative increasing integer $O(2^{32})$
Gauge	nonnegative floating integer $O(2^{32})$
TimeTicks	counter in $10ms$ increments
Opaque	unformatted text

about their line of products. A MIB is organized hierarchically, so that MIB searches can be systematized easily as MIB graph traversals, and optimized using graph theory. At the time of this writing, the current effort is towards standardization of a global, framework independent MIB, where Internet, ISO, ITU, all fit. MIB objects are generically defined in ASN.1.

MIBs are divided in *groups.* For example, MIB-II features the following groups: `system`; `interfaces`; `ip`; `icmp`; `tcp`; `udp`; `egp`; `transmission`; `snmp`. The names are self-explanatory for the most part. The `system` group represents the system where the entity resides. The `interface` group represents each specific interface of a network device. ICMP is the control message protocol for TCP/IP, EGP (Exterior Gateway Protocol) is the IP routing protocol between autonomous systems, and the respective groups concern the relevant variables. The `transmission` group represents the physical media entities.

In each group, the necessary data objects are defined, obeying to the data types specified in the SMI. These data types correspond to attributes of one or more entities, and are specified according to the several management function needs. For example, Table 24.4 presents a few of the MIB-II `tcp` group objects, for configuration and for performance management. Other objects exist, in these and in other groups, for accounting management, fault management, and so forth.

One problem haunting management platforms is the network load caused by polling remote devices, specially in large-scale systems. An interesting standard addressing this problem is the RMON MIB, Remote Network Monitoring. The underlying idea is the concept of *remote monitoring device.* Such a device will supposedly assist network management, by gathering data in remote places, and making it accessible to the managing nodes. The RMON 1 standard is Ethernet based. Later, it was extended to the higher layers, 3 through 7, in RMON 2. This greatly increased the efficiency and accuracy of the remote monitoring device, also called *probe.* For example, it can track the traffic of a web-based application between two specific hosts. RMON devices can be either dedicated, or live as software modules in functional devices, such as hubs, bridges, routers, or PCs. However, the performance of the normal functions of host devices may be affected by RMON monitoring. For example, PC-based

Table 24.4. Example tcp group MIB Objects

Objects for Configuration Management	
Object	Description
tcpRtoMin	minimum retransmission timeout
tcpRtoMax	maximum retransmission timeout
tcpMaxConn	maximum number of open connections
tcpCurrEstab	current number of open connections
Objects for Performance Management	
Object	Description
tcpAttemptFails	number of failed connection attempts
tcpEstabResets	number of connection resets
tcpInErrs	number of reception errors
tcpOutSegs	rate of transmitted segments

monitoring of a network requires enabling promiscuous mode reception, i.e., receiving all passing frames.

An RMON device only has to implement part of the MIB it is concerned with. It can be normally off-line, and can be programmed to only contact the manager (notification) when certain conditions are met: a failure; certain thresholds passed; other conditions dictated by the manager (e.g., new host found). RMON2 allows a probe to only return the values that have changed since the last poll. The probe can also post-process data (e.g., create composite variables), and execute complex actions (e.g., host discovery), offloading work from the manager host. Finally, the probe supports multiple management platforms, that is, it can serve several managers. Incidentally, note the analogy of remote monitoring with distributed sensing, and of probes with representatives (*see Entities and Representatives* and *Input/Output* in Chapter 12).

24.4 MANAGEMENT AND CONFIGURATION TOOLS

What is required of management tools? In essence, whatever we have discussed in the previous chapters that can be performed in an automated fashion. For example:

- console management
- event monitoring and management
- remote control
- current management functions (configuration, accounting, etc.)
- software distribution
- backup management
- storage management
- server pool management

- security management (protection, intrusion detection, etc.)

There are a number of tools for these several different purposes, and in general one can talk about the following classes:

- testers
- network analyzers
- management software packages
- distributed management protocol stacks
- integrated management systems
- help desk systems
- trouble ticket systems

24.4.1 Testers

The simplest tools, testers are, as the name implies, devoted to testing basic hardware functions. Hardware probes test continuity of electrical circuits and cables, or digital levels (logic probe). Signal generators serve to inject signals in circuits or cables and test their reaction. Time domain reflectometers (TDR) are sophisticated devices that test the state of transmission lines and cables carrying high frequency signals. When these lines are twisted, smashed or simply flickering, they do not propagate high frequency signals adequately (e.g. 100Mb/s Ethernet), while still passing a simple continuity test. This is mainly because spurious reflected signals develop on the cable and disturb the original signal, hence the name of the detector. Impedance meters measure impedance, another parameter that only makes sense at moderate to high frequency, and must exhibit a stable value throughout the whole link between two nodes. Electrical field meters measure radiated power (e.g., strength, direction), for wireless communication.

24.4.2 Network Analyzers and Monitoring Tools

One level of abstraction up we have protocol signalling. Network (or protocol) analyzers can follow *all* the communication between two (or more) parties, and dissect the execution of a given protocol. Industrial network analyzers are normally dedicated machines with real-time operating systems and fast communication adapters that work in promiscuous mode, listening to everything on a medium. These machines normally have powerful filtering functions that allow pinpointing: packet types; origin and/or destination; contents; dialogues, etc. They are also capable of detecting errors in the protocol execution. Modern network analyzers are multi-protocol, i.e., they understand several of the major protocols. It is also possible to configure a software-based network analyzer, by using an adequately powerful machine (e.g., a high-end PC) and a software package, normally comprised of a modified network driver (e.g., Ethernet) that works in promiscuous mode, and a filtering, processing and rendering module.

Several of these analyzers for local area networks, also known as sniffers, are discussed later in this chapter (*see* Section 24.7).

Two of the main attributes of analyzers and monitoring tools are non-intrusiveness, the degree in which the tool does not disturb the system under observation, and precision, the degree in which the measurements correspond to the magnitudes being measured. Monitoring tools with hardware sensors exhibit better precision and smaller intrusiveness. On the other hand, they provide significant amounts of raw information that must be processed. Software sensors are implemented through the instrumentation of code. They exhibit moderate precision and are significantly intrusive. However, they address high level information (e.g., a certain step of the code), and can act at very specific points to gather information. An example passive, software-based performance measurement tool is described in (Malan and Jahanian, 1998). Hybrid systems take the best of both worlds, by using low-level instrumentation in hardware, hooked to software sensors. Consider the following example problem, complicated enough to deserve some hybrid tooling:

"Accurately measuring the interval between the reception instants of the first frame after a hardware trigger, at two network adapters in the same LAN"

That is, there is a trigger signal, after which we want to catch the first frame on the LAN, and measure the interval between the reception of that frame at two different network adapters. Note that this interval is of the order of the microseconds, and depends on the difference in network propagation delay and in reception processing speed at each adapter. As a hybrid solution, we make a hardware sensor by hooking the hardware trigger signal to a hardware interrupt line. Then, we build a software sensor activated by the trigger sensor. The software sensor is composed of monitoring code that waits for the next received frame, and produces an interrupt when that happens. The interrupt handling code activates a hardware actuator, for example by flipping one of the control pins of an unused RS-232C serial line connector. All these steps are done at both hosts. By measuring the interval between the pin flips at the two hosts with a simple external device such as an oscilloscope or an interval meter, we have an extremely accurate monitoring tool with relatively little hardware apparatus.

Amongst the existing monitoring and analyzing tools, we emphasize a few easily available ones. The *SNMP MIB Browser*, based on the WWW, is developed in the Technical University of Braunschweig (Germany). It is a simple CGI script written in the Tool Command Language (Tcl), which uses the Tnm Tcl extension for network management applications, and allows browsing SNMP MIB contents in a structured away. *Beholder*, or BTNG, is an RMON compliant Ethernet network monitor developed at the Technical University of Delft (Netherlands), which can be remotely queried by means of SNMP. Beholder is accompanied by the Tricklet package: a set of SNMP utilities for OS/2 and UNIX. *Ethereal* is a network protocol analyzer for Unix, in the context of the GNU-GPL effort. It examines both real-time network data and data from a capture file on disk. It allows a user to browse the data, viewing packets with programmable levels of detail. Ethereal features a display filter language and the ability to view the ASCII contents of a TCP connection.

Multi Router Traffic Grapher, or MRTG, is a tool developed at the Swiss Federal Institute of Technology (Switzerland), to monitor the traffic load on network links, as seen for example through routers. MRTG generates HTML pages containing GIF images of the traffic. *RRDtool* from the same institution builds on MRTG to provide fast round-robin database storage and display of time-dependent data (i.e. network bandwidth, machine-room temperature, server load average).

META is an interesting example of distributed monitoring tool is META (Wood, 1991), developed in Cornell University (USA). META runs on UNIX. It is distributed, and has a neat layered structure. Its functional part is a rule-based system written in a special language, Lomita. META not only monitors but can also control the managed system. For that, it features a sensor/actuator layer that deals with the infrastructure itself. This layer conceals the specific characteristics of the devices, and on top of it an abstract data model of the system is constructed, on whose state the rules of the policy layer are applied. Its data model is implemented on an active real-time database, capable of: triggering actions based on modifications of the state of the database (e.g., **when** condition **do** action); and expressing conditions depending on time durations (e.g., X **until** Y).

24.4.3 Management Software Packages and Protocols

The core of modern management systems are specialized software packages that implement specific management functions. Remote operation is possible if those packages can talk with devices through a common protocol. This is where management communication protocol stacks fit, such as CMIP or SNMP, which should be supported by both managing and managed hosts. On top of these stacks, management processes including the above-mentioned software packages dialogue. Software packages can be used to build tools that perform one or several related functions, such as configuration, or performance management (Zeltserman and Puoplo, 1998). These tools normally reside in the managing host(s), or manager(s), and talk remotely to the other (managed) hosts or devices. For example, *Zebra* is a free routing software (GNU Generic Public License or GPL) package that manages TCP/IP based routing protocols. It supports Border Gateway Protocol, BGP-4 (RFC1771) as well as RIPv1, RIPv2 and OSPFv2. Zebra software, unlike traditional, Gated based, monolithic architectures.

Tcl Extensions for Network Management Applications, or *Scotty*, is a software package developed at the Technical University of Braunschweig (Germany), which simplifies the implementation of portable, script-based, specific network management functions. Scotty is based on the scripting language Tcl, and has two main components. The first one is the Tnm Tcl Extension which provides access to network management information sources. The second component is the Tkined network editor which provides a framework for constructing an extensible network management system.

24.4.4 Application Configuration and Construction

Tools and environments for system and application configuration programming have emerged in the past few years. *Regis* is a programming environment for constructing distributed programs and systems, developed in the University of London, Imperial College (UK). Structural aspects of distributed programs— which are orthogonal to its algorithmics— are expressed in a configuration language. Regis is based on the architectural configuration language Darwin. Other aspects, such as inter-process communication, are orthogonal to the application structure. Programmers can create new communication classes (not necessarily RPC) independently from the program and the interaction style. Regis is assisted by companion software, such as the Software Architect's Assistant, and a Tcl/Tk Graph Widget. *AAA*, "Agents Anytime Anywhere", is an agent development environment, created at INRIA (France). It is materialized as a Java agent-based platform, using the message-bus paradigm. It supports the configuration and development of modular and configurable applications. The *TACOMA* project focuses on operating system support for agents. TACOMA is a collaboration between the University of Tromsø (Norway), Cornell University (USA) and the University of California San Diego (USA). An agent in TACOMA can be installed and executed on a remote computer, and may explicitly migrate to other hosts in the network during execution. Tacoma currently features a web agent that can be used to construct management tools, for example, for remote monitoring. A striking example is StormCast, a wide-area network weather and environmental monitoring application accessible over the internet.

24.4.5 Integrated Management Systems and Applications

When several subsystems, even heterogeneous, fall under the realm of a single management system, we say we have an integrated management system (IMS). Several freeware and commercial products fall into this designation. Current, significant IMS are built according to standardized frameworks, such as those originating from ISO or IETF (CMISE/CMIP or SNMP/RMON), which are manufacturer-independent and support open systems. They are often complemented with powerful database management, and versatile graphical interfaces.

Besides the many commercial IMSs, there are a few freely available ones. *LANdb* provides network managers a means of cataloging all connections, closets, and network hardware on a network. It uses scripting and database query languages in order to provide an efficient web-based frontend to a complete network management package. LANdb is distributed under the GNU-GPL. *MibMaster* is intended to be used on any SNMP-compliant network environment, MIB Master allows performing web-based management. SNMP agents can be viewed and/or modified using any Web browser. *UTopia* is an ATM management tool developed at the University of Twente that helps managers to configure their ATM switches. UTopia is web-based. The web client part is implemented as a JAVA applet.

GxSNMP is a GNU-GPL network management application developed in the context of the GNOME project. GNOME is the GNU Network Object Model Environment. The GNOME project intends to build a complete, easy-to-use desktop environment and application development framework.

SMB-SNMP is a management application based on MSFT Windows, developed at the University of Pisa (Italy). It maps SNMP MIBs as files of a Windows or WNT directory. MIB variables are represented as text ot HTML files. It uses the Samba file system to transparently map file operations on SNMP primitives. Whenever a variable is read/modified an SNMP get/set is issued to the remote SNMP agent. SNMP resources can thus be managed without any specialized software: the files can be edited by any normal editor.

24.4.6 Help Desks and Trouble Ticket Systems

Open systems, beyond a certain scale, require help desk functionality, normally provided through a center that ensures information providing and first-line assistance to users. A management help desk is little different from other kinds of help desks. It may be either public or private, e.g., fully open or confined to employees of an enterprise. Several media can be used, sometimes in alternative or complement, to serve the user. Telephone is the most obvious and the one with greater promptness. Fax or email provide a means for deferred attention, which can reduce the costs of a telephone interaction. More recently, web-based interfaces started proliferating. This kind of interface offers a potential for multimedia not available in the other means.

The help desk can have have a flat structure for simple systems (and simple problems). Otherwise, in a large/complex system, it may be structured by hierarchy and specialization. That is, personnel is stratified by expertise, and that expertise may be divided between groups. Calls arrive at the front-line, less expert and more generalist personnel, and may go up the ladder should that be required. Routing of requests may be semi- or fully-automated.

The main requirements to be satisfied by a help desk system are:

- interactive access to management information (if possible, concerning the function or device being asked about)
- interactive diagnosis guide (set-by-step methodology for arriving at the root of the problem)
- frequently asked questions (FAQ) manual (both for user and manager)
- knowledge base (a "FAQ" with sophisticated search methods)
- trouble ticket system (a complement to the help desk function, to track problems that remain unsolved for a while)

Some requests addressed to the help desk derive from malfunctions or other kinds of problems that require further attention. There are several reasons for organizing the response to these anomalies: to log requests, for legal and administrative reasons; to sequence requests, eventually prioritizing them; to classify problems, creating a typology for them; to create technical memory in the system, for frequent or recurrent events. This is done through a Trouble Ticket System.

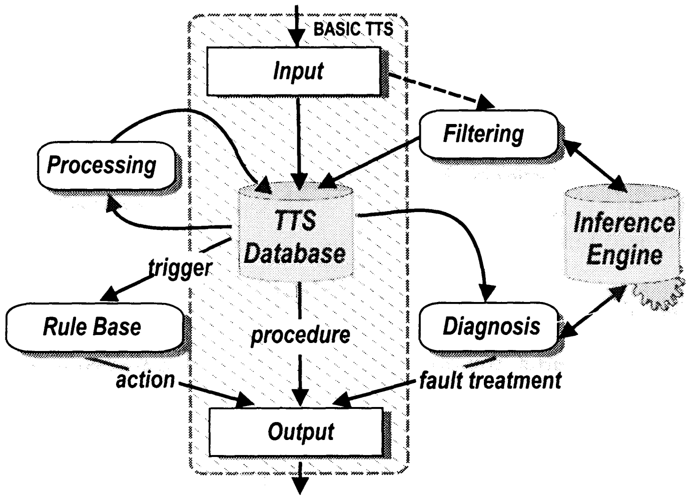


Figure 24.1. Structure of a Trouble Ticket System

A TTS is mainly an anomaly record and response system. It receives events, stores and processes them, and prepares the adequate response. Given the amount and kind of information it gathers, a TTS can be integrated with fault management functions to enhance the latter (Lewis and Dreo, 1993). TTS events can be generated by users, but they can also be generated automatically, by devices. After being processed, they originate a *trouble ticket (TT)*. The notification primitive that we studied in the managed object model can be used to send events to a TTS. The structure of a TTS is depicted in Figure 24.1, and has the following main blocks: input and output subsystems for interface with the outside (users, infrastructure, and managers); database for storing all necessary information (e.g., TTs, historical records); filtering and processing modules for the main TTS functions (e.g., housekeeping, filtering irrelevant requests); rule base and diagnosis modules for automating some of the TTS functions (e.g., automatic generation of actions upon certain triggers, fault diagnosis based on previous knowledge processed by an inference engine).

