Project IST-1999-11583

Malicious- and Accidental-Fault Tolerance for Internet Applications

REFERENCE MODEL AND USE CASES

University of Newcastle upon Tyne (UK), FCUL, Lisboa (P),
DERA, Malvern (UK), Universität des Saarlandes (D),
CNRS-LAAS (F), IBM Zurich Research Lab (CH)

MAFTIA deliverable D1

Public document

25 AUGUST 2000

Technical Report CS-TR-707, University of Newcastle upon Tyne

LAAS-CNRS Report No. 00280.

Technical Report DI/FCUL TR-00-5, Universidade de Lisboa

IBM Research Report, RZ 3259 (#93305), IBM Research, Zurich

List of Contributors

Christian Cachin ..... IBM Research, Zurich

Jan Camenisch ..... IBM Research, Zurich

Marc Dacier ..... IBM Research, Zurich

Yves Deswarte ..... LAAS-CNRS, Toulouse

John Dobson ..... University of Newcastle upon Tyne

David Horne[1] ..... Tradezone

Klaus Kursawe ..... IBM Research, Zurich

Jean-Claude Laprie ..... LAAS-CNRS, Toulouse

Jean-Claude Lebraud[1] ..... Rockwell-Collins

Derek Long[1] ..... CISA Ltd

Tom McCutcheon ..... DERA, Malvern

Jurgen Müller ..... IBM Research, Zurich

Frank Petzold ..... IBM Research, Zurich

Birgit Pfitzmann ..... Universität des Saarlandes

David Powell ..... LAAS-CNRS, Toulouse

Brian Randell ..... University of Newcastle upon Tyne

Mathias Schunter ..... Universität des Saarlandes

Victor Shoup ..... IBM Research, Zurich

Paulo Veríssimo ..... Universidade de Lisboa

Gilles Trouessin[1] ..... CESSI[1]

Robert Stroud ..... University of Newcastle upon Tyne

Michael Waidner ..... IBM Research, Zurich

Ian Welch ..... University of Newcastle upon Tyne

[1] Member of MAFTIA industrial advisory board

Table of Contents

Table of Figures

Reference Model and Use Cases

[1]C. Cachin, [1]J. Camenisch, [1]M. Dacier, [2]Y. Deswarte, [3]J. Dobson, [4]D. Horne, [1]K. Kursawe, [2]J.-C. Laprie, [5]J.-C. Lebraud, [6]D. Long, [7]T. McCutcheon, [1]J. Müller, [1]F. Petzold, [8]B. Pfitzmann, [2]D. Powell, [3]B. Randell, [8]M. Schunter, [1]V. Shoup, [9]P. Veríssimo, [10]G. Trouessin, [3]R. J. Stroud, [1]M. Waidner, [3]I. S. Welch

[1]IBM Zurich, Research Labs (CH), [2]CNRS-LAAS (F), [3]University of Newcastle (UK), [4]Tradezone (UK), [5]Rockwell-Collins (F), [6]CISA Ltd (UK), [7]DERA, Malvern (UK), [8]Universität des Saarlandes, [9]FCUL, Lisboa (P), [10]CESSI (F)

25 August 2000

Abstract

This document constitutes the first deliverable of MAFTIA work package 1. The objective of this work package is to define a consistent framework for ensuring the dependability of distributed applications in the face of a wide class of threats. In particular, the aim is to develop a coherent set of concepts for an architecture that can tolerate deliberately malicious faults, such as intrusions, in applications distributed over the Internet. The intrusions of concern include not only those perpetrated by external penetrators, but also those carried out by corrupt insiders, i.e., users who are authorized to access the system but not authorized for the accessed data, program or resource, and administrators who misuse their rights. Although intrusions are the primary class of targeted faults, the architecture should also be adequately robust towards accidental physical faults and accidental design faults.

Keywords

Dependability, Security, Fault-tolerance.

Chapter 1 Scenarios and Use Cases

1.1 Introduction

J. Dobson, R. J. Stroud, ***University of Newcastle upon Tyne (UK)***

This chapter contains a number of scenarios and use cases that are informed by the members of the Industrial Advisory Board and the experience of project members themselves. The first scenario is concerned with the issue of industrial espionage and illustrates the many ways in which an Illegal Information Broker can obtain confidential information about a target company. The second scenario is based on marine transportation, and highlights some of the complex accountability issues that can arise when the chain of dependencies in a commercial transaction involves multiple organisations working across international boundaries. The third scenario discusses some of the problems of securing a corporate intranet for a large multinational company. The fourth scenario is concerned with the issue of privacy for mental health records, and the fifth scenario argues that the traditional attributes of security (confidentiality, integrity and availability) need to be extended with a notion of auditability in order to address legal concerns about accountability and proof when deploying a nation-wide electronic healthcare system. Two further scenarios deal with the problems of setting up a fair and trustworthy electronic market place for e-commerce, and the security issues involved in allowing communication of sensitive information between an aircraft and its airline company.

It should be noted that these scenarios are not intended to be a scientific contribution of the project, nor have they been analysed in order to derive specific requirements for the project. Instead, their purpose is to motivate discussions within the project on information security issues and models that are applicable to a wide class of applications in various kinds of organisations.

There are two organisational contexts in which the security mechanisms in a computer system need to be considered. The first is one of re-engineering: new technology means that more efficient or effective mechanisms can be put in place either in an existing system or in a new system that is an incremental adaptation of an existing system whose engineering is well-understood. The second context, which is much less frequent and much more painful, is re-institutionalisation: an organisation needs to re-invent itself, perhaps in response to changes in the market, or perhaps because the impact of new technology is so fundamental that the old assumptions no longer apply and so incremental adaptation is not an adequate response.

These two contexts are very different in almost every respect, but the one we want to focus on here is the relationship between policies (in particular, security policies but the discussion is more general than that) and mechanisms. In a context of re-engineering, the policies are more or less unchanged (and indeed, the stability of policy is constitutive of re-engineering), so that the policies still govern the choice of mechanism, a choice which is being re-evaluated.

In contrast, the relationship between policies and mechanisms changes completely in a context of re-institutionalisation. Here, the need to choose mechanisms creates a demand for policy: a demand that often cannot be fulfilled because, until the re-institutionalisation is more or less underway, it cannot be known what the policies are. It is almost always premature to assume that the policies conceived at the beginning of a re-institutionalisation process will be the most appropriate ones once the process is complete.

It follows that the choice of mechanisms and the relationship of this choice to policy is quite different in the two cases. In the first, it is simply an engineering decision: will these mechanisms implement this policy? But in the second case, the question is often: what is the *range* of possible policies that need to be supported? This becomes an architectural question.

It would be naïve to think that the technologist has no role in the architectural discourse. However, the role is not to remain silent until the policy is decided and then implement the decision. Rather, the role of the technologist is to inform as to technological possibility and to suggest alternatives that are feasible. For example, a re-institutionalisation may well be raising a requirement for a more fluid definition of role, with its attendant difficulties for a role-based access control security policy. The security technologist could contribute to the discourse by explaining the possibilities afforded by an approach based on an information flow model rather than a role-based model.

But of course the role of the technologist as informant in a policy debate differs from the technologist's role as implementer of a pre-defined policy. To be effective in the former role, the technologist needs to know how to transform the rhetoric of policy into a form that is suitable as input into an engineering process. This requires a way of formalising the language of policy, which often involves concepts such as role, responsibility, communication, 'need-to-know' and so on. One approach to this issue of formalisation is presented in Sections 3.7 and 3.8 of this deliverable, and will be further developed during the project.

1.2 Orchestrated Information Attacks

D. Long, ***CISA Ltd (UK)***

1.2.1 Introduction

The need to obtain information is driven by many factors and its acquisition may lead to success in wealth, war, power, and capital. The capabilities of organisations dedicated to finding and processing information can be extraordinary and the amount that they will invest can be considerable, as the rewards for a single transaction of the correct information magnitude can be millions. One of the most significant changes to occur in relation to information in the past two decades is that information that was previously within the domain and protection of the State, has now, under the information explosion and the free market economy, become accessible to individuals or groups dedicated to acquiring information, manipulating data or denying others legitimate access.

Throughout history there has been a cadre of those who would collect information gathered for one purpose and fuse or meld it with other data, in order to achieve personal or organisational advantage. The legitimacy of this is a matter of perception. Undertaken for research, combining information in this way can be viewed as a major benefit to society. Used by the Police for the detection and prosecution of crime, such actions are also acceptable. However, it would probably be considered unacceptable for the same information to be accessed in the same manner by a private detective agency, information broker or foreign intelligence organisation.

1.2.2 The Information Broker

We are now seeing the rise of the Information Broker (IB) as an Information Assurance threat. Prior to the IT age, an IB provided market research services to business and industry via the capturing and collation of information from media, official registers of companies, annual reports, economic surveys, etc.

In the IT age we now see the industrial IB moving towards searching both public and private databases. Throughout both periods, the unscrupulous have operated in the public and private domains to obtain sensitive information of value. Thus IB is a term that includes a wide variety of activity. It can be an entirely proper business activity but it can also involve:

- An organisation that sets out to obtain information – or influence - through agents, and then trades that onward, sometimes for corrupt purposes.
- Individuals, or opportunists acting on their own who might also trade information and influence often gained innocently as a result of the naïve attitude of staff and/or poor control procedures.

1.2.3 The Broker as a Potential Protagonist

An Illegal Information Broker (IIB) may be:

- An opportunistic individual working alone.
- An individual co-ordinating and using others.
- A trading organisation, perhaps offering consultancy services with a network of agents who develop company insiders.

An IIB frequently targets large industrial organisations to take advantage of imprudent company contacts. Frequently they seek to penetrate projects and the contract process by developing relationships with relevant personnel and might use bribery, blackmail or social recruitment. They will also seek to gain access electronically and remotely via the Web into the corporate Intranets and thence to critical domains within the company structure.

The IIB seeks to obtain and then trade:

- Influence, usually bid lists, evaluations, scope of work etc.
- Information, mainly tenders, procurement, budgets, new formula and business critical IPRs.

Frequently the IIB is an ex-member of the sector or actual organisation using specialist knowledge to target and identify the specific 'information gems' that will bring the most gain when offered to the market.

In late 1999, a review of the web revealed 135 advertisements on one site alone for 'Information Broker' services. Mainly based in the US and Europe (Switzerland) they offered various services but all related to the ability for them to gain access to databases such as Social Security, Court records, Credit Card information, etc. This would indicate that IIBs have followed the trend towards IT database consolidation and are 'harvesting' these new sources. The New York Times reported that a Social Security Clerk could receive $25 for an item of database information that would be retailed by an IIB for $300.

In addition to the targeted recruiting or coercion of sources within companies, a further recent trend has been the exploitation by IIBs of Hackers. Frequently the Hacker is unaware of the manipulation that is occurring, as he is in dialogue over the Internet with the IIB who appears to be another Hacker, usually of apparently greater skills. The Hacker is set challenges or tasks by the IIB, and accepts them as a technology challenge. The resulting break-in methodology or information gained is then passed to the IIB as proof of success. In due course, this may lead to receiving rewards or indeed 'funded commissions' from the IIB. The use of this method is on the increase, as it reduces the risk that the IIB will be exposed or entrapped during the normal recruitment period, which is when the IIB is at greatest risk should the attempt fail. The use of the Web provides both speed and anonymity for recruitment, targeting and information dissemination.

A company may be the target of an IIB for several reasons:

- Perhaps as a major purchaser of vehicles, hardware & software systems, stationary etc., the contracts may be lucrative and the knowledge of opponent's bids or the win price is high value.
- Information on new IPR investments, new partnerships etc. can be highly valuable to the other market players.
- If a corporation handles client information, knowledge of those clients, their requirements and business or State operations is valuable.

The IIB is unlikely to wish to achieve anything other than a Confidentiality attack, but his methods can also be used by those who are members of extremist pressure groups. The true IIB seeks information that may be resold or meets a commission's specific needs, the extremist may also attempt to cause damage or service/system denial. In extreme circumstances, Integrity may be affected if the IIB is able to change a contract price/budget but that is highly unusual.

1.2.4 Scenario

An IIB is approached at a Conference and asked to provide some deep background information on a new `.com` company such as its budget, technical performance, numbers of 'hits' per day, system suppliers, business plan for the stock release, stock options held, etc. For example:

- is this new venture being supported by banks?
- which international corporations are using the system?
- is there a possibility that a future tie-up may be made with key ISPs?

The IIB sets in place a comprehensive, overlapping plan to obtain the information and validation via a multiple set of activities and contacts. The first objective is to find out more information on the whole operation, the types of business that are being sought and the methods used. The IIB notes from the press that the target is making strong inroads into its chosen market and that there appears to be no equal supplier. Rumour abounds that the target may be sold to an established Web company and consolidated into its existing business. The IIB makes plans to find out details concerning the sale, and approaches a known university graduate who is willing to act as an insider. The graduate applies to an Agency that supplies staff to the Information Support helpdesk of the target company. With some false records provided by the IIB, the graduate is hired and starts work. From the inside, he is able to monitor and build up the network design of the company. The team appreciate his skills and after some manoeuvring he moves to assist the Technical Design Architect in some support solutions for the roll-out – if the bid goes through. As part of the forward business plan core team he has access to details of the company's size, future developments, abilities and targets. The graduate also acts as a 'talent spotter' for the IIB and highlights some team members

who are short of funds or who have personal problems that can be exploited. These are followed up and in turn these contacts volunteer names of others. Using other previous contacts in Telecommunication companies and specialist web software, the IIB will also insert electronic access gateways into the internal telephone exchange for modem dial-in and software 'holes' that provide exploitation points. Each is disguised to look like a legitimate remote user, whilst the network monitoring devices are disabled by the insider.

As a backup, the IIB arranges for the waste-bags from the company to be dropped off in nearby bushes for collection and sorting.

Using multiple identities on various European and South American web sites, the IIB uses the bulletin boards and chat lines to see if any technical chat or financial trading tips are being traded about the companies. On occasions he will stimulate debate by making provocative statements and reviewing the response. Electronic dialogues develop over the globe from the original 'net-chats'.

Given some of the basic information from the 'insider', the IIB begins to set 'challenges' to hackers on the dedicated Boards. In turn, some are caught but a few do succeed and post their success, methods and information gleaned. The IIB logs this and uses the results to refocus his efforts or validate information gained though conventional methods.

The IIB turns his attention to the company's suppliers; using both electronic attacks, deception telephone calls, and insiders he seeks to find out what contracts are being placed, and the dates and delivery times of new equipment that support the company expansion and which might be used by the new owner. By posing as an employee or as a sub-supplier, this method provides a backdoor into future technological expansion and aids corporate valuation.

In time, a set of contacts in the key business areas, usually at the PA/Secretarial level is used to collect information. On occasions, these contacts download information onto discs to be smuggled out. A few Sales staff and Technical support staff will freely provide information to those who they see as friends (or potential new employers if they think they might be made redundant during the forthcoming takeover).

The IIB can provide the information requested but now has additional information on contract win prices, tips for stock-market investments but more importantly a portfolio of contacts who can be reactivated and used for other purposes and needs in the future. The original request for information may have been legitimate but by passing the objective down the chain, more illegal methods have come into operation.

The IIB will use 'open source' commercial information to build his attack plan and identify the more difficult areas where key information is held. He may even commission a commercial marketing intelligence report or front such an organisation. The IIB will use multiple methods to gain access. Where possible he would use a country that had weak laws in prosecuting such activities. A blend of physical contact,

an insider, and remote electronic access is still the most reliable method of capturing information.

1.3 Intelligent Transport Systems

D. Long, ***CISA Ltd (UK)***

1.3.1 Introduction

This scenario is related to a new EU eEurope initiative on "Intelligent Transport Systems". In "An Information Society For All", the following problem is identified:

> "Maritime transport safety is hampered by lack of information and requires closer identification and monitoring of traffic along the coasts of Member States, in particular, of ships carrying polluting goods"[2].

1.3.2 Scenario

A Greek registered container ship enters European waters from the Atlantic and proceeds to Felixstowe in Suffolk. On board are 1200 mixed containers, 400 of which are to be unloaded at Felixstowe before the vessel collects 300 containers and moves onto Hamburg. The Master has a loading plan, Felixstowe has a master list of containers, and contents and position. The loading port (New Orleans) had prepared bills of lading that are also held in databases. HM Customs & Excise have the bills of lading sent to them for processing by their system, CHIEF.

Certain containers are listed as containing 'hazardous cargo' and these are stored with due security.

At Felixstowe the containers are unloaded and stacked on the dock. Lorries arrive to load and collect containers, which then leave by road for distribution throughout the UK. Hazardous containers are marked and the lorries display the correct agreed EU hazardous warnings for the Emergency Services.

Due to a failure of the software or malicious interference by a 'Green Group', the electronic record identifying a particular hazardous container gets lost. The lorry transporting the container is involved in an accident whilst passing through a built-up area. The Emergency Services, unaware of the cargo within the container, use the wrong chemicals to contain a fire and the whole incident rapidly accelerates out of control.

---

[2] "An Information Society For All", the Draft Action Plan prepared by the European Commission for the European Council in Feira 19-20 June 2000, under the section entitled "Intelligent Transport Systems" on page 26.

2.3.3 Trust Model

The system that should have marked the container as hazardous has had to interact from transhipment in the US, to New Orleans Port Authority, through US Customs, the Shipping Agent, the Master, UK Customs, Felixstowe Port, and road transport. All these organisations have had to have secure, reliable e-commerce interaction to ensure that the right information is passed around the international private/government chain. Communications bearers are required, the software applications have to accept the right 'formats', and 'trust' is required at every level throughout a mixed judicial environment.

2.3.4 Analysis

The security model for this application is unclassified but commercially sensitive. The threats and risks to the application are accidental loss of information, unintentional changes, and malicious faults. Contingency plans in the event of a malicious attack from an intruder or corrupt insider are minimal, due to the international chain of organisations involved in the transaction. Fault tolerance mechanisms exist but only in basic form. For example, electronic tagging of individual containers and GPS tracking does occur but is expensive. This is very much a 'belt and braces' approach. The dependability of the system is really only concerned with 'availability' in order to keep traffic flowing, and the prevention of incidents further down the delivery chain is regarded as being out of scope.

2.4 Information Systems in Mental Health

J. Dobson, ***University of Newcastle upon Tyne (UK)***

2.4.1 Introduction

The issue of confidentiality and invasion of privacy has become increasingly prominent as the use of computers becomes widespread. The importance of individual privacy has long been recognised in a democratic society. Specific laws and regulations, such as privacy and freedom of information acts, have been introduced to assure and safeguard individual rights. However, the issues involved in protecting these rights when using a computerised database system have not been clearly understood. In order to illustrate some of the problems, a case study based on mental health care is presented. The goal is not so much to define a security policy for this application area as to highlight some of the issues that such a policy should address.

2.4.2 Confidentiality and Privacy Issues in the Mental Health Profession

The confidence and trust between the patient and doctor is an important concern within the medical profession and healthcare system. Maintaining confidentiality of a patient's medical data is of great importance. This issue is particularly sensitive in a mental health delivery system and has become a controversial topic when

computerised information systems are being considered to handle mental health data. It is the fear of many medical professionals that the confidentiality of medical and personal data will not be maintained. Such a fear is not totally unsupported.

The current scenario envisages that a proposal has been made for the development of an information system that can provide the needed information and data to improve the quality and efficiency of mental health care delivery in a Regional Health Authority. The Authority has only limited resources available but is responsible for using those resources effectively and efficiently in the development and management of a system-wide mental health delivery system. The Authority's staff will use the system to gather planning data about individuals who need mental health care in order to develop a system of mental health care facilities that reflects the patterns of the mental health patient population. Information may also be available for supporting medical, psychological, and social studies, and for monitoring possible epidemic situations. All these functions may have opportunities for the compromise of the required confidentiality and for the invasion of a patient's privacy.

The responsible administrator and technical staff have repeatedly assured the mental health community that the proposed information system will provide adequate controls for the protection of confidentiality and individual privacy as demanded by regulations and ethical standards of the medical profession. However, their confidence has not been shared by the mental health professionals.

Information and data recorded in the patient's medical records are subject to many potential abuses. A patient needs confidence in his or her doctor in order to build a trust for obtaining appropriate and quality medical attention and care. Patients with mental health problems are particularly vulnerable. The simple fact of seeking mental health care can have potential abuses in many organisational, political, and social circumstances. Mental patient records often contain sensitive private and confidential data about certain organisations or other individuals that could but should not be shared by others. Basic requirements for handling patient records have been established, rigidly stated, and are compulsory by laws and organisational regulations.

On the other hand, there are organisations and individuals that are directly and indirectly related to the patient mental health care process or to the patient, and have certain legitimate needs to access relevant information from the patient's record. These individuals include medical support personnel such as nurses, pharmacists, social workers, medical administrative staff, insurance agents, patients' relatives and legal guardians, mental health systems administrators, researchers, and law enforcement officials who are responsible for protecting the patient and the public. They all have justifiable needs for accessing a patient's medical or personal data for fulfilling their professional, legal, and/or moral responsibilities. Relationships between these individuals are complicated. The conditions under which they are permitted to access information are not clearly understood and issues related to these conditions are often emotional, controversial and ambiguous. To have a clear understanding of these conditions is an extremely difficult task.

Confidentiality and individual privacy issues have not been taken seriously by many administrators. Their main responsibilities and concerns are economical, particularly operating costs. Many of them do not have a full appreciation of confidentiality and privacy issues and have a very limited understanding of the technical limitations of information and data security. Thus, although the power of computerised information systems has been recognised by many mental health administrators, proposals for using computers to increase the efficiency and quality of mental health delivery systems have often been made without a clear understanding of the potential information security implications and consequences.

The issues of confidentiality have also not been fully comprehended by computing professionals. The semantics of information security requirements when using a computerised database system have not been understood by computing professionals. Most database security researchers and developers have only just begun to appreciate the problem.

The most difficult concern is that administrators and computing professionals have not been given the legal, ethical, and moral responsibility for information security. They have not been held legally liable and there is no law to require them to do so.

2.4.3 Information Security Demands

In this section, the information security requirements and demands in a mental health delivery system are briefly presented and analyzed. The analysis is by no means complete or absolutely accurate. The characteristics of these mental health security demands are identified along with some possible approaches or suggestions for the computerisation of mental health information systems. These characterisations and suggested approaches are presented for the purpose of stimulating discussions. They should not be viewed as formal research results from a well-formed analysis or investigation.

2.4.3.1 Information Security Focuses on the Meaning of the Data

Information security focuses more on the meaning of the data instead of the data itself. Security systems that process classified data assume that sensitive data is the valuable resource to be protected. Information security systems concern mainly the functions and interests of the application and its organisation. Information security functions and features must be developed within the domain of the application data model. The database user’s application level responsibility plays an important role in information access and control. Access to information should not just be based on the trustworthiness of the user but more on the user’s need for information in order to perform the assigned application functions.

2.4.3.2 Application-dependent Security Constraints

A security policy must be developed as an integral part of the application. The information security semantics must reflect data semantics germane to the application

and relevant to the role of the user who performs data access tasks. An application system is considered secure if it provides the needed information to a legitimate user for accomplishing application objectives and prevents individuals from the unauthorised access of information. Application-dependent security constraints are often complicated. They are frequently dependent upon the contents of several data items and the context of the database structure. Certain constraints are time- or event-dependent. The data access control may have a temporal relationship to the status of some events or time-dependent data values.

2.4.3.3 A User-role Based and Action Oriented Information Security Approach

In a mental health information system, the control on who can access what information is an important feature. The emphasis on the user-role and data access actions within the application domain is essential. The user's need for information and its permitted data access actions are the main factors that determine the allowable data access operations.

Only authorised users should be allowed to access data. Based on their roles, all users must have predefined information access rights. Changes to the state of the data must be due to authorised actions that carry out data access operations on certain permissible data sets as specified in the user's access rights. All actions are traceable.

Individuals in medical and related professions have specific legal and moral responsibilities in accessing and maintaining patients' medical records. An individual's data access rights are often dependent upon his or her relationship with other medical personnel or the appearance of certain values such as permission. The authorised data access rights are often content- and context-dependent. The development of appropriate techniques for handling these types of access controls is certainly a non-trivial task.

2.4.3.4 Only Authorised and Registered Users are Allowed to Access Mental Health Information

The computerised mental health information system has inherited legal and professional responsibilities of medical professionals for maintaining patients' confidentiality and privacy. The system has to be designed to allow access only by authorised and registered users.