Not all TTS are as sophisticated as the model given in Figure 24.1, however, a basic TTS has at least the functionality included in the dashed part. A TTS in action is briefly as follows. The notification event is received, and filtered. The problem is logged with all the necessary information, in what makes a trouble ticket (TT): origin and location of the fault, nature of the fault, possible cause, timestamp, contact person. The TT is classified, against ordering, type and priority, and then dispatched to a manager on duty. If the TTS allows data mining, the historical records are searched for a match with a similar problem. A *current problem log* is also opened, and will be maintained until the problem is solved, with all relevant information, such as further findings or steps taken to remedy. Automated instrumentation (management tools) may be used to diagnose the problem better and attempt to correct it. If the problem persists or is too complex, the TT is routed to a more senior manager. In background, both the managed system and the TTS operation should be quality controlled: Are there many problems? Are we solving most of them? Are we solving them well?

24.5 MANAGEMENT PLATFORMS

What is a management platform? We have seen that management requires the assistance of several functions that are normally performed by isolated tools. A platform is a working base where the manager has access to a set of tools that act in a coordinated way, interact among themselves, and share the same raw data. Platforms are a pre-condition for integrated management, inasmuch as distributed management frameworks are a pre-condition for open platforms to work.

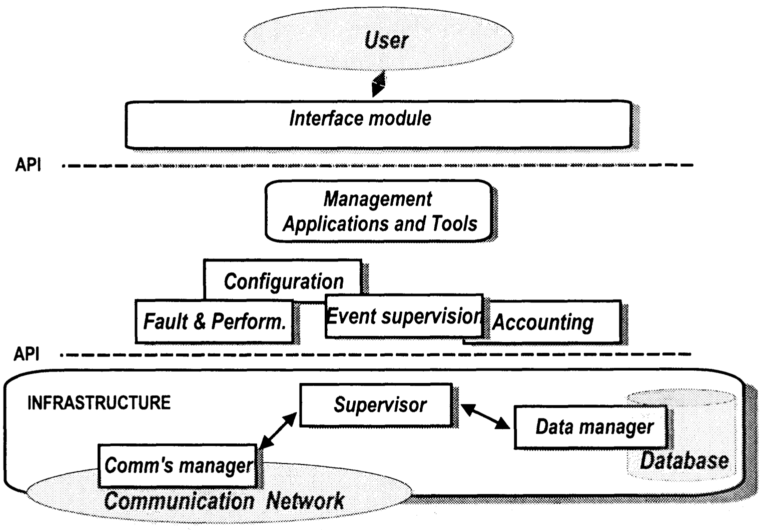


Figure 24.2. Generic Structure of a Management Platform

A generic block diagram of a platform is given in Figure 24.2. The platform is supposed to be distributed, and has essentially three levels: the user interface level, where all graphical rendering facilities lie; the managing level, where management functions are concentrated; the infrastructure level, where the device specific parts are located (managed objects, agent functionality, etc.).

Let us consider the manager host. The *management console* runs in this host, with the *interface level* software. The several software packages running the necessary management functions run at the *managing level* (most modern platforms are modular with this respect, one can install only the tools needed), provide a substrate for building management applications and tools. They communicate with the user interface through a API common to all application modules, so that display is homogeneous. They communicate with the *infrastructure level* through another API, also common to all application modules. At this level, the *supervisor* is the kernel-level part that handles and dispatches all management related requests. It interacts with the *data manager*, which controls the database (MIB) operations, and with the *communications manager*, which runs the management communication protocols, that dialogue with the other managed hosts in the system. Through this protocol, the platform can invoke operations on remote managed hosts and objects.

A device may have more or less powerful machinery for management. Let us consider a managed host, possibly subtending several simpler devices. It will only have an implementation of the infrastructural level, that will dialogue through the management communication protocols with the managing host, where the rest of the platform functionality resides. The infrastructure can have a simplified supervisor, a MIB database specific of the subtended devices, and a fully-fledged communication stack (e.g., SNMP).

The integrated management model (*see* Section 23.5) is normally materialized around management platforms. There are several commercial management platforms, amongst which Sun Solstice, IBM Tivoli, HP OpenView. The latter will be discussed with some detail.

HP OpenView The reference example of management platform is OpenView from HP. The architecture of OpenView is presented in Figure 24.3. The architecture is modular, is pretty much in line with the generic platform structure that we have just presented.

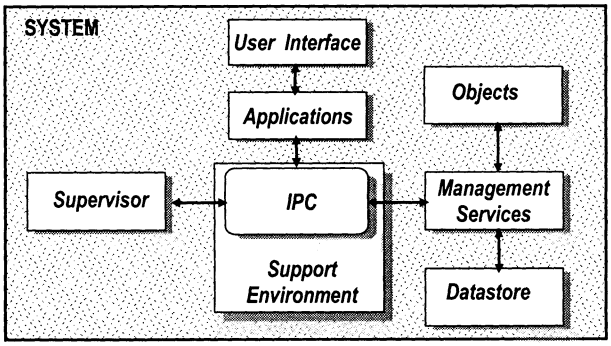


Figure 24.3. HP OpenView Management Platform

Currently available on several operating systems, OpenView was one of the first open platforms based on standards, namely ISO protocols and Internet MIBs. Although it is a proprietary system, it soon became a market reference, and was one of the main contributors to the Distributed Management Environment initiative of the OSF. OpenView is distributed, and its components are spread throughout all the managed hosts. The managing host runs the manager console, and the platform server, the OpenView Network Management Server. In each host one may install only the necessary modules. In Figure 24.3 we can see the several modules of OpenView:

system	the host where the modules are installed
supervisor	the kernel level manager of local resources
IPC	interprocess communication, also called *postmaster*, handles message transactions between platform modules, local or remote, through the adequate protocols (local IPC, remote operations, CMIP, SNMP, etc.)
support environment	the platform modules rely on a set of local support services, such as directories, file transfer, etc.
objects	the representation of the managed objects themselves
management services	management function modules necessary in this host
datastore	management of the data repositories
user interface	interface with a user console when necessary
applications	management applications

High-Level Platforms The management models have evolved, as we have studied in Chapter 23, and so have management platforms. Support began to emerge for the global planning and configuration aspects, and for the strategic management of large and/or loosely coupled (e.g., federated) facilities, in the form of high-level platforms. High-level platforms perform strategic support to management in a decentralized fashion, such as helping with the definition and refinement of policies. For that reason, they must be capable of integrating several platforms under their realm, such that it makes sense to call them **platforms of platforms**. One of the enabling factors for this integration to be possible is what has been denominated *platform middleware*, the set of support technologies for high level integration of applications. Object request broker, message bus and agent technologies are promising options for platform middleware. This technology maps onto the decentralized management model (*see* Section 23.6).

Examples of high-level platforms are OperationCenter from HP and Spectrum from Cabletron. Essentially, they provide an environment-independent platform, based on a virtual network machine, to which all underlying platforms are translated. The machine can thus dialogue with the tools and applications resident in the underlying platforms, and it can also interface directly the several standard management protocols, or support new ones. Clients interface this virtual platform and run high-level applications, mostly of the strategic nature we have suggested above. Clients of high-level platforms supposedly belong to strategic management teams.

24.6 DME: DISTRIBUTED MANAGEMENT ENVIRONMENT

The Distributed Management Environment or DME is a multi-vendor distributed management platform originally developed by the Open Software Foundation (Chappell, 1992). It was based on several contributing technologies, amongst which Bull, IBM, HP and Tivoli. DME was important in that it launched the generic architectural foundations of distributed systems management platforms. DME lost momentum because of the hesitations of vendors to endorse it, since a common platform would shave any competitive advantages of their products. Namely, it ceased being supported by the now-called Open Group, who has discarded plans to port CORBA to DME as its object broker.

The success of the Web also brought lightweight, desktop-oriented, *web-based management*, a dressing of the integrated management models which changes the focus of platform development towards the desktop. Research on web-based management models is very active. Standardization is lead by the Open Group (OG) Management Program, and by the Distributed Management Task Force (DMTF) (*see* Table 24.6 for URLs). Both are consortia of companies operating in the field. Together, they are defining the object-oriented Common Information Model. The DMTF is investing on a model for vendor-independent remote management operations, has defined a Desktop Management Interface (DMI) for the purpose, and has clearly endorsed the Web-Based Enterprise Management (WBEM) model.

We can say DME fulfilled its role, in influencing a whole generation of platforms. Vendors have in most cases adopted DME concepts, and provide some of its proposed functionality in their products. For these reasons, it is worthwhile analyzing DME. Its functional structure consists of the following modules: object management framework, network management option, distributed services. The *object management framework* is the central piece, based on cooperating peer-to-peer objects, oriented to distributed systems management, on top of which are based the *distributed services.* These are infrastructure independent, and extensible.

DME is object based. Both managed objects and all functions are specified and implemented as objects, in the sense of CORBA and ODP (*see* Section 23.8). However, with regard to our generic management model, where there was a distinct hierarchy between manager and managed object or agent, these objects in DME are *cooperating*, that is, they have a peer relationship. Of course, a manager/agent relationship may be superimposed (and in fact often is) on DME objects. The architecture of DME is depicted in Figure 24.4, and consists of the following building blocks: object services; management services; management applications; management user interface (MUI); support services.

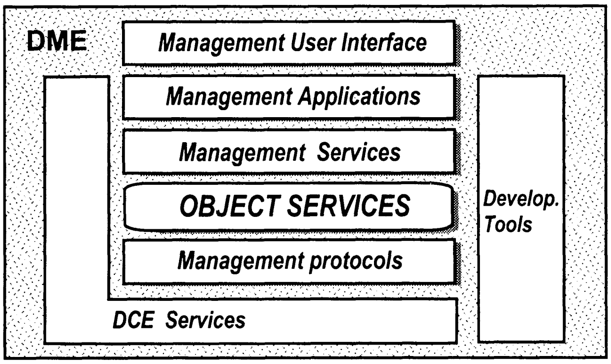


Figure 24.4. Architecture of DME

The *Object Services* are the core of DME. They are implemented by *object servers*, and also feature a notification service, the *event service*, which is crucial to implement interactions of the above-mentioned peer-to-peer nature, awkward to build with mere RPC. The object services supply the basic modules for the construction of the rest of the management services and applications.

The *Management Services* are built on top of the object services, and consist of building blocks for applications in the several system management areas.

The *Management Applications* rely both on the object services and on the management services. They interact with the user through a *Management User Interface* module. The *Support Services* consist of: DCE (*see DCE* in Chapter 4); management protocols; and development tools.

24.7 MANAGING SECURITY ON THE INTERNET

Internet protocols have their *design vulnerabilities*. In order to trace them and neutralize their effect, the *security management* of a system is of extreme importance. The functions relevant to security management are listed in Table 24.5.

First of all, in order to look for vulnerabilities we have to know about them. Then, we also might appreciate advice on how to remove them. Several institutional entities cooperatively centralize incident notices (e.g., new viruses, attacks), and cures for them (e.g., patches, releases, scripts). One relevant example is the CERT, Computer Emergency Response Team, whose URL is given in Table 24.6, as well as the URLs of most of the systems described in this chapter. The technologies cited above are freeware tools available to assist the administrator or to enhance security of installations. The insertion of this kind of technologies in a coherent management framework has been addressed in Chapter 23, where we discussed System Management strategies and tactics. In this section, we are going to briefly review the technologies themselves.

Table 24.5.

security enhancement tools	in this class we have tools that render machines and software more robust, for example cryptographic communication software, filtering and wrapping software, or packages that encrypt, sign or checksum critical software, to detect modifications; examples of such software are Tripwire, Xinetd, Tcpwrapper, Portmapper, and Cracklib
fault diagnosis tools	in this class we have for example packages that scan the facility looking for design or configuration vulnerabilities; examples of such software are Crack, COPS, Tiger, ISS, Satan, Merlin, Trojan
intrusion detection tools	in this class we have for example packages that perform real-time supervision, looking for anomalous behavior or state of the system, or abnormal patterns of usage, in order to detect intrusions; examples of such software are Scan-detector, CPM, AID, AAID, NID, ASAX, Hummer
auditing tools	in this class we have for example packages that perform logging and build audit trails of the system, in order for the administrator to analyze events a posteriori e.g., correlate attacks to detect intrusion campaigns; examples of such software are Tcpdump, Analyzer, Swatch, Logdaemon, Netlog, Netman

24.7.1 Security Enhancement Tools

Tripwire is an integrity monitor tool for Unix systems. It uses message digest algorithms to checksum files and guarantee their integrity, by detecting tampering with file contents, in result of intrusions. *Xinetd* is a replacement for inetd, the internet daemon in UNIX systems. It supports access control based on the address of the remote host and the time of access. It also provides extensive logging capabilities, including server start time, remote host address, remote username, server run time, and actions requested. *Tcpwrapper* allows monitoring and controlling connections to the main inetd communication ports: `tftp`, `exec`, `ftp`, `rsh`, `telnet`, `rlogin`, `finger`, and `systat` ports. Also includes a library so that other programs can be controlled and monitored in the same fashion. *Portmapper3* is a replacement of the original `portmapper` program, known to have security flaws such as allowing anyone to read or modify its tables and forwarding any request so that it appears to come from the local system. Portmapper3 does essentially the same as Tcpwrapper, but for RPC based programs invoked by the standard `portmapper`. The Securelib shared library (eecs.nwu.edu:/pub/securelib.tar) implements access control for all kinds of (RPC) services, not just the `portmapper`. *Cracklib* is inspired by the Crack program (*see* next section). It is a library containing C functions that may be used in a passwd-like program. CrackLib prevents users from choosing passwords that Crack could guess, by filtering them out at the source.

24.7.2 Fault Diagnosis Tools

Crack guesses eight-character standard Unix passwords, by standard guessing techniques. It is written to be flexible, configurable and fast. *COPS*, the Computer Oracle and Password System (COPS) package, examines a system for a number of known weaknesses and alerts the system administrator to them; in some cases it can automatically correct these problems. *Tiger* is similar to COPS, but more up to date, and easier to configure and use. It consist of system monitoring scripts that scan a Unix system looking for security problems. *ISS* is a multi-level security scanner that checks a UNIX system for a number of known security holes such as problems with sendmail, improperly configured NFS file sharing, etc. ISS can be used to probe entire network facilities. *Satan*, the System Administrator Tool for Analyzing Networks, is a network security analyzer. SATAN scans systems connected to the network, notifying about the existence of well-known, often exploited vulnerabilities. SATAN can scan the system from the outside, as hackers do, and thus provide a realistic analysis. *Courtney* monitors the network and identifies the source machines of SATAN probes/attacks, because SATAN is also used by hackers. Courtney receives input from tcpdump. If one machine connects to numerous services within a short time window, Courtney identifies that machine as a potential SATAN host. *Merlin* is a tool for managing and enhancing existing security tools. It can provide a graphical front-end to many popular tools, such as Tiger, COPS, Crack, and Tripwire. Merlin makes these tools easier to use, while at the same time extending their capabilities. *Trojan* is a trojan horse checking program. It examines a given search path and looks at all of the executables in the search path for people who can create a trojan horse that root can execute.

24.7.3 Intrusion Detection Tools

CPM Sniffer Detector, or Check Promiscuous Mode, checks a system for any network interfaces in promiscuous mode; this may indicate that an attacker has broken in and installed a sniffer program. *Scan-detector* is a tool to monitor for port scans of a Unix system. This is a frequent attack, and normally the first to be performed against a facility. Detecting port scans can give a security administrator precious early warning. *Adaptive Intrusion Detection System (AID)* is designed for network audit based monitoring of local area networks and used for investigating network and privacy oriented auditing. The system has a client-server architecture consisting of a central monitoring station and several agents (servers) on the monitored hosts. The central station hosts a manager (client) and an expert system. Heterogeneous UNIX environments are supported by having the agents produce OS independent data formats. Audit data are analyzed at the central station by a real-time expert system. Secure RPC is used for the communication between the manager and the agents. *ASAX*, Advanced

Security audit trail Analysis on uniX, is a distributed audit trail analysis system that also incorporates configuration analysis. The audit trail analysis system consists of a central master host and one or more monitored machines. The latter analyze their local audit data and send relevant events to the central host. Heterogeneity is achieved by using an O.S. independent data format. The system is rule-based, detecting known penetration patterns. *Hummer* is a distributed component for any intrusion detection system, that allows any two Internet hosts to share security information. It enables cooperative intrusion detection using data sharing between distinct sites, to counter the present threat of distributed intrusion campaigns— systemic attacks involving multiple hosts.

24.7.4 Auditing Tools

Tcpdump and *Analyzer* (formerly Windump) are packages for network monitoring and logging. Tcpdump is the best known and the most used such package. It programs the driver to be in promiscuous mode and grabs the network traffic. Analyzer is the port to Windows95 and WNT. These packages are normally assisted by analysis software, since they grab huges quantities of bulk data. *Swatch* aims at monitoring events on a large number of systems. It modifies certain programs to enhance their logging capabilities, and monitors the system logs for specific messages. *Logdaemon* is a package that provides modified versions of rshd, rlogind, ftpd, rexecd, login, and telnetd. These versions log significantly more information than the standard vendor versions, enabling better auditing of problems via the logfiles. It also includes support for the S/Key one-time password package. *Netlog* is a package that contains a TCP and UDP traffic logging system. It can be used for locating suspicious network traffic. *Netman* is a fairly complete toolbox for network monitoring and visualisation. Two of the tools provide a real-time picture of network communications, while the other provides retrospective packet analysis. These tools are designed to allow network managers to passively monitor a network and diagnose common network problems as quickly and efficiently as possible. Etherman is an X11 based tool which displays a representation of real-time Ethernet communications. Interman focusses on IP connectivity within a single segment. As with Etherman, this tool allows a real-time representation of network communications to be displayed. Packetman is a retrospective Ethernet packet analyser. This tool allows the capture and analysis of an Ethernet packet trace.

24.8 SUMMARY AND FURTHER READING

This chapter gave examples of systems and platforms for distributed systems management. We started by addressing tools and platforms and explaining their differences. We talked about testers, network analyzers, management software packages, distributed management protocol stacks, integrated management systems, help desk systems, trouble ticket systems and monitoring systems, integrated platforms. Then we discussed the two main management frameworks, ISO CMISE/CMIP and IETF SNMP, presenting the relevant protocols, followed by a study of the main standard MIBs. Next, we addressed DME as the reference environment for distributed management platforms, and finalized by presenting a number of tools for managing Internet performance and security.

Further study is suggested on several areas. Management of, and based on, the World-Wide Web has deserved great attention, as exemplified by (Pras et al., 1997; Schonwalder and Toet, 1997; Hong et al., 1997; Thompson, 1998). High-level notations for the specification of network management functions help bridging the semantic gap between the sometimes simplistic management function standards and the often sophisticated requirements of application and tool builders (Brites et al., 1994; Pavlou et al., 1998; Hughes, 1993). Agent technology can be used to develop new generation management platforms, as discussed in (Muller, 1997).

Table 24.6 gives a few pointers to information about some of the systems described in this chapter. However, for further practical study on CMIP and SNMP, we suggest (Leinwand and Conroy, 1996) or (Stallings, 1999), besides the standards and RFC documents themselves. The Distributed Management Environment (DME) is further treated by Autrata & Strutt in (Sloman, 1994). Omnipoints (OMNIPoint, 1993) is a Network Management Forum initiative for achieving interoperability of management systems.

Table 24.6. Pointers to Information about Management Systems and Platforms

Sys. Class	*System*	*Pointers*
ISO		www.iso.ch
ITU	(ex-CCITT)	www.itu.int
RFCs	(IETF)	www.rfc-editor.org
ICANN	(Names & Nrs)	www.icann.org
OpenGroup	(ex-OSF)	www.opengroup.org/management
DMTF		www.dmtf.org
	(DMI spec.)	www.dmtf.org/spec/dmis.html
NMF		www.nmf.org
		www.tmforum.org
OMG	(Objects)	www.omg.org
GNOME	(GNU GPL)	www.gnome.org
ASN.1	(Syntax Not.)	www-sop.inria.fr/rodeo/personnel/hoschka/asn1.html
Management	**SmurfWeb**	netman.cit.buffalo.edu
Related	**SimpleWeb**	www.simpleweb.org
Sites	**SimpleTimes**	www.simple-times.org
	Agentlink	www.agentlink.org
Managem.	**CMISE/CMIP**	www.cs.ucl.ac.uk/research/osimis/share.htm
Protocols	**SNMP**	ftp.net.cmu.edu/pub/snmp
		www.gaertner.de/snmp
		ucd-snmp.ucdavis.edu
		www.snmpworld.com
	SNMPv3	www.ibr.cs.tu-bs.de/ietf/snmpv3
	Agentx	www.scguild.com/agentx
		ftp.net.cmu.edu/pub/agentx
	MIB Browser	www.ibr.cs.tu-bs.de/cgi-bin/sbrowser.cgi
	Beholder	dnpap.et.tudelft.nl/pub/btng/README
	Vendor MIBs	www.simpleweb.org/ietf/enterprise.html
	Ethereal	ethereal.zing.org
	MRTG	ee-staff.ethz.ch/~oetiker/webtools/mrtg
		ee-staff.ethz.ch/~oetiker/webtools/rrdtool
	Zebra	www.zebra.org
	Scotty	wwwhome.cs.utwente.nl/~schoenw/scotty
		www.ibr.cs.tu-bs.de/projects/scotty
Managem.	**AAA**	www.dyade.fr/en/actions/aaa
and	**GxSNMP**	www.gxsnmp.org
Configur.	**LANdb**	avenir.dhs.org/landb
Packages	**MibMaster**	www.equival.com.au/mibmaster
Tools and	**Regis**	www-dse.doc.ic.ac.uk/~regis
Systems	**SMB-SNMP**	jake.unipi.it/~deri/SMB_SNMP
	SAMBA	samba.anu.edu.au/samba
	StormCast	www.cs.uit.no/forskning/DOS/StormCast
	TACOMA	www.tacoma.cs.uit.no/
	Tcl	www.scriptics.com
	UTopia	www.simpleweb.org/nm/research/projects/utopia
Management	**HP OpenView**	www.openview.hp.com
Platforms	**Tivoli**	www.tivoli.com
	Spectrum	www.aprisma.com
	Solstice	www.sun.com/solstice
	DCE	www.opengroup.org/dce

Table 24.6 *(continued)*
. Pointers to Information about Management Systems and Platforms

Sys. Class	*System*	*Pointers*
Security Manag. Tools	**CERIAS**	www.cerias.purdue.edu
	tripwire	ftp.cerias.purdue.edu/pub/tools/unix/ids/tripwire
	Xinetd	qiclab.scn.rain.com/pub/security
		www.synack.net/pub/xinetd
	Tcpwrapper	ftp.ox.ac.uk/pub/comp/security/software/monitors/
		ftp.cerias.purdue.edu/pub/tools/unix/netutils/tcp_wrappers
	Portmapper	ftp.cerias.purdue.edu/pub/tools/unix/netutils/portmap
	Cracklib	ftp.cerias.purdue.edu/pub/tools/unix/libs/cracklib
	Crack	ftp.cerias.purdue.edu/pub/tools/unix/pwdutils/crack
	COPS	ftp.cerias.purdue.edu/pub/tools/unix/scanners/cops
	Tiger	ftp.cerias.purdue.edu/pub/tools/unix/scanners/tiger
	ISS	ftp.cerias.purdue.edu/pub/tools/unix/scanners/iss
	Satan	www.cs.purdue.edu/coast/satan.html
		www.fish.com/~zen/satan/satan.html
		ftp.cerias.purdue.edu/pub/tools/unix/scanners/satan
	Courtney	ciac.llnl.gov/pub/ciac/sectools/unix/courtney
		ftp.cerias.purdue.edu/pub/tools/unix/logutils/courtney
	Merlin	ciac.llnl.gov/pub/ciac/sectools/unix/merlin/merlin.tar.gz
	Trojan	ftp.cerias.purdue.edu/pub/tools/unix/sysutils/trojan
	Sniff Det	ftp.cerias.purdue.edu/pub/tools/unix/sysutils/cpm
		ciac.llnl.gov/pub/ciac/sectools/unix/sniffdetect
	Scan Det	ftp.cerias.purdue.edu/pub/tools/unix/logutils/scan-detector
	AID	www-rnks.informatik.tu-cottbus.de/~sobirey/aid.e.html
	ASAX	www.info.fundp.ac.be/~amo/publications.html
	Hummer	www.csds.uidaho.edu/~hummer
	Tcpdump	ftp.cerias.purdue.edu/pub/tools/unix/netutils/tcpdump
		ftp.ee.lbl.gov
	Analyzer	netgroup-serv.polito.it/analyzer
	Swatch	ftp.cerias.purdue.edu/pub/tools/unix/logutils/swatch
	Logdaemon	ftp.cerias.purdue.edu/pub/tools/unix/logutils/logdaemon
	Netlog	ftp.cerias.purdue.edu/pub/tools/unix/logutils/netlog
	Netman	ftp.cerias.purdue.edu/pub/tools/unix/netutils/netman

25 CASE STUDY: MANAGING VP'63

This chapter finalizes our case study: managing the (VintagePort'63) Large-Scale Information System. VP'63 became significantly complex, and the company depends heavily on it. Its operation must remain stable, and its reconfiguration made as easy as possible. Tactical management mechanisms implementing strategic management policies will be studied, and developed around an integrated management platform.

25.1 ESTABLISHING MANAGEMENT STRATEGIES AND POLICIES

The reader should recall that this is the next step of a project implementing a strategic plan for the modernization of VP'63, started in Chapter 5, and continued in the Case-Study *chapters of each part of this book. The reader may wish to review the previous parts, in order to get in context with the project.*

The current infrastructure is managed on an ad hoc, uncoordinated way, since there was not until now a real distributed systems approach to the problem. The networking infrastructure evolved with the introduction of new segments and modules, and the corresponding network management points. Systems and applications are managed by staff local to the facilities. This situation is depicted in Figure 25.1.

Before attempting to do any change, a management strategy should be defined. The objectives of this investment on VP'63 are: to have a global and seamless information flow that serves decision making in the company; to allow corporate management decisions to be impressed as fast as possible on the information system. Corporate management is centralized, and as such, centralized strategic management is the option to make. The Chief Information Officer (CIO) helps define this strategy, in the form of management policies, and is responsible for its implementation by the tactical management team. Management policies should be defined in terms of resources (information and services) and users. They concern, amongst other things, the generic management policy for each service, and the characteristics of operations of users on resources.

Current management personnel expertise should be preserved, but perhaps reallocated under the viewpoint of the new organization. The core management team should be allocated to one, at most two, physical sites, from where they should be able to run the infrastructure. Then, more important facilities and specially those hosting factory automation subsystems will have some dedicated management personnel.

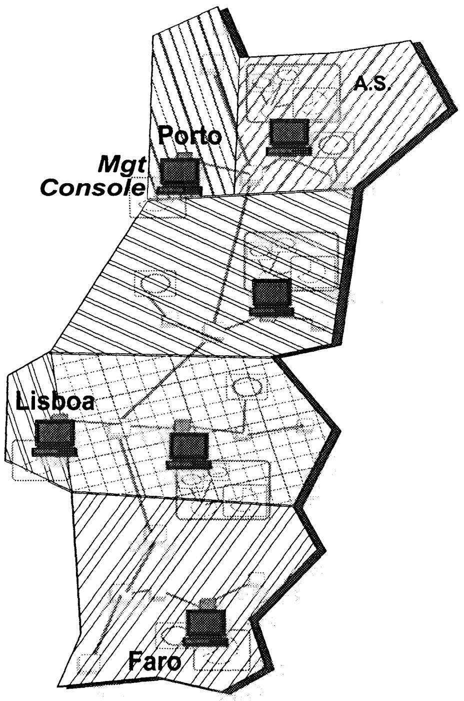


Figure 25.1. Uncoordinated Management

25.2 TOWARDS INTEGRATED MANAGEMENT

Integrated management is the best suited model to pursue the strategy underlined in the previous section. Given the geographical dispersion of the company, an integrated management platform should be selected that allows to perform remote management on all managed resources in the company domain, composed of all facilities interconnected by the secure tunnels.

If the platform supports it, a composite management structure would prove quite effective in this system: hierarchical management, with mid-level agents located in the Gateway Facilities, each acting on the managed resources of their facility, and responding to the platform manager console above; and cooperative management among those mid-level agents.

The desired setting is shown in Figure 25.1a. The platform and its services are installed in the main facility at Porto, where the main management console is also installed. Given that important services also exist in Lisboa, a secondary management console is also installed there. The detail of the hierarchy to the inside of each facility is omitted in the drawing. All equipment should comply with the standard management communication protocol selected (e.g., SNMPv2, migrating to SNMPv3 a.s.a.p.), and with the standard MIB formats, such as Internet MIB-II and RMON2 MIB.

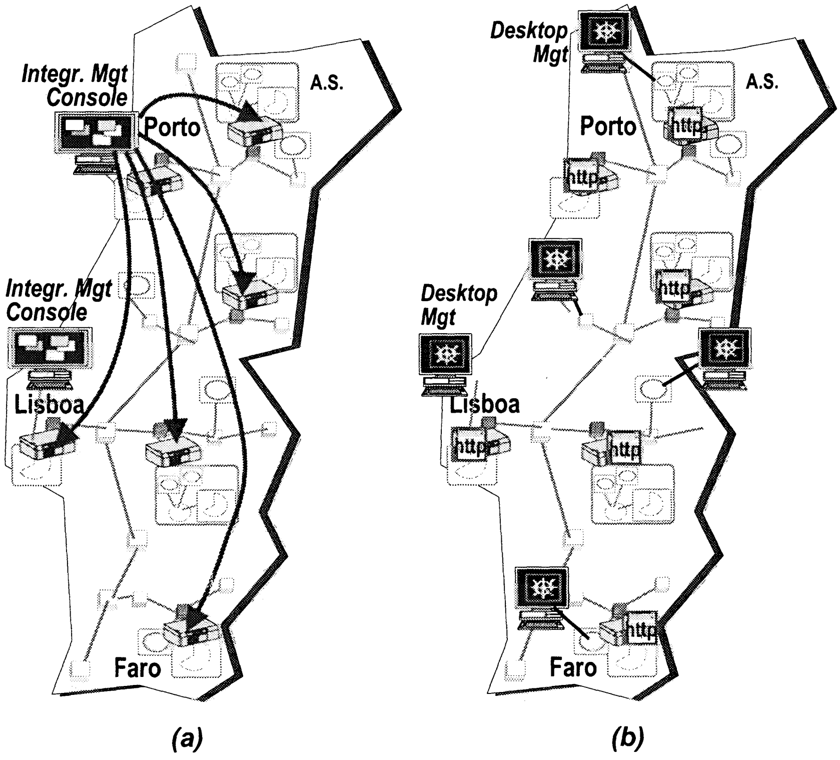


Figure 25.2. (a) Integrated Management; (b) Desktop Management

In a later phase, after the integrated management concept has stabilized, and the informatics culture of the company is more mature, an evolution towards desktop management may be envisaged, as depicted in Figure 25.1b. Most of the current structure may remain. The integrated management agents and middle managers will be provided with HTTP servers. Many emerging equipments are already provided with individual web servers allowing web-based management. This evolution should be made as compatible as possible with the emerging DMI standard. This will allow a moderate but desirable decentralization of management, specially low-level local functions, since a desktop with a browser can virtually manage any equipment, depending on the access control capabilities of the user.

Further Issues

These issues need some refinement now, and the reader was assigned the study of a few questions that were still left unsolved:

Q.5. 11 *Select the actual tools that should equip a platform managing a system like VP'63.*

Q.5. 12 *Define a minimal structure for an enterprise-wide Help Desk and Trouble Ticket System.*

Q.5. 13 *Monitoring is addressed both as an industrial system (SCADA) function, and as a management function. Can they be aggregated or do they have different characteristics?*

References

Abadi, M. and Needham, R. (1994). Prudent engineering practice for cryptographic protocols. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy.*

Abdelzaher, T. and Shin, K. (1998). End-host architecture for qos-adaptive communication. In *Procs. of the 4th IEEE Real-Time Technology and Applications Symposium*, Denver, USA.

Abrams, M. (1998). *World Wide Web, Beyond the Basics.* Prentice Hall.

Abrams, M., Jajodia, S., and Podell, H., editors (1995). *Information Security.* IEEE CS Press.

ADA 83 (1983). Ans reference manual for the ada programming language. Technical Report Mil/Std 1815A-1983, American National Standards Institute.

Agnew, B., Hofmeister, C., and Purtilo, J. (1994). Planning for change. *IEEE/ IOP/BCS Distributed Systems Engineering Journal*, 1(5):313–322.

Agrawal, D. and El-Abbadi, A. (1990). Efficient techniques for replicated data management. In *Proceedings of the IEEE Workshop on the Management of Replicated Data*, pages 48–52, Houston, USA.

Agrawal, D. and El-Abbadi, A. (1991). An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Transaction on Computer Systems.*

Aguilera, M., Chen, W., and Toueg, W. (1998). Failure detection and consensus in the crash-recovery model. In *Proc. 12th Int. Symposium on DIStributed Computing (formerly WDAG)*, pages 231–245, Andros, Greece. Sringer-Verlar LNCS 1499.

Ahamad, M. and Ammar, M. (1991). Multidemensional voting. *ACM Transactions on Computer Systems*, 9(4):339–431.

Ahamad, M., Hutto, P., and John, R. (1991). Implementing and programming causal distributed shared memory. In *Proceedings of the 11th IEEE International Conference on Distributed Computing Systems*, pages 274–281, Arlington, Texas, USA.

Aidarous, S. and Plevyak, T. (1998). *Telecommunications Network Management - Technologies and Implementations.* IEEE Press.

Almeida, C. and Veríssimo, P. (1996). Timing failure detection and real-time group communication in *quasi-synchronous* systems. In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*, L'Aquila, Italy.

Alpern, B. and Schneider, F. (1987). Recognizing safety and liveness. *Distributed Computing*, (2):117–126.

Alvisi, L., Elnozahy, E., Rao, S., Husain, S., and DeMel, A. (1999). An analysis of communication induced checkpointing. In *Digest of Papers, The 29th IEEE International Symposium on Fault-Tolerant Computing*, Madison - USA.