However, what are the legal and professional responsibilities of computer and systems staff who have the opportunity to access data, and how are these to be defined and developed? What mechanisms can be used to ensure that they, and only they, take the actions necessary to discharge these responsibilities?

2.4.3.5 Traceability of Data Access Activities

Traceability on activities of generation and use of the data may be required for the enforcement of an individual user's accountability as required by laws and regulations in mental health delivery.

The system must be able to maintain a trace of data access activities. The trace should be able to tell who generated and accessed what information, and when such an action took place in order to determine individual users' legal and professional responsibilities. How can such an auditing procedure be effectively designed? Can the auditing process itself contribute to the potential of the invasion of privacy?

2.4.3.6 Temporal Controls may be Necessary

Granting permission to permit certain information access by a patient or medical personnel implies the need for temporal logic and control. A data access action to be executed may depend on the value of a temporally varying variable (e.g. the patient's age ≥ 18) or the appearance of a given event (e.g. the doctor's confirming signature). The precedence relationships between various types of events will have to be used to form temporally related control systems. A doctor's reporting obligations may trigger a sequence of activities. These chained activities will require the security system to have certain propagation transaction controls and will require the use of rollback techniques in the event that it fails to complete the anticipated activities. The development of temporal entities and temporally related information security access control is another difficult but very intriguing task.

2.4.3.7 Information Flow Control Problems

The chained activities involve much more than just a simple information flow. At each stage the system must interpret the meaning of the information flow in order to take an appropriate next step. For example, when a parent attempts to access a patient's data, the system must determine if the patient's permission is needed. This depends on the patient's age and whether the doctor thinks the patient is capable of making such a decision. The system also has to determine that the individual attempting to access the data is indeed the parent of the patient. The system has to take an appropriate action at each step. Understanding the meaning and consequences of the information flow is a difficult research task. Are recently developed AI techniques and knowledge-based systems concepts helpful in this regard?

2.4.3.8 Verifying the Source of the Information

When the system requires certain information, it must also verify that the appropriate authorised individual generated the information. For example, how does the system know that permission is indeed given by the patient or the doctor? An individual may deny having accessed certain information - how can the system prove whether or not the individual is telling the truth? Can electronic signature techniques be applied to solve this problem in a mental health information system?

2.4.3.9 Data Accuracy and Completeness Problems

Although doctors are required to maintain an accurate and complete patient record, there are no valid legal and technical means to ensure accuracy and completeness of information. Is this an open problem?

2.4.3.10 Inference and Aggregation Problems

The control of inference and aggregation issues in mental health information systems is a common problem that is encountered in all statistical database systems. Appropriate means for controlling illegal inferences and deductions must be introduced to reduce possible damages.

Different types of statistical studies have different data access characteristics and needs. Inference control experts must examine each proposed study in order to determine what inference control techniques are to be used. This is not something that can be automated by the system.

2.4.3.11 Applicability of Multi-level Security to Mental Health Information Systems

The need to protect different levels of sensitive information is not unique to a mental health system. There are similarities with corporate management information systems that must protect proprietary and market sensitive information, and systems that process classified data in military applications. However, the existing military classification system may not be directly applicable to mental health delivery systems. Different levels of clearance may have to be assigned to different types and levels of medical, administrative, and technical personnel.

Can the existing principles and concepts of multi-level security systems be made applicable to mental health information systems? Can a partially ordered classification system be appropriately developed for controlling the sensitive information in a mental health information system?

2.5 Multinational Intrusion Detection Systems

M. Dacier, ***IBM Research, Zurich (CH)***

In today's world, the economy is global and many companies have a global presence too. The Internet offers simple ways to create world-wide virtual private networks that link all the sites of a given company together. Enforcing security within such a distributed and, frequently, heterogeneous environment is a difficult task. In addition to the usual techniques for preventing intrusions, such as firewalls and packet filters, intrusion detection systems are now also becoming part of the security officer's toolbox.

Initially these systems were fairly simple, focusing on the "outside" border of the network, looking for intrusions originating from the Internet. However, with the

growing demand for sophisticated services to enable e-commerce, back-end systems are less and less confined to the secure “Intranet”. Indeed, the notions of Intranet and Extranet are fading out. As a consequence, the number of intrusion detection sensors is increasing because such sensors now have to be deployed in what used to be seen as the ‘secure Intranet’. They not only have to be placed at different locations but they also need to use different technologies in order to cope with the kind of system they are supposed to protect. This proliferation of sensors generates several problems that are typical of large scale distributed applications and which, so far, have not been addressed appropriately. In the next few paragraphs, we will try to outline an idealised scenario for the deployment of a large-scale intrusion-detection system.

It is not our intent to be exhaustive in listing the various problems that have to be solved. On the contrary, we limit ourselves to a few examples of things that, in a real-world and concrete example, need to be addressed.

In our scenario, the intrusion detection system is supposed to protect a large number of sites, located in different countries. The term “site” denotes a, potentially, complicated network. For economical reasons, the people in charge of security are centralized in one location. In order to achieve their work efficiently they need to be able to:

i) Remotely manage all the intrusion detection software at every site (i.e. install, start, stop, configure, etc.).

ii) Verify that, at any point in time, the integrity of these sensors has not been violated.

iii) Collect all the alarms generated by all sensors in a central location.

iv) Do some reasoning based on the alarms received, using a *correlation engine*.

v) Launch countermeasures in case of attacks.

It should be clear from this list of requirements that there is a need for secure information communication between the centralized system where correlation takes place and all the sensors, distributed in every site. One could imagine using Virtual Private Networks (VPNs) between each sensor and the correlation engine. However, in our example, the security officer cannot restrict his or her view to the threat caused by the outsider. A large multinational company may be employing many contractors, with “trusted” connections to several other companies in order to run its business. Furthermore, some of the company’s sites may have outsourced part of their infrastructure or have it remotely managed by some specialized company. For all these reasons, the “insider” threat is of high importance and the security officer may have deployed several hundreds of intrusion detection sensors in each site or perhaps even installed simple intrusion detection probes on each desktop machine. Therefore, it is quite likely that sensors will very frequently be added and removed from every site. Using classical and rather static VPNs in such an environment is doomed to fail. This highlights the need for more sophisticated means to communicate between the sensors and the correlation engine. A solution has to be found that can deal with this

changing set of sensors in an efficient and scalable way. Secure group membership and group key exchange protocols provide one possible solution that will be explored by the MAFTIA project.

Also, this scenario indicates that the system where correlation takes place is itself a single point of failure. If an insider can control it, the whole intrusion detection system becomes useless. One way of avoiding this problem is for the correlation service to be distributed across several platforms that each receives all the alarms from all the sensors. This probably requires some form of secure and ordered multicast protocol. Last but not least, these correlation engines should eventually report their conclusion to the security officer in a consistent way. This requires the use of some form of voting protocol since we assume that some of the correlation engines could have been corrupted.

This example is intriguing in the sense that it presents a large-scale internet application that requires the technology the MAFTIA project is about to work on but, at the same time, the application itself is something that will be investigated and is needed to protect other applications. We have here some form of recursion. MAFTIA technology is being used to support MAFTIA technology, and this scenario is best described by the term “intrusion-tolerant large-scale intrusion-detection”.

2.6 Healthcare Information System Security

G. Trouessin, *CESSI (F)*

2.6.1 Introduction

Information System Security (ISS) has been historically viewed as IT security, in other words the combination of the usual *Availability*, *Integrity* and *Confidentiality* properties. However, more recently, the definition of Healthcare Information System Security (HISS) has been broadened to include the juridical-technical concepts of *Auditability*; first, for legal and privacy reasons, but also to enable some convergence with potential safety and reliability needs.

For this reason, defining, applying and verifying security is not considered to be enough by most responsible health actors involved in serious healthcare projects. Auditability is seen as a sort of juridical-technical bridge between pure legislation such as legal rights and obligations, and pure technical solutions such as digital signatures (as opposed to the more recent electronic signature).

As these systems are considered by human rights, health and life preservation organisations, the challenge of providing HISS is that it embraces privacy, reliability and safety; increasingly, HISS must be provided over a multimedia network dedicated to medical activity, say a Health Information Network (HIN).

2.6.2 Typical Scenarios

2.6.2.1 The Medical Scenario (tele-medicine):

A patient and their General Practitioner (GP) communicate about the results of a test and a draft diagnosis. There is an off-line consultation between the GP and another GP, video-advice from a specialist in another country, and a final decision as to whether to proceed with a surgical operation or not.

What for?	high confidentiality and integrity, and actor authentication
Why?	deontology/code of practice and privacy
How?	authorisation, encryption and digital/electronic signature
When?	over the HIN when transporting sensitive personal data and availability during the remote consultation

2.6.2.2 The HealthCare Scenario (electronic reimbursement claim)

Ensuring that a patient is reimbursed by the social security agency for the payment the patient has made to his GP for a series of consultations, tests, and a referral to a specialist.

What for?	confidentiality and integrity, and organisational authentication
Why?	medical-administrative secrecy (confidentiality); electronic evidence of the reimbursement claim and financial high integrity and accountability
How?	authentication, encryption, non-repudiation
When?	on the GP's hard disk, in the healthcare insurance computer and during the transport of the electronic reimbursement claim

2.6.2.3 The SocialCare Scenario (electronic benefit claim)

The costs of providing hospital care for an unemployed member of a disadvantaged family are paid for out of Social Security funds. Anonymised statistics about the provision of such benefits are gathered nationally for analysis and to inform policy.

What for?	confidentiality and integrity, and organisational authentication; confidentiality-anonymity for statistical studies
Why?	administrative secrecy (confidentiality) and financial high integrity and accountability
How?	authentication, encryption, non-repudiation; anonymisation (one-way function) processes

When? in the social institution's computers and during the transportation of the electronic reimbursement (from the social security to the bank organisation)

2.6.3 Analysis

	The Medical *scenario*	*The* HealthCare *scenario*	*The* SocialCare *scenario*
Needs			
IT security: ISS			
Availability	high	low	medium
Integrity	yes	yes	yes
Confidentiality	secrecy	privacy	privacy
Health security: HISS			
Auditability	medical evidence	financial/administrative evidence	financial evidence
Dependability			
Reliability	high (for high availability)	network	network
Safety	emergency / surgery		
Maintainability	for the GP devices	yes	yes
Usability	yes	yes	yes
…			
Means			
Sub-objectives			
Dependable middleware	high importance	medium importance	medium importance
Intrusion detection	high importance	strong importance	strong importance
Distributed authorisation	high importance	medium importance	low importance
Dependable TTP	certification-TTP and confidentiality-TTP	certification-TTP	certification-TTP

2.6.3.1 What are the Contingency Plans in the Case of Malicious Attacks?

Confidentiality-secrecy: a priori robust encryption.

Confidentiality-seclusion: a priori irreversible anonymisation.

Integrity: usual security and dependability techniques.

Availability: network availability essentially.

2.6.3.2 Utility of Mechanisms

Fault-and-intrusion-tolerant atomic transactions, intrusion detection, trusted third parties (TTPs) and distributed authorisation schemes are useful for all three scenarios.

In particular, authenticity can be provided by certification-TTPs and availability by confidentiality-TTPs. Fault- and intrusion- tolerant group communication and consensus protocols are useful in the first scenario, particularly in the case where remote advice is received from more than two agents. The matrix above shows in more detail how these mechanisms can be used as the means of achieving the security needs of each application scenario.

2.7 E-Procurement Application

D. Horne, *Tradezone (UK)*

2.7.1 Introduction

The following scenarios are taken from Tradezone e-procurement workflows where the issues addressed by MAFTIA, fault- and intrusion- tolerance for large-scale Internet applications, are, or might be, of importance.

The overall application domain of Tradezone has the following characteristics:

- Electronic purchasing of goods and services that can either (a) be represented in catalogue form (i.e., definable product types and attributes), or (b) can be defined by a request for quotation / tender workflow.
- A domain with three principal actors (see Figure 1):
    - Purchasers – individuals and corporate groups of individuals requisitioning, approving, placing orders, tracking fulfilment and making payments.
    - Suppliers – creating and managing product information, customer account classes and pricing, processing and completing orders, responding to tenders.
    - Market operators – running on-line market places facilitating purchaser and supplier trading.

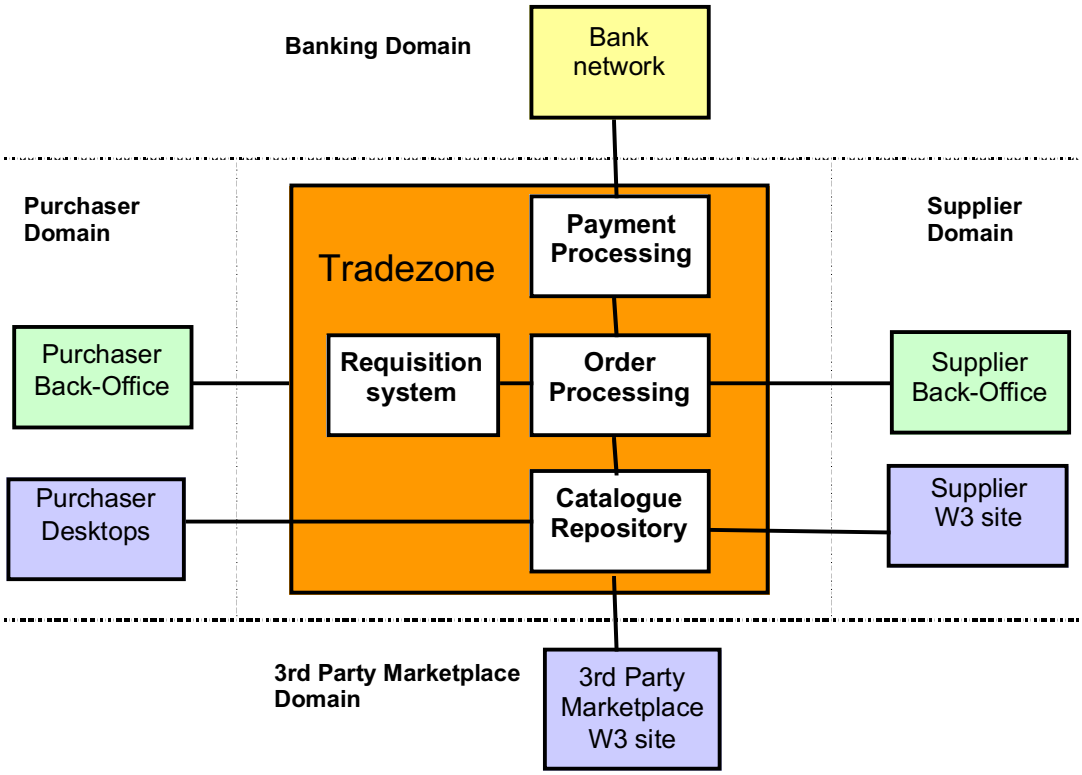

Figure 1 — Tradezone Application Domains

Figure 2 shows an overview of the electronic marketplace envisaged by Tradezone. Purchasers are able to source and order products through a custom browser window accessing their approved suppliers, and through a supplier's web site directly. There is a range of approval, management and reporting functions supporting the procurement process. Suppliers use a cataloguing system to create and maintain their product information, and define different account classes with specific pricing. They collect orders online and process through to fulfilment.

Market operators facilitate trading between partners in specific sectors (e.g., Internet portal businesses are typical customers). They attract suppliers and purchasers to the market, and can also import existing suppliers' catalogues where relevant. But the service as seen by an individual supplier or purchaser remains a one- to one- channel between themselves and each purchaser or supplier with whom they have a business relationship, i.e., it is not normally a public market where all suppliers are visible (although it can be).

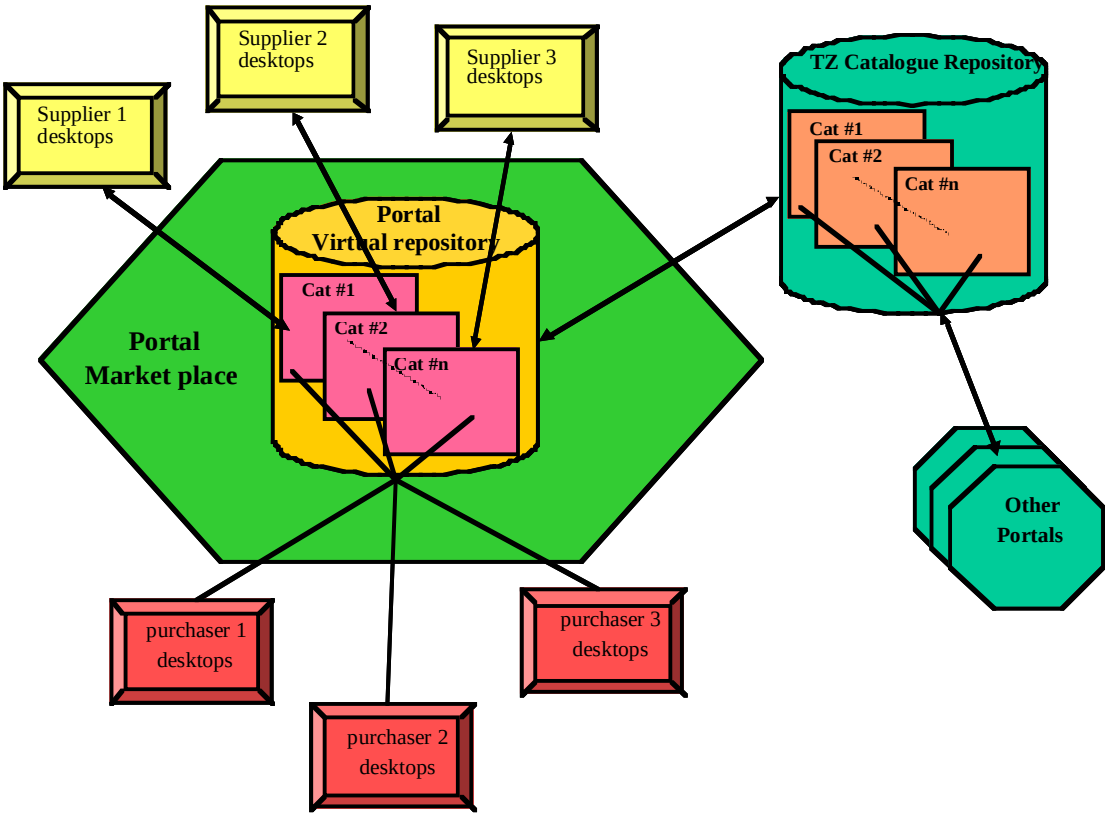

Figure 2 — Overview of Electronic Marketplace

Here are two or three scenarios that suggest themselves as candidates for MAFTIA.

2.7.1.1 Catalogue Management and Publishing

Suppliers maintain their catalogue offline using software that communicates with the online repository for uploading and classification consistency purposes. Information, especially pricing structures, is sensitive and has to be protected from unauthorized disclosure. Problems could occur if communication fails at any time.

2.7.1.2 Creation and Placement of Order

Requisitions and orders are persistent items created through workflow processes, and finally communicated via Tradezone to suppliers. The transactions are digitally signed and time-stamped and their integrity is essential to the legal acceptability of the service. Tradezone will eventually be a distributed set of service operators, so that these transactions will be communicated between servers operated by different operators who have to be trusted parties. This requires a plan for dealing with communication failures.

2.7.1.3 Tendering

Tradezone will extend purchasing systems to include support for the tendering process using flexible documentation and workflow services. All such documents will

be integral to workflow management, not just within organisations but between organisations within the commercial framework. Maintaining document and process integrity across the systems boundaries of Tradezone and customer sites is important not only for functional coherence, but also for legal and financial reasons.

2.7.1.4 Tradezone's Key Interests

The main interest areas that might be relevant to MAFTIA are as follows:

- Reliable systems support for distributed systems.
- Distributed certification / Certification Authorities (CAs).
- Protection of transactions and content in a multi-operator network.

2.7.2 Catalogue Management

Prices are changed off line and have to be confirmed by other individuals in the supplier organisation (verification, anti-fraud). They then need to be approved by all affected customers (customer-supplier relationships, trust) except for non-account "retail" pricing; this requires sensitive price information to be sent to specific customers, followed by confirmation or rejection, raising authentication, privacy and completeness issues.

Suppliers must develop product types that are consistent with the existing type definitions. A type approval service enforces this by preventing the publication of catalogues until all product types are approved. This process could be compromised by the interruption of application level communications.

Once published live, content needs to be updated in any other physical servers holding affected subset catalogues. It is necessary to ensure that all replicas are consistent at any point in time.

What for?	Maintaining up to date product information and price schedules relating to individual customers or groups of customer. Maintaining consistency with a global product type space.
Why?	Buyers want current product and custom pricing online. Suppliers want control over content. Tradezone wants to maintain global product definitions to ensure true multi-supplier comparisons by buyers.
How?	Strong authentication of each user. Separation of content management and price management roles, established by the authentication. Reconciliation between supplier's local catalogue copy and published copy in live service environment. Workflow associated with publishing (verification by supplier, price change approval by affected contract customers before going live).

When? During upload and download of catalogue information, during pricing updates, and during refresh of subsidiary catalogue repositories.

2.7.3 Order Creation

A person wishing to order goods (the requisitioner) assembles a list of items, and when satisfied proceeds to raise an order. This could be hours or days later. The order will be only be created if certain constraints are met (buying limits, allowed supplier, allowed product type etc), otherwise additional approval is required. A robust workflow is needed to allow for errors or interruptions in the process.

Once created, the order is transmitted to the server and possibly to the supplier's order processing system. There is a danger of inconsistency between (a) the transaction logs of Tradezone and the user's applications, (b) the buyer's internal product file and Tradezone catalogue.

As an additional complication, the whole process may require messaging between two physically distinct Tradezone infrastructures (e.g., in different countries).

What for? Purchase of goods from supplier – private, secure, non-repudiable.

Why? Reducing purchasing costs and delivery times, delegation of buying, more purchasing control.

How? Electronic presentation of requisitions uses catalogue access to create items, subjected to approval rules. Tests can result in immediate order transmission or require single or multiple approvals first. Users have to be strongly authenticated, workflow can't break down (otherwise orders never created), order–fulfilment and dispatch is a single process with many possibilities for breakdown.

When? On the requisitioner's or approver's PC desktop, during approval workflow process, during order transmission and presentation to supplier.

2.7.4 Tendering

The creation of a tender requires a digital signature to be attached; this may be transmitted to multiple (known) recipients. The sender has to trust that this process has not been compromised. It is important for competitive tenders that receipt and circumstances should be acknowledged. Preparing the response might involve a multiparty workflow – each stage has to be authenticated and logged. Secure archiving may be needed for legal obligations, with each stage in the document workflow being logged to form an audit trail.

What for?	Purchasing of complex goods and services that can't be represented as off the shelf. Elicitation of competitive offers as opposed to comparison of pre-defined offers.
Why?	Cost reduction, faster turn round.
How?	Use of documentation system allied with workflow process enables complex transactions over period of time. Audit trail, accountability, secure archiving.
When?	During document creation/editing, during transmission, during co-operative development of responses.

2.7.5 Analysis

The trust model for this application requires validation of the digital identities of each human party and of the servers involved. This implies the need for a TTP network, something that is not presently available. The security policy aims at providing strong protection of the server side and separating application components whilst maximising server-side functionality. Encryption and digital signatures are used for all transactions to ensure privacy and detect tampering. The goal is to encapsulate the user-to-user transaction space in a single security shell.

The main risks and threats to the application are:

- Tampering with transaction contents (items, values, delivery information, etc.).
- Malicious or unintended visibility of information intended for one class of account.
- Tampering with supplier catalogue content/pricing.
- Growth of inconsistencies in content or transaction logs between different servers due to communication faults.
- Spoofing of digital identities.
- Server-side unauthorised human intervention.
- Lack of TTP certification of digital identities.

The following mechanisms are used to guard against malicious attacks from intruders or corrupt insiders:

Privacy and integrity:	Strong encryption and digital signatures
Repudiation:	Digital signatures and time stamps
Authorisation:	Digital certificates with PIN (and smart card carrier)

Availability:	Network and server redundancy (plus hot swap)

Most of the proposed mechanisms being developed by the MAFTIA project are relevant to all of these scenarios, with the possible exception of fault and intrusion tolerant consensus. However, clearly fault and intrusion tolerant transactions, group communications, trusted third parties, distributed authorisation and intrusion detection services are all important.