Alvisi, L. and Marzullo, K. (1993). Non-blocking and orphan-free message logging protocols. In *Digest of Papers, The 23rd IEEE International Symposium on Fault-Tolerant Computing*, pages 145–154, Toulouse, France.

Ames, S., Gasser Jr., M., and Schell, R. (1983). Security kernel design and implementation: An introduction. *Computer*, 16(7):14–22.

Amir, Y., Dolev, D., Kramer, S., and Malki, D. (1992). Membership algorithms for multicast communication groups. In *Proceedings of the 6th International Workshop on Distributed Algorithms*, pages 292–312, Haifa, Israel.

Amir, Y., Dolev, D., Kramer, S., and Malki, D. (1993a). Transis: A communication sub-system for high-availability. In *Digest of Papers, The 22nd IEEE Int. Symp. on Fault-Tolerant Computing Systems*, pages 76–84.

Amir, Y., Moser, L., Melliar-Smith, P., Agarwal, D., and Ciarfella, P. (1993b). Fast message ordering and membership using a logical token-passing ring. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 551–560, Pittsburgh, Pennsylvania, USA.

Ananda, A., Tay, B., and Koh, E. (1992). A survey of asynchronous remote procedure calls. *Operating Systems Review*, 266(2):92.

Anderson, J. (1972). Computer security technology planning study. Technical Report ESD-TR-73-51, Hanscom AFB.

Anderson, R. and Needham, R. (1995). Robustness principles for public key protocols. In *Proceedings of the Advances in Cryptology- CRYPTO'95*. Springer-Verlag.

ANSA (1987). *ANSA Reference Manual, Release 00.03.* Advanced Networked Systems Architecture, ESPRIT technical week edition.

ANSA (1990). *ANSAware Release 3.0 Reference Manual.* APM Ltd, Cambridge.

ANSI X9.9 (1986). *American National Standard for Financial Institution Message Authentication (Wholesale).*

Arlat, J., Aguera, M., Crouzet, Y., Fabre, J., Martins, E., and Powell, D. (1990). Fault injection for dependability validation: a methodology and some applications. *IEEE Trans. on SW Engineering*, Special Issue of Experimental Computer Science.

ASN.1 (1990). *Information Technology – Open Systems Interconnection – Specification of Abstract Notation One (ASN.1).* ISO/IEC.

Audsley, N. (1993). *Flexible Scheduling of Hard Real-Time Systems.* PhD thesis, University of York, UK.

Aurrecoechea, C., Campbell, A., and Hauw, L. (1998). A survey of QoS architectures. *Multimedia Sys. Journal, Special Issue on QoS Arch.*, 6(3):138–151.

Babaoğlu, m. (1987). On the reliability of consensus-based fault-tolerant distributed computing systems. *ACM Transactions on Computer Systems*, 5(3):394–416.

Babaoğlu, m., Baker, M., Davoli, R., and Giachini, L.-A. (1994). RELACS: A communications infrastructure for constructing reliable applications in large-scale distributed systems. In *Procs. of the 28th Hawaii Int'l Confer. on System Sciences.*

Babaoğlu, m. and Marzullo, K. (1993). Consistent global states of distributed systems: Fundamental concepts and mechanisms. In Mullender, S., editor, *Distributed Systems (2nd edition)*, chapter 4. Addison-Wesley.

Babaoğlu, O., Bartoli, A., and Dini, G. (2000). Programming partition-aware network applications. In Krakowiak, S. and Shrivastava, S., editors, *Recent Advances in Distributed Systems*, volume 1752 of *LNCS*, chapter 8. Springer-Verlag.

Babaoğlu, O., Drummond, R., and Stephenson, P. (1986). The impact of communication network properties on reliable broadcast protocols. *IEEE Transactions on Software Engineering*, (6):212–217.

Baker, S. (1997). *Corba Distributed Objects : Using Orbix.* Number ISBN: 0201924757. Addison-Wesley.

Bal, H. and Tanenbaum, A. (1988). Distributed programming with shared data. In *Procs. of the IEEE Conf. on Computer Languages*, pages 82–91.

Baldi, M., Gai, S., and Picco, G.-P. (1997). Exploiting code mobility in decentralized and flexible network management (ma'97). In *Proceedings of the 1st International Workshop on Mobile Agents 97*, pages 13–26, Berlin, Germany. Springer, Lecture Notes on Computer Science vol. 1219.

Banker, K. and Mellquist, P. (1995). Snmp++: A portable object-oriented approach to network management programming. *ConneXions*, 9(3):35–41.

Barabanov, M. and Yodaiken, V. (1997). Real-time linux. *Linux Journal.*

Barbacci, M., Weinstock, C., Doubleday, D., Gardner, M., and Lichota, R. (1993). Durra: A structure description language for developing distributed applications. *Sofware Engineering Journal*, 8(2):83–94.

Barborak, M., Malek, M., and Dahbura, A. (1993). The consensus problem in fault-tolerant computing. *ACM Computing Surveys*, 25(2):171–220.

Barrett, P., Bond, P., Hilborne, A., Rodrigues, L., Seaton, D., Speirs, N., and Veríssimo, P. (1990). The Delta-4 Extra performance architecture (XPA). In *Digest of Papers, The 20th IEEE International Symposium on Fault-Tolerant Computing*, Newcastle-UK. also as INESC AR/21-90.

Bartlett, J., Gray, J., and Horst, B. (1987). Fault tolerance in Tandem computer systems. In Avizienis, A., Kopetz, H., and Laprie, J., editors, *Dependable Computing and Fault-Tolerant Systems*, volume 1, pages 55–76. Springer-Verlag.

Bauer, A., Bowden, R., Browne, J., Duggan, J., and Lyons, G. (1991). *Shop Floor Control Systems.* Chapman Hall, London.

Becker, L., Pereira, C., Dias, O., Teixeira, I., and Teixeira, J. (2000). Mosys a methodology for automatic object identification from system specification. In *Procs. of ISORC 2000, the 3rd IEEE Int'l Symp. on Object-Oriented Real-Time Distributed Computing*, pages 198–201, Newport Beach, USA.

Beekmann, D. (1989). CIM-OSA: Computer integrated manufacturing - open system architecture. *Int'l Journal Computed Integrated Manufacturing.*

Beertema, P. (1993). Common dns data file configuration errors. Technical Report RFC 1537, USc Inf. S. Inst.

Bell, D. and LaPadula, L. (1973). Secure computer systems: Mathematical foundations and model. Technical report, MITRE Corp.

Bellissard, L., Atallah, S., Boyer, F., and Riveill, M. (1996). Distributed application configuration. In *Proceedings of the 16th IEEE International Conference on Distributed Computing Systems*, pages 579–585, Hong-Kong.

Bellovin, S. and Merritt, M. (1992). Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Computer Society Conference on Research in Security and Privacy*, pages 72–84.

Ben-Ari, M. (1990). *Principles of Concurrent and Distributed Programming.* Prentice Hall.

Berndtsson, M. and Hansson, J. (1995). Issues in active real-time databases. In *Proceedings of the First ACM International Workshop on Active and Real-Time Database Systems*, pages 142–157, Skovde, Sweden.

Berners-Lee, T. and Cailliau, R. (1990). Worldwideweb: Proposal for a hypertext project. Technical report.

Bernstein, P., Hadzilacos, V., and Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems.* Addison-Wesley.

Bershad, B. and Zekauskas, M. (1991). Midway: Shared memory paralel programming with entry consistency for distributed memory multiprocessors. Technical Report CMU-CS-91-170, Carnegie-Mellon University.

Bhide, A., Elnozahy, E., and Morgan, S. (1991). A highly available network file server. In *Proceedings of the USENIX Winter Conference*, pages 199–205.

Biba, K. (1977). Integrity considerations for secure computer systems. Technical Report 76-372, U.S. Air Force Electronic Systems Division.

Birman, K., editor (1996). *Building Secure and Reliable Network Applications.* Number ISBN 1-884777-29-5. Manning Publications Co.

Birman, K. and Joseph, T. (1987). Reliable Communication in the Presence of Failures. *ACM, Transactions on Computer Systems*, 5(1).

Birman, K., Schiper, A., and Stephenson, P. (1991a). Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3).

Birman, K., Schiper, A., and Stephenson, P. (1991b). Lightweight Causal and Atomic Group Multicast. *ACM Transacs. on Computer Systems*, 9(3):272–314.

Birman, K. and van Renesse, R., editors (1994). *Reliable Distributed Computing With the ISIS Toolkit.* Number ISBN 0-8186-5342-6. IEEE CS Press.

Birrell, A. and Nelson, B. (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1).

Bloom, J. and Dunlap, K. (1986). Experiences implementing bind, a distributed name server for the darpa internet. In *USENIX Summer*, pages 172–181.

Bodilsen, S. (1994). Scheduling theory and ada 9x. *Embedded Systems Programming*, pages 32–52.

Boehm, B. (1988). A spiral model of software development and enhancement. *IEEE Computer*, pages 61–72.

Boly, J., Bosselaers, A., Cramer, R., Michelsen, R., Mjolsnes, S., Muller, F., Pedersen, T., Pfitzmannn, B., de Rooji, P., Schoenmakers, B., Schunter, M., Vallée, L., and Waidner, M. (1994). The esprit project cafe – high security digital payment system. In *Proceedings of the Third ESORICS, European Symposium on Research in Computer Security*, pages 217–230. Springer-Verlag, Vol.875.

Boorstin, D. (1983). *The Discoverers.* Gradiva/Random House.

Bowen, N., Antognini, J., Regan, R., and Matsakis, N. (1997a). Availability in parallel systems: automatic process restart. *IBM Systems Journal*, 36(2):284–300.

Bowen, N., Elko, D., Isenberg, J., and Wang, G. (1997b). A locking facility for parallel systems. *IBM Systems Journal*, 36(2):202–220.

Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., and Yovine, S. (1998). Kronos: a model-checking tool for real-time systems. In *Proceedings of CAV'98, the 10th IEEE Conference Computer-Aided Verification.*

Bradley, D., Dawson, D., Burd, N., and Loader, A., editors (1991). *Mechatronics, Electronics in Products and Processes.* Chapman and Hall.

Brands, S. (1995). Electronic cash on the internet. In *Proceedings of the Internet Society 1995 Symposium on Network and Distributed Systems Security*, pages 64–84. IEEE Computer Society Press.

Brewer, E., Gauthier, P., Goldberg, I., and Wagner, D. (1995). Basic flaws in internet security and commerce. Technical report, Dpt. of CS, Berkeley University.

Brites, A., Simões, P., Leitão, P., Monteiro, E., and Fernandes, F. (1994). A high-level notation for the specification of network management applications. In *Proceedings of the INET'94/JENC5*, pages 5611–8.

Budhiraja, N., Marzullo, K., Schneider, F., and Toueg, S. (1993). The primary-backup approach. In Mullender, S., editor, *Distributed Systems, 2nd Edition*, ACM-Press, chapter 8. Addison-Wesley.

Burns, A. and Welling, A. (1996). Advanced fixed priority scheduling. In Joseph, M., editor, *Real-Time Systems.* Prentice-Hall.

Burns, A. and Wellings, A. (1995). *Hard Real-Time HOOD: A Structured Design Method for Hard Real-Time Ada Systems.* Elsevier.

Burns, A. and Wellings, A. (1996). *Real-Time Systems and Programming Languages.* International Computer Science Series. Addison-Wesley.

Buttazzo, G., editor (1997). *Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications.* Kluwer Academic Publishers.

Callahan, J. and Montgomery, T. (1996). Approaches to verification and validation of a reliable multicast protocol. *ACM Software Engeneering Notes*, 21(3).

CAN (1993). *Int'l Std.11898- Road vehicles - Interchange of digital information - Controller Area Network (CAN) for high-speed communication.* ISO.

Carreira, J., Madeira, H., and Silva, J. (1998). Xception: A technique for the experimental evaluation of dependability in modern computers. *Transactions on SW Engineering*, 24(2):125–136.

Carriero, N. and Gelertner, D. (1986). The S/Net's Linda Kernel. *ACM Transactions on Computer Systems*, 4(2).

Cart, M., Ferrie, J., and Mardyanto, S. (IFIP, 1987). Atomic broadcast protocol, preserving concurrency for an unreliable broadcast network. In Cabanel, J., Pujole, G., and Danthine, A., editors, *Local communication systems: LAN and PBX.* North-Holland.

Carter, J., Bennettt, J., and Zwanepoel, W. (1991). Implementation and performance of Munin. In *Proceedings of the 13th ACM Symposium on Operating System Principles*, pages 152–164.

Carter, W. and Schneider, P. (1968). Design of dynamically checked computers. In *Proc. IFIP'68 World Computer Congress*, pages 878–883.

CC-ITSE (1998). *Common Criteria for Information Technology Security Evaluation.* ISO/IEC JTC 1.

Chandra, T., Hadzilacos, V., and Toueg, S. (1996). On the impossibility of group membership. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC '96)*, pages 322–330.

Chandra, T. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 34(1):225–267.

Chandy, K. and Lamport, L. (1985). Distributed snapshots: Determining global states of distributed systems. *ACM*, 3(1):63–75.

Chang, J. and Maxemchuck, N. (1984). Reliable broadcast protocols. *ACM, Transactions on Computer Systems*, 2(3):251–273.

Chappell, D. (1992). The osf distributed management environment. *ConneXions*, 6(10):10–15.

Chaum, D. (1983). Blind signatures for untraceable payments. In *Proceedings of the Advances in Cryptology- Crypto'82*, pages 199–203. Plenum Press.

Chaum, D. (1992). Achieving electronic privacy. *Scientific American*, 267(2):96–101.

Chen, L. and Avizienis, A. (1978). N-version programming: A fault-tolerance approach to reliability of software operation. *8th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-8)*, pages 3–9.

Chen, W., Toueg, S., and Aguilera, M. (2000). On the quality of service of failure detectors. In *Procs. of DSN 2000, the IEEE/IFIP Int'l Conf. on Dependable Systems and Networks.*

Chereque, M., Powell, D., Reynier, P., Richier, J.-L., and Voiron, J. (1992). Active replication in Delta-4. In *Digest of Papers, The 22nd IEEE Int'l Symp. on Fault-Tolerant Computing Systems*, page 28.

Cheriton, D. and Mann, T. (1989). Decentralizing a global naming service for improved performance and fault tolerance. *ACM Transactions on Computer Systems*, 7(2):147–183.

Cheriton, D. and Skeen, D. (1993). Understanding the limitations of causally and totally ordered communication. In *Proceedings of the 14th Symposium on Operating Systems Principles*, Asheville, NC, USA.

Cheriton, D. and Zwaenepoel, W. (1985). Distributed process groups in the V-kernel. *ACM Tran. on Computer Systems*, 3(2).

Cheswick, W. and Bellovin, S. (1997). *Internet Security: Firewalls and Gateways, 2nd edition.* Addison-Wesley.

Cheung, S., Ammar, M., and Ahamad, M. (1990). The grid protocol: A high performance scheme for maintaining replicated data. In *Proceedings of the 6th Internation Conference on Data Engineering*, pages 438–445.

Chung, S., Lazowska, E., Notkin, D., and Zahorjan, J. (1989). Performance implications of design alternatives for remote procedure call stubs. In *Proceedings of the 9th Int'l IEEE Conference on Distributed Computing Systems*, pages 36–41, Newport Beach - USA.

Clark, D. and Wilson, D. (1987). A comparison of commercial and military computer security policies. In *Proc. of the IEEE Symp. on Security and Privacy*, pages 184–194.

Clegg, M. and Marzullo, K. (1996). Clock synchronization in hard real-time distributed systems. Technical Report CS96-478, University of California, San Diego, Department of Computer Science and Engineering.

CMIP (1988). *Open Systems Interconnection – Management Information Protocol Definition, Part 2: Common Management Information Protocol.* ISO.

CMISE (1988). *Open Systems Interconnection – Management Information Service Definition, Part 2: Common Management Information Service.* ISO.

CNMA (1993). CNMA Implementation Guide, Revision 6.01. Technical report, ESPRIT Project 7096.

Comer, D. (1991). *Internetworking With TCP/IP: Principles, Protocols, Architecture.* Prentice Hall.

Comer, D. (1997). *The Internet Book.* Prentice Hall.

Cooper, E. (1985). Replicated distributed programs. In *Procs. of the 10th ACM Symposium on Operating Systems Principles*, Berkeley– USA.

Corneillie, P., Deswarte, Y., Goodson, J., Hawes, A., Kaâniche, M., Kurth, H., Liebisch, G., Manning, T., Moreau, S., Steinacker, A., and Valentin, C. (1999). Squale– dependability assessment criteria (4th draft). Technical Report 98456, ACTS proj. AC097, LAAS, Squale Consortium.

Cornhill, D., Sha, L., Lehoczky, J., Rajkumar, R., and Tokuda, H. (1987). Limitations of ada for real-time scheduling. In *Proceedings of the ACM International Workshop on Real Time Ada Issues*, Ada Letters, pages 33–39.

Cosquer, F., Antunes, P., and Veríssimo, P. (1996). Enhancing dependability of cooperative applications in partitionable environments. In *Dependable Computing - EDCC-2*, volume 1150 of *LNCS*, pages 335–352. Springer-Verlag.

Crane, S., Dulay, N., Fossa, H., Kramer, J., Magee, J., Sloman, M., and Twidle, K. (1995). Configuration management for distributed software services. In *Proceedings of the 4th IFIP/IEEE Int'l Symposium on Integrated Network Management (ISINM'95)*, Santa Barbara, USA.

Cristian, F. (1988). Reaching agreement on processor group membership in synchronous distributed system. Technical Report RJ 5964 (59426), IBM Almaden Research Center.

Cristian, F. (1989). Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–148.

Cristian, F. (1990). Synchronous atomic broadcast for redundant broadcast channels. *The Journal of Real-Time Systems*, 2(1):195–212.

Cristian, F. (1994). Abstractions for fault-tolerance. In *Proceedings of the 13th IFIP World Computer Congress*, Hamburg.

Cristian, F., Dancey, B., and Dehn, J. (1996). Fault-tolerance in air traffic control systems. *ACM Transaction on Computer Systems.*

Cristian, F. and Fetzer, C. (1998). The timed asynchronous system model. In *Proceedings of the 28th IEEE Annual International Symposium on Fault-Tolerant Computing*, pages 140–149, Munich, Germany.

Cristian, F., H., A., Strong, R., and Dolev, D. (1985). Atomic broadcast: From simple message diffusion to byzantine agreement. In *Digest of Papers, The 15th IEEE International Symposium on Fault-Tolerant Computing*, Ann Arbor-USA.

Dana, P. (1996). Global positioning system (gps) time dissemination for real-time applications. *Journal of Real-Time Systems, this issue.*

Daneshgar, F. and Ray, P. (1997). Cooperative management based on awareness modelling. In *Proceedings of the 8th IFIP/IEEE Int'l Workshop on Distributed Systems Operations and Management (DSOM'97)*, Sydney, Australia.

Danzig, P., Obraczka, K., and Kumar, A. (1992). An analiysis of wide-area name server traffic. In *Proceedings of ACM SIGCOM 1992*, pages 281–292.

Davcev, D. (1989). A dynamic voting scheme in distributed systems. *IEEE Transactions on Software Engineering*, 15(1):93–97.

Debar, H., Dacier, M., and Wespi, A. (1999). A revised taxonomy for intrusion-detection systems. Technical Report 53, IBM Research, Zurich Research Laboratory.

Deering, S. (1989). Host extensions for ip multicasting. Technical Report RFC 1112, Stanford University, Stanford, CA, USA.

Deering, S., Estrin, D., Farinacci, D., Jacobson, V., and Liu, C-G. andWei, L. (1996). The PIM architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162.

Deitel, H. and Deitel, P. (2000). *Internet and World Wide Web: How to Program.* Number ISBN: 0130161438. Prentice Hall.

Denning, D. (1976). A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243.

Denning, D. and Sacco, G. (1981). Time-stamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536.

Deri, L. and Ban, B. (1997). Static vs. dynamic cmip/snmp network management using corba. In *Proceedings of the 4th Int'l Conference on Intelligence in Services and Networks (IS&N'97).*

DES (1977). *Data Encryption Standard.*

Deswarte, Y., Blain, L., and Fabre, J.-C. (1991). Intrusion tolerance in distributed systems. In *Proc. of the IEEE Symp. on Security and Privacy*, pages 110–121, Oakland - USA.

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654.

DigiCash (1994). World's first electronic cash payment over computer networks. Technical report, DigiCash Press Release.

Dolev, D., Dwork, C., and Stockmeyer, L. (1983). On the minimal synchronism needed for distributed consensus. *24th Annual IEEE Symp. on Foundations of Computer Science.*

Dolev, D., Kramer, S., and Malki, D. (1993). Early delivery totally ordered multicast in asynchronous environments. In *Digest of Papers, The 23th IEEE Int'l Symp. on Fault-Tolerant Computing*, pages 544–553, Toulouse, France.

Dolev, D. and Yao, A. (1981). On the security of public key protocols. In *Proc. of the 22nd Annual Symp. on the Foundations of Computer Science*, pages 350–357.

Drummond, R. and Babaoğlu, O. (1993). Low Cost Clock Synchronization. *Distributed Computing*, 6:193–203.

DSS (1994). *Digital Signature Standard.*

Dupuy, F., Nilsson, G., and Inoue, Y. (1995). The tina consortium: Toward networking telecommunications information services. *IEEE Communications Magazine*, pages 78–83.

Dwork, C., Lynch, N., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323.

EES (1994). *Escrowed Encryption Standard.*

ElGamal, T. (1985). A public-key cryptosystem and a signature scheme based on discrete logarithms. In *Proc. of Advances in Cryptology- CRYPTO'84*, pages 10–18. Springer-Verlag.

Ellison, C. and Schneier, B. (2000). Ten risks of pki: What you're not being told about public key infrastructure. *Computer Security Journal*, XVI(1).

Elnozahy, E., Alvizi, L., Wang, Y., and Johnson, D. (1999). A survey of rollback-recover protocols in message-passing systems. Technical Report CMU-CS-99-148, Carnegie Mellon University.

Elnozahy, E. and Zwaenepoel, W. (1992a). Manetho: Transparent Rollback-Recovery with Low Overhead, Limited Rollback and Fast Output Commit. *IEEE Transactions on Computers*, 41(5):526–531.

Elnozahy, E. and Zwaenepoel, W. (1992b). Replicated distributed process in Manetho. In *Digest of Papers, The 22nd IEEE International Symposium on Fault-Tolerant Computing Systems*, page 18.

Eppinger, J.and Mummert, L. and Spector, A. (1991). *Camelot and Avalon.* Morgan Kaufmann Publishers, Inc.

Eswaran, K., Gray, J., Lorie, R., and Traiger, I. (1976). The notions of consistency and predicate locks in a database system. *Communics. of the ACM*, 19(11):624–633.

Ezhilchelvan, P., Macedo, R., and Shrivastava, S. (1995). Newtop: A fault-tolerant group communication protocol. In *Proc. of the 15th IEEE Int'l Conference on Distributed Computing Systems*, pages 296–306, Vancouver, Canada.

Fabre, J., Nicomette, V., Pérennou, T., Stroud, R., and Wu, Z. (1995). Implementing fault tolerant applications using reflective object-oriented programming. In *Digest of Papers of the 25th IEEE International Symposium on Fault-Tolerant Computing Systems*, pages 489–498.

FDDI, X. (1986). *FDDI documents: Media Access Layer, Physical and Medium Dependent Layer, Station Mgt.*

Felber, P., Grabinato, B., and Guerraoui, R. (1996). The design of a CORBA group communication service. In *Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems*, pages 150–159, Niagara-on-the-Lake, Canada.

Felber, P., Guerraoui, R., and Schiper, A. (1997). Replicating objects using the corba event service. In *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 14–19, Tunis, Tunisia.

Felten, E., Balfanz, D., Dean, D., and Wallach, D. (1996). Web spoofing: an internet con game. Technical Report 540-96, Princeton University, Department of CS.

Fetzer, C. and Cristian, F. (1996). Fail-awareness in timed asynchronous systems. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC '96)*, pages 314–321, New York, USA.

Fetzer, C. and Cristian, F. (1997a). Fail-awareness: An approach to construct fail-safe applications. In *Proc. of the 27th IEEE Annual Int'l Fault-Tolerant Computing Symposium*, pages 282–291, Seattle, USA.

Fetzer, C. and Cristian, F. (1997b). Integrating external and internal clock synchronization. *Journal of Real-Time Systems*, 12(2).

FIP (1990). *General Purpose Field Communication System – Part 3, WorldFIP.*

Fischer, M., Lynch, N., and Paterson, M. (1985). Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32:374–382.

Fisher, T., editor (1990). *Batch Control Systems: Design, Application and Implementation.* ISA.

Fohler, G. (1995). Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In *Proceedings of RTSS'95, the IEEE Real-Time Systems Symposium*, pages 152–161, Pisa, Italy.

Forestier, J., Forarino, C., and Franci-Zannettacci, P. (1989). Ada++: A class and inheritance extension for ada. In *Procs. of the Ada-Europe Int'l Conf.*, Ada Companion Series, Madrid-Spain. Cambridge Univ. Press.

Fosså, H. (1997). *Interactive Configuration Management for Distributed Systems.* PhD thesis, University of London, Imperial College.

Friday, A., Davies, N., Blair, G., and Cheverst, K. (1999). Developing adaptive applications: The MOST experience. *Journal of Integrated Computer-Aided Engineering*, 6(2).

Fritzke Jr., U., Ingels, P., Moustefaoui, A., and Raynal, M. (1998). Fault-tolerant total order multicast to asynchronous groups. In *Proc. 17th IEEE Symp. on Reliable Distributed Systems*, pages 228–234, West Lafayette, USA.

Furht, B., Grostick, D., Gluch, D., Rabbat, G., P., J., and McRoberts, M. (1991). *Real-time Unix Systems Design and Application Guide.* Kluwer.

Garcia-Molina, H. (1982). Elections in distributed computer systems. *IEEE Transactions on Computers*, C-31(1):48–59.

Garcia-Molina, H. and Barbara, D. (1985). How to assign votes in distributed system. *Journal of the ACM*, 32(4):841–860.

Garcia-Molina, H. and Spauster, A. (1991). Ordered and reliable multicast communication. *ACM Transactions on Computers Systems*, 9(3):242–271.

Garfinkel, S. and Spafford, G. (1997). *Web Security and Commerce.* O'Reilly.

Gharachorloo, K., Lenoski, D., Laudon, J., Gibsons, P., Gupta, A., and Henessy, J. (1990). Memory consistency and event ordering in scalable shared-memory multiprocessors. In *Proceedings of the 17th International Symposium on Computer Architecture*, pages 15–26, Seattle, Washington, USA.

Gifford, D. (1979). Weighted Voting For Replicated Data. In *Proceedings of the Seventh ACM Symposium on Operating System Principles*, pages 150–162.

Golding, R. (1992). Weak consistent group communication for wide-area systems. In *Proceedings of the Second IEEE Workshop on the Management of Replicated Data*, pages 13–16, Monterey, California.

Goldszmidt, G. and Yemini, Y. (1995). Distributed management by delegation. In *Proc. of the 15th IEEE Int'l Conference on Distributed Computing Systems.*

Gollmann, D. (2000). On the verification of cryptographic protocols – a tale of two committees. *Electronic Notes in Theoretical Computer Science*, 32.

Gong, L. (1992). A security risk of depending on synchronized clocks. *Operating Systems Review*, 26(1):49–53.

Goodman, J. (1989). Cache consistency and sequencial consistency. Technical Report 61, SCI Commitee.

Gorur, R. and Weaver, A. (1988). Setting target rotation times in an IEEE Token Bus network. *IEEE Transactions on Industrial Electronics*, 35(3).

Graw, G., Herrmann, P., and Krumm, H. (2000). Verification of uml-based real-time system designs by means of ctla. In *Proceedings of ISORC 2000, the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 86–95, Newport Beach, USA.

Gray, C. and Cheriton, D. (1989). Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 202–210.

Gray, J. (1978). *Notes on Database Operating Systems*, volume 60 of *Lecture Notes in Computer Science*, pages 393–481. Springer-Verlag.

Gray, J. (1986). Why do computers stop and what can be done about it? In *Proceedings of the 5th IEEE Sycmp. on Reability in Distributed Software and Database Systems*, pages 3–12, Los Angeles, USA.

Gray, J. and Reuter, A. (1993). *Transaction processing: concepts and techniques.* Series in Data Management Systems. Morgan Kaufmann.

Guedes, P. and Castro, M. (1993). Distributed shared object memory. In *Proceedings of the Fourth IEEE Workshop on Workstation Operating Systems*, pages 142–149, Napa, California, USA.

Guerraoui, R., Hurfin, M., Mostefaoui, A., Oliveira, R., Raynal, M., and Schiper, A. (2000). Consensus in asynchronous distributed systems: a concise guided tour. In Krakowiak, S. and Shrivastava, S., editors, *Advances in Distributed Systems*, LNCS 1752, chapter 2, pages 33–47. Springer Verlag.

Guerraoui, R. and Schiper, A. (1997). Total order multicast to multiple groups. In *IEEE 17th Intl. Conf. Distributed Computing Systems*, pages 578–585.

Guo, K. and Rodrigues (1997). Dynamic light-weight groups. In *Proceedings of the 17th IEEE International Conference on Distributed Computing Systems (ICDCS'17)*, Baltimore, Maryland, USA.

Gusella, R. and Zatti, S. (1989). The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3bsd. *IEEE Transactions on Software Engineering*, 15(7):847–853.

Guy, R., Page, T., Heidemann, J., and Popek, G. (1990). Name transparency in very large scale distributed file systems. In *Proceedings of the IEEE Workshop on Experimental Distributed Systems*, pages 20–25, Huntsville, Alabama.

Haberman, S., Falciani, A., and Riggsby, M. (2000). *Mastering Lotus Notes and Domino R5 Premium Edition.* Number ISBN: 0782126359.

Hachiga, J. (1992). The Concepts and Technologies of Dependable and Real-time Computer Systems for Shinkansen Train Control. In *Proc. of the 2nd Int'l Workshop on Responsive Computer Systems*, Tokyo, Japan. Springer-Verlag.

Hadzilacos, V. and Toueg, S. (1994). A modular approach to the specification and implementation of fault-tolerant broadcasts. Technical Report TR94-1425, Department of Computer Science, Cornell University, Ithaca- USA.

Halpern, J. and Moses, Y. (1987). Knowledge and common knowledge in a distributed environment. Technical Report RJ4421, IBM Research Laboratory.

Halpern, J., Simons, B., Strong, R., and Dolev, D. (1984). Fault-Tolerant Clock Synchronization. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pages 89–102, Vancouver, Canada.

Halpern, J. and Suzuki, I. (1991). Clock synchronization and the power of broadcasting. *Distributed Computing*, 5(2):73–82.

Halsall, F. (1994). *Data Communications, Computer Networks and Open Systems, 3rd Ed.* Addison-Wesley.

Harper, R., Lala, J., and Deyst, J. (1988). Fault-tolerant parallel processor architecture overview. In *Digest of Papers, the 18th FTCS, IEEE Int'l Symp. on Fault-Tolerant Computing*, pages 252–257, Tokyo - Japan.

Hawking, S. (1988). *A Brief History of Time - from the Big Bang to Black Holes.* Gradiva.

Hayden, M. (1998). *The Ensemble System.* PhD thesis, Cornell University, Computer Science Department.

Hayes, S. (1993). Analyzing network performance management. *IEEE Communications Magazine*, pages 52–58.

Hedenetz, B. (1998). A development framework for ultra-dependable automotive systems based on a time-triggered architecture. In *Proceedings of RTSS'98, the IEEE Real-Time Systems Symposium*, pages 358–367, Madrid, Spain.

Hegering, H.-G. and Abeck, S. (1994). *Integrated Network and System Management.* Addison-Wesley.

Heiner, G. and Thurner, T. (1998). Time-triggered architecture for safety-related distributed real-time systems in transportation systems. In *Digest of Papers, The 28th IEEE Int'l Symp. on Fault-Tolerant Computing Systems*, Munich, Germany.

Henning, M. and Vinoski, S. (1999). *Advanced CORBA Programming with C++.* Number ISBN: 0201379279. Addison-Wesley.

Herlihy, M. and Wing, J. (1990). Linearizability: a correctness condition for concurrent objects. *ACM Transac. on Programming Languages and Systems*, 12(3):463–492.

Hiltunen, M. and Schlichting, R. (1993). An approach to constructing modular fault-tolerant protocols. In *Proceedings of the 12th IEEE Symposium on Reliable Distributed Systems*, pages 105–114, Princeton, New Jersey.

Hiltunen, M. and Schlichting, R. (1994). Properties of membership services. Technical report, University of Arizona, Department of Computer Science, Tucson, AZ.

Hong, J., Kim, J.-S., and Park, J.-K. (1999). A corba-based quality of service management framework for distributed multimedia services and applications. *IEEE Network*, 13(2):70–79.

Hong, J., Kong, J.-Y., Yun, T.-H., and Kim, J.-S. (1997). Web-based intranet services and network management. *IEEE Communic's Magazine*, 35(10):100–110.

Hood, C. and Ji, C. (1996). Probabilistic network fault detection. In *Proceedings of the IEEE Globecom'96*, pages 1872–1876, London, UK.

Hopkins, A., Smith, T., and Lala, J. (1978). FTMP - A highly reliable fault-tolerant multiprocessor for aircraft. *Proceedings of IEEE*, 66(10):1221–1239.

Hughes, D. (1993). Esl-a script language for snmp (and then some!). Technical report, Bond University.

Hutchison, D., Coulson, G., Campbell, A., and Blair, G. (1994). Quality of service management in distributed systems. In Sloman, M., editor, *Network and Distributed Systems Management*, chapter 11. Addison-Wesley.