2.8 Networking in an Aeronautic Environment

J.-C. Lebraud, ***Rockwell-Collins (F)***

2.8.1 Introduction

This section describes possible scenarios for the MAFTIA project in the context developed by Rockwell Collins for Airlines in the I^2S (Integrated Information System).

The general idea of I^2S is to provide an Airline with a means of data communication between an airborne network and the ground network of the company; i.e. to consider the aircraft as an extension of the airline network. Any equipped aircraft should be able to land in any equipped airport and get a connection with its company network (see Figure 3).

The connection between the aircraft and the ground network occurs only on the ground when the aircraft is at an airport. The connection is wireless using the 802.11 standard for wireless IP connections.

The airport is equipped with a wireless access point at the gates, with all airlines sharing the same communication media. Once on the ground, data are routed from aircraft to the company network using either private or public networks.

On the aircraft side, the LAN is flexible enough to accommodate mobile users (the crew moves from aircraft to aircraft, using the same laptop). This network is also a wireless LAN to simplify connectivity. The network is built around a network server, which is COTS (commercial off the shelf) equipment (repackaged for the aircraft environment). The wireless LAN is also COTS equipment.

A firewall function allows LAN users (airborne or ground) to access avionics systems using a proprietary protocol.

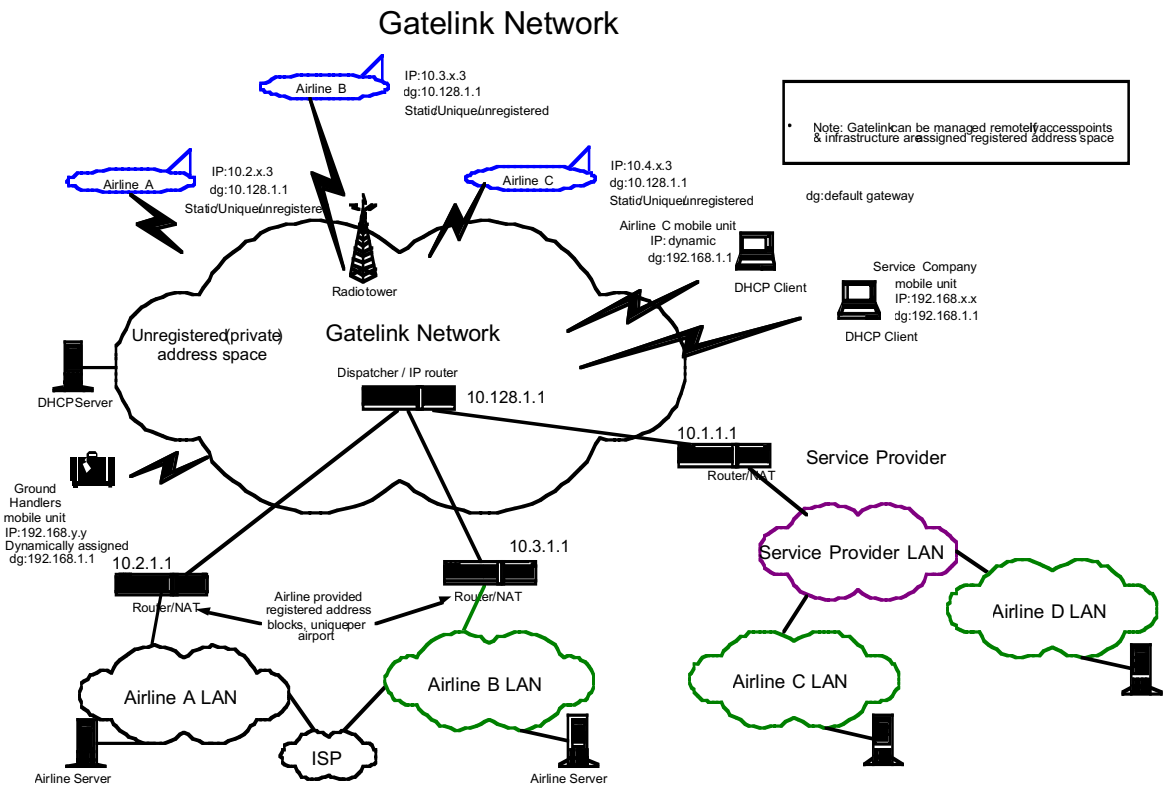

Figure 3 — Description of an I²S Airport

Several aspects of security need to be considered:

- Aircraft protection: protect avionics and all actions that impact aircraft dispatch. This is a highly important feature. The system should be, at least, as safe as the current manual procedures.
- Protect data between aircraft and airline from competing companies. The airline world is a very competitive environment, and companies are very jealous about their data (quality of flight, flight information…)
- Ensure that data are sent by the correct transmitter and not by an impostor. For example, it is highly important to be sure that flight information sent to an aircraft is from the operation centre of the company.
- The next step will be to open services such as email and Internet access up to passengers, using the same system. Sharing a private highly-controlled world with an open one will really become an issue.

In the I²S project, security issues have been solved using end-to-end encryption and certificate based user authentication.

2.8.2 Scenarios

Two kinds of scenario can be proposed, one regarding the access to aircraft and to avionics, the second regarding protection of company privacy.

2.8.3 Aircraft Access

A hacker or corrupt insider, using the same COTS product as the aircraft system, may listen to radio communication, and spoof the airborne server, use some existing user account and access applications. Some maintenance applications are very sensitive as they can access avionics and change the content of their firmware. Thus, possible threats to the application include someone modifying avionics information such as the flight plan, or the airborne system becoming overloaded and thus inoperative for normal usage. The potential risk could vary from delayed departure through hijacking to aircraft destruction.

One possible trust model would be to limit use of the radio link to authorized users. Users would need to logon to the system first, but this authentication would only take place once during a session, so a hacker might be able to spoof the system by taking the place of a real user. Thus, the system would need to perform user authentication checks randomly throughout the session, and perhaps detect doubtful users and discard any requests from them using anomaly detection systems. The most relevant MAFTIA technologies would be fault- and intrusion- tolerant atomic transactions, trusted third parties, and intrusion detection.

2.8.4 Ground Access

Company private information travels from aircraft to the ground company network, using radio links and public networks. These media are shared between airlines and service companies. The risk is to have sensitive data misrouted, re-routed or duplicated to a competitive company. For example, such information might be used by a competitor to claim that the airline company didn’t maintain its aircraft to a sufficiently high level.

A strong authentication of the communicating peer may avoid misrouting, but does not prevent someone from listening to data. End-to-end encryption can be used but does not prevent some third party listener. The security policy would be for the message handling system used between aircraft and ground to provide server authentication, with users being separately authenticated. All of the proposed MAFTIA mechanisms would be useful for dealing with this scenario.

Chapter 3 Basic Concepts

Y. Deswarte, J.-C. Laprie, D. Powell, *LAAS-CNRS, Toulouse (F)*

The purpose of this chapter is to show how basic dependability concepts can be extended to deal with malicious faults and intrusions. We first recall some core dependability concepts, extracted mostly literatim from [Laprie *et al.* 1998, chapter 1][3]. We then extend and refine these definitions in the context of security and intrusion tolerance/detection. Readers familiar with the core dependability concepts may proceed directly to Section 3.2, page 40.

3.1 Core Dependability Concepts

3.1.1 Basic Definitions

Dependability is that property of a computer system such that *reliance can justifiably be placed on the service it delivers.* The **service** delivered by a system is its behaviour *as perceived* by its user(s); a **user** is another system (human or physical) interacting with the system considered.

According to the application(s) of the system, different facets of dependability may be highlighted. This is tantamount to stating that dependability can be viewed according to different but complementary properties that allow its *attributes* to be defined:

- *readiness for usage* leads to **availability**;
- *service continuity* leads to **reliability**;
- non-occurrence of catastrophic consequences for the environment leads to **safety**;
- non-occurrence of unauthorised disclosure of information leads to **confidentiality**;
- non-occurrence of inadequate information alterations leads to **integrity**;
- ability to conduct repairs and introduce evolutions leads to **maintainability**.

Security is generally considered as the combination of confidentiality, integrity and availability [ITSEC], in particular relative to the authorised actions.

A **failure** of the system occurs when the delivered service deviates from implementing the system **function**, that is, from what the system *is intended for*. An **error** is that part of the system state that *may lead to a failure*: an error affecting the

[3] The most recent readily-available version of these concepts may be found in [Laprie 1995].

service is an indication of a failure occurring or which has occurred. The *adjudged or hypothesised cause* of an error is a **fault**.

Development of a dependable system requires the combined use of a set of methods that can be listed as follows:

- **fault prevention**: how to prevent the occurrence or introduction of faults;
- **fault tolerance**: how to provide a service capable of implementing the system function despite faults;
- **fault removal**: how to reduce the presence (number, severity) of faults;
- **fault forecasting**: how to estimate the presence, creation and consequences of faults.

The notions that have been introduced can be listed under three main headings (as shown in Figure 4):

- **impairments** to dependability: faults, errors, failures; these are undesirable — but not unexpected — circumstances, causes or results of un-dependability (that can be simply derived from the definition of dependability: trust can no longer, or will no longer, be put in the service delivered);
- the **means** for dependability: fault prevention, fault tolerance, fault removal, fault forecasting; these are the methods and techniques giving the system the ability to deliver a service conforming to the accomplishment of its function, and to place trust in this ability;
- **attributes** of dependability: availability, reliability, safety, confidentiality, integrity, maintainability: these enable a) expression of the properties expected from the system, and b) assessment of the quality of the service delivered, as resulting from the impairments and the means used to avoid them.

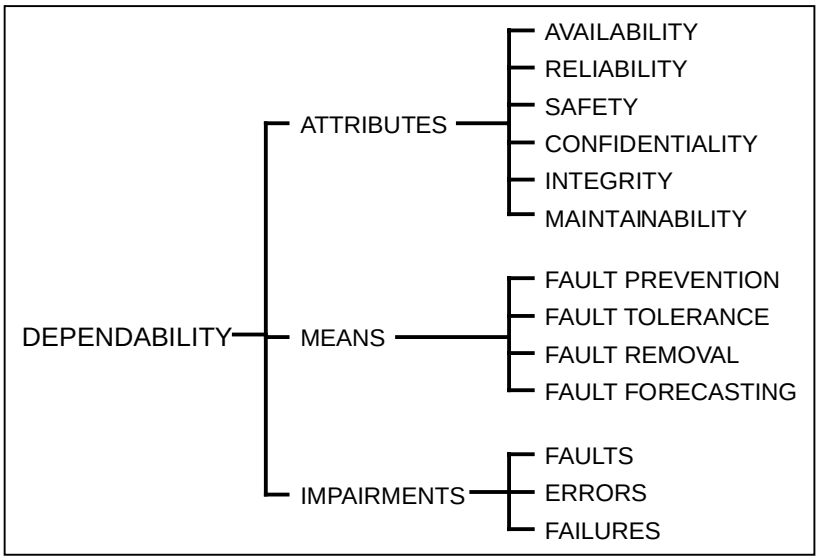

Figure 4 — The Dependability Tree

3.1.2 On the Function, Behaviour and Structure of a System

So far, a **system** has — implicitly — been considered as a whole, emphasis being placed on behaviour as perceived from the outside. A definition according to this "black box" vision is: a system is an entity having interacted or interfered, interacting or interfering, or likely to interact or interfere, with other entities, that is, other systems. The latter make up or will make up the **environment** of the system considered.[4] A system user is part of the environment *interacting* with the latter: a user provides inputs to the system and/or receives outputs from it. In other words, what distinguishes a user from the other parts of the environment is the fact that he uses *the service* delivered by the system.

As already pointed out in Section 3.1.1, the function of a system is what it is *intended for*. The **behaviour** of a system is what it *does*. What *enables it to do what it does* is its **structure** [Ziegler 1976]. Adopting the spirit of [Lee & Anderson 1990], a definition of a system from a structural point of view ("white box" or "glass box") is the following: a system is a set of components interconnected in order to interact; a **component** is another system, etc. Decomposition ends when a system is considered **atomic**: in other words, no subsequent decomposition can be envisaged either by nature or because it is devoid of interest. The term "component" must be understood in its broad sense: layers of a system as well as intra-layer components; in addition, a component being itself a system encompasses the relationships between the components that make it up. A more conventional definition of the structure of a system is what the system *is*. This definition remains appropriate as long as dependability impairments are not considered and, therefore, the structure is considered to be frozen. However, as dependability impairments can be structural changes, or can cause or result from structural changes, a structure can have states.[5] Hence a definition of the notion of state: a **state** is a condition of being *relative to a*

---

[4] a) Giving recursive definitions allows the relativity of the notion of system to be underlined according to the point of view considered: a system will not be looked at in the same way by a designer, its users and the maintenance teams.

b) Use of the past, present and future is intended to show that the system environment will change, particularly during the various phases of the life cycle. For example, the notion of "programming environment" can be integrated with the definition given, as well as with the physical environment to which a system can be confronted during its operational life

[5] One can therefore say that a structure also features a behaviour, particularly, relative to dependability impairments, even if the pace of changes considered relative to, on the one hand, the user's requests and, on the other the dependability impairments, are — as should be noted — radically different.

set of circumstances; this definition applies to the behaviour of a system as well as to its structure.[6]

Owing to its definition (the behaviour perceived by a user), the service delivered by a system is clearly an *abstraction* of the latter's behaviour. It is worth pointing out that this abstraction directly depends on the application for which the system is used. One example of this is the role played by time of this abstraction: time granularities of a system and of its users are usually different and vary according to the application concerned. In addition, the notion of service is not, of course, limited to outputs only but includes all interactions of interest to the user; for example, to scan sensors is clearly part of the service expected from a monitoring system.

So far, we have used the singular for function and service. Usually, a system implements more than one function and delivers more than one service. Thus, function and service can be considered as composed of function elements and service elements. For clarity, we will use the plural — functions, services — when it is necessary or useful to make a distinction between several elements of function or service.

Given the preceding definition for the structure of a system, the notions of function and service naturally apply to components. This is particularly relevant in the design process when pre-existing hardware or software components are incorporated into a system: the designer is more interested in the function of the component or service it delivers than its detailed (internal) behaviour.

The **specification** of the system, that is, an *agreed*[7] description of the function or service expected from the system, plays a pivotal role in dependability. Generally, the function or service is described or specified first in terms of what should be implemented or delivered according to the primary purpose of the system (for example, to carry out transactions, order or monitor a process, pilot a plane or guide a missile, etc.). With respect to security or safety systems, this description is usually completed by a statement of what should not occur (for example, hazardous states that could cause a catastrophe or the disclosure of sensitive information). This latter description leads to the identification of additional functions the system should implement to reduce the possibilities of what should not occur (e.g., identification of a user and verification of his rights).

In addition, the specification of these diverse functions can be:

---

[6] This definition is designed to lay the stress on a notion of state that depends directly on the phenomena and circumstances considered; for example: states relative to the information processing activities, states relative to the occurrence of failure, etc.

[7] An agreement is usually struck between two persons or groups of persons, physical or moral: the system vendor (in its broad sense: designer, manufacturer, seller, etc.) and its human users. The agreement may be implicit — during the purchase of a system with its specification and the user's manual or when employing off-the-shelf systems.

- expressed according to various points of view or degrees of detail: specification of the needs, design specification, implementation specification, etc.,
- decomposed in accordance with the absence or presence of a failure; the first case relates to what is usually referred to as the *nominal* mode of operation and the second may deal with the so-called *degraded* mode of operation, if the remaining resources are no longer adequate for the nominal mode to be provided.

As a result, there exist several specifications, not one only, and a system can fail relative to one of them while still satisfying the others.

The expression of the functions of a system is an activity that is naturally initiated in the very early stages of a system development. However, generally, it is not limited to this phase of a system's lifetime. In fact, experience has shown that the process of specifying the system functions has to be pursued throughout the system's lifetime, as it is difficult to identify what is expected of it.

3.1.3 Human-made Faults

Faults and their sources are highly diverse. Five main points of view can be considered to classify them. These are the phenomenological cause, nature, phase of creation or occurrence, situation relative to the system boundaries, and persistence [Laprie *et al.* 1998, chapter 1] (see Figure 5).

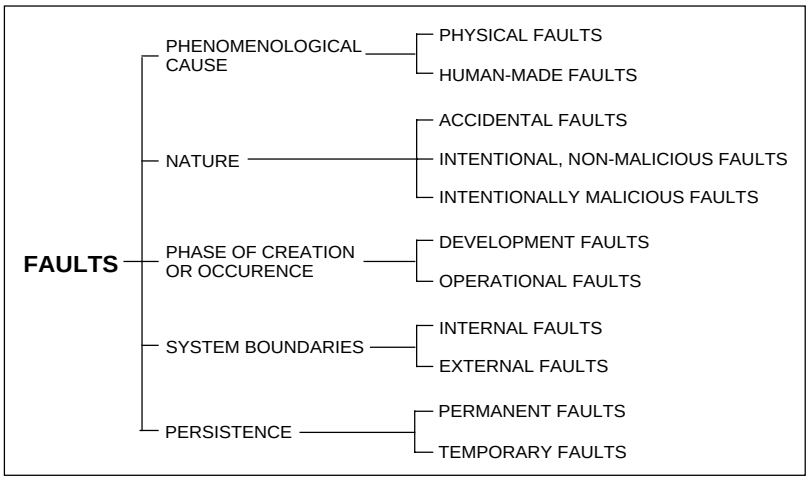

Figure 5 — Classes of Elementary Faults

Of particular interest to MAFTIA, are *human-made faults*, which correspond to four classes of combined faults:

- **design faults**, which are development faults, accidental or intentional with no malicious intent;
- **interaction faults**, which are external faults, accidental or intentional with no malicious intent;

- **malicious logic**, which consists of internal, intentionally malicious faults;
- **intrusions**, which are external, operational, intentionally malicious faults.

Some comments upon these classes of human made faults:

1. Intentional design faults with no malicious intent usually result from tradeoffs during the development, made with a concern for maintaining a suitable level of system performance or for facilitating system use, or even for economic reasons; these faults can be sources of security impairments in the form of hidden channels. Intentional interaction faults with no malicious intent can result from an operator attempting to address an unexpected event or deliberately acting in breach of procedures without realizing the detrimental effects of his action. Generally, intentional faults performed without malicious intent are only identified as such after they have caused an unacceptable behaviour of the system, hence a failure.
2. Interaction faults are defined above as a class of human-made external faults that includes both accidental faults and intentional faults without malicious intent. These sub-classes should not be confused with the two error classes commonly considered for operators [Norman 1983]: intention errors (i.e., errors in the formulation of the interaction objective) and execution errors (i.e., errors in implementing these intentions).
3. Malicious logic covers development faults such as Trojan horses, trapdoors, logic or timing bombs, and operational faults (for the system considered) such as viruses and worms [Landwehr *et al.* 1994]. These faults can be defined as follows:

    - a **logic bomb** is part of a program that remains dormant in the host system till a certain time or an event occurs, or certain conditions are met, unleashing devastating consequences for the host system;
    - a **Trojan horse** is a program performing an illegitimate action while giving the impression of being legitimate; the illegitimate action can be the disclosure or modification of information (attack against confidentiality or integrity) or a logic bomb;
    - a **trapdoor** is a means of circumventing access control mechanisms; it is a flaw in the security system due to an accidental or intentional design fault (Trojan horse in particular);
    - a **virus** is a program segment that replicates itself and joins another program (system or application) when it is executed, thereby turning into a Trojan horse; a virus can carry a logic bomb;
    - a **worm** is an independent program that replicates itself and propagates without the users being aware of it; a worm can also carry a logic bomb.

4. As will be detailed in Section 3.3.1, intrusions can only be successful if vulnerabilities such as design faults are present; evident and nevertheless interesting similarities can be found between an intrusion and an external accidental fault "taking advantage" of a lack of shielding. However, as noted in Section 3.3.1, despite their external nature, intrusions may also be attempted by operators or managers of the system overriding their commission.

3.1.4 Fault Tolerance

Fault tolerance [Avizienis 1967] is carried out by error processing and by fault treatment [Anderson & Lee 1981]. **Error processing** is aimed at removing errors from the computational state, if possible before failure occurrence; **fault treatment** is aimed at preventing faults from being activated — again.

Error processing can be carried out via three primitives:

- **error detection**, which enables an erroneous state to be identified as such;
- **error diagnosis**, which aims to assess the damage caused by the detected error, or by errors propagated before detection;
- **error recovery**, where an error-free state is substituted for the erroneous state; this substitution may take on three forms:
  - **backward recovery**, where transformation of the erroneous state consists of bringing the system back to a state already occupied prior to error occurrence; this involves the establishment of recovery points, which are points in time during the execution of a process for which the then current state may subsequently need to be restored;
  - **forward recovery**, where transformation of the erroneous state consists of finding a new state, from which the system can operate (frequently in a degraded mode);
  - **compensation**, where the erroneous state contains enough redundancy to enable its transformation into an error-free state.

When backward recovery or forward recovery are utilised, error detection must precede error recovery. These techniques are not antagonistic: a backward recovery may first be attempted; if the error persists, forward recovery may then be undertaken. In the latter case, the error diagnosis must take place before undertaking recovery; error diagnosis is not — in theory — necessary in the case of a backward recovery provided the mechanisms for implementing error recovery have not been affected [Anderson & Lee 1981].

The addition of error detection mechanisms to the component's functional processing capabilities leads to the notion of **self-checking component**, for the hardware [Carter & Schneider 1968, Wakerly 1978, Nicolaïdis *et al.* 1989] or software [Yau & Cheung 1975, Laprie *et al.* 1990]. One of the main advantages of self-checking components is

the possibility of clearly defining *error confinement domains* [Siewiorek & Johnson 1982]. When error compensation is carried out in a system made up of self-checking components partitioned into given classes of task execution, error recovery reduces to a switchover from a failed component to a non-failed one within the same class. On the other hand, error compensation may be applied systematically, even in the absence of errors, thereby providing **fault masking** (e.g., through a majority vote). Error detection is not, then, strictly speaking, needed to perform recovery. However, to avoid an undetected decrease in the redundancy available during a component failure, practical implementations of masking usually include an error detection facility, which may in this case be initiated *after* recovery.

The operational time overhead (in terms of execution) needed for error processing may vary considerably according to the technique used:

- in the case of error recovery based on backward recovery or forward recovery, the time overhead is more important when an error occurs than when it does not. In the case of backward recovery, the time overhead consists in establishing recovery points and, therefore, in laying the groundwork for error processing;
- in error compensation, the time overhead remains unchanged or almost the same whether or not there exists an error.[8]

In addition, error compensation is much faster than with a backward recovery and forward recovery owing to the much more important structural redundancy. This remark:

- carries a certain weight in practice because it often conditions the choice of a fault tolerance strategy relative to the time granularity of the system user;
- introduces a relationship between operational time overhead and structural redundancy. More generally, a redundant system always provides redundant behaviour, incurring at least some operational time overhead. The time overhead may be small enough not to be perceived by the user, which means only that the *service* is not redundant. An extreme form of time overhead is "time redundancy" (redundant behaviour obtained by repetition), which needs to be at least initiated by a structural redundancy, even in a limited form. Typically, the greater the structural redundancy, the lower the time overhead.

The first step in fault treatment is **fault diagnosis**, which consists in determining the causes of errors in terms of localisation and nature. Then, the steps needed to fulfil the main objective of fault treatment are carried out to prevent faults from being reactivated, hence **fault isolation**. To do so, the components deemed faulty are

[8] In all cases, the time to update the system state tables increases the time overhead.

removed from the subsequent execution process[9]. If the system is no longer able to provide the same service as before, a **reconfiguration** may be envisaged by modifying the system structure so that fault-free components provide an adequate, although degraded, service. Reconfiguration may mean scrapping a number of tasks, or reallocating some of them to the remaining devices.

If it is thought that the fault has vanished after error processing or if its probability of recurrence is low enough, isolation becomes unnecessary. As long as fault isolation has not been undertaken, a fault is considered **soft**; undertaking isolation means that the fault is **hard**, or **solid**. At first sight, the notions of soft fault and hard fault may seem synonymous with that of temporary fault and permanent fault. Indeed, temporary faults can be tolerated without the need for fault treatment since error recovery should theoretically directly suppress the effects of a temporary fault, which will vanish unless a permanent fault is created by the propagation process. In fact the notions of soft fault and hard fault are useful, for the following reasons:

- distinguishing between a permanent fault and a temporary one is not easy and highly complex since a temporary fault vanishes after a certain amount of time, generally, before diagnosis is carried out and distinct classes of faults can give rise to similar errors; thus, the notion of soft or hard fault carries implicitly the subjectivity associated with these difficulties, including the fact that a fault can be considered soft following unsuccessful diagnosis;

- their ability to take into consideration subtleties in the action modes of certain transient faults; for example, can a dormant fault due to the action of alpha particles, or of heavy ions in space, on memory elements (in the broad sense of the word, including flip-flops) be regarded a *temporary* fault*?* This fault is definitely a *soft* fault however.

The foregoing considerations apply to physical faults as well as design faults: the fault classes that can be tolerated in practice depend on the fault assumptions made in the design process, which are conditioned by the independence of redundancies relative to the fault creation and activation processes. An example is provided by considering tolerance of physical faults and tolerance of design faults. A (widely used) method to attain fault tolerance is to perform multiple computations through multiple channels. When tolerance of physical faults is foreseen, the channels may be identical, based on the assumption that hardware components fail independently. However, such an approach is not suitable for the tolerance of design faults. To tolerate design faults, the multiple channels have to provide *identical services* through *separate designs and implementations* [Elmendorf 1972, Randell 1975, Avizienis 1978], i.e., through **design diversity.** Design diversity is intended to tolerate permanent design faults. On the other hand, a backward recovery based error processing usually enables temporary design faults to be tolerated [Gray 1986].

---

[9] Usually, removed faulty components can be repaired and re-inserted into the system; such curative maintenance can be considered as an ultimate form of fault-tolerance.

An important aspect of co-ordinating the activities of multiple components is ensuring that the propagation of errors has no effect on fault-free components. This is particularly important when a given component must transmit a piece of information to other components. Typical examples of *information obtained from a single source* are local sensor data, the value of a local clock, the local perception of the state of the other components, etc. As a result, fault-free components must *agree* on how to use consistently the information obtained and, therefore, protect against possibly inconsistent failures (e.g., atomic broadcast [Cristian *et al.* 1985], clock synchronisation [Lamport & Melliar-Smith 1985, Kopetz & Ochsenreiter 1987] or membership protocols [Cristian 1988]). It should be noted, however, that the unavoidable presence of structural redundancies in any fault tolerant system requires a resource distribution at one level or another, leading to the persistence of the consensus problem. Geographically localised fault-tolerant systems may employ solutions to the agreement problem that would be deemed too costly in a "classical" distributed system of components communicating by messages (e.g. inter-stages [Lala 1986], multiple stages for interactive consistency [Frison & Wensley 1982]).

The knowledge of certain properties of the system may allow the required redundancy to be limited. Classical examples are given by regularities of a structural nature: error detecting and correcting codes [Peterson & Weldon 1972], robust data structures [Taylor *et al.* 1980], multiprocessors and networks [Pradhan 1986, Rennels 1986], algorithm-based fault tolerance [Huang & Abraham 1982]. The faults that can be tolerated are then dependent upon the properties considered since these properties are directly involved in the fault assumptions made during the design.

Warning users about the failure of a component is extremely important. This can be taken into consideration within the framework of exceptions [Melliar-Smith & Randell 1977, Cristian 1980, Anderson & Lee 1981]. *Exception handling* facilities provided in some languages may constitute a convenient way for implementing error recovery, especially forward recovery[10].

Fault tolerance is (also) a recursive concept; the mechanisms designed to tolerate faults must be protected against the faults likely to affect them. Examples are given by the replication of voters, self-checking controllers [Carter & Schneider 1968], through the notion of "stable" memory [Lampson 1981] in recovery data and programs.

Fault tolerance is not limited to accidental faults. Protecting against intrusions has long relied on cryptography [Denning 1982], which can be viewed as a form of tolerance in that ciphered information can be inspected by an intruder without compromising its confidentiality. Certain error detection mechanisms are designed for accidental as well as intentional faults (e.g., memory access protection techniques),

---

[10] The term "exception", due to its origin of coping with exceptional situations — not only errors — should be used carefully in the framework of fault tolerance: it could appear as contradicting the view that fault tolerance is a natural attribute of computing systems, taken into consideration from the very initial design phases, and not an "exceptional" attribute.

and approaches have been put forward to tolerate both intrusions and physical faults [Fray *et al.* 1986, Rabin 1989], and to tolerate malicious logic faults [Joseph & Avizienis 1988]. The MAFTIA project aims to follow this very approach, by building on this earlier work and extending it to the case of large distributed systems.

3.2 Security Properties

Dependability has been defined in Section 3.1.1 as "that property of a computer system such that *reliance can justifiably be placed on the service it delivers*". According to the application requirements, the service may be requested to exhibit certain functional or non-functional properties, such as, for instance, accuracy of the computation results, respect of real-time deadlines, or other "quality-of-service" characteristics. Some of these properties have to be fulfilled continuously: this is generally the case of *safety* and *confidentiality* properties (an item of information that is no longer secret cannot become secret again). Other properties can alternate between being fulfilled and not being fulfilled, as long as they are fulfilled when the service is delivered: this is generally the case of *integrity* properties (an item of information only needs to be correct at the instant it is read).

Many security properties can be defined in terms of the confidentiality, integrity and availability of the information or the service itself, or of some meta-information[11] related to the information or service. Examples of such meta-information are:

- time of a service delivery, or of creation, modification or destruction of an item of information;
- identity of the person who has realised an operation: creator of an item of information, author of a document, sender or receiver of an item of information, etc.;
- location or address of an item of information, a communication entity, a device, etc.;
- existence of an item of information or of the service;
- existence of an information transfer, or a communication channel, or of a message, etc.;
- occurrence of an operation;
- sensitivity level of an item of information or meta-information;
- certainty or plausibility level of an item of information or meta-information;
- etc.

[11] Of course, at some level (e.g., at the operating system level), meta-information is "real" information.

For example, *accountability* [CEN 13608-1, ISO 7498-2, Trouessin 2000] corresponds to the availability and integrity of a set of meta-information concerning the existence of an operation, the identity of the person who realised the operation, the time of the operation, etc. *Anonymity* is the confidentiality of the identity of the person, for instance, who realised (or did not realise) an operation. *Traffic analysis* is an attack against the confidentiality of communication meta-information, to gain knowledge of the existence of a channel, of the existence of a message, identities, locations or addresses of the message sender and receiver, the time of a communication, etc.

Privacy is confidentiality with respect to personal data, which can be either "information" (such as the content of a registration database), or "meta-information" such as the identity of a user who has performed a particular operation, or sent a particular message, or received the message, etc.

Authenticity is the property of being "genuine". For a message, authenticity is equivalent to integrity of both the message content (information integrity) and of the message origin, and possibly of other meta-information such as time of emission, classification level, etc. (meta-information integrity). Similarly, a document is authentic if its content has not been altered (information integrity) and optionally if the declared author is the real author and not a plagiarist, if the publication date is correct, etc. (meta-information integrity). Likewise, an alleged user is authentic if the declared identity is the real identity of that person. *Authentication* is the process that gives confidence in authenticity.

Non-repudiation corresponds to the availability and integrity of some meta-information, such as creator identity (and possibly time of creation) for non-repudiation of origin, or such as reception and receiver identity for non-repudiation of reception.

It is conjectured that all security properties can be expressed in terms of the availability, integrity and confidentiality properties applied to information and meta-information.

3.3 Intrusion-tolerance Concepts

3.3.1 Intrusion vs. Attack

A first terminology issue concerns the use of the very word "intrusion" to designate an *external*, operational, intentionally malicious fault (cf. Section 3.1.3, page 34). Etymologically, the word "intrusion" comes from the Latin *intrudere* (to thrust in) but current usage covers both senses of "illegal penetration" and "unwanted interruption". Even a malicious interaction fault perpetrated by an *insider* can thus be classed as an intrusion since the intent is to carry out an operation on some resource that is unwanted by the owner of that resource.

A possible alternative to “intrusion” would be the word “attack”. However, it would seem that both terms are necessary, but for different concepts. A system can be attacked (either from the outside or the inside) without any degree of success. In this case, the attack exists, but the protective “wall” around the system or resource targeted by the attack is sufficiently efficacious to *prevent* intrusion. An attack is thus an *intrusion attempt* and an intrusion results from an attack that has been (at least partially) successful.

In fact, there are two underlying causes of any intrusion:

1. The malicious act or *attack* that attempts to exploit a weakness in the system,
2. At least one weakness, flaw or *vulnerability*, which is an accidental fault, or a malicious or non-malicious intentional fault, in the requirements, the specification, the design or the configuration of the system, or in the way it is used.

This is a similar situation to that of external (or, to be precise, “externally-induced”) physical faults: a heavy ion approaching the system from outside is like an attack. The aim of shielding is to prevent the heavy ion from penetrating the system. If the shielding is insufficient, a fault will occur (e.g., a bit-flip). Mechanisms can be implemented inside the system to tolerate such “external” faults.

Since we are essentially concerned with techniques aimed at providing security guarantees in spite of imperfect “shielding” of the considered system, in the sequel we will prefer to refer to just vulnerabilities and the intrusions that result from their successful exploitation by an attacker (who is, *ipso facto*, an *intruder*). Typical examples are:

1. An outsider penetrating a system by guessing a user password: the vulnerability lies in the configuration of the system, with a poor choice of password (too short, or susceptible to a dictionary attack).
2. An insider abusing his authority (i.e., a misfeasance): the vulnerability lies in the specification or the design of the (socio-technical) system (violation of the principle of least privilege, inadequate vetting of key personnel).
3. An outsider using “social engineering”, e.g., bribery, to cause an insider to carry out a misfeasance on his behalf: the vulnerability is the presence of a bribable insider, which in turn is due to inadequate design of the (socio-technical) system (inadequate vetting of key personnel).
4. A denial-of-service attack by request overload (e.g., the February 2000 DDoS[12] attacks of Web sites): the vulnerability lies partly in the very requirements of the system since it is contradictory to require a system to be completely open to all well-intended users and closed to malicious users. This particular type of attack also exploits design or configuration faults in the many Internet-connected hosts that were penetrated to insert the zombie daemons required to mount a co-

[12] DDoS: Distributed Denial of Service.

ordinated distributed attack [Garber 2000]. A third vulnerability, that prevents effective countermeasures from being launched, resides in a design fault caused by Internet service providers not implementing ingress/egress filtering (which would enable the originating IP source address to be traced).

3.3.2 Outsiders vs. Insiders

In the previous section, we referred to attackers as being either "outsiders" or "insiders". What exactly is the distinction between the two?

In common parlance, an insider is "a person within a society, organisation, etc. or a person privy to a secret, especially when using it to gain advantage" [OMED 1992].

The first part of this definition can be interpreted in terms of the *rights* of the considered person. A person has a *right* on a specified object within the system if and only if he is authorised to perform a specified operation on that object — a right is thus an object-operation pair. The set of rights of the considered person is that person's *privilege*. An *outsider* might thus be defined as a person who has no privilege, i.e., no rights on any object in the system. Inversely, an *insider* is thus any individual who has some privilege, i.e., some rights on objects in the system.

Consider now the case of an "open" system, such as a public web server. Such systems allow any user at least read access rights on certain objects within the system so, with the above definitions, all users would be considered as insiders. The very notion of an outsider, as defined above, is only relevant for *closed* systems.

An alternative distinction is thus necessary for open systems. The second part of the dictionary definition of an insider relates both to the *knowledge* of the considered person and the *illegal use* of this knowledge[13]. The distinction between outsider and insider must thus be made in terms of the types of attack that can be perpetrated by the considered person: unauthorised increase in privilege or abuse of privilege. The distinction between "inside" or "outside" must be made with respect to a subset of the universe of object-operation pairs of the considered system, rather than the complete system:

- **outsider**: a human user not authorised to perform any of a set of specified operations on a set of specified objects, i.e., a user whose (current) privilege does not intersect the considered domain of object-operation pairs.
- **insider**: a human user authorised to perform some of a set of specified operations on a set of specified objects, i.e., a user whose (current) privilege intersects the considered domain of object-operation pairs.

[13] The relationship between *knowledge* and *right* needs to be explored further, especially in terms of concepts such as the *need to know* and the *principle of least privilege*.

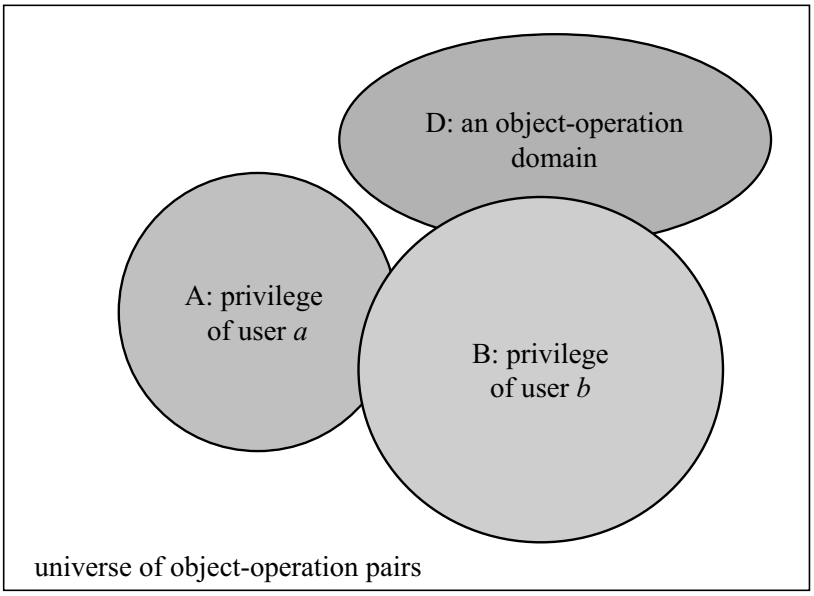

Figure 6 — Outsider (user a) vs. Insider (user b) with respect to Domain D

In Figure 6, user *a* is currently an outsider with respect to domain *D* since his privilege does not intersect the considered domain *D.* User *b* is an insider with respect to domain *D* but an outsider with respect to sub-domain *D-B.* According to these definitions of outsider and insider, two basic forms of intrusion are:

- **outsider intrusion**: an unauthorised increase in privilege, i.e., a change in the privilege of a user that is not permitted by the system's security policy.
- **insider intrusion**: an abuse of privilege or misfeasance, i.e., an improper use of authorised operations.

3.3.3 Security methods

Equating *intrusion* and *vulnerability* with fault, and applying the definitions given in Section 3.1.1, we can obtain *a priori* eight methods for ensuring or assessing security. However, not all of these eight methods are distinguishable or indeed meaningful. First, since an intrusion cannot occur in the absence of vulnerability, intrusion tolerance and vulnerability tolerance are equivalent in the sense that tolerance of an intrusion implies tolerance of the vulnerability or vulnerabilities that were exploited to perpetrate the intrusion. To conform to current usage, we will refer to *intrusion tolerance*. Second, fault removal normally refers to verification methods (including testing) aimed at finding *internal* faults. As such, only vulnerability removal is meaningful, since intrusions are external faults. Countermeasures aimed against intruders are better classified as a form of intrusion *tolerance* since, like maintenance or other fault treatment actions, they aim to maintain or to restore the ability of the system to fulfil its function. We thus obtain six meaningful methods:

vulnerability prevention: how to prevent the occurrence or introduction of vulnerabilities;

This includes measures going from formal specification, rigorous design and system management procedures, up to and including user education (e.g., choice of passwords).

intrusion prevention: how to prevent the occurrence of intrusions;

This includes measures such as social pressure or deterrence, as well as vulnerability prevention (see above).

intrusion tolerance: how to provide a service capable of implementing the system function despite intrusions;

Admitting that intrusion prevention, and vulnerability prevention and removal are always imperfect, intrusion tolerance aims to ensure that the considered system provides security guarantees in spite of partially successful attacks.

vulnerability removal: how to reduce the presence (number, severity) of vulnerabilities;

This covers verification procedures such as formal proof, model-checking and testing, specifically aimed at identifying flaws that could be exploited by an attacker. Identified flaws may then be removed by applying a security patch, withdrawing a given service, changing a password, etc.

vulnerability forecasting: how to estimate the presence, creation and consequences of vulnerabilities.

This includes the gathering of statistics about the current state of knowledge regarding system flaws, and the difficulties that an attacker would have to take advantage of them.

intrusion forecasting: how to estimate the presence and consequences of intrusions.

This includes the gathering of statistics about how frequently people run attacks, and against whom and how.

3.3.4 Intrusion Detection

In Section 3.1.1, a *fault* is defined to be the adjudged or hypothesised cause of an *error*, the latter being that part of the system state that *may lead to a failure*.

Whereas a security failure is naturally defined in terms of loss of confidentiality, integrity or availability, there is currently no agreed definition of what constitutes an *error* from the security viewpoint. However, current literature refers to “intrusion detection” which, from the dependability concept viewpoint, might lead one to equate

intrusion with "error", rather than "fault"[14]. In reality, current literature uses the term "intrusion detection" to cover a *spectrum* of techniques. To paraphrase [Halme & Bauer]: "Intrusion detection may be accomplished:

- after the fact (as in post-mortem audit analysis),
- in near real-time (supporting SSO[15] intervention or interaction with the intruder, such as a network trace-back to point of origin), or
- in real time (in support of automated countermeasures)."

From the dependability concept viewpoint, these three types of intrusion detection can be interpreted respectively as:

- off-line fault diagnosis (as part of curative maintenance);
- error detection and on-line fault diagnosis (to an operator-assisted fault treatment facility);
- error detection (as a preliminary to automatic error recovery), or error detection and on-line fault diagnosis (as a preliminary to automatic fault treatment).

Further confusion is introduced by the opposition in [Halme & Bauer] between a "manually reviewed IDS"[16] (called a passive IDS in [Debar *et al.* 1999]) and "Intrusion Countermeasure Equipment (ICE)" or "autonomously acting IDS" (sic) (called an active IDS in [Debar *et al.* 1999]), which clearly go beyond just detection.

Again, from [Halme & Bauer], intrusion detection techniques can be divided into:

- anomaly detection techniques, that compare observed activity against normal usage profiles (in [Debar *et al.* 1999], these are called behaviour-based methods);

---

[14] Note, however, that it is also quite common in the literature on tolerance of physical faults to find the term "fault detection" used in one of two ways:

a) As a clumsy synonym for "error detection" (since detection of an error implies, rather indirectly and perhaps falsely, the "detection" of its cause)

b) As the designation of a mechanism that seeks out (dormant) faults by running a test procedure to activate them as errors that can be detected by an error detection mechanism.

[15] SSO: System Security Officer.

[16] IDS: Intrusion Detection System.

- misuse detection techniques, that check for known undesired activity profiles (in [Debar *et al.* 1999], these are called knowledge-based methods).[17]

Here "anomaly" is definitely being used in the traditional sense of "error", whereas "misuse" has an element of fault diagnosis since error patterns related to previously-identified intrusions are being searched for (see Figure 7).

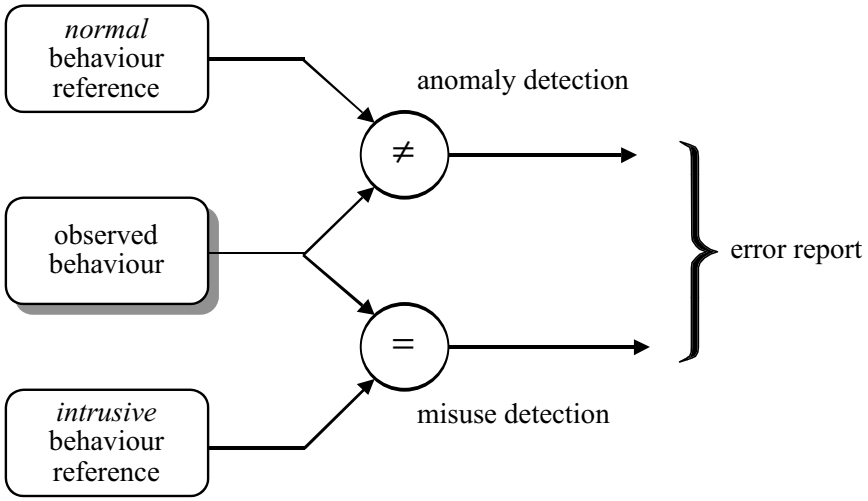

Figure 7 — Detection Paradigms

"Intrusion detection" is thus commonly used in a very broad sense to indicate a set of system administration tools that provide error detection *and* some degree of fault diagnosis ("these errors — anomalies and misuses — were probably due to intrusions"), and reports these partial conclusions either:

- to a system administrator (the SSO) who might carry out further diagnosis and initiate appropriate countermeasures and/or litigation, or
- to an automatic countermeasure mechanism.

Given this current broad usage of the term, we will avoid using "intrusion detection" in the limited sense of "error detection". However, we will also refrain from including automatic countermeasures as part of intrusion *detection* since this notion clearly goes beyond *detection* to embrace a form of fault treatment. Instead, we will adopt the following definition that concords both with current usage and with traditional dependability concepts:

intrusion detection system: a facility aimed at discovering the presence of intrusions; the facility consists of a set of sensors or error detectors (including anomaly and misuse detectors) and an intrusion diagnosis

---

[17] [Halme & Bauer] actually also identifies "hybrid misuse/anomaly detection" and "continuous system health monitoring". The former is clearly not a separate form of detection and the latter can be viewed as a form of anomaly detection, since it applies to "suspicious changes in system-wide activity measures and system resource usage".

mechanism that collates sensor outputs to decide whether the detected errors are symptomatic of intrusion.

It should be noted that most currently available intrusion detection systems do not include any intrusion diagnosis mechanisms. The explicit recognition of the fact that misuses and anomalies are indeed errors that can be caused by any sort of fault is an initial result of the MAFTIA project. Indeed, a good intrusion detection system *requires* such a fault diagnosis mechanism to minimise the rate of false alarms caused by errors due to other classes of faults (e.g., design faults in the reference for defining "misuse" or "anomalies", accidental interaction faults such as mistyping a password, etc.).

3.3.5 Loss of Confidentiality

There is no equivalent in traditional fault-tolerance of "loss of confidentiality". Moreover, loss of confidentiality may be very difficult to detect. Loss of confidentiality means that some information is present in a place (computer storage, paper, some user's brain…) where it should not be. This is an error, i.e., "part of the system state which may lead to a failure", the failure being the disclosure of this information outside the considered system. This error, at least theoretically, can be detected: the system state is different to what it would be in the absence of error, so comparison of different copies should detect that. Moreover, if "undetachable" labels are attached to items of information, it may be possible to detect that this information should not be where it is, e.g., presence of information labelled as secret in an unclassified file (the labels form a sort of error-detecting code). As for other errors, a confidentiality error can be detected by a detected failure (e.g., publication of the confidential information in a newspaper). The failure may remain undetected (the information is propagated to unauthorised users, but not published) in the same way as other failures can remain undetected for a long time (e.g., the Therac system).

Another issue with loss of confidentiality is the inherent difficulty of recovery. Since a lost secret cannot become secret again, the only apparent way to recover confidentiality is to substitute a new secret for the lost secret, e.g., by asking a user to choose a new password (i.e., a form of forward recovery). This can be done pre-emptively in order to limit the duration of the threat posed by a compromised secret.

3.3.6 Security Guarantees

In this section, we discuss a fundamental restriction on what sort of security guarantees can be given to system users. To do this, we introduce the notion of a system access point as an intrusion containment region. This notion is introduced by analogy with fault containment regions and correctness properties in Byzantine agreement.

The problem of Byzantine agreement [Lamport *et al.* 1982] is stated in terms of a transmitter (the commanding general) and a set of receivers (the lieutenant generals) to which an order must be sent. The loyal (i.e., non-faulty) generals must agree on the

order, despite the fact that traitorous (i.e., faulty) generals can alter, delay or otherwise destroy messages that the agreement protocol requires them to send. Byzantine agreement is achieved if the following two properties are satisfied:

- All loyal lieutenants obey the same order.
- If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

In this example, it is clearly inappropriate to require a disloyal lieutenant to obey the same order as the loyal lieutenants or to require that loyal lieutenants obey the "order" of a disloyal commanding general (a disloyal commanding general could give conflicting orders to different lieutenants). The important issue is that any solution to the problem should not place any requirement on what an arbitrarily faulty component may do[18]. Consequently, correctness of the solution can only be defined with reference to *fault-free* components.