IEEE-RT (1994). Special issue on real-time systems. In *Procs. of the IEEE.*

ISO10040 (1992). *Information Technology – Open Systems Interconnection – Systems Management Overview.* ISO/IEC.

ISO10164 (1992). *Information Technology – Open Systems Interconnection – Systems Management Functions.* ISO/IEC.

ISO10165 (1992). *Information Technology – Open Systems Interconnection – Structure of Management Information.* ISO/IEC.

ISODE (1993). *ISODE Volume 1: Overview of ISODE.* England.

Issarny, V. (1993). An exception handling mechanism for parallel object-oriented programming: Towards reusable, robust distributed software. *Journal of Object-Oriented Programming*, 6(6):29–40.

ITSEC (1991). *Information Technology Security Evaluation Criteria, Ver. 1.2.*

Iyer, V. and Joshi, S. (1985). FDDI's 100 mb/s protocol improves on 802.5 specs 4mb/s limit. *EDN.*

Jahanian, F. and MoranJr, W. (1992). Strong, weak, and hybrid group membership. In *Proceedings of the Second IEEE Workshop on the Management of Replicated Data*, pages 34–38, Monterey, California.

Jalote, P. (1994). *Fault Tolerance in Distributed Systems.* Prentice-Hall.

Janetzky, D. and Watson, K. (1986). Token bus performance in MAP and PROWAY. In *Proceedings of the IFAC Workshop on Distributed Computer Protocol Systems.*

Jeffay, K. (1993). The real-time producer/consumer paradigm: A paradigm for construction of efficient, predictable real-time systems. In *Procs. of the ACM/SIGAPP Symp. on Applied Computing*, Indianapolis, USA.

Jeffay, K., Stanat, D., and Martel, C. (1991). On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of RTSS'91, the 12th IEEE Real-Time Systems Symposium*, pages 129–139.

Jensen, E. (2000). A proposed initial approach to distributed real-time java. In *Procs. of ISORC 2000, the Third IEEE Int'l Symp. on Object-Oriented Real-Time Distributed Computing*, pages 2–6, Newport Beach, USA.

Jensen, E. and Northcutt, J. (1990). Alpha: A non-proprietary os for large, complex, distributed real-time systems. In *Procs. of the IEEE Workshop on Experimental Distributed Systems*, pages 35–41, Alabama, USA.

Kaashoek, M. and Tanenbaum, A. (1991). Group communication in the Amoeba distributed operating system. In *Procs. of the 11th IEEE Int'l Conference on Distributed Computing Systems*, pages 222–230, Arlington, USA.

Kahn, D. (1967). *The Codebreakers: The Story of Secret Writing.* Macmillan Publishing Co.

Kaiser, J. and Livani, M. (1998). Invocation of real-time objects in a can-bus system. In *Proceedings of the ISORC 1998, IEEE International Symposium on Object-Oriented Real-Time Distributed Computing.*

Kaiser, J. and Mock, M. (1999). Implementing the real-time publisher/subscriber model on the controller area network. In *Procs. of the ISORC 1999, IEEE Int'l Symp. on Object-Oriented Real-Time Distributed Computing.*

Kaliski, B. (1993). A survey of encryption standards. *IEEE Micro*, 13(6):74–81.

Kalogeraki, V., Melliar-Smith, P., and Moser, L. (2000). Dynamic scheduling for soft real-time distributed object systems. In *Proceedings of ISORC 2000, the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 114–121, Newport Beach, USA.

Kaufman, C., Perlman, R., and Speciner, M. (1995). *Network Security, Private Communication in a Public World.* Prentice-Hall.

Kemme, B., Pedone, F., Alonso, G., and Schiper, A. (1999). Processing transactions over optimistic atomic broadcast protocols. In *Procs. of the 19th IEEE Int' l Conference on Distributed Computing Systems*, pages 424–431, Austin, USA.

Kent, S. and Atkinson, R. (1998). Security architecture for the internet protocol. Technical Report Request for Comments 2401, IETF.

Kieckhafer, R., Walter, C., Finn, A., and Thambidurai, P. (1988). The MAFT architecture for distributed fault tolerance. *IEEE Trans. on Computers*, 37(4).

Kim, K. and You, J. (1990). A highly decentralized implementation model for the programmer-transparent coordination (ptc) scheme for cooperative recovery. In *Digest of Papers, 20th IEEE Intl. Symposium on Fault-Tolerant Computing Systems*, pages 282–289, Newcastle - England.

Koch, T. and Kramer, B. (1995). Toward a comprehensive distributed systems management. *Open Distributed Processing*, pages 259–270.

Koops, B.-J. (1999). Crypto law survey. Technical Report Version 14.3.

Kopetz, H. (1992). Sparse Time versus Dense Time in Distributed Systems. In *Proc. of the 12th IEEE Int'l Conf. on Distributed Computing Systems*, Yokohama, Tokyo.

Kopetz, H. (1997). *Real-Time Systems.* Kluwer.

Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C., and Zainlinger, R. (1989a). Distributed fault-tolerant real-time systems: The Mars approach. *IEEE Micro*, pages 25–41.

Kopetz, H. and Grunsteidl, G. (1993). TTP - a Time-Triggered Protocol for Fault-Tolerant Real-Time Systems. In *Digest of Papers, The 23rd IEEE Int'l Symp. on F/T Computing*, Toulouse, France.

Kopetz, H., Grunsteidl, G., and Reisinger, J. (1989b). Fault-tolerant Membership Service in a Synchronous Distributed Real-time System. In *Proceedings of the IFIP WG10.4 Int'l Working Conference on Dependable Computing for Critical Applications*, Sta Barbara - USA.

Kopetz, H. and Ochsenreiter, W. (1987). Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, C-36(8):933–940.

Kopetz, H. and Veríssimo, P. (1993). Real-time and Dependability Concepts. In Mullender, S., editor, *Distributed Systems, 2nd Edition*, ACM-Press, chapter 16, pages 411–446. Addison-Wesley.

Korth, H., Soparkar, N., and Silberschatz, A., editors (1996). *Time-Constrained Transaction Management: Real-time Constraints in Database Transaction Systems.* Kluwer Academic Publishers.

Koymans, R. (1990). Specifying real-time properties with metric temporal logic. *Journal of Real-Time-Systems*, 2(4):255–299.

Kronenberg, N., Levy, H., and Strecker, W. (1987). Vaxclusters: A closely-coupled distributed system. *ACM Trans. on Computer Systems*, 4(3):130–146.

Kumar, A. and Cheung, S. (1991). A high availability $\sqrt{N}$ hierarquical grid algorithm for replicated data. *Information Processing Letters*, (40):311–316.

Ladin, R., Liskov, B., Shrira, L., and Ghemawat, S. (1992). Providing high availability using lazy replication. *ACM Transactions on Computer Systems*, 10(4):360–391.

Lai, X. (1992). *On the Design and Security of Block Ciphers.* ETH Series in Information Processing, Vol.1. Konstanz Hartung-Gorre Verlag.

Lamb, J. and Lew, P. (1996). *Lotus Notes Network Design : For Notes Release 3 and 4.* Computer Communications. McGraw-Hill.

Lamport, L. (1978a). The implementation of reliable distributed multiprocess systems. *Computer Networks 2*, (1978):95–115.

Lamport, L. (1978b). Time, Clocks and the Ordering of Events in a Distributed System. *CACM*, 21(7):558–565.

Lamport, L. (1979). How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transacs. on Computers*, 28(9):690–691.

Lamport, L. (1981). Password identification with insecure communications. *Communications of the ACM*, 24(11):770–772.

Lamport, L. (1984). Using Time Instead of Timeout for Fault-Tolerant Distributed Systems. *ACM Transactions on Prog. Lang. and Systems*, 6(2).

Lamport, L. (1994). The temporal logic of actions. *ACM Transactions on Programming Language and Systems*, 16(3).

Lamport, L. and Melliar-Smith, P. (1985). Synchronizing Clocks in the Presence of Faults. *Journal of the ACM*, 32(1):52–78.

Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Transactions on Prog. Lang. and Systems*, 4(3).

Lampson, B. (1974). Protection. *Operating Systems Review*, 8(1):18–24.

Lampson, B. (1981). Atomic transactions. In *Distributed Systems - Architecture and Implementation: An Advanced Course*, volume 105 of *Lecture Notes in Computer Science*, chapter 11, pages 246–265. Springer-Verlag.

Lampson, B. (1993). Authentication in distributed systems. In Mullender, S., editor, *Distributed Systems, 2nd Edition*, ACM-Press, chapter 21. Addison-Wesley.

Lampson, B., Abadi, M., and Wobber, E. (1992). Authentication in distributed systems: Theory and practice. *ACM Trans. on Computer Systems*, 10(4):265–310.

Langsford, A. (1994). Osi management model and standards. In Sloman, M., editor, *Network and Distributed Syst. Management*, chapter 4. Addison-Wesley.

Laplante, P. (1997). *Real-time systems design and analysis: an engineer's handbook, 2nd Edition.* IEEE Press.

Laprie, J. (1987). Dependability: A Unifying Concept for Reliable Computing and Fault-Tolerance. In *Resilient Computing Systems*, volume 2. Collins and Wiley.

Laprie, J.-C. (1992). Dependability: A unifying concept for reliable, safe, secure computing. In *IFIP Congress*, volume 1, pages 585–593.

Laprie, J.-C., editor (1998). *Dependability Handbook*, volume Report Nr.98-346 of *Laboratory for Dependability Engineering.* LAAS.

Lauer, H. and Satterwaite, E. (1979). The impact of mesa on systems design. In *Procs. of the 4th IEEE Int'l Conf. on Software Engineering*, pages 174–182.

Le Lann, G. and Rivière, N. (1993). Real-time communications over broadcast networks: the CSMA-DCR and the DOD-CSMA-CD protocols. Technical Report 1863, INRIA.

Lea, D. (1997). *Concurrent Programming in Java. Design Principles and Patterns.* Addison-Wesley.

Lee, P. and Anderson, T. (1990). *Fault-Tolerant: Principles and Practice, Second Edition.* Springer-Verlag.

Lehoczky, J. (1998). Scheduling communication networks carrying real-time traffic. In *Procs. of RTSS'98, the 19th IEEE Real-Time Systems Symp.*

Leiner, B., Cerf, V., Clark, D., Kahn, R., Kleinrock, L., Lynch, D., Postel, J., Roberts, L., and Wolff, S. (1997). The past and future history of the internet. *Communications of the ACM*, 40(2):102–108.

Leinwand, A. and Conroy, K. (1996). *Network Management: a Practical Perspective.* UNIX and Open Systems. Addison-Wesley.

Leslie, I., McAuley, D., Black, R., Roscoe, T., Barham, P., Evers, D., Fairbairns, R., and Hyden, E. (1996). The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas in Communication*, 14(7).

Leung, J. and J., W. (1982). On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250.

Levine, P. (1987). The apollo domain distributed file system. In *Theory and Practice of Distributed Operating Systems.* Springer Verlag, NATO ASI Series.

Lewis, L. and Dreo, G. (1993). Extending trouble ticket systems to fault diagnostics. *IEEE Network*, 7(6):44–51.

Li, K. and Hudak, P. (1989). Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359.

Lin, J. and Paul, S. (1996). RMTP: a reliable multicast transport protocol. In *Proceedings of the IEEE INFOCOM'96*, pages 1414–1424.

Linn, J. (1993). Privacy enhancement for internet electronic mail: Part I-message encipherment and authentication procedures. Technical Report RFC1421, IETF.

Linn, J. (1996). Generic security service application programming interface (GSS-API), version 2. Technical Report Internet Draft, IETF.

Liskov, B. (1985). The ARGUS language and system. In *Distributed Systems, Methods and Tools for Specification*, volume 190 of *LNCS*. Springer-Verlag.

Liskov, B., Castro, M., Shrira, L., and Adya, A. (1999). Providing persistent objects in distributed systems. In *Proceedings of the ECOOP'99 - Object Oriented Programming*, number 1628 in Lecture Notes in Computer Science, pages 230–257, Lisbon, Portugal. Springer-Verlag.

Liskov, B., Ghemawat, S., Gruber, R., Johnson, P., and Shrira, L. (1992). Efficient recovery in harp. In *Proceedings of the Second IEEE Workshop on the Management of Replicated Data*, pages 104–106, Monterey, California.

Liskov, B., Scheifler, R., Walker, E., and Weihl, W. (1987). Orphan detection. In *Digest of Papers, The 17th IEEE International Symposium on Fault-Tolerant Computing*, Pittsburgh-USA.

Liu, C. and Layland, J. (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61.

Lockhart Jr., H. (1994). *OSF DCE.* McGraw-Hill.

Lowe, G. (1995). An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133.

Lundelius, J. and Lynch, N. (1984a). A New Fault-Tolerant Algorithm for Clock Syncr onization. In *Proceedings of the 3rd ACM SIGACT-SIGOPS Symp. on Prin ciples of Distrib. Computing*, pages 75–88, Vancouver-Canada.

Lundelius, J. and Lynch, N. (1984b). An upper and lower bound for clock synchronization. *Information and Control*, 62:190–204.

Lupu, E. and Sloman, M. (1997). Towards a role based framework for distributed systems management. *Journal of Network and Syst. Manag't*, 5(1).

Lynch, N. (1996). Data link protocols. In *Distributed Algorithms*, pages 691–732. Morgan-Kaufmann.

Lyu, M., editor (1995). *Software Fault Tolerance.* Wiley.

M. Shapiro, M. (1986). Structure and Encapsulation in Distributed Systems: the Proxy Principle. In *Proceedings 6th IEEE Intl. Conf. on Distributed Computing Systems*, pages 198–204, Cambridge, USA.

Macedo, R., Ezhilchelvan, P., and Shrivastava, S. (1995). Flow control schemes for a fault-tolerant multicast protocol. Technical report, Univ. Newcastle upon Tyne.

Madeira, H. and Silva, J. (1994). Experimental evaluation of the fail-silent behaviour in computers without error masking. In *Digest of Papers, Fault-Tolerant Computers Symposium*, pages 350–359.

Magedanz, T. and Eckardt, T. (1996). Mobile software agent: A new paradigm for telecommunications management. In *Proceedings of the IEEE/IFIP Network and Management Operations Symposium (NOMS)*, Kyoto, Japan.

Magee, J., Dulay, N., and Kramer, J. (1993). Structuring parallel and distributed programs. *IEEE Software Engineering Journal*, 2(8):73–82.

Magee, J., Dulay, N., and Kramer, J. (1994). Regis: A constructive development environment for distributed programs. *IEEE/IOP/BCS Distributed Systems Engineering Journal*, 1(5):304–312.

Magee, J., Kramer, J., and Sloman, M. (1989). Constructing distributed systems in conic. *IEEE Transactions on Software Engineering*, SE-15(6):663–675.

Mahony, D., Peirce, M., and Tewari, H. (1997). *Electronic Payment Systems.* Artech House.

Malan, G. and Jahanian, F. (1998). An extensible probe architecture for network protocol performance measurement. In *Procs. of ACM SIGCOMM'98.*

Malkhi, D. and Reiter, M. (1998). Byzantine quorum systems. *Distributed Computing*, 11(4):203–213.

Manna, Z. and Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems.* Springer, New York.

Mansouri-Samani, M. and Sloman, M. (1994). Monitoring distributed systems. In Sloman, M., editor, *Network and Distributed Systems Management*, chapter 12, pages 303–347. Addison-Wesley.

MAP (1985). *Manufacturing Automation Protocol Specification V2.1.*

Martin-Flatin, J. (1999). Push vs. pull in web-based network management. In *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management (IM'99)*, pages 3–18, USA.

Martin-Flatin, J.-P., Znaty, S., and Hubaux, J.-P. (1999). A survey of distributed network and systems management paradigms. *Journal of Network and Systems Management*, 7(1):9–26.

Marzullo, K. (1983). Maintaining the time in a distributed system. *ACM*, pages 295–305.

Marzullo, K. (1990). Tolerating failures of continuous valued sensors. *ACM Transactions on Computer Systems*, 8(4):284–304.

Maxion, R. and Olszewski, R. (1998). Improving software robustness with dependability cases. In *Digest of Papers, The 28th Int'l Symp. on Fault-Tolerant Computing Systems*, Munich, Germany. IEEE.

Mazumdar, S. (1996). Inter-domain management between corba and snmp: Web-based management - corba/snmp gateway approach. In *Proceedings of the 7th IEEE Int'l Workshop on Distributed Systems Operations and Management (DSOM'96)*, pages 28–30, L'Aquila, Italy.

McGraw, G. and Felten, E. (1997). *Java Security, Hostile Applets, Holes and Antidotes.* John Wiley.

Melliar-Smith, P., Moser, L., and Agrawala, V. (1990). Broadcast protocols for distributed systems. *IEEE Trans. on Parallel and Distributed Systems*, 1(1):17–25.

Menezes, A., Van Oorschot, P., and Vanstone, S. (1999). *Handbook of Applied Cryptography, 4th ed.* CRC.

Merkle, R. (1978). Secure communication over insecure channels. *Communications of the ACM*, 21(4):294–299.

Merkle, R. and Hellman, M. (1981). On the security of multiple encryption. *Communications of the ACM*, 24(7):465–467.