When defining the correctness of a mechanism designed to tolerate *intrusions*, a similar restriction to fault-free components must apply since no assumptions can be made about what an intruder or a corrupted component can or cannot do.

An intrusion-tolerant system is aimed at guaranteeing certain security properties, despite the fact that some components of the system might be compromised, by either corrupt system administrators or corrupt users. Consider now that users (and administrators) of the considered computer system access the latter by means of an "access point", i.e., a terminal or a workstation (Figure 8).

[18] In practice that there is always *some* assumption about what a faulty component is not allowed to do in the sense that it should not be able to change the *structure* of the considered fault-tolerant system. For example, in Byzantine agreement, a disloyal general is only allowed to change messages (in arbitrary ways), but is not allowed to kill his colleagues, or to create clones of himself to falsify the majority.

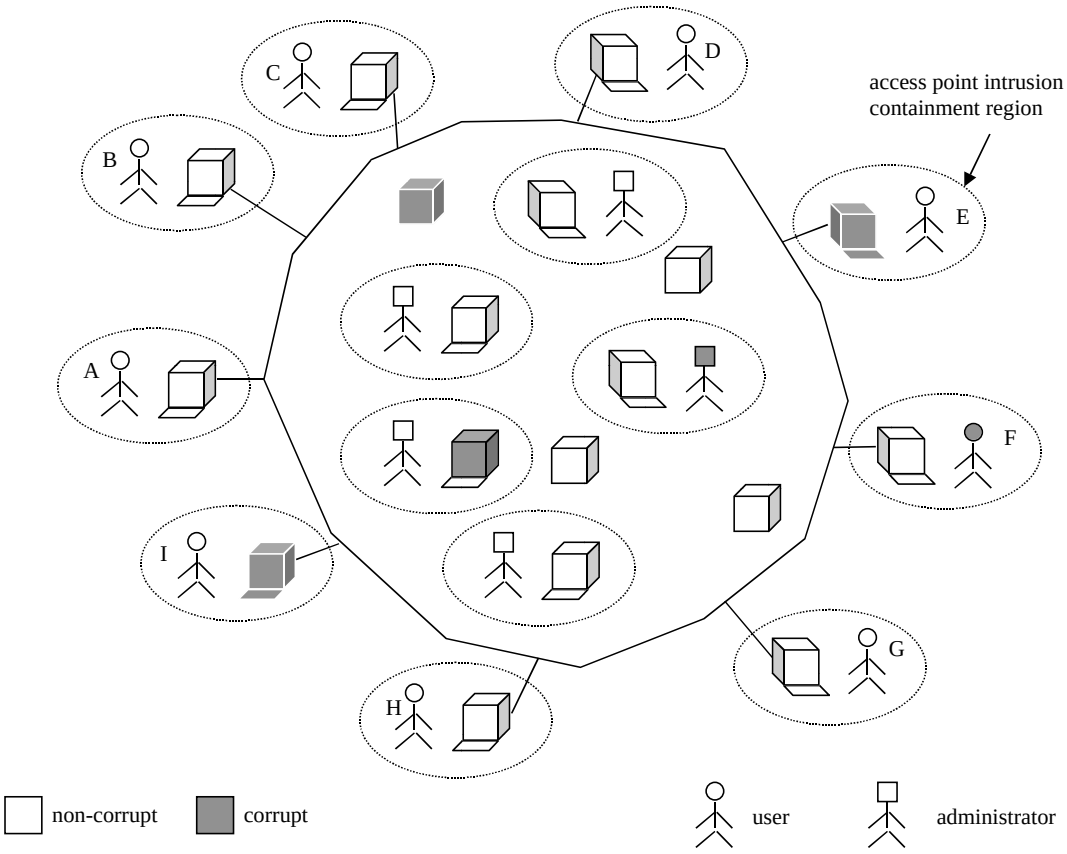

Figure 8 — Corrupt vs. Non-corrupt Access Points and Users

If an access point has been corrupted (e.g., by a Trojan horse that logs or modifies confidential inputs or outputs) then it is clear that the user of that access point cannot be given any security guarantees (case of users E and I in Figure 8).[19] Also, it is of no interest to give a security guarantee to a corrupt user, even if his access point is non-corrupt (case of user F in Figure 8). Indeed, from the security viewpoint, a user and his corresponding access point constitute a single "intrusion containment region" (by analogy with "fault containment regions" of traditional fault-tolerance[20]): it is of no import to the rest of the system whether a user or his access point is corrupt.

Consequently, it is clear that security guarantees can be given only with respect to a set of non-corrupt access point intrusion containment regions, e.g., access points A, B, C, D, G and H in Figure 8. For these users, an intrusion-tolerant system should be able to provide guarantees about the confidentiality, integrity and availability of the data owned (or, equivalently, the service purchased) by those users, despite the fact that there are (a certain number of) corrupt components, administrators or users of the system. A similar concept is introduced in [Pfitzmann & Waidner 1994], where

---

19 The protection of an access point against intrusions should thus be under the responsibility of the corresponding user: a reckless user cannot be given any security guarantees.

20 A fault containment region is a set of components that is considered to be atomic from the point of view of fault-tolerance. See, for example, [Smith 1986].

security properties are specified in terms of a subset of access points that together constitute the interface to the concerned parties, i.e., those parties who mutually trust each other but distrust other parties (other users, access points or system components).

3.3.7 Intrusion Tolerance

In the definitions relative to fault tolerance in Section 3.1.4 a distinction is made between *error processing* (error detection, error diagnosis and error recovery), aimed at preventing errors from leading to (catastrophic) failure, and *fault treatment* (fault diagnosis, fault isolation, and reconfiguration), aimed at preventing the recurrence of errors.

Error detection is a necessary preliminary to achieving backward or forward recovery, or compensation by switchover, but is not strictly necessary if compensation is carried out systematically (i.e., fault masking). However, irrespectively of the error recovery method employed (if any), error detection is necessary if subsequent fault treatment or curative maintenance actions are to be undertaken.

It is interesting to re-examine these notions when the set of faults to be tolerated includes intrusions.

By definition, error-processing techniques apply to all errors irrespectively of the specific faults that caused them. This does not mean that the *design* of an error-processing technique is independent of the hypothesised fault model, but it does mean that the possible *recovery* actions do not depend on particular faults within that model. For example, as discussed in Section 3.3.4, misuse detection is a form of error detection that would be meaningless if the considered fault model did not include intrusions. However, if the detection of misuse is to trigger an error recovery action, that action will be triggered even if the detected “misuse” was due, say, to an accidental power glitch.

Remedial actions that are specific to intrusions can only be carried out when fault diagnosis has been carried out, to decide whether the detected errors were indeed caused by intrusions. In the core dependability concepts, these remedial actions are part of fault treatment. When applied to intrusions, these remedial actions can include countermeasures such as intrusion isolation (e.g., blocking traffic from a host that is diagnosed as corrupt) and system reconfiguration (e.g., changing the settings of firewalls or routers), as well as curative maintenance (e.g., vulnerability removal) and the initiation of litigation.

The various views of intrusion detection and intrusion tolerance developed in this section and the previous section are illustrated in Figure 9.

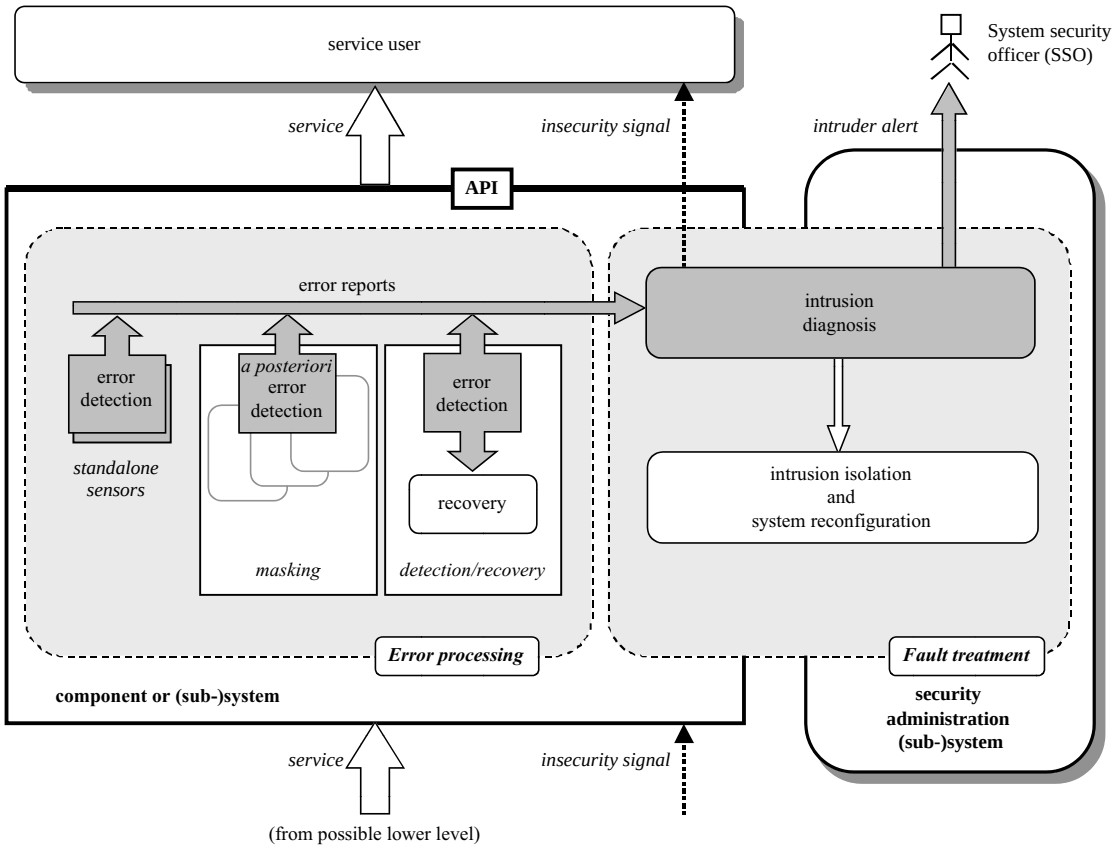

Figure 9 — Intrusion-detection and Tolerance Framework

The figure shows a component or (sub-)system offering some service over an API to a higher level component, using the service(s) offered by possible lower level components. Taking inspiration from the "ideal fault-tolerant component" of [Anderson & Lee 1981], these top and bottom interfaces include "insecurity signals" aimed at informing the service user that the service has been (might have been) compromised.

The considered component may implement fault-tolerance using either a masking scheme (e.g., the FRS technique, which can compensate errors due to both accidental faults and intrusions [Fraga & Powell 1985, Rabin 1989, Deswarte *et al.* 1991, Fabre *et al.* 1994]) or an error detection and recovery scheme. Whether masking or detection-and-recovery is used, detected errors are reported to the fault treatment facilities, which can be partly internal to the considered component or situated within a global *security administration (sub-)system.* The fault-treatment facilities include means for intrusion diagnosis, intrusion isolation, and automatic or manual system reconfiguration or intrusion-specific countermeasures.

The considered component may also include other standalone sensors in the form of misuse, anomaly and other error detectors, in addition to those error detectors included within any internal fault-tolerance mechanisms. The intrusion-detection system (in the sense defined in the previous section) consists of the set of all such sensors, and the intrusion diagnosis mechanisms included within the security

administration system. The components of the intrusion-detection system are shown in dark grey.

3.4 Glossary

> This glossary is provided as an aid to reading this chapter. It should not be considered independently of the body of the chapter.
>
> For some terms, only dictionary definitions have been given. It was felt necessary to include these terms in the glossary, but the corresponding definitions are particularly subject to change after further scientific discussion.
>
> Some terms, indicated by the symbol '†', are not directly relevant to this chapter. These might have to be moved elsewhere in (or deleted from) the next version of the reference model.

access control: the prevention of use of a resource by unidentified and/or unauthorised entities in any other than an authorised manner [ECMA TR/46] ; the determination as to whether a requested access to an information item is to be granted or denied; see also, *authorisation*.

accidental: unintentional.

accountability: availability and integrity of some meta-information related to an operation (e.g., identity of the user realising the operation, time of the operation, etc.).

admissible evidence †: see *evidence (admissible ~)*.

anonymisation: process that gives confidence in *anonymity*.

anonymity: *confidentiality* of the identity of a person, e.g., who has realised an operation, or has not realised an operation.

attack: an *intrusion* attempt.

auditability: availability and integrity of some meta-information related to all operations.

authentic: of undisputed origin, genuine [OMED 1992].

authentication: process which gives confidence in *authenticity*.

authenticity: integrity of some information and meta-information; integrity of the meta-information representing the link between some information and its origin (e.g., the meta-information relating the claimed identity of a subject to the real identity of the subject).

authorisation: the granting of access to a security object [ECMA TR/46]; the determination as to whether a requested operation is to be granted or denied, according to the security policy; see also, *access control.*

availability : *dependability* with respect to the readiness for usage [Laprie 1992]; measure of *correct service* delivery with respect to the alternation between *correct service* and *incorrect service* [Laprie 1992].

component (system ~): another system, which is part of the considered system [Laprie 1992].

confidentiality: dependability with respect to the non-occurrence of unauthorised information disclosure.

correct service: see *service (correct ~).*

coverage: measure of the representativeness of the situations to which a *system* is submitted during its validation compared to the actual situations it will be confronted with during its operational life [Laprie 1992].

dependability: property of a computer system such that reliance can be justifiably placed on the service it delivers [Laprie 1995].

dependence: the state of being dependent on other support [OMED 1992]; reliance, trust, confidence [OMED 1992].

dependent: depending, conditional or subordinate [OMED 1992].

effort †: an expense (of bodily or mental energy) to achieve a desired end [LMED 1976].

error: part of the state of a *system* liable to lead to *failure* [Laprie 1992]. manifestation of a *fault* in a *system* [Laprie 1992].

event: a thing that happens or takes place [OMED 1992]; a change in *state.*

evidence (admissible ~) †: (law) evidence that can be taken into account in court.

evidence (irrefutable ~) †: (law) evidence that cannot be refuted, i.e., proven wrong.

failure: event occurring when the delivered *service* deviates from fulfilling the *system function* [Laprie *et al.* 1998]; transition from *correct service* to *incorrect service* [Laprie 1992].

failure model: a *fault model* defined in terms of the *failures* of the *components* of a *system.*

false negative: the *event* corresponding to the occurrence of an *intrusion* that is not detected as such by an *intrusion detection system* (i.e., no alarm raised

due to a lack of *coverage*, including excessive latency); also called a "miss".

false positive: the *event* corresponding to an alarm generated by an *intrusion detection system* in the absence of intrusion (i.e., a false alarm).

fault: adjudged or hypothesised cause of an *error* [Laprie 1992]; *error* cause which is intended to be avoided or tolerated [Laprie 1992]; consequence for a *system* of the *failure* of another *system* which has interacted or is interacting with the considered *system* [Laprie 1992].

fault forecasting: see *forecasting (fault ~).*

fault model: set of assumptions about the *faults* that are taken into account during *fault prevention*, *tolerance*, *removal* or *forecasting*.

fault prevention: see *prevention (fault ~).*

fault removal: see *removal (fault ~).*

fault tolerance: see *tolerance (fault ~).*

forecasting (fault ~): methods and techniques aimed at estimating the present number, the future incidence, and the consequences of *faults* [Laprie *et al.* 1998].

identity: representation of a person in a *system*.

incorrect service: see *service (incorrect ~).*

insider: a human *user* authorised to perform some of a set of specified operations on a set of specified objects, i.e., a user whose (current) *privilege* intersects the considered domain of object-operation pairs.

insider intrusion: see *intrusion (insider ~).*

insurance †: a measure taken to provide for a possible contingency [OMED 1992].

integrity: *dependability* with respect to the non-occurrence of inadequate information alterations [Laprie *et al.* 1998].

intentional: voluntary, deliberate.

intrusion: a *malicious* interaction *fault* resulting from an *attack* that has been (at least partially) successful in exploiting a *vulnerability.*

intrusion (insider ~): an abuse of *privilege* or *misfeasance*, i.e., an improper use of authorised operations.

intrusion (outsider ~): an unauthorised increase in privilege, i.e., a change in the *privilege* of a user that is not permitted by the system's security policy.

intrusion detection system: a facility aimed at discovering the presence of intrusions; the facility consists of a set of sensors or error detectors (including anomaly and misuse detectors) and an intrusion diagnosis mechanism that collates sensor outputs to decide whether the detected errors are symptomatic of intrusion.

irrefutable evidence †: see *evidence (irrefutable ~).*

liability †: the state of being liable [OMED 1992]; legal or financial responsibility.

liable †: legally bound [OMED 1992].

malicious: intending or intended to do harm [OMED 1992].

misfeasance: the illegal or improper performance of an action in itself lawful [LMED 1976]; an *intrusion* through the abuse of *privilege*.

object: information container.

outsider: a human *user* not authorised to perform any of a set of specified operations on a set of specified objects, i.e., a user whose (current) *privilege* does not intersect the considered domain of object-operation pairs.

outsider intrusion: see *(intrusion, outsider ~).*

prevention (fault ~): methods and techniques aimed at preventing *fault* occurrence or introduction [Laprie *et al.* 1998].

privacy: *confidentiality* of personal information.

privilege: set of *rights* of a *subject.*

removal (fault ~): methods and techniques aimed at reducing the presence (number, seriousness) of *faults*[Laprie *et al.* 1998].

responsibility: the state of being responsible [OMED 1992].

responsible: obliged to account; being the cause of; accountable for.

rights: a subject has a given right on a specified *object* if and only if he is authorised to perform a specified operation on that object; elements of a subject's *privilege.*

security: *dependability* with respect to the prevention of unauthorised access and/or handling of information [Laprie 1992]; the combination of *confidentiality, integrity* and *availability.*

security policy : description of 1) the security properties which are to be fulfilled by a computing system; 2) the rules according to which the system security state can evolve.

service (correct ~): *service* that fulfils the *system function* [Laprie *et al.* 1998].

service (incorrect ~): *service* that does not fulfil the *system function* [Laprie *et al.* 1998].

service: *system* behaviour as perceived by a *system user* [Laprie 1992].

state (**system ~)**: a condition of being with respect to a set of circumstances [Laprie 1992].

subject: active entity in a computer *system* — a process is a subject, a human *user* is also a subject.

system: entity having interacted, interacting or able to interact with other entities [Laprie 1992]; set of components bound together in order to interact [Laprie 1992].

system function: that for which the *system* is intended [Laprie *et al.* 1998].

system user: see *user (system ~)*.

tolerance (fault ~): methods and techniques aimed at providing a *correct service* in spite of *faults* (adapted from [Laprie *et al.* 1998]).

trust: reliance on the truth of a statement etc. without examination [OMED 1992].

trusted: adjective to describe a statement etc. on which *trust* has been placed.

user (system ~): another *system* (physical, human) interacting with the considered *system*.

value †: the worth, desirability, or utility of a thing [OMED 1992][21].

vulnerability: an accidental fault, or a malicious or non-malicious intentional fault, in the requirements, the specification, the design or the configuration of the system, or in the way it is used, that could be exploited to create an intrusion.

[21] If this notion is really needed, it would be better to use the term *worth* instead of value, in order to avoid confusion with *value* as used in *value domain, value failure,* etc.

Chapter 4 Conceptual Models

4.1 Introduction

R. J. Stroud, *University of Newcastle upon Tyne (UK)*

This chapter consists of a series of conceptual models describing different aspects of a secure distributed information system. These models are largely orthogonal but no attempt has yet been made to unify them. Together these conceptual models constitute a first version of the MAFTIA reference model, which will be used to inform the subsequent work of the project, including the development of the MAFTIA architecture in work package 2.

The first conceptual model is concerned with the failure model. It deals with the different failure assumptions, both from the system perspective and the user perspective. The second conceptual model is concerned with the issue of synchrony and proposes that the architecture should be based on the notion of a Trusted Timely Computing Base, a mechanism for guaranteeing system timeliness properties despite the presence of malicious faults. The third conceptual model is the topological model. It deals with issues of locality and scaling in a potentially global network environment. In particular, it distinguishes the different characteristics of local area networks, enterprise networks, and the global Internet. The fourth conceptual model builds on these foundations and characterises the kinds of services that will be provided by the MAFTIA middleware. These include basic cryptography and group communication protocols as well as more application oriented services such as trusted third party, timestamping and distributed authentication.

The remaining conceptual models are more abstract and are concerned with notions of trust, responsibility, accountability, and communication. The trust based verification model explores the nature of proof for a cryptographic system and identifies where trust is required in the verification process. The responsibility model discusses the nature of responsibility and delegation in an organisation, thus providing some insight into the problem of a corrupt insider exceeding his or her authority. Finally, the inter-agent communication model explores the nature of communication and identifies the different levels of meaning at which a failure can occur.

4.2 Failure Model

P. Verissimo, *Universidade de Lisboa (P)*

A crucial aspect of any architecture is the failure model upon which the system architecture is conceived, and component interactions are defined. The failure model conditions the correctness analysis, both in the value and time domains, and dictates crucial aspects of system configuration, such as the placement and choice of components, level of redundancy, types of algorithms, and so forth. There are

essentially two different kinds of failure model: *controlled failure* assumptions; and *arbitrary failure* assumptions.

4.2.1 Failure Assumptions

Controlled failure assumptions specify qualitative and quantitative bounds on component failures. For example, the failure assumptions may specify that components only have timing failures, and that no more than *f* components fail during an interval of reference. Alternatively, they can admit value failures, but not allow components to spontaneously generate or forge messages, nor impersonate, collude with, or send conflicting information to other components. This approach is extremely realistic, since it represents very well how common systems work under the presence of accidental faults, failing in a benign manner most of the time. It can be extrapolated to malicious faults, by assuming that they are qualitatively and quantitatively limited. However, it is traditionally difficult to model the behaviour of a hacker, so we have a problem of coverage that does not recommend this approach unless a solution can be found.

Arbitrary failure assumptions specify no qualitative or quantitative bounds on component failures. Obviously, this should be understood in the context of a universe of "possible" failures of the concerned operation mode of the component. For example, the possible failure modes of interaction, between components of a distributed system are limited to combinations of timeliness, form, meaning, and target of those interactions (let us call them messages). In this context, an arbitrary failure means the capability of generating a message at any time, with whatever syntax and semantics (form and meaning), and sending it to anywhere in the system. Moreover, practical systems based on arbitrary failure assumptions very often specify quantitative bounds on component failures, or at least equate tradeoffs between resilience of their solutions and the number of failures eventually produced [Babaõglu 1987]. Arbitrary failure assumptions are costly to handle, in terms of performance and complexity, and thus are not compatible with the user requirements of the vast majority of today's on-line applications.

Hybrid assumptions combining both kinds of failure assumptions would be desirable. They are a known framework in dependable system design vis-à-vis accidental failures [Meyer & Pradhan 1987] [Powell *et al.* 1988] [Verissimo *et al.* 1997]. Generally, they consist of allocating different assumptions to different subsets or components of the system, and have been used in a number of systems and protocols. Hybrid models allow stronger assumptions to be made about parts of the system that can justifiably be assumed to exhibit fail-controlled behaviour, whilst other parts of the system are still allowed an arbitrary behaviour. This is advantageous in modular and distributed system architectures such as MAFTIA. However, it is only feasible when the model is well-founded, that is, the behaviour of every single subset of the system can be modelled and/or enforced with high coverage, and this brings us back, at least for parts of the system, to the problem identified for controlled failure assumptions.

4.2.2 Composite Failure Model

The problems identified in our discussion of failure assumptions point to the need for the MAFTIA failure model to have characteristics enabling the definition of intermediate, hybrid assumptions, with adequate coverage. A first step in this direction is the definition of a composite failure model specifically aimed at representing the failures that may result from several classes of malicious faults. A second step is the definition of a set of techniques that act at different points within this composite failure model and which, combined in several ways, yield dependability vis-à-vis particular classes of faults. We are going to base our reasoning on two guiding principles:

- the sequence: *attack + vulnerability → intrusion → failure*

- the recursive use of fault tolerance and fault prevention

Concerning the mechanisms of failure, Figure 10 represents the fundamental sequence: *attack + vulnerability → intrusion → failure*. It distinguishes between several kinds of faults capable of contributing to a security failure. Recalling the definitions made in Chapter 3, vulnerabilities are the primordial faults existing inside the components, essentially design or configuration faults (e.g., coding faults allowing program stack overflow, files with root setuid in UNIX, naïve passwords, unprotected TCP/IP ports). Attacks are malicious interaction faults that attempt to activate one or more of those vulnerabilities (e.g., port scans, email viruses, malicious Java applets or ActiveX controls). An attack that successfully activates a vulnerability causes an intrusion. This further step towards failure is normally characterized by an erroneous state in the system that may take several forms (e.g., an unauthorized privileged account with telnet access, a system file with undue access permissions to the hacker). Such erroneous states can be unveiled by intrusion detection, as we will see ahead, but if nothing is done to process the errors resulting from the intrusion, failure of one or more security properties will occur.

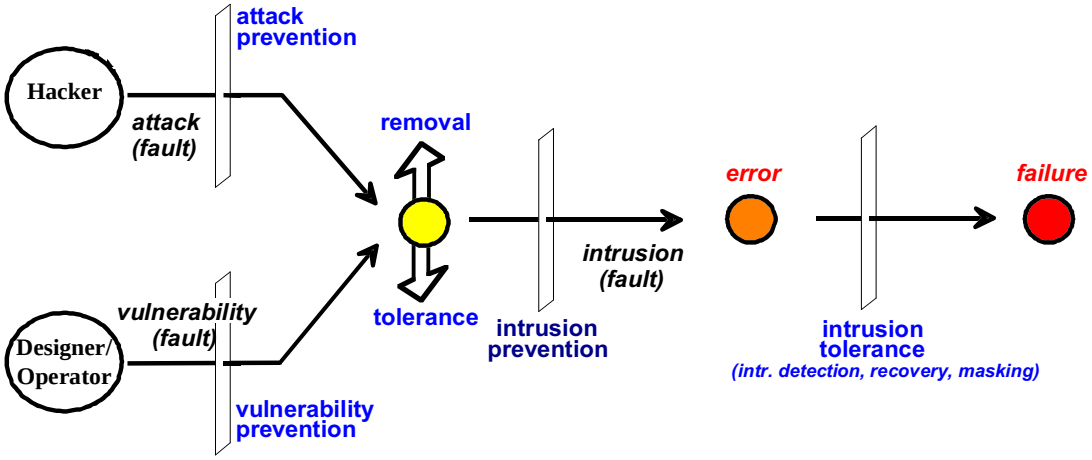

Figure 10 — The Composite Failure Model of MAFTIA

The composite model embraced in MAFTIA allows the combined introduction of several techniques. Note that two causes concur to create an intrusion, as shown in Figure 10: vulnerabilities and attacks.

To begin with, we can prevent some attacks from occurring, thereby reducing the level of threat imposed on the system. *Attack prevention* can be performed, for example, by selectively filtering access to parts of the system (e.g., if a component is behind a firewall and cannot be accessed from the Internet, it cannot be attacked from there). However, it is impossible to prevent all attacks (e.g., some components have to be placed outside the firewall in a Demilitarised Zone[22]), and in consequence, other measures must be taken.

On the vulnerability side, *vulnerability prevention* helps to reduce the degree of vulnerability by construction. However, many systems are assembled from COTS components that contain known vulnerabilities. When it is not possible to prevent the attack(s) that would activate these vulnerabilities, a first step would be to attempt *vulnerability removal*. Sometimes this is done at the cost of eliminating the system functions that contain the vulnerabilities. The above-mentioned approaches can be complemented, still at the attack level, with *tolerance* measures achieved by combinations of the classic techniques: detection, recovery, and masking. The detection of port scans or other anomalous activity at the external border of the system forms part of the functionality of some systems generically known as Intrusion Detection Systems (IDS). Although an intrusion has not yet occurred, there is an erroneous symptom that can be addressed by several attack countermeasures. For example, honey-pots and other evasive measures at the periphery of the system can be used to mask the effects of the attack.

The various combinations of techniques discussed above provide a range of alternatives for achieving *intrusion prevention* (see Figure 10), i.e. attempting to avoid the occurrence of intrusions. Whilst this is a valid and widely used approach, its absolute success cannot be guaranteed in all situations, and for all systems. The reason is obvious: it may not be possible to handle all attacks, either because not all attacks are known or new ones may appear, or because not all attacks can be guaranteed to be detected or masked. Similar reasoning applies to vulnerabilities. In consequence, some attacks will succeed in producing intrusions, requiring forms of *intrusion tolerance*, as shown in the right part of Figure 10, in order to prevent system failure. Again, these can assume several forms: detection (e.g., of intruded account activity, of Trojan horse activity); recovery (e.g., interception and neutralization of intruder activity); or masking (e.g., voting between several components, including a minority of intruded ones).

---

[22] A Demilitarized Zone is an area typically outside the inner protection perimeter of the firewall, and sometimes between an outer and an inner firewall, where public servers are placed.

The above discussion has laid the foundations for achieving our objective: a well-founded hybrid failure model, that is, one where different components have different faulty behaviours. Consider a component for which a given controlled failure assumption was made. How can we achieve coverage of such an assumption, given the unpredictability of attacks and the elusiveness of vulnerabilities? The key is in a recursive use of fault tolerance and fault prevention. Think of the component as a system: it can be constructed through the combined use of removal of internal vulnerabilities, prevention of some attacks, and implementation of intrusion tolerance mechanisms internal to the component, in order to prevent the component from exhibiting failures.

Looked upon from the outside now, at the next higher level of abstraction, the level of the outer system, the would-be component failures we prevented restrict the system faults the component can produce. In fact we have performed *fault prevention*, that is, we have a component with a controlled behaviour vis-à-vis malicious faults. This principle:

- establishes a divide-and-conquer strategy for building modular fault-tolerant systems;
- can be applied in different ways to any component;
- can be applied recursively at as many levels of abstraction as are found to be useful.

Components exhibit a coverage that is justifiably given by the techniques used in their implementation, and can subsequently be used in the construction of fault-tolerant protocols under the hybrid failure model.

Since we are dealing with systems of interacting components we should also address *error confinement*. Errors can propagate from one component to another when messages are sent from one component to another component. One approach to confining errors is to replicate components, and apply a message selection protocol that validates the messages generated by the replicated components before they are propagated. The selection protocol cross-checks each message against equivalent data messages generated by the other replicas and only sends messages that the majority of replicas agree upon. Thus, messages are only propagated if the replicas reach a consensus about them. The drawback of this *validate-before-propagate* [Powell 1991] approach is that every replica must agree to send a message, so propagation is limited by the slowest replica in the group. A more efficient approach is *propagate-before-validate* [Powell 1991]. With this approach the first replica to generate a message propagates the message to other components without any validation taking place. However, at some later point the computation being carried out by the components is suspended until all previous computation is validated. One plausible implementation approach would be to use a transactional framework. All interacting components would join a transaction and before commitment could take place, all message sending would be suspended and all messages sent during the transaction would be validated by cross-checking the messages generated by component replicas. If any of

the messages were invalid then the transaction would be aborted and rollback would take place, otherwise the transaction would be committed and the effects of the message interchange made permanent.

This approach was originally developed for components that were active replicas. We would like to apply the concept to general components. However, standard transactional frameworks are unsuitable for this as they support only backward error recovery and a general component may not support rollback. Therefore the framework used for error confinement should support forward recovery techniques as well as backward recovery techniques. The coordinated atomic actions (CA actions) concept [Xu *et al.* 1995] [Xu *et al.* 1999] may offer a suitable framework. CA actions are intended for cooperating or competing general components involved in a joint interaction and impose strong controls on error confinement and error recovery activities in the event of failure. The implementation of the CA action mechanism would need to be made both fault tolerant and secure to be used as a framework for error confinement in a intrusion tolerant system. Therefore, as part of the MAFTIA project we intend to investigate the usefulness of an extended concept of CA actions for error confinement

4.3 Synchrony Model

P. Verissimo*, Universidade de Lisboa (P)*

Research in distributed systems algorithms has traditionally been based on one of two canonical models: fully asynchronous and fully synchronous models [Verissimo & Raynal 2000]. Asynchronous models do not allow timeliness specifications. They are time-free, that is, they are characterized by an absolute independence of time, and distributed systems based on such models typically have the following characteristics:

Pa 1 Unbounded or unknown processing delays

Pa 2 Unbounded or unknown message delivery delays

Pa 3 Unbounded or unknown rate of drift of local clocks

Pa 4 Unbounded or unknown difference of local clocks[23]

Asynchronous models obviously resist timing attacks, i.e. attacks on the timing assumptions of the model, which are non-existent in this case. However, because of their time-free nature, asynchronous models cannot solve timed problems. For example, they cannot address Quality of Service (QoS) specifications, which are of increasing importance in the measure of the quality of transactional systems in open

[23] **Pa3** and **Pa4** are essentially equivalent but are listed for a better comparison with the synchronous model characteristics listed below. Since a local clock in a time-free system is nothing more than a sequence counter, clock synchronization is also impossible in an asynchronous system.

networks such as the Internet (e.g., stock exchange, e-commerce). They cannot reason in terms of time, either relative or absolute, which dictates a lot of the functionality of interactive applications. In addition, asynchronous models preclude the deterministic solution of interesting problems, such as consensus or Byzantine agreement [Fischer *et al.* 1985]: only probabilistic solutions work in this case. Besides, the only way asynchronous models can reason in terms of causality between events is in a logical way, and this is insufficient if hidden communication channels exist between participants. Causality or cause-effect order is a crucial factor of correctness in some interactive and competitive applications, such as on-line operations on the stock market. "False" asynchronous algorithms have been deployed over the years, exhibiting subtle but real failures, thanks to the inappropriate use of timeouts in a supposedly time-free model.

In practice, many of the emerging applications we see today, particularly on the Internet, have interactivity or mission-criticality requirements. That is, service must be provided on time, either because of user-dictated quality-of-service requirements (e.g., network transaction servers, multimedia rendering, synchronized groupware), or because of dependability constraints (e.g., air traffic control). Synchronous models allow timeliness specifications. In this type of model, it is possible to solve all hard problems (e.g., consensus, atomic broadcast, clock synchronization) [Chandra & Toueg 1996]. In consequence, such models solve timed problem specifications, one precondition for at least a subset of the applications targeted in MAFTIA, for the reasons explained above. Synchronous models have the following characteristics:

Ps 1 There exists a known bound for processing delays

Ps 2 There exists a known bound for message delivery delays

Ps 3 There exists a known bound for the rate of drift of local clocks

Ps 4 There exists a known bound for the difference among local clocks

However, synchronous models are fragile in terms of the coverage of timeliness assumptions such as: positioning of events in the timeline and determining execution durations. Indeed, to achieve these characteristics, some additional timing assumptions must be made concerning the behaviour of the system with regard to component failures. These timing bounds are assumed to be valid for correct processes, and conversely their achievement must not be disturbed by incorrect processes. It is easy to see that synchronous models are susceptible to timing attacks, since they make strong assumptions about things happening on time. For example, algorithms based on messages arriving by a certain time, or on reading the actual global time from a clock, may fail in dangerous ways if manipulated by an adversary [Gong 1992]. Likewise, causal delivery order of messages or event trace analysis based on physical timestamps may be disturbed to the advantage of a hacker who, for example, manipulates the time-stamping facility.

4.3.1 Partial Synchrony

The introductory words above explain why synchronism is more than a mere circumstantial attribute in distributed systems subjected to malicious faults: absence of time is detrimental to quality of service; presence of time increases vulnerability. Restrictions to the asynchrony of time-free systems have been addressed in earlier studies [Dolev *et al.* 1987, Dwork *et al.* 1988], but timed partially synchronous models have deservedly received great attention recently. These intermediate models provide better results, essentially for three reasons: (i) they allow timeliness specifications; (ii) they admit failure of those specifications; (iii) they provide timing failure detection.

We are particularly interested in a model based on the existence of a *timely computing base*, which is both a timely execution assistant and a timing failure detection oracle that ensures time-domain correctness of applications in environments of uncertain synchronism [Verissimo *et al.* 2000]. The timely computing base model addresses a broader spectrum of problems than those solved by previous timed partially synchronous models, such as the quasi-synchronous [Veríssimo & Almeida 1995] and the timed-asynchronous models [Cristian & Fetzer 1998]. All these works share the same observation: *synchronism or asynchronism are not homogeneous properties of systems*. That is, they vary with time, and they vary with the part of the system being considered. However, each model has treated these asymmetries in its own way: some relied on the evolution of synchronism with time, others with space or with both; synchronism assumptions were not totally transparent to applications; and architectural constructs were rarely used to enforce these assumptions. In the timely computing base model, it is assumed that systems have an architecture such that applications, however asynchronous they may be and whatever their scale, can rely on services provided by a special module which is timely and reliable.

Research in this area has focused on benign (non-arbitrary, non-malicious) failure models. However, the architectural characteristics of the timely computing base enables the model to be extended so as to be resilient to value- as well as time-domain failures.

This would contribute to supporting what we described earlier as a hybrid failure model, encompassing malicious faults whilst guaranteeing system timeliness in a much more robust way than fully synchronous models would. In essence, the timely computing base must follow a few construction principles that guarantee its behaviour in the face of faults:

- Interposition: it must by construction be interposed between vital resources and any attempt to interact with them.
- Shielding: it must be shielded (tamperproof) from any contamination by the rest of the system.
- Validation: it must be verifiable, in order to ensure very high coverage of its properties.

We call such an extended model, whose development we will pursue in the MAFTIA project, a *Trusted Timely Computing Base*, or TTCB. In one sense, a TTCB has similar design principles to the very well known paradigm in security of a Trusted Computing Base (TCB) [Abrams *et al.* 1995]. However, the objectives are drastically different. A TCB aims at fault prevention and ensures that the whole application state and resources are tamper-proof. It is based on logical correctness and makes no attempt to reason in terms of time. In contrast, a TTCB aims at fault tolerance: application components can be tampered with, but the whole application should not fail. In other words, a TTCB is an architectural artefact supporting the construction and trusted execution of fault-tolerant protocols and applications running under a partially synchronous model.

The architecture of a system with a TTCB is suggested by Figure 11. The first relevant aspect is that the heterogeneity of system properties is incorporated into the system architecture. There is a generic or *payload* system, over a global network or *payload* channel. This prefigures what is normally "the system" in homogeneous architectures, that is, the place where the protocols and applications run. The latter can have any degree of synchronism, and be subjected to arbitrary attacks. Additionally, there is a *control* part, made of local TTCB modules, interconnected by some form of medium, the *control* channel. We will refer to this set up as the distributed TTCB, or simply TTCB when there is no ambiguity. The second relevant aspect of the TTCB is that its well-defined properties are preserved by construction, regardless of the properties of applications running with its assistance: it is synchronous, and it is trusted to execute as specified, being resilient to intrusions.

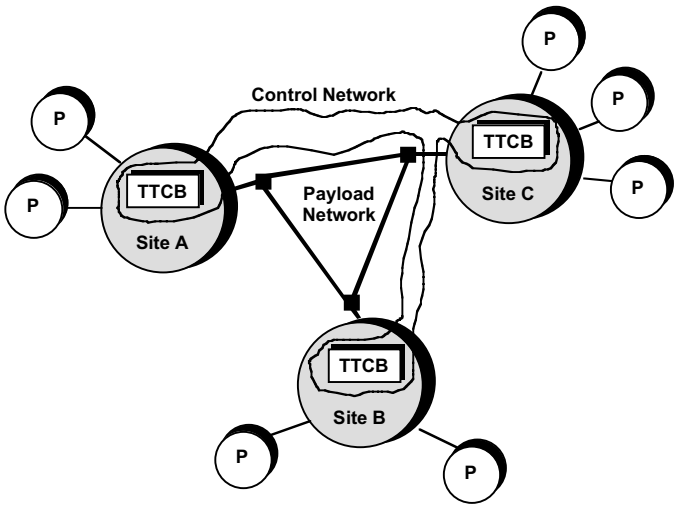

Figure 11 — Trusted Timely Computing Base Model

Unlike the classic TCB, the TTCB can be a fairly modest and simple component of the system, used as an assistant for parts of the execution of the payload protocols and applications. Moreover, depending on the type of application, it is not necessary that all sites have a local TTCB. Consider the development of a fault-tolerant TTP (Trusted Third Party) based on a group of replicas that collectively ensure the correct behaviour of the TTP service vis-à-vis malicious faults. One possibility is for the

replica group activity to be based on algorithms that support an arbitrary failure assumptions model (e.g., asynchronous randomized Byzantine Agreement), with the corresponding penalty in performance and lack of timeliness. Alternatively, the replica group management may rely on simpler algorithms that are at least partially executed in a synchronous subsystem with a benign (intrusion-free) failure model. Running these parts of the algorithm on a distributed TTCB substantiates the coverage of these assumptions.

A TTCB should be built in a way that secures both the synchronism properties mentioned earlier, and its correct behaviour vis-à-vis malicious faults. In consequence, a local TTCB can be either a special hardware module, such as a tamperproof device, an auxiliary firmware-based microcomputer board, or a software-based real-time and security kernel running on a plain desktop machine such as a PC or workstation. Likewise, a distributed TTCB assumes the existence of a timely inter-module communication channel. This channel can assume several forms exhibiting different levels of timeliness and resilience. It may or not be based on a physically different network from the one supporting the *payload* channel. Virtual channels with predictable timing characteristics coexisting with essentially asynchronous channels are feasible in some current networks, even over the Internet [Schulzrinne *et al.* 1996]. Such virtual channels can be made secure through virtual private network (VPN) techniques, which consist of building secure cryptographic IP tunnels linking all TTCB modules together, and these techniques are now supported by standards [Kent & Atkinson 1998]. On a timeliness side, it should be observed that the bandwidth required of the control channel is bound to be much smaller than that of the payload channel. In more demanding scenarios, one may resort to alternative networks (real-time LAN, ISDN connection, GSM or UMTS Short Message Service, Low Earth Orbit satellite communication).

A TTCB also allows partial synchrony to be explored from a time-free perspective. A time-free approach is necessary when the criticality of operations is such that an arbitrary failure assumptions model is needed to maximize coverage and prevent timing attacks by resorting to an asynchronous model. However, this setting does not offer timeliness guarantees and that would be the price to pay. The optimistic synchrony model that we intend to pursue in the MAFTIA project attempts to improve on this situation. A fully asynchronous model is assumed as a baseline framework for constructing probabilistic malicious-fault resilient building blocks, for example, randomized Byzantine agreement. However, whenever the system exhibits enough synchrony, the system switches to the partially synchronous versions of those building blocks, still malicious-fault resilient, but exhibiting better performance.

4.4 Topological Model

P. Verissimo, *Universidade de Lisboa (P)*

Previous work on large-scale open distributed systems has shown the value of *topology awareness* in the construction of efficient protocols [Rodrigues & Verissimo 2000], from both functionality and performance viewpoints. The principle is

explained very simply: (i) the topology of the system is set up in ways that may enhance its properties; (ii) protocols and mechanisms in general are designed in order to recognize system topology and take advantage from it. This is achieved both by creating adequate physical topologies (the part of the system under the control of the organization, e.g., the local networks) and by logically reorganizing existing physical topologies (the part of the system outside the control of the organization, e.g. the Internet).

We intend to extrapolate the virtues of topology awareness to security in the MAFTIA architecture, through a few principles for introducing topological constructs that facilitate the combined implementation of malicious fault tolerance and fault prevention. The first principle is to use topology to facilitate separation of concerns: the site-participant separation in the internal structure of system hosts separates communication from processing; and the WAN-of-LANs duality at network level separates communication amongst local aggregates of sites, which we call facilities, from long-haul communication amongst facilities. The second principle is to use topology to construct clustering in a natural way. Two points of clustering seem natural in the MAFTIA large-scale architecture: sites and facilities. We clarify these principles in the next sections.

4.4.1 Sites and Participants

The MAFTIA architecture supports interactions among entities in different hosts (e.g. processes, tasks, etc.). We call them generically *participants*. Participants, which execute distributed activities, can be senders or recipients of information, or both, in the course of the aforementioned activities. The local topology of hosts is such that they are divided into a *site* part, which connects to the network and takes care of all inter-host operations, i.e., communication, and a *participant* part, which takes care of all distributed activities and relies on the services provided by the site-part modules. Participants interact via the respective site part, which handles all communication aspects on behalf of the former, as represented in Figure 12. From now on, we will refer to sites, when taking the communication/networking viewpoint on the system, and we will refer to participants, when taking the activity/processing viewpoint.

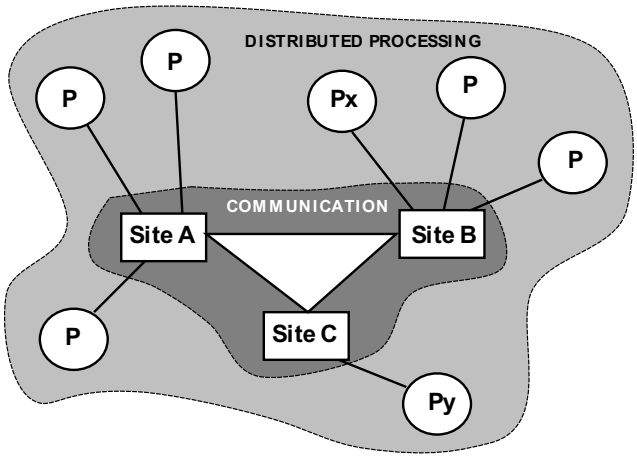

Figure 12 — Site-participant Duality

A system built according to the site-participant duality model provides a framework for defining realms of different synchrony, reliability and security. For example, intra-site communication can be assumed to have better properties of synchrony and reliability with regard to both accidental and malicious faults, in contrast with inter-site communication. In consequence, while site failure detection is unreliable in a network of uncertain synchronism, participant failures can be reliably detected. Likewise, different assumptions can be made concerning trustworthiness of the participant and site parts.

4.4.2 The Two-tier WAN-of-LANs

The global topology of the networking infrastructure is seen as a logical two-tier WAN-of-LANs. Large-scale computing infrastructures exhibit what appears to be a two-tier organization: pools of sites with privately managed high connectivity links, such as LANs or MANs or ATM fabrics, interconnected in the upper tier by a publicly managed point-to-point global network (e.g., the Internet). More concretely, we mean that the global network runs standard, de jure or de facto, protocols whilst each local network is run by a single, private, entity[24], and can thus run specific protocols alongside standard protocols.

Again, this structure offers opportunities for making different assumptions regarding the types and levels of threat and degrees of vulnerability of the local network versus the global network part. Incidentally, this does not necessarily mean considering intra-facility networking threat-free. For example, certain port scans or pings in the global network may mean absolutely nothing, whereas they may mean an attack if performed inside the facility. On the other hand, an intruder working from the inside of the facility may have considerably more power than one working from the outside.

4.4.3 Clustering

Clustering seems one of the most promising techniques to cope with large-scale distributed systems, providing the means to implement effective divide-and-conquer strategies. Whilst this enhances scalability and performance, it also provides hooks for the combined implementation of fault-tolerant mechanisms (e.g., cryptographic group management protocols) and fault-preventive protection strategies (e.g., firewalls). We identify at least two clustering opportunities: (i) the *Facility* as a cluster of nodes, or more appropriately sites, if seen from the viewpoint of the network; (ii) the *Site* as a cluster of participants, the ones that reside in the relevant node.

[24] E.g.: bridged LAN compound of a university campus, MAN of a large industrial complex, LAN of a regional company department.

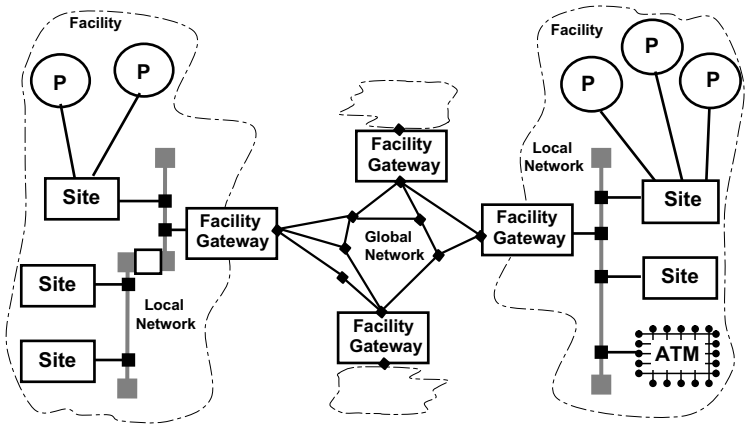

Figure 13 — Two-tier WAN-of-LANs and Clustering

The first clustering level is obviously compatible with the two-tier architecture identified in the previous section, and is illustrated in Figure 13. Clustering sites that coexist in the same local network can simplify inter-network addressing, communication and administration of these nodes. These sites are hidden behind a single entry-point, a *Facility Gateway*, a logical gateway that represents the local network members, for the global network. The second level of clustering consists of taking advantage of a multiplying factor between the number of sites and the (sometimes large) number of participants that are active in communication.