Metcalfe, R. and Boggs, D. (1976). Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7).

Meyer, J. (1992). Performability: A retrospective and some pointers to the future. *Performance Evaluation North Holland*, 14, 3-4:139–155.

Meyer, J., Muralidhar, K., and Sanders, W. (1989). Performability of a token-bus network under transient faults. In *The 19th Annual IEEE International Symposium on Fault-Tolerant Computing*, Chicago-USA.

Micali, S. (1993). Fair public-key cryptosystems. In *Proceedings of the Advances in Cryptology-CRYPTO'92*, pages 113–138. Springer-Verlag.

MIL-STD-1553B (1988). *Field Bus Based on MIL-STD-1553B.*

Miller, C. (1999). *Multicast Networking and Applications.* Addison Wesley.

Mills, D. (1991). Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493.

Minar, N. (1999). A survey of the ntp network. Technical report, MIT.

Mintzberg, H. (1989). *Mintzberg on Management, Inside Our Strange World of Organizations.* Free Press, MacMillan.

Mishra, S., Peterson, L., and Schlichting, R. (1993). Consul: A communication substracte for fault-tolerant distributed programs. *Distributed Systems Engineering*, 1(2):87–103.

Mishra, S. and Schlichting, R. (1992). Abstractions for constructing dependable distributed systems. Technical Report TR 92-19, The University of Arizona, Departement of Computer Science, Tucson, Arizona, USA.

MMS (1990). *MMS Specification - Part 1: Service definition, Part 2: Protocol specification.* International Organization for Standardization.

Mok, A. (1983). *Fundamental Design Problems of Distributed Systems for the Hard Real-time Environment.* PhD thesis, MIT, Cambridge-Mass., USA.

Morris/Satyanarayanan, Conner, M., Howard, J., Rosenthal, D., and Smith, F. (1986). Andrew: a Distributed Personal Computing Environment. *Communications of the ACM*, 29(3).

Moser, L., Amir, Y., Melliar-Smith, P., and Agarwal, D. (1994). Extended virtual synchrony. In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, pages 56–65, Poland.

Moser, L., Melliar-Smith, P., Agarwal, A., Budhia, R., Lingley-Ppadopoulos, C., and Archambault, T. (1995). The Totem system. In *Digest of Papers of the 25th IEEE Int. Symp. on Fault-Tolerant Computing Systems*, pages 61–66.

Mossé, D., Melhem, R., and Ghosh, S. (1994). Analysis of a fault-tolerant multiprocessor scheduling algorithm. In *Digest of papers, the 24th FTCS, IEEE International Symposium on Fault-Tolerant Computing.*

Moy, J. (1994). Multicast routing extension for ospf. *Communications of the ACM*, 37(8):61–66.

Mullender, S., editor (1993). *Distributed Systems, 2nd Edition.* ACM-Press. Addison-Wesley.

Muller, N. (1997). Improving network operations with intelligent agents. *Journal of Network and Systems Management*, 7(3):116–126.

Needham, R. (1993). Cryptography and secure channels. In Mullender, S., editor, *Distributed Systems, 2nd Edition*, ACM-Press, chapter 20. Addison-Wesley.

Needham, R. and Schroeder, M. (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999.

Needham, R. and Schroeder, M. (1987). Authentication revisited. *Operating Systems Review*, 21(1):7.

Neuman, B. and Stubblebine, S. (1993). A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10–14.

Neuman, B. and Ts'o, T. (1994). Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38.

Neumann, P. (1995). *Computer Related Risks.* Addison-Wesley.

Neves, N. and Fuchs, W. (1998). RENEW: A tool for fast and efficient implementation of checkpointing protocols. In *Digest of Papers, The 28th IEEE Int'l Symp. on Fault-Tolerant Computing Systems*, Munich, Germany.

Nicomette, V. and Deswarte, Y. (1997). An authorization scheme for distributed object systems. In *Proc. of the IEEE Symp. on Security and Privacy*, pages 21–30.

Norton, B. (1994). Integrating network discovery with network monitoring: The nsfnet method. In *Proceedings of the INET'94/JENC5*, page 5631 to 5636.

ODP (1987). *Proposed Revised Text for the New Work Item on the Basic Reference Model for Open Distributed Processing.* ISO/TC97/SC21 N1889.

Okamoto, T. and Ohta, K. (1992). Universal electronic cash. In *Proceedings of the Advances in Cryptology- CRYPTO'91*, pages 324–337. Springer-Verlag.

Oki, B., Pfuelgl, M., Siegel, A., and Skeen, D. (1993). The information bus- an architecture for extensible distributed systems. *ACM SIGOPS*, pages 58–68.

OMG (1997a). The Common Object Request Broker: Architecture and Specification.

OMG (1997b). CORBAservices: Common Object Services Specification.

OMNIPoint (1993). Understanding omnipoint, white paper. Technical report.

Otway, D. and Rees, O. (1987). Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10.

Ozden, B., Rastogi, R., and Silberschatz, A. (1996). Research issues in multimedia storage managers. *ACM Computing Surveys.*

Panzieri, F. and Roccetti, M. (2000). Responsive protocols for distributed multimedia applications. In Krakowiak, S. and Shrivastava, S., editors, *Advances in Distributed Systems*, volume 1752 of *LNCS*, chapter 7. Springer-Verlag.

Panzieri, F. and Shrivastava, S. (1988). Rajdoot: a remote procedure call mechanism supporting orphan detection and killing. *IEEE*, 14(1).

Paris, J.-F. and Sloope, P. (1992). Dynamic management of highly replicated data. In *Procs. of the 11th IEEE Symposium on Reliable Distributed Systems*, pages 20–28.

Parkinson, B. and Gilbert, S. (1983). Navstar: Global positioning system— ten years later. *Proceedings of the IEEE*, 71(10):1177–1186.

Patterson, D., Gibson, G., and Katz, R. (1988). A case for redundant arrays of inexpensive disks (RAID). In *Proc. of the ACM SIGMOD Conference*, volume 17, pages 109–116.

Paul, S. (1998). *Multicasting on the Internet and its Applications.* Kluwer Academic Publishers.

Pavlou, G., Liotta, A., Abbi, P., and Ceri, S. (1998). Cmis/p++: Extensions to cmis/p for increased expressiveness and efficiency in the manipulation of management information. *IEEE Network*, pages 10–20.

Pease, M., Shostak, R., and Lamport, L. (1980). Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2).

Peden, J. and Weaver, A. (1988). The utilization of priorities on token ring networks. In *Proc. of the 13th Conf. on Local Computer Networks*, Minneapolis, USA.

Pedone, F., Guerraoui, R., , and Schiper, A. (1998). Exploiting atomic broadcast in replicated databases. In *Proceedings of Europar Conference*, number 1470 in Lecture Notes in Computer Science, pages 513–520. Springer-Verlag.

Peterson, L., Buchholz, N., and Schlichting, R. (1989). Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–146.

Pfister, G. (1998). *In search of clusters. Second Edition.* Prentice Hall.

Pfleeger, C. (1996). *Security in Computing, 2nd Edition.* Prentice-Hall.

Pimentel, J. (1990). *Communication Networks for Manufacturing.* Prentice-Hall.

Pleinevaux, P. and Decotignie, J. (1988). Time critical communication networks: Field buses. *IEEE Network*, 2(3).

Poledna, S., editor (1995). *Fault-Tolerant Real-Time Systems: the Problem of Replica Determinism.* Kluwer Academic Publishers.

POSIX (1995). *Portable Operating System Interface (POSIX) - Part 1: API C Language - Real-Time Extensions.* ISBN 1-55937-375-X.

Post, M., Shen, C.-C., and Wei, J. (1996). The manager/agency paradigm for distributed network management. In *Proceedings of the IEEE Network Operations and Management Symposium (NOMS'96)*, pages 44–53.

Postel, J. (1978). Internetwork protocol specification - version 4. Technical Report IEN-41.

Powell, D., editor (1991). *Delta-4 - A Generic Architecture for Dependable Distributed Computing.* ESPRIT Research Reports. Springer Verlag.

Powell, D. (1992). Failure mode assumptions and assumption coverage. In *Proc. of The 22nd IEEE Int. Symp. on Fault-Tolerant Computing Systems*, page 386.

Powell, D. (1994). Distributed fault tolerance: Lessons from delta-4. *IEEE Micro*, pages 36–47.

Powell, D., Arlat, J., Beus-Dukic, L., Bondavalli, A., Coppola, P., Fantechi, A., Jenn, E., Rabejac, C., and A., W. (1999). GUARDS: A generic upgradable architecture for real-time dependable systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):580–599.

Powell, D., Seaton, D., Bonn, G., Veríssimo, P., and Waeselynk, F. (1995). The Delta-4 approach to dependability in open distributed computing systems. In Suri, N., Walter, C., and Hugue, M., editors, *Advances in Ultra-Dependable Distributed Systems.* IEEE CS Press. Reprinted from Digest of Papers, The 18th IEEE International Symp. on Fault-Tolerant Computing, Tokyo - Japan, June 1988.

Pras, A., Hazewinkel, H., and van Hengstum, E. (1997). Management of the worldwide web. In *Procs. of the SBRC'97*, pages 340–345, São Carlos, Brazil.

Preparata, F., Metze, G., and Chien, R. (1967). On the connection assignment problem of diagnosable systems. *IEEE Trans. on Electronic Computers*, 16(6):848–854.

Prodromides, K. and Sanders, W. (1993). Performability evaluation of csma/cd and csma/dcr protocols under transient faults. *IEEE Transactions on Reliability*, 42(1):166–127.

Profibus (1991). *General Purpose Field Communication System – Part 2, Profibus.*

Purimetla, B., Sivasankaran, R., Ramamritham, K., and Stankovic, J. (1995). Real-time databases: Issues and applications. In Son, S., editor, *Advances in Real-Time Systems.* Prentice-Hall.

Quinn, S. (1996). Unix host and network security tools. Technical report, National Institute of Standards and Technology.

Radestock, M. and Eisenbach, S. (1996). Agent-based configuration management. In *Proceedings of the Seventh IFIP/IEEE International Workshop on Distributed Systems: Operation and Management.*

Rajkumar, R., Gagliardi, M., and Sha, L. (1995). The real-time publisher/ subscriber interprocess communication model for distributed real-time systems: Design and implementation. In *Proceedings of RTAS'95, the IEEE Real-Time Technology and Applications Symposium.*

Ramamritham, K. (1995). The origin of tcs. In *Proceedings of the First ACM International Workshop on Active and Real-Time Database Systems*, pages 50–62, Skovde,Sweden. Springer-Verlag.

Ramamritham, K. (1996a). Dynamic priority scheduling. In Joseph, M., editor, *Real-Time Systems.* Prentice Hall.

Ramamritham, K. (1996b). Real-time databases. *International Journal of Distributed and Parallel Databases*, 1(2):199–226.

Ramamritham, K., Stankovic, J., and Zhao, W. (1989). Distributed sheduling of tasks with deadlines and resource requirements. *IEEE Transactions on Computers*, 38(8):1110–1123.

Ramanathan, P., Kandlur, D., and Shin, K. (1990). Hardware-Assisted Software Clock Synchronization for Homogeneous Distributed Systems. *IEEE Trans. Computers*, C-39(4):514–524.

RAND (1955). *A Million Random Digits with 100,000 Normal Deviates.* Free Press Publishers.

Randell, B. (1975). System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, SE-1(2).

Raynal, M., Schiper, A., and Toueg, S. (1991). The causal ordering abstraction and a simple way to implement it. *Information Processing Letters*, 39(6):343–350.

Redmond, K. and Smith, T. (1980). *Project Whirlwind - The History of a Pioneer Computer.* Digital Press.

Reiter, M. (1996). Distributing trust with the rampart toolkit. *Communications of the ACM*, 39(4).

Rennels, D. (1984). Fault-tolerant computing - concepts and examples. *IEEE Transactions on Computers*, 33(12):1116–1129.

Ricart, G. and Agrawala, A. (1981). An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, 24(1):9–17.

Ricciulli, L. and Shacham, N. (1997). Modelling correlated alarms in network management systems. In *Procs. of the Conference on Communication Networks and Distributed Syst. Modeling and Simulation, CNDS'97*, pages 9–16.

Rivest, R. (1992). The MD5 message digest algorithm. Technical Report RFC 1321, IETF.

Rivest, R. and Shamir, A. (1984). How to expose an eavesdropper. *Communications of the ACM*, 27(4):393–395.

Rivest, R., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2).

Rodrigues, L., Fonseca, H., and Veríssimo, P. (1996). Totally ordered multicast in large-scale systems. In *Proceedings of the 16th IEEE International Conference on Distributed Computing Systems*, pages 503–510, Hong Kong.

Rodrigues, L., Guerraoui, R., and Schiper, A. (1998a). Scalable atomic multicast. In *Proc. of the Seventh IEEE International Conf. on Computer Communications and Networks (IC3N'98)*, pages 840–847, Lafayette, USA.

Rodrigues, L., Guimarães, M., and Rufino, J. (1998b). Fault-tolerant clocks synchronization in can. In *Proc. of the 19th IEEE Real-Time Systems Symp.*

Rodrigues, L. and Raynal, M. (2000). Atomic broadcast in asynchronous crash-recovery distributed systems. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems (ICDCS'20)*, pages 288–295, Taipe, Taiwan.

Rodrigues, L., Siegel, E., and Veríssimo, P. (1994). A Replication-Transparent Remote Invocation Protocol. In *Proceedings of the 13th IEEE Symposium on Reliable Distributed Systems*, Dana Point, California.

Rodrigues, L. and Veríssimo, P. (1995). Causal separators for large-scale multicast communication. In *Proceedings of the 15th IEEE International Conference on Distributed Computing Systems*, pages 83–91, Vancouver, British Columbia, Canada.

Rodrigues, L. and Veríssimo, P. (2000). Topology-aware algorithms for large-scale communication. In Krakowiak, S. and Shrivastava, S., editors, *Advances in Distributed Systems*, LNCS 1752, chapter 6, pages 127–156. Springer-Verlag.

Rodrigues, L., Veríssimo, P., and Rufino, J. (1993). A low-level processor group membership protocol for LANs. In *Proc. of the 13th IEEE International Conference on Distributed Computing Systems*, pages 541–550, Pittsburgh, USA.

Rodrigues, L. and Veríssimo, P. (1992). xAMp: a Multi-primitive Group Communications Service. In *Proceedings of the 11th IEEE Symposium on Reliable Distributed Systems*, Houston, Texas.

Rom, R. (1988). A reconfiguration algorithm for a double-loop token-ring local area network. *IEEE Transactions on Computers*, 37(2).

Romão, A. (1994). Tools for DNS debugging. Technical Report RFC 1713, USc Inf. S. Inst.

RTS Journal (1997). The challenge of global time in large-scale distributed real-time systems, Schmid,U., ed. *Special Issue of the Journal of Real-Time Systems*, 12(1-3).

Rufino, J., Veríssimo, P., and Arroz, G. (1999). A Columbus' egg idea for CAN media redundancy. In *Digest of Papers, The 29th IEEE International Symposium on Fault-Tolerant Computing Systems*, Madison, Wisconsin - USA.

Rufino, J., Veríssimo, P., Arroz, G., Almeida, C., and Rodrigues, L. (1998). Fault-tolerant broadcasts in CAN. In *Digest of Papers, The 28th IEEE Int'l Symp. on Fault-Tolerant Computing Systems*, Munich, Germany.

Rufino, J. and Veríssimo, P. (1992). A study on the inaccessibility characteristics of ISO 8802/4 Token-Bus LANs. In *Proceedings of the IEEE INFOCOM'92 Conference on Computer Communications*, Florence, Italy. also INESC AR 16-92.

Ryan, P., Schneider, S., Roscoe, B., Goldsmith, M., and Lowe, G. (2000). *The Modelling and Analysis of Security Protocols.* Addison-Wesley.

Sahai, A. and Morin, C. (1998). Towards distributed and dynamic network management. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, New Orleans, USA.

Saksena, M., da Silva, J., and Agrawala, A. (1995). Design and implementation of Maruti-II. In Son, S., editor, *Advances in R-T Systems.* Prentice-Hall.

Saltzer, J. and Schroeder, M. (1975). The protection of information in computing systems. *Proceedings of the IEEE*, 63(9):1278–1308.

Sandberg, R. (1985). The sun network filesystem: Design, implementation and experience. In *Proc. of the Summer 1985 USENIX Confer.*, pages 119–130.

Schell, R. (1984). Security kernel design principles. Technical Report 84-2-7, Auerbach.

Schmid, U. and Schossmaier, K. (1997). Interval-based clock synchronization. *Journal of Real-Time Systems*, 12(2):173–228.

Schneider, F. (1987). Understanding protocols for byzantine clock synchronization. Technical report, Cornell University, Ithaca, New York.

Schneider, F. (1993). Replication management using the state-machine approach. In Mullender, S., editor, *Distributed Systems, 2nd Edition*, ACM-Press, chapter 7. Addison-Wesley.

Schneider, F., Gries, D., and Schlichting, R. (1984). Fault-tolerant broadcasts. *Science of Computer Programming*, (4):1–15.