Organization-dependent clustering allows specific protocols to be run behind the Facility Gateways, without conflicting with the need to use standard protocols in wide-area networking. Global network communication is then performed essentially among Facility Gateways. From a security viewpoint, participant-site clustering allows investing in the implementation of fault-tolerant and fault-preventive mechanisms at node level to collectively serve the applications residing in the node. On the other hand, the opportunities offered by site-facility clustering with regard to security are manifold: firewalling at the Facility Gateway; establishing inter-facility secure tunnels ending in the facility agents; inspecting incoming and outgoing traffic for attack and intrusion detection; ingress and egress traffic filtering; internal topology hiding through network address translation, etc.

4.4.4 Recursivity

The view we have just presented can be recursively applied, in order to represent very-large-scale organizations. On an intra-facility level, further hierarchies, namely those already deriving from hierarchical organization of subnetworks and domains, are not precluded, if the Facility Gateway role is respected. Protocols can easily take advantage of further topology refinements. On an intra-organization level, the topology depicted in Figure 13 can be instantiated as representing an organization with multiple geographically dispersed facilities interconnected by secure tunnels whose end points are the Facility Gateways. The only role of the Facility Gateways is to implement the Virtual Private Network (VPN) that ensures this internal communication, so they would be better named as Internal Facility Gateways. However, the organization still needs to be connected to the Internet. On an inter-

organization level, Figure 13 can be re-instantiated with each top-level facility at the organisational level now representing multiple inter-organizational facilities interconnected through Internet Facility Gateways. Communication with other organizations through the Internet is ensured through normal gateways, i.e., External Facility Gateways. On an extra-organizational level, the Facility Gateway, which is a logical entity and does not necessarily correspond to a single machine, offers the necessary services, such as incoming email, web presence, e-commerce, and so forth, implemented by extranet-based servers, serving potentially many thousands of clients.

4.4.5 System Components

The architecture of a MAFTIA node is represented in Figure 14, in which the local topology and the dependence relations between modules are depicted by the orientation of the ("depends-on") arrows. In Figure 14 the set of layers is divided into site and participant parts. The site part has access to and depends on a physical networking infrastructure, not represented for simplicity. The participant part offers support to local participants engaging in distributed computations. The lowest layer is the *Multipoint Network* module, *MN*, created over the physical infrastructure. Its main properties are the provision of multipoint addressing and a moderate best-effort error recovery ability, both depending on topology and site liveness information. The MN layer hides the particulars of the underlying network to which a given site is directly attached, and is as thin as the intrinsic properties of the latter allow.

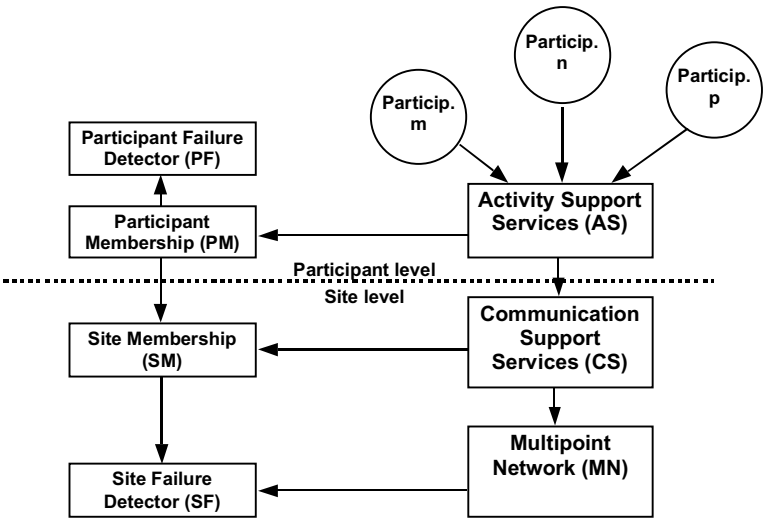

Figure 14 — Architecture of a MAFTIA Node

In the site part, the *Site Failure Detector* module, *SF*, is in charge of assessing the connectivity and correctness of sites, and the Multipoint Network module depends on this information. The SF module depends on the network to perform its job, and thus is not completely reliable, due to the uncertain synchrony and susceptibility to attacks of at least parts of the network. The universe of sites being monitored can be parameterized, for example: all sites inside a facility, all sites having to do with ongoing computations at this site, all facility agents, etc. The *Site Membership*

module, *SM*, depends on information given by the SF module. It creates and modifies the membership (registered members) and the view (currently active, or non-failed, or trusted members) of sets of sites, which we call site-groups. The *Communication Support Services* module, *CS*, implements basic cryptographic primitives (e.g., secure channels and envelopes), Byzantine agreement, group communication with several reliability and ordering guarantees, clock synchronization, and other core services. The CS module depends on information given by the SM module about the composition of the groups, and on the MN module to access the network.

In the participant part, the *Participant Failure Detector* module, *PF*, assesses the liveness of all local participants, based on local information provided by sensors in the operating system support. The *Participant Membership* module, *PM*, performs similar operations as the SM, on the membership and view of participant groups. Note that several participant groups, or simply *groups*, may exist in a single site. The site-participant clustering and separation of concerns stated earlier is thus implemented by making a separation between groups of participants (performing distributed activities), and site-groups of the sites where those participants reside (performing reliable communication on behalf of the latter). Clustering can be further enhanced by mapping more than one group onto the same site-group, in what are called *lightweight groups* [Rodrigues *et al.* 1996]. The PM module monitors all groups with local members, depending on information propagated by the SM and by the PF modules, and operating cooperatively with the corresponding modules in the concerned remote sites. The *Activity Support Services* module, *AS*, implements building blocks that assist participant activity, such as replication management (e.g., state machine, voting), leader election, transactional management, key management, and so forth. It depends on the services provided by the CS module, and on the membership information provided by the PM module.

Consequentially, the protocols implementing the layers described above all share the topology awareness property. As such, they may run differently depending on their position in the topology, although this happens transparently. For example, the SF protocol instantiated at the Facility Gateways may wish to aggregate all liveness/failure information from the site it oversees, and gather that same information from the corresponding remote Facility Gateways. These considerations may obviously be extended to topology-aware attack and intrusion detection.

4.4.6 Interaction Styles

In MAFTIA we intend to support different styles of basic interactions among the participants. These interaction styles also assume topological importance, because they can be combined to construct complex software architectures, but the possible combinations are conditioned by the system topology.

Although the main goal of MAFTIA is to provide security in the face of malicious faults, the MAFTIA architecture must also provide a versatile functional support in order to be useful. Consequentially, MAFTIA will support the main interaction styles used in distributed computing, namely:

- *client-server*, for service invocations

- *multipeer*, for interactions amongst peers

- *dissemination*, of information in push or pull form

- *transactions*, for encapsulation of multiple actions

Client-server interactions can be implemented by two different mechanisms: in *closed loop*, usually performed through RPC [Birrell & Nelson 1984], or in *open loop*, usually performed through group communication, pioneered by Cheriton [Cheriton & Zwaenepoel 1985], Birman [Birman & Joseph 1987] and Cristian [Cristian *et al.* 1985], and followed by many others. From previous works [Birman *et al.* 1991] [Ladin *et al.* 1990], it is known to be difficult to scale service access when clients need to be strongly coupled (for instance, when all messages need to be ordered). However, there are a number of services, especially in large-scale systems, where the coupling among clients, and between a client and the server, can be weakened. In addition, techniques supporting reliable large-scale remote server access have been deployed, allowing services to be easily replicated and invoked transparently, without necessarily implying a degradation of strong failure semantics [Rodrigues *et al.* 1994]. Both approaches are easily implemented using group-based open-loop mechanisms.

Another style of interaction is *multipeer*, conveying the notion of spontaneous, symmetric interchange of information, amongst a collection of peer entities. This paradigm appeared as early as in [Powell *et al.* 1988] where it is called multipoint association, and also in [Peterson *et al.* 1989] where it is called conversation, a term that we avoid in order not to cause confusion with a different paradigm with the same name described in [Campbell & Randell 1986], and also discussed below. Multipeer interactions are the kind of interaction one might wish among managers of a distributed database, a group of commerce servers, a group of TTP servers, or a group of participants running a cryptographic agreement (e.g., contract signing). Communication requirements may be heavy in ordering and reliability requirements, and a notion of composition or membership may be required (for example, to provide explicit control over who is currently in the group). Again, the highly interactive nature of the multipeer style of interactions prevents per se the number of participants in real applications from exceeding the small-scale threshold.

Next, we have *dissemination*, which combines the information push and pull approaches. Information is published by *publishers*, and is made available on a repository. Message subscription can be implemented using two different alternatives: the *push* strategy or the *pull* strategy. With the push strategy, subscribers just register their interest in receiving a certain class of messages with the server, and the server is then responsible for dissemination of these messages to the interested subscribers. With the pull strategy, it is up to the subscriber to contact the server periodically to fetch the messages on request. The number of recipients may be rather high, while the number of senders will not.

Finally, *transactions* provide the capability of performing multiple actions encapsulated with certain reliability guarantees. There are three candidates for a transactional interaction style — atomic transactions, conversations and coordinated atomic actions — each providing different guarantees. *Atomic transactions* are a well known structuring mechanism that are best suited to *competitive* interactions. Atomic transactions guarantee the properties of atomicity, consistency, isolation and durability (ACID). *Conversations* [Campbell & Randell 1986] are traditionally used for *cooperative* systems and employ coordinated exception handling for tolerating faults. *Coordinated atomic actions* (or CA actions) [Xu *et al.* 1995] [Xu *et al.* 1999] are a structuring mechanism that integrates and extends conversations and atomic transactions. The former are used to control cooperative interactions and to implement coordinated error recovery whilst the latter are used to maintain the consistency of shared resources in the presence of failures and competitive concurrency. Coordinated exception handling is supported by distributed exception resolution algorithms [Xu *et al.* 1998]. Given the possibly complex operations triggered in some of the applications envisaged for MAFTIA, transactions may be coordinated in a distributed way, and may be long-lived.

As said in the beginning, the several styles referred to above can be combined to form more complex interaction styles. For example, transactions may encapsulate several interactions built using the other styles. Note also that the extensive use of open-loop client server mechanisms, multipeer interactions, replication, and distributed atomic actions is yet another justification for the emphasis of the group-orientation paradigm in the architecture of MAFTIA.

4.5 Services Model

C. Cachin, J. Camenisch, K. Kursawe, J. Müller, F. Petzold, V. Shoup, M. Waidner, ***IBM Research, Zurich (CH)***

We consider several distributed application services with high availability and security requirements. These can be built on top of the services provided by the Activity Support layer of the MAFTIA architecture described in Section 4.4.5.

4.5.1 Threshold-cryptography Services

Threshold cryptography allows a group of processors to perform cryptographic operations securely even though a fraction (minority) of them may be unavailable or may leak information to the outside. From the point of the cryptographic service offered by the group, one must assume that they are completely under control of the adversary and are called corrupted here. Robust protocols, such as those considered here, allow the system to continue its operation despite arbitrary malicious behaviour of the corrupted parties.

Building secure intrusion-tolerant protocols for groups requires several cryptographic services:

- Threshold Pseudo-Random Functions: A pseudo-random function can be viewed as a random-access source of unpredictable coin flips (similar to a key distribution centre that derives session keys from a master key). The master random seed of all coins is maintained in a shared way among the parties such that the adversary cannot gain information about it before it is revealed. The value of a coin with a certain name is revealed only if a suitable majority processes a request to flip the coin.

- Threshold Signatures: A threshold signature is a distributed signature scheme, in which the signing key is maintained in a distributed way and never assembled. A signature may be generated from appropriate signature shares that are issued by a party when they process a request to sign a particular message.

4.5.2 Fair Exchange

Fair Exchange protocols are useful in electronic commerce for digital content selling, certified email or electronic contract signing. The fairness property ensures that either both parties that wish to exchange items get the item they are supposed to, or get neither.

One way to exchange digital items fairly is to slice the items in small parts and take turns in sending them to the other party, thus reducing the other party's entropy step by step. For a more efficient approach, which is better suited for exchanging larger items, we use protocols that need an additional party in the protocol, a trusted third party (TTP).

One easy approach is that both parties send their respective item to the TTP, which then forwards them to the other party. This approach makes it necessary that the TTP becomes involved in every transaction. In the *optimistic* approach proposed by Asokan et al [Asokan *et al.* 1997], [Asokan *et al.* 2000] the TTP is only involved when something goes wrong, e.g., one party tried to cheat or messages were lost.

In a normal protocol run, the party which goes first, called the *sender*, starts by preparing and sending a *promise* for the item he wishes to send. The other party, the *receiver*, then sends back a promise for his item. Upon receipt of the receiver's promise, the sender sends the item in the clear, or the keys for decrypting the item if it was sent as part of the promise. The receiver then does the same and the transaction is complete.

The TTP is used to force a resolve of the exchange in case one party quits the protocol prematurely, or to ensure a permanent abort. For these functions, the TTP needs to have a database of transactions it handled together with their respective state. The requests received have to be handled atomically per transaction.

For the TTP to be able to resolve a transaction, it must be able to substitute the role of the player that quit. One way to enable it to play that role is to include an escrow of the item to be exchanged in the promises. If one party wishes to resolve, the TTP can

then open the escrow and give the item to the party. The escrow can be realized by an encryption of the item under the TTP's public key. Aborting via the TTP is necessary because a sender might not want to wait forever for a promise of the receiver, and therefore make sure that the TTP will never resolve the transaction.

4.5.3 Certification Authority

A *Certification Authority* (CA) is a service run by a trusted organization that verifies and confirms the validity of a public key. The issued *certificate* usually also confirms that the real-world user defined in the certificate is in control of the corresponding private key. It links the public key to an ID by signing the two together under the CA's private signing key.

In the view of the service user, it works as follows. The CA has published its own public key, of which all users are supposed to posses an authentic copy (how this is done is not of interest here). When a user wants to obtain a certificate for his public key, he sends the key together with his ID and some credentials to the CA. The ID might consist of any information, such as name, address, email, and other data to identify the holder. The CA then verifies the credentials, produces a certificate and sends it back to the user. The user can then verify his certificate against his public key with the public key of the CA.

In order to render its certificates meaningful, the CA will only issue one if the credentials are valid and the user was authorized to get a certificate. The ID in the certificate must belong to the holder of the public/secret key pair. It is in the interest of the CA to have established a well-known policy for issuing certificates.

4.5.4 Time-stamping Service

Digital time-stamping services (TSS) are a means to assure the (relative) temporal order of the creation of digital documents. They are a prerequisite for fully digitalizing legal processes such as the issuing of patents. A time-stamping service involves users and a time-stamping authority (TSA) that has made available some system parameters such as a cryptographic hash-function. The TSA publishes regularly, say once a day, some string (*day stamp*) in such a way that these strings cannot be lost or altered. They could for instance be published in a daily newspaper or be stored in a distributed database. To get a document time-stamped, a user sends the TSA a cryptographic hash of the document. The TSA answers each request with a time-stamping certificate. The certificate is intended to guarantee that the document was published on a certain day.

The certificate service provided by a TSA ideally satisfies the properties of *validity*, *accountability*, *provability*, *security*, *unforgeability*, and *privacy*.

In more detail, validity ensures that if the TSA is honest, a user will receive a time-stamping certificate that is valid w.r.t. the next day stamp to be published. Accountability means that whenever the TSA issues invalid time-stamping

certificates, the affected user can convince a third party that the TSA misbehaved. Provability means that given a document, a time-stamping certificate, and a day stamp published by the TSA on a particular day, it should be possible to prove whether the certificate is valid w.r.t. the document and the day stamp.

A day stamp is secure if the TSA can no longer issue valid time-stamp certificates for a day after the day stamp has been published. Furthermore, given all day stamps, their order can be uniquely determined. For the time-stamping scheme to be secure, one must be able to determine uniquely the order in which any given pair of time-stamp certificates were issued. Furthermore, once a time-stamping certificate is issued, the TSA can no longer generate another one that appears to have been issued previously.

Unforgeability means that only the TSA is able to publish day stamps and time-stamping certificates. In addition, privacy means that the TSA does not get any information about the contents of a document that is time-stamped.

Digital signatures may be used for time-stamping, but the uncertainty about the lifetime of cryptographic keys seriously limits the usefulness of this method because once the TSA's signing key is exposed in the future, all time stamps created by the TSA lose their validity. The problem is only aggravated by the fact that time stamps become more valuable over time.

Concrete protocols for time-stamping that are not based on digital signatures and avoid this danger are described in [Haber & Stornetta 1991].

4.5.5 Authentication Service

The basic task of an authentication service is to verify the claimed identity of a user or a process acting on behalf of a user. The user must present secret information that identifies her or carry out a zero-knowledge identification protocol. If verification succeeds, the service will take some action to grant the request, like establishing a session or replying with a cryptographic token to be used. Often, the answer contains a freshly generated, random session key as in Kerberos [Neuman & Ts'o 1994], [Steiner *et al.* 1988]. Such an authentication server is also called a key distribution centre (KDC). Communication between the authentication service and clients may be encrypted and signed with the public key of the service.

The security assumption about the authentication service is that it acts honestly when verifying a password or an identification protocol and never grants a request without having seen the proper identification. The reference data against which the verification occurs is assumed to be public but immutable; this is the case for Unix-style password authentication and for zero-knowledge identification protocols, for instance. But a key distribution centre based on symmetric-key cryptography must also protect the corresponding master secret key.

4.5.6 Mix Service

The concept of a mix network was introduced by Chaum [Chaum 1981] as a primitive for privacy. A mix network can for instance be used to achieve anonymous communication over the Internet. A mix network consists of some number of *mix-servers*, S_1, ... S_n say, each of which makes available a public key of a suitable encryption scheme. A user wanting to send a message anonymously encrypts the message and the name of its receiver under the public key of S_n. She then encrypts the result under the public key of S_{n-1}, the result of which she encrypts under the public key of S_{n-2} and so on. The final encryption is then sent to S_1.

As a single message routed through the mix network would be perfectly observable, establishing anonymity requires hiding individual messages among a certain number of similar messages. Therefore, S_1 collects many messages before processing them; we say that messages are processed in rounds. In the first round, S_1 removes the first layer of encryption from all messages, and sends them in a random order to S_2. Then S_2 processes them in the second round, and so on. Eventually, S_n will receive the messages, remove the last layer of encryption and send them to their final destination.

A mix network should satisfy conditions of robustness and privacy, meaning that if a sufficiently large subset of mix servers is honest, then all messages are delivered correctly and it is infeasible to link messages from the input to the output of a round, guaranteeing anonymity.

4.6 Trust-based Verification Model[25]

B. Pfitzmann, M. Schunter, ***Universität des Saarlandes (D)***

M. Waidner, ***IBM Research, Zurich (CH)***

There are many different informal definitions of security. However, most people would agree that requiring a secure system to be correct is a necessary but not sufficient condition for it to be secure.[26] In this section, we consider what aspects beyond the usual notion of correctness are needed and propose a framework for trust-based specification that is outlined in Section 4.6.1. Section 4.6.2 then refines this notion by outlining our approach towards formal evaluation of cryptographic systems.

4.6.1 Trust-based Specification

An overview of our framework for trust-based specification is given in Figure 15 below. Rectangles and normal arrows denote objects and actions of computer science

---

[25] This section is based on [Pfitzmann & Waidner 1994].

[26] Safety specialists might object that their systems need not be “correct”, only fulfil some important properties correctly. We regard this as correctness with respect to a special specification.

(ideally formalised); clouds and dotted arrows are necessarily informal. Some problems and solutions are simplified. We strive for completeness with the design stages; in concrete designs, some stages may be omitted.

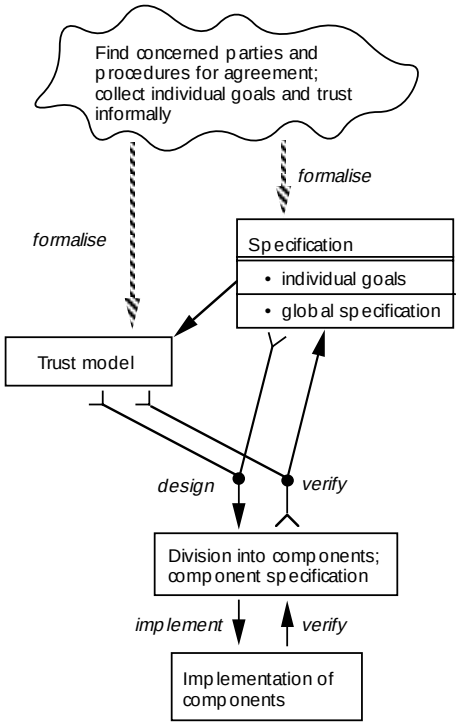

Figure 15 — Stages in the Design of a Secure System

4.6.1.1 Before Development of a Specification

At the beginning of the design of a secure system, the designers must somehow agree

- who the concerned parties are,
- what their individual requirements are, and
- how they will try to agree on common requirements.

These questions are largely outside computer science. However, security only makes sense if this agreement really covers the interests of everybody concerned. Usually, this step is missing in current security criteria. Only the designer and the future owner of the system are considered. However, the design of a system such as a power plant control or a payment system clearly concerns other people, too. The agreement is framed in terms of “values” and “goals” [Biskup 1993]. Values are independent of the system (such as the health of the people concerned) and goals are what the specific system should achieve. This distinction between values and goals is helpful intuitively, but we do not consider it further in the following sections since we do not see how to formalize it (which was not intended in [Biskup 1993]).

Below, we show that considering the concerned parties separately is important not only socially, but also in the security specifications. We use the term **concerned**

party for everyone who has wishes about the system, whereas **user** means someone who interacts with the system at its interface; **system** means an information technology system and comprises neither of these people[27].

4.6.1.2 Specification

Computer science starts, at the latest, when specifications are written. By specification we mean a description of the system behaviour at its external interface, as independent as possible of the internal structure that will be designed. We will not dwell on the usual aspects of specifications, e.g., that there are specifications with different degrees of formality and that they are developed in cycles. However, we do wish to discuss general security-specific aspects of specifications.

4.6.1.3 Two-phase Structure

Most importantly, the specification technique has to reflect that there may be several concerned parties, each with their own interests. Thus, in the general case, there is a two-phase structure in the specification:

1. There are **individual goals** of individual parties.
2. These individual goals are unified and refined into a **global specification**.

The individual goals need not be in a bijective relation with the parties concerned — a party can have many goals that it cannot immediately specify coherently.

Furthermore, groups of concerned parties may have more individual goals than each of them has alone. For instance, with a communication system, only sets of at least two users have the goal of being able to communicate.

The global specification should be free of contradictions[28]; if the goals turn out to be contradictory, they have to be modified by discussion between the concerned parties.[29] Furthermore, a global specification is often assumed to be "complete" [Dierstein 1991, Biskup 1993] (where this notion is left to the intuitive understanding of the reader — necessarily, since the notion of specification is so informal that there are several options). One reason to require completeness is that this makes an informal review from a different perspective possible, at least if the global

---

27 In practice, this separation may be too restrictive since it does not allow the behaviour of users to be specified. System administrators with fixed guidelines, for example, may be considered as components following a given algorithm.

28 Otherwise, no system can fulfil all of them.

29 This can be made less likely by assigning priorities to individual goals [Brüggemann 1993]. Of course, the priorities must also be agreed upon. Typically, preformalized goals for classes of participants can be overruled by more specific goals.

specification is in its own specification language, and not just defined as a "completion" of the goals collected. Another reason is the nothing-else aspect discussed below.

Note that we assume that the individual goals are formalized (at least in theory), whereas in [Dierstein 1991, Biskup 1993], the formal process starts with a complete global specification. However, a complete specification is a complicated object to produce in one step, and we have found that the relation between individual goals and global specifications is an important aspect of several research areas within security. We will show that individual goals are important in combination with trust models.

4.6.1.4 "Nothing-else" Aspect

It is sometimes required that a secure system does nothing else than what is specified [Dierstein 1991, Biskup 1993]; this notion, just like completeness, is also left vague because of the vague notion of specification. This nothing-else aspect goes beyond correctness, i.e., the mere notion of fulfilling a specification. It is important in some applications, e.g., if one requires the absence of Trojan horses or covert channels, but we do not claim that it is always needed.

4.6.1.5 Trust Model

Another characteristic step in the design of a secure system is to determine a *trust model*. This is needed in addition to the usual specification because not every concerned party trusts all components, all steps of the design process, and all other users.

Conversely, one often speaks of a model of *adversaries* or *attackers* or *threats*. By saying "trust", we want to stress three aspects:

- It is a relation, i.e., it depends not only on the component or person that is trusted or not, but also on the concerned party who trusts or not.
- The default value in the design of a secure system should be that something or someone is *not* trusted. This does not mean that all untrusted components will in fact be malicious but rather that we do not assume that they are not.
- Both malicious behaviour as well as accidental faults are meant.

4.6.1.6 Trust Subjects

In the real world, the subjects who express trust are the concerned parties. However, the trust sometimes depends on the individual goal. An important example is graceful degradation, where one has a series of weaker and weaker goals, and needs less and

less trust to achieve them. Hence, formally, the subjects in the trust model are just the *individual goals*.[30]

Note that the goals of sets of concerned parties are formally treated just like those of individual parties, and so is their trust. However, in research areas with general rules to derive the trusted objects from the parties concerned, one needs rules for sets, too.

For a global specification that is not just a collection of individual goals, building in the trust model is more complicated: the subjects can only be the concerned parties, but these are not usually mentioned in a normal specification. Hence one has to introduce them for this purpose.

4.6.1.7 Trust Objects

Usually, a concerned party can express the circumstances under which it trusts a component using the following **trust criteria**:

- Who will design / implement / verify / test / ship the component? (And with how much care and which tools?)
- Where and under whose control will the component be placed?
- What physical and organisational protection will there be for it?

Note that these trust criteria should be applied to "basic" components, i.e., those that concerned parties trust or do not trust as a whole — combining such components into more trusted subsystems is another design stage.

4.6.1.8 Trust Degrees

In many cases, the trust model simply contains binary decisions concerning whether certain objects are trusted for an individual goal or not. However, there may be more degrees of trust, described by a combination of statements of the following types:

- Worst possible behaviour of untrusted objects, e.g.:
  - One trusts that the computational resources of adversaries are limited, or that adversaries cannot break a so-called cryptologic assumption, e.g., *factor 660-bit integers*.
  - One trusts that data may be observed in a component, but the component does not deviate from its component specification.
- Characterisation of sets of simultaneously trusted objects, e.g.:

---

[30] Note that the trust model makes the individual design goals relevant formally, even if one has a global specification too. The goals will have to be proved under these different trust assumptions.

    - One trusts at least *k* out of certain *n* people (such as designers of components).

- Quantification of trust, e.g.:
    - For the reliability of hardware components: parameters such as the mean time to failure.
    - For the trustworthiness of a physical protection measure: the price an adversary would probably have to pay to break it.

Additionally, trust may be a function of time.

4.6.1.9 Component Division and Implementation

Given the specification and the trust model (and assuming one knows the semantics of those two together), one starts building a system. Among the many steps of such an implementation, (at least) one is particular to security, the **division into components** in the sense of the trust objects of Section 4.6.1.7. One determines the number of components, the trust criteria that each component fulfils, and how they are connected.[31] If the trust model offers the possibility to design and place a component so that it is trusted for all individual goals, one is lucky: one can implement the whole system in this component without further trust considerations and thus use all known design and verification techniques. (One may, of course, still prefer a decentralised solution for efficiency reasons.) Well-known strategies for dealing with other cases include storing data under the control of the party who has the primary interest in them, storing copies in different components, or distributing the data with a threshold scheme [Shamir 1979] so that it cannot be read by observing any single component.

Finally, the components have to be implemented. This will usually involve a division into sub-components and several design steps. However, as soon as one has a whole component in one trust domain (including the current designer and the tools) and the whole set of requirements on its interface behaviour, there is nothing security-specific to be considered.

4.6.2 Formal Evaluation

The goal of a formal evaluation is to **verify** that the system with given components fulfils the specification with respect to the trust model. Briefly, each individual goal has to be proved with only the given assumptions about the components. In particular, if subjects either trust objects completely or not at all, only the specifications of the former components can be used in the proof, whereas the other components can behave arbitrarily.

---

[31] Note that objects like networks, which have their own trust criteria, must be considered as components.

We now sketch our formal model for cryptographic security and how it relates to the general specification framework described in Section 4.6.1.

4.6.2.1 Security of Reactive Systems

The model for security of reactive systems [Pfitzmann *et al.* 2000a, Pfitzmann *et al.* 2000b] defines a formal comparison relation "as secure as" that compares the security of two systems providing the same service. Intuitively, one system $\mathbf{Sys_1}$ is at least as secure as another system $\mathbf{Sys_2}$, if and only if whatever an adversary can achieve in the former, it can also achieve in the latter. This relation can be used to specify the behaviour of a system by defining an idealised system providing the desired service.[32] The real system is defined to be secure if it is *at least as secure* as the idealised system.

Formally, a system consists of a set of structures for different trust assumptions. Each structure defines a set of trusted components (formalised by interactive Turing machines) as well as their user-interface. The model compares each structure of the first system with another structure of the second system. In this comparison, the untrusted components are subsumed by one adversary component **A**. The network connections among the correct components and their connections to the adversary component are defined by a connection structure ***G.*** The user-interface as well as the ports that are not connected are connected to a component **H** that models the users of the system[33]. These connected components result in a closed system that has no free input ports. This system is called a configuration.

Using this notation, a system $\mathbf{Sys_1}$ is at least as secure as another system $\mathbf{Sys_2}$, if and only if for any configuration of $\mathbf{Sys_1}$, (including an arbitrary adversary $\mathbf{A_1}$ and any behaviour for the user **H**) there exists a configuration of $\mathbf{Sys_2}$ with the same user **H** (but another adversary $\mathbf{A_2}$) such that the views at the user-interfaces U_1 and U_2 are indistinguishable. Figure 16 shows an example of comparing two systems according to this definition. The dashed line denotes the user-interface, the fine lines are connections between components, the two views at the grey lines are required to be indistinguishable.

---

[32] Note that the idealised system is usually much simpler than the real system. Since it usually defines the service by one trusted machine, it is also called *trusted host*.

[33] Like all other components, the users are state-keeping machines. This models that a system can as well be "used" by another system.

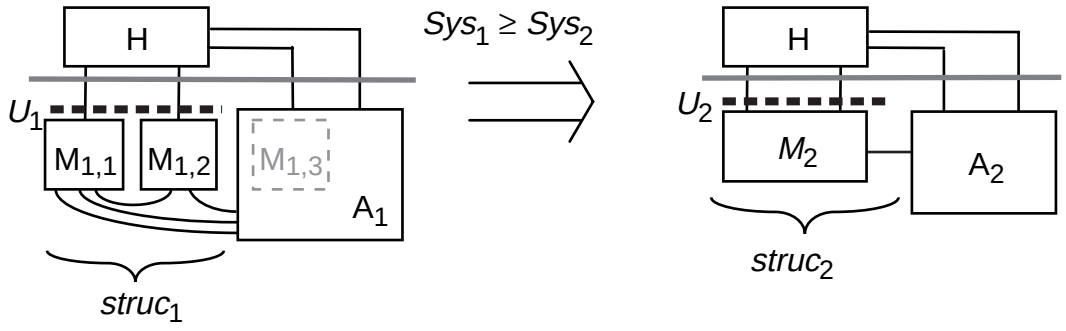

Figure 16 — Comparing Two Configurations

Note that this definition of security covers the intuitive notions of secrecy and integrity:

- If one system is "as secure as" another, it is also "as confidentiality protecting as" the other. If the adversary $\mathbf{A_1}$ can obtain any knowledge φ, this knowledge can be output to **H**. As a consequence, our definition requires that $\mathbf{A_2}$ is able to output the same knowledge, i.e., is able to obtain any knowledge that $\mathbf{A_1}$ is able to obtain.
- If a system is "as secure as" another, the behaviour of the two systems is indistinguishable from the user's point of view, i.e., an integrity requirement expressed by some formula over the user inputs and outputs either holds for both of the configurations or neither of them.

4.6.2.2 Trusted-host-based Specifications

Before actually being able to evaluate the security of a given set of components formally, we have to translate the generalised specification as described in Section 4.6.1 into a form that enables formal verification in our model. This includes translation of the system to be evaluated as well as translating the specification.

In our model, a system to be evaluated is a set of structures. Therefore, for each different trust assumption, we define one structure as well as its user-interface by removing all untrusted components. This assumes a binary trust relation since a component either works correctly or else may behave arbitrarily.

A service specification for a set of trusted machines needs to be "translated" into an idealised and usually non-distributed system providing the desired service. Again, each structure of trusted components will usually be specified by one idealisation. In order to enable efficient implementations, this idealised system not only specifies the idealised service to the user but also idealised services to the adversary. For example, a system sending encrypted messages usually does not hide the fact that a message was sent. As a consequence, the idealised system should provide a service that informs the adversary whether a message was sent or not (e.g., by setting a busy-bit).

These two steps result in an actual system consisting of structures with trusted components and an adversary. This can be evaluated against an ideal system and its adversary using our formal model.

4.6.2.3 Tool-support for Evaluation of Complex Cryptographic Systems

In order to evaluate complex systems using formal methods, tool-support is desirable to manage the high complexity. Since cryptographic systems are probabilistic (i.e., require reasoning about the probabilities of certain events), existing tools cannot handle them. As a consequence, we cannot hope for tools verifying complete cryptographic systems. For many systems, however, existing tools are able to verify whether certain undesirable behaviour occurs at all or whether one system is a refinement of another[34].

To enable the tool-supported proof of complex probabilistic systems, we separate the probabilistic cryptographic components from the deterministic protocols using them. This enables us to use tools for proving the deterministic protocols while proving the cryptographic components "by hand" (see below for an example). In order to show that this separation is viable, we need to apply a composition theorem: In [Pfitzmann & Waidner 2000], we have shown that if

- a system **X** built on idealised system **SpecY** is as secure as another idealised system **SpecX** specifying the behaviour of **X**, and
- a system **Y** is as secure as its idealised version **SpecY,**

then the system **X** built on the (real) system **Y** is also as secure as **SpecX**. Intuitively, this means that a system can be designed using the idealised versions of its secure sub-systems without compromising security. In order to enable tool-supported proofs of cryptographic systems, this theorem is applied as follows (see Figure 17):

- Lower-level (probabilistic) components are identified and proven by hand against a deterministic idealisation specifying their behaviour.
- Higher-level components based on lower-level deterministic specifications can be evaluated against higher-level deterministic specifications using tools such as FDR / CSP [Roscoe 1998].
- The composition theorem guarantees the security of the composed system (i.e., lower-level and high-level components).

For example, Figure 17 illustrates how the theorem can be applied to prove that Certified Mail can be designed using idealised cryptography and shown to be secure if the real cryptography is secure.

---

[34] Intuitively, one system is a refinement of another if all events in the latter are also events in the former.

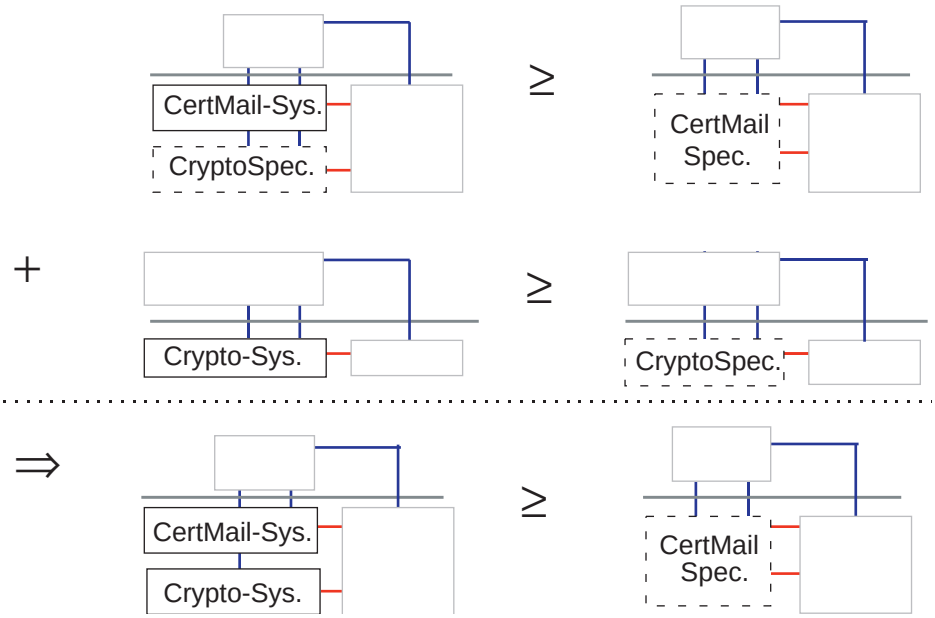

Figure 17 — An Application of the Composition Theorem

In the certified mail system[35] depicted in Figure 17, one first identifies the required cryptographic primitives and defines an appropriate probabilistic subsystem. Then, one defines a deterministic idealisation of this cryptographic subsystem and proves that the cryptographic implementation is as secure as its idealisation. Finally, one proves that the certified mail protocol based on the deterministic cryptographic idealisation is as secure as an idealised system for certified mail. If one can keep all probabilistic actions inside the cryptographic module, the certified mail protocol is deterministic. Thus, as a consequence, this second proof only involves deterministic systems, i.e., tools can be used to support this proof.

Given these two results, the overall security of the composed system results from applying the composition theorem as described above.

4.7 Responsibility Model

J. Dobson, ***University of Newcastle (UK)***

In this section we discuss the structure of responsibility in an organisation. One reason for doing this is that until the responsibilities are understood it is not possible to determine whether an individual is intruding or merely carrying out some responsibility of which the detector is unaware. (And this implies, of course, that we may have to understand the responsibilities of the detector.) Another reason for looking at responsibility structures is that it gives an understanding of privacy, in the following way. The role of an individual is a statement of the responsibilities held by that individual. Privacy is essentially a matter of role separation: the right to expect

35 Certified mail sends a mail such that a receipt is issued if and only if the message is received.

that information generated in the context of one role is not interpreted in the context of another role.

As will be seen, the definition we use of responsibility leads to a basic ontology of relationships rather than of individual agents. As a consequence, we see a communication system and its applications in terms of the relationships they support rather than the individual activities they implement. This gives rise to the model of inter-agent communication discussed in Section 4.8**.**

Together these two models relate to the intentional and extensional aspects of communication: what the purpose of a communication is in terms of the human purposes it is intended to serve; and how this is related to the physical messages that bear this intended communication. It is our belief that it is the policies which dictate the models that drive the choice of mechanisms, and the policies cannot be understood without understanding how the human purposes have dictated the articulation of responsibilities.

The concept of responsibility allows us to make a distinction between an insider attack and an outsider attack in the following way. In both cases the purpose of an attack is to cause harm. An insider attack is characterised by the fact that the intention is to cause harm to those to whom the attacker has a responsibility not to harm (a duty of care). Conversely, an outsider attack is the intention to harm those to whom the attacker does not hold such a responsibility, as a consequence of which the potential attackee must assume a responsibility to protect that which has been entrusted to their care. A vulnerability analysis would therefore look at “the responsibility not to harm” and “the responsibility to protect” as being distinct, the failure mode of the first being an abuse of an existing responsibility whereas the second is the failure to discharge a required responsibility. The main implication is on the nature of the records made during the discharge of responsibility (for example, in order to show that the responsibility was adequately discharged). If insider attacks are considered a threat, there must be an obligation on individual agents to show that they did not abuse their privileges during the discharge of their responsibilities since the keeping of such a record is in their interests in the event they come under unwarranted suspicion. Conversely, if outsider attacks are considered a threat, the obligation on individual agents is to show what protection mechanisms they deployed.

4.7.1 The Core Concepts: Agents and Responsibility

The core concept in our way of looking at organisations is the agent. We describe these as the primary manipulators of the state or structure of the system, but essentially they are the people in the socio-technical system, although it is possible for a machine to behave as an agent. What an agent represents is an ‘office’ in the sense of a role holder, and this can be any size of group from an individual to a whole organisation.

Following from the concept of the agent is the concept of the relationship between agents. We call these **structural relationships** because we regard them as the

skeleton of the socio-technical system. Every relationship between one agent and another implies a conversation and the need for some sort of communication link between them permitting the exchange of information. Structural relationships are thus central in two respects: (i) in that they embody the organisational structure in terms of authorisation and power structures, and (ii) in that they impose requirements in terms of information and communication structures on any system installed.

Figure 18 — A Structural Relationship between Agents

The key to modelling structural relationships is the realisation that they are basically responsibility relationships between agents. This brings us to the second core concept in the modelling scheme, that of **responsibility**. We take the view that the function of the organisation is manifest in the responsibilities held by agents, and that the structure of the organisation is manifest in the responsibility relationships between agents. The rationale behind this view will therefore be explained in depth as it forms the central tenet of our conceptual modelling framework.

SUMMARY

CORE CONCEPTS

AGENT - a manipulator of the system

STRUCTURAL RELATIONSHIPS - relationships between agents

- they are indicative of structure
- they embody responsibility relationships
- they imply conversations and communication links
- they impose requirements for communications and information

4.7.2 Responsibility and the Responsibility Relationship

4.7.2.1 The Nature of the Responsibility Relationship

Being responsible can mean either being accountable for a state of affairs without necessarily any implication of a direct causal connection, or being the primary cause of a result. We have named these two distinct types of responsibility **consequential** and **causal** responsibility respectively. Consequential responsibilities are indicative of the purpose of the organisation and the enduring organisational structure, whereas causal responsibilities are dynamic in nature, being the relationship between an agent

and an event. An example taken from the "Herald of Free Enterprise" disaster[36] illustrates the distinction. The ship's captain is always consequentially responsible for the state of the ship, and in this case was blamed (with others) for the disaster although he did not cause it directly. However, consequential and causal responsibilities are often closely associated as in the case of the deckhand who did not close the hold doors. He was causally responsible when the ship capsized, but he also held consequential responsibility for the state of the hold doors all the time he held the role of deckhand.

In what follows we are attempting to model the enduring organisational structure so the responsibilities referred to throughout this section are only of the consequential type implying accountability, blameworthiness or liability of the responsibility holder. Models of causal responsibility, which in many ways are simpler, are for discussion in a later version of this deliverable.

We define responsibility as a relationship between two agents regarding a specific state of affairs, such that the holder of the responsibility is responsible to the giver of the responsibility, the responsibility principal (Figure 19).

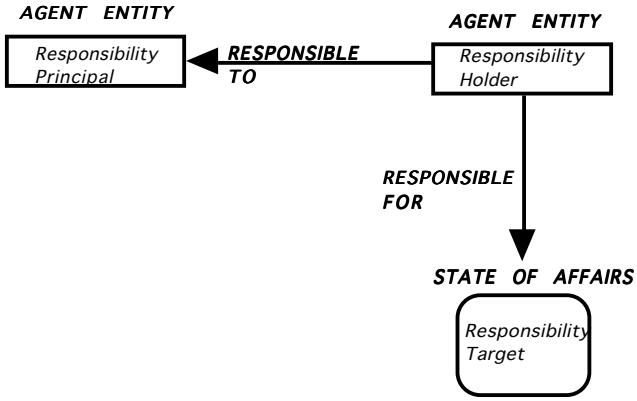

Figure 19 — A Responsibility Relationship between Two Agents

The definition of a responsibility consists of:

a) who is responsible to whom;

b) the state of affairs for which the responsibility is held;

c) a list of obligations held by the responsibility holder (what the holder must do, or not do, in order to fulfil the responsibility);

---

[36] This was a famous accident some years ago in which a car ferry sank and many lives were lost. Briefly, the salient facts were these: a deckhand forgot to close the bow doors before the ship started her voyage, so that water entered the car storage area and capsized the ship; but the captain had no way of knowing directly whether or not the bow doors were closed prior to commanding the starting of the voyage.

d) the type of responsibility (this includes accountability, blameworthiness, legal liability, etc.).

The important point here is that responsibilities cannot be looked at on their own but must always be considered as a relationship between two agents. The states of affairs for which responsibilities are held may be at any level of granularity of the organisation. For example, the responsibilities may be at a very high level (e.g. responsibility for the adequacy of the service provided, for the continuity of a process, for safety, for security, for the accuracy of information and suchlike), or they may be at an individual level for a very specific state (e.g. whether a door is closed, or whether a form is correctly filled in).

4.7.2.2 The Responsibility - Obligation - Activity Relationship

The distinction between **responsibilities**, **obligations** and **activities**, and the relationship of activities to responsibilities through obligations is central to our conceptual modelling framework. This is based on the precept that people execute activities in order to discharge the obligations imposed on them by virtue of the responsibilities they hold. These obligations effectively describe their "jobs" or roles, and are the link between their responsibilities and the activities they execute. We can choose whether it is more appropriate to model responsibilities, obligations or activities depending on what view of the organisation we want to take and what stage we are at in the system design process.

The distinction between responsibilities and obligations is apparent from the words we use: a responsibility is **for** a state (of affairs), whereas an obligation is **to do** (or not do) something that will change or maintain that state of affairs. Thus a set of obligations must be discharged in order to fulfil a responsibility. As such, obligations define in what way the responsibility holder is responsible, and what must be done to fulfil the responsibility. Responsibilities therefore tell us **why** agents do something, whereas obligations tell us **what** they should do. Although we make a clear distinction between responsibilities and obligations (since this distinction is particularly valuable in that we can choose to model either responsibilities or obligations), it should be understood that responsibilities and obligations are closely linked: every responsibility must have obligations attached to it and every obligation must be related to a responsibility.

The distinction between obligations and activities is that obligations define **what** has to be done rather than **how** it is done. As such we regard obligations as an abstraction away from activities. Activities are defined as operations that change the state of the system. Roleholders may (or may not) have a wide choice of activities that discharge the obligations they hold.

It should be emphasised here that, although we have suggested that the activity – obligation – responsibility sequence is progressively more abstract in nature, responsibilities are not abstracted activities. A responsibility model captures more about an organisation than an activity model. Responsibilities represent aspects of

structure and policy as well as function, and are, for example, indicative of commitment by the responsibility holder. We also focus on obligations in preference to activities since an obligation model provides us with an abstract template of the process within the organisation and avoids the trap of working from a model of the system as it is instantiated at present.

SUMMARY

RESPONSIBILITIES

•	Signify Structure:	WHO is responsible to whom
•	Imply Requirements:	WHEN obligations are discharged
•	Responsibilities are:	FOR a state of affairs
•	Obligations are:	TO DO something regarding that state of affairs

THE RESPONSIBILITIES - OBLIGATIONS - ACTIVITIES RELATIONSHIP

•	Responsibilities	WHY agents do something
•	Obligations	WHAT agents must do to fulfil their responsibilities
•	Activities	HOW agents discharge their obligations

4.7.2.3 Delegation of Responsibility

The concept of the responsibility relationship allows us to give an account of the delegation process in terms of responsibilities and obligations. We shall see below that the delegation process is essentially a transfer of obligations from one agent to another thereby establishing a new responsibility relationship between them.

Although it is common to speak of responsibilities being transferred or delegated, and thus as having a dynamic aspect, the fact that a responsibility is a relationship between two agents means that a responsibility holder cannot independently transfer responsibilities to another agent. However, what may be happening in the case of apparent transfer is that the responsibility principal reallocates the responsibility to a new holder by destroying the relationship with the previous holder and establishing a new one with a new holder. The case of apparent delegation of responsibilities is accounted for by the fact that, although he cannot transfer his responsibilities, a responsibility holder can transfer his *obligations* to another agent. The result of this process is the establishment of a new responsibility relationship between the two agents. The first agent becomes the principal of the new responsibility relationship and the other agent is the new responsibility holder. We will now examine this process in detail.

Obligations held by one agent by virtue of the responsibilities they hold may be passed to another agent provided that this transfer is permitted by the relationship between the two agents within the organisational structure. This process is illustrated

in Figure