Schneier, B. (1996). *Applied Cryptography, 2nd edition.* John Wiley.

Schonwalder, J. and Toet, M. (1997). Management of the world-wide web. In *Proceedings of the 8th IFIP/IEEE Int'l Workshop on Distributed Systems Operations and Management (DSOM'97)*, Sydney, Australia.

Schossmaier, K., Schmid, U., Horauer, M., and Loy, D. (1997). Specification and implementation of the universal time coordinated synchronization unit (UTCSU). *Journal of Real-Time Systems*, 12(3).

Sha, L., Rajkumar, R., and Lehoczky, J. (1990). Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. on Computers*, 39:1175–1185.

Shin, K. (1991). Harts: Distributed real-time architecture. *IEEE Computer*, 24(5):25–35.

Shirley, J., Hu, W., Magid, D., and Oram, A. (1994). *Guide to Writing Dce Applications.* Number ISBN: 1565920457. O'Reilly & Associates.

Shrivastava, S., Dixon, G., and Parrington, G. (1991). An Overview of the Arjuna Distributed Programming System. *IEEE Software.*

Siamwalla, R., Sharma, R., and Keshav, S. (1999). Discovering internet topology. In *Proceedings of the IEEE Infocom'99*, pages 21–25.

Silberschatz, A., Galvin, P., and Gagne, G. (2000). *Applied Operating System Concepts.* Number ISBN: 0471365084. Wiley.

Silva, L. and Silva, J. (1992). Global checkpointing for distributed programs. In *Procs. of the 11th IEEE Symp. on Reliable Distributed Systems*, pages 155–164.

Sinha, P. and Suri, N. (1999). On the use of formal techniques for analyzing dependable real-time protocols. In *Proc. of the 20th IEEE Real-Time Systems Symp.*

Siqueira, F. and Cahill, V. (2000). An open qos architecture for corba applications. In *Proc. of ISORC 2000, the Third IEEE Int'l Symposium on Object-Oriented Real-Time Distributed Computing*, pages 328–335, Newport Beach, USA.

Skeen, D. (1985). Determining the last process to fail. *ACM Trans. on Computer Systems*, 3(1).

Slade, R. (1995). *Computer Viruses, 2nd edition.* Springer-Verlag.

Sloman, M., editor (1994). *Network and Distributed Systems Management.* Addison-Wesley.

Sloman, M. and Twidle, K. (1994). Domains: a framework for structuring management policy. In Sloman, M., editor, *Network and Distributed Systems Management*, chapter 16. Addison-Wesley.

Solomon, M., Landweber, L., and Neuhengen, D. (1982). The csnet name server. *Computer Networks*, 6(3):161–172.

Son, S. (1987). Using replication for high performance database support in distributed real-time systems. In *Proc. of RTSS'87, the 8th IEEE Real-Time Systems Symp.*

Song, X. and Liu, J. (1992). How well can data temporal consistency be maintained? In *Proceedings of the IEEE Symposium on Computer-Aided Control Systems Design.*

Spainhour, S. and Quercia, V. (1996). *Webmaster in a Nutshell.* O'Reilly.

Spector, A. (1987). Camelot: a distributed transaction facility for Mach and the Internet — an interim report. Research paper. CMU-CS-87-129, Carnegie Mellon University, CS Dept., Pittsburgh, PA, USA.

Speirs, N. and Barrett, P. (1989). Using passive replicates in Delta-4 to provide dependable distributed computing. In *Digest of Papers, The 19th IEEE International Symposium on Fault-Tolerant Computing*, Chicago-USA.

Sprunt, B., Sha, L., and Lehoczky, J. (1989). Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1(1):27–60.

Srikanth, T. Kand Toueg, S. (1987). Optimal Clock Synchronization. *Journal of the Association for Computing Machinery*, 34(3):627–645.

Stallings, W. (1998). Security comes to snmp: The new snmpv3 proposed internet standard. *The Internet Protocol Journal*, 1(3):2–12.

Stallings, W. (1999). *Cryptography and Network Security: Principles and Practice, 2nd Ed.* Prentice-Hall.

Stankovic, J. (1988). Misconceptions about real-time computing. *IEEE Computer.*

Stankovic, J. and Ramamritham, K. (1991). The Spring Kernel: A New Paradigm for Real-time Systems. *IEEE Software.*

Steiner, J., Neumann, C., and Schiller, J. (1988). Kerberos: An authentication service for open network systems. In *Proc. of USENIX Winter Conference*, pages 191–202.

Steiner, M., Tsudik, G., and Waidner, M. (1998). CLIQUES: A new approach to group key agreement. In *Proc. 18th IEEE International Conference on Distributed Computing Systems (ICDCS'98)*, pages 380–387, Amsterdam.

Stephenson, P. (1991). *Fast Causal Multicast.* PhD thesis, Cornell Univ.

Steusloff, H. (1981). The impact of distributed computer control systems on software. *Pergamon Press.*

Stewart, J. (1999). *BGP4.* Addison-Wesley.

Stiffler, J. (1978). Fault coverage and the point of diminishing returns. *Journal of Design Automation & Fault Tolerance Computing*, 2(4).

Strom, R. and Yemini, S. (1985). Optimistic recovery in distributed systems. *ACM Trans. on Computer Systems*, 3(3):204–226.

Suri, N., Hughe, M., and Walter, C. (1994). Synchronization issues in real-time systems. In *Proceedings of IEEE, Special Issue on Real Time Systems.*

Tanenbaum, A. (1992). *Modern Operating Systems.* Prentice-Hall.

Tanenbaum, A. (1995). *Distributed Operating Systems.* Prentice-Hall.

Tanenbaum, A. (1996). *Computer Networks.* Prentice-Hall, 3rd edition.

TCSEC (1985). *Trusted Computer System Evaluation Criteria.*

Thomas, R. (1979). A Majority Concensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transactions on Database Systems*, 4(2):180–209.

Thompson, J. (1998). Web-based enterprise management architecture. *IEEE Communications Magazine*, 36(3):80–86.

Tindell, K. (1994). *Fixed Priority Scheduling of Hard Real-Time Systems.* PhD thesis, University of York, UK.

Tindell, K. and Burns, A. (1994). Guaranteeing message latencies on Controler Area Network. In *Proc. of the 1st Int'l CAN Conference*, Mainz, Germany. CiA.

Tindell, K., Burns, A., and Wellings, A. (1994). Calculating Controller Area Network (CAN) message response times. In *Proceedings of the IFAC Workshop on Distributed Computer Control Systems*, Toledo, Spain.

Tindell, K., Burns, A., and Wellings, A. (1995). Analysis of hard real-time communications. *Real-Time Systems*, 9(2):147–171.

Token Bus (1985). *Token Passing Bus Access Method.*

Tokuda, H., Nakajima, T., and Rao, P. (1990). Real-time mach: Towards a predictable real-time system. In *Proceedings of the USENIX Mach Workshop.*

Tovar, E., Vasques, F., and Burns, A. (1999). Adding local priority-based dispatching mechanisms to p-net networks: A fixed priority approach. In *Proceedings of the 11th IEEE Euromicro Conference on Real-Time Systems*, pages 175 – 184, York, England.

Tschichholz, M., Tschammer, V., and Dittrich, A. (1996). Integrated approach to open distributed management. *Computer Communications*, (19):76–87.

Turek, J. and Shasha, D. (1992). The many faces of consensus in distributed systems. *IEEE Computer*, 25(6):8.

van Renesse, R., Birman, K., and Maffeis, S. (1996). Horus: A flexible group communications system. *Communications of the ACM*, 39(4):76–83.

Veríssimo, P. (1996). Causal delivery protocols in real-time systems: A generic model. *Journal of Real-Time Systems*, 10(1):45–73.

Veríssimo, P. and Almeida, C. (1995). Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models. *Bulletin of the Tech. Commit. on Operating Systems and Application Environments (TCOS)*, 7(4):35–39.

Veríssimo, P., Rodrigues, L., and Baptista, M. (1989). AMp: A highly parallel atomic multicast protocol. In *Proceedings of the ACM SIGCOM'89 Symposium*, pages 83–93, Austin, USA.

Veríssimo, P., Rodrigues, L., and Casimiro, A. (1997). Cesiumspray: a precise and accurate global clock service for large-scale systems. *Journal of Real-Time Systems*, 12(3):243–294.

Veríssimo, P. (1988). Redundant media mechanisms for dependable communication in token-bus LANs. In *Proceedings of the 13th IEEE Local Computer Network Conference*, Minneapolis-USA.

Veríssimo, P. (1993). Real-time Communication. In Mullender, S., editor, *Distributed Systems, 2nd Ed.*, ACM-Press, chapter 17, pages 447–490. Addison-Wesley.

Veríssimo, P. (1994). Ordering and Timeliness Requirements of Dependable Real-Time Programs. *Journal of Real-Time Systems, Kluwer*, 7(2):105–128.

Veríssimo, P., Barrett, P., Bond, P., Hilborne, A., Rodrigues, L., and Seaton, D. (1991). The Extra Performance Architecture (XPA). In Powell, D., editor, *Delta-4 - A Generic Architecture for Dependable Distributed Computing*, ESPRIT Research Reports, pages 211–266. Springer Verlag.

Veríssimo, P., Casimiro, A., and Fetzer, C. (2000). The timely computing base: Timely actions in the presence of uncertain timeliness. In *Procs. of DSN 2000, the IEEE/IFIP Int'l Conf. on Dependable Systems and Networks.*

Veríssimo, P. and Marques, J. (1990). Reliable broadcast for fault-tolerance on local computer networks. In *Procs. of the 9th IEEE Symp. on Reliable Distributed Systems*, Huntsville, Alabama-USA. Also INESC AR/24-90.

Veríssimo, P., Melro, S., Casimiro, A., and Silva, L. (1996). Distributed industrial information systems: Design and experience. In *Proceedings of BASYS'96, the 2nd IEEE/IFI International Conference on Information Technology for Balanced Automation SYStems in Manufacturing.*

Veríssimo, P. and Raynal, M. (2000). Time, clocks and temporal order. In Krakowiak, S. and Shrivastava, S., editors, *Recent Advances in Distributed Systems*, volume 1752 of *LNCS*, chapter 1. Springer-Verlag.

Veríssimo, P. and Rodrigues, L. (1992). A posteriori Agreement for Fault-tolerant Clock Synchronization on Broadcast Networks. In *Digest of Papers, The 22nd IEEE Int'l Symp. on F/T Computing*, Boston - USA.

Veríssimo, P., Rufino, J., and Ming, L. (1997). How hard is hard real-time communication on field-buses? In *Digest of Papers, The 27th IEEE International Symposium on Fault-Tolerant Computing*, Seattle - USA.

Vogels, W., Rodrigues, L., and Veríssimo, P. (1992). Fast group communication for standard workstations. In *Proceedings of the OpenForum'92 Technical Conference*, Utrecht, the Netherlands. EurOpen, UniForum.

Volg, C., Wolf, L., Herrtwich, R., and Wittig, H. (1996). Heirat - quality of service management for distributed multimedia systems. *Multimedia Systems Journal.*

Wakerly, J. (1978). *Error detecting codes, self-checking circuits and applications.* North Holland.

Waldo, J. (1999). Jini technology architectural overview. Technical report, Sun Microsystems.

Wang, Y.-M. and Fuchs, W. (1992). Optimistic message logging for independent checkpointing in message-passing systems. In *Proceedings of the 11th IEEE Symposium on Reliable Distributed Systems*, pages 147–154.

Watt, A. and Watt, M. (1992). *Advanced animation and rendering techniques - Theory and practice.* Addison-Wesley, New York.

Wayner, P. (1993). Should encryption be regulated? *Byte.*

Wellings, A., Beus-Dukic, L., and Powell, D. (1998). Real-time scheduling in a generic fault-tolerant architecture. In *Proceedings of RTSS'98, the IEEE Real-Time Systems Symposium*, pages 390–398, Madrid, Spain.

Wensley, J. H., Lamport, L., Goldberg, J., Green, M. W., Levitt, K. N., Melliar-Smith, P. M., Shostak, R. E., and Weinstock, C. B. (1978). SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255.

Wies, R. (1994). Policies in network and systems management - formal definition and architecture. *Journal of Network and Sys. Management*, 2(1):63–83.

Wikander, J.and Svensson, B., editor (1998). *Real-Time Systems in Mechatronic Applications.* Kluwer Academic Publishers.

Wilson, D. (1985). The stratus computer system. In *Resilient Computing Systems*, pages 208–231.

Wood, M. (1991). *Fault-Tolerant Management of Distributed Applications Using a Reactive System Architecture.* PhD thesis, Cornell University, USA.

Wood, M. (1993). Replicated RPC using Amoeba closed group communication. In *Proceedings of the 13th IEEE International Conference on Distributed Computing Systems*, pages 499–507, Pittsburgh, Pennsylvania, USA.

X.509 (1997). *Information technology - Open Systems Interconnection– The Directory: Authentication Framework.*

XTP (1998). *The Xpress Transport Protocol Specification.* XTP Forum Inc., 1394 Greenworth Place, Santa Barbara, USA.

Xu, J., Randell, B., Romanovsky, A., Stroud, R. Zorzo, A., Canver, E., and von Henke, F. (1999). Rigorous development of a safety-critical system based on coordinated atomic actions. In *Digest of Papers, The 29th IEEE International Symposium on Fault-Tolerant Computing Systems*, pages 68–75, Madison- USA.

Xu, J., Randell, B., Rubira-Calsavara, C., and Stroud, R. (1995). Toward an object-oriented approach to software fault tolerance. In *Recent Advances in Fault-Tolerant Parallel and Distributed Systems*, Computer Society Press, pages 226–233. IEEE.

Yau, S. and Karim, F. (2000). Component customization for object-oriented distributed real-time software development. In *Proceedings of ISORC 2000, the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 156–163, Newport Beach, USA.

Zeltserman, D. and Puoplo, G. (1998). *Building Network Management Tools with Tcl/Tk.* Prentice-Hall.

Zhang, T. and Covaci, S. (1997). Java-based mobile intelligent agents as network management solutions. In *Proceedings of the 8th Joint European Conference (JENC8)*, pages 12–15, Edinburgh, Scotland.

Zheng, Q. and Shin, K. G. (1992). Fault-tolerant real-time communication in distributed computing systems. In *Digest of Papers, The 22nd IEEE International Symposium on Fault-Tolerant Computing Systems*, pages 86–93.

Zimmermann, H. (1980). OSI Reference model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, COM-28(4):425–432.

Zimmermann, P. (1995). *The Official PGP User's Guide.* MIT Press.

Znaty, S., Genilloud, G., Gaspoz, J.-P., and Hubaux, J.-P. (1995). Networked systems: Introducing network management models into odp. In *Proceedings of the IEEE Globecom'95*, pages 121–126.

Znaty, S. and Hubaux, J.-P. (1997). Telecommunications services engineering: Principles, architectures and tools. In *Procs. of the ECOOP'97*, Finland.

Zuberi, K. and Shin, K. (1995). Non-preemptive scheduling of messages on controller area networks for real-time control applications. In *Procs. of RTAS'95, the IEEE Real-Time Technology and Applications Symp.*

Index

About the Authors

Paulo Veríssimo Paulo Veríssimo is a professor of the Department of Informatics, University of Lisboa Faculty of Sciences. He leads the Navigators research group, at the University of Lisboa. He has been in the coordinating team of several major national projects in informatics, and headed the participation of the group in several CEC ESPRIT projects. He belonged to the Executive Board of the CABERNET- ESPRIT Network of Excellence. Paulo Veríssimo is a member of Ordem dos Engenheiros, IEEE and ACM. He has served as program co-chair of the (ex-FTCS/DCCA) IEEE DSN 2001 conference and on the programme committees of several other conferences (IEEE, AFCET, ACM), and is an associate editor for the Telecommunications Systems Journal (Baltzer). He is author of more than 80 refereed publications in international scientific conferences and journals, and over a 100 technical reports. He is co-author of four books in distributed systems and dependability. He organised and lectured in the LISBOA'92 Advanced Course on Distributed Systems, and was director of the 3rd European Seminar on Advances on Distributed Systems, ERSADS 99.

Luís Rodrigues Luís Rodrigues graduated (1986), has a Master (1991) and a PhD (1996) in Electrotechnic and Computers Engineering, by the Instituto Superior Técnico de Lisboa (IST). He is Assistant Professor at Department of Informatics, University of Lisboa Faculty of Sciences. Previously he was at the Electrotechnic and Computers Engineering Department of Instituto Superior Técnico de Lisboa (he joined IST in 1989). From 1986 to 1996 he was a member of the Distributed Systems and Industrial Automation Group at INESC. Since 1996, he is a senior researcher of the Navigators group and a founding member of the LASIGE laboratory at University of Lisboa. He participated and contributes to several national and international projects such as, Delta-4, Estímulo, Broadcast, GODC, etc. His current interests include fault-tolerant and real-time distributed systems, group membership and communication, and replicated data management. He has more than 40 publications in these areas. He is a member of the Ordem dos Engenheiros, IEEE and ACM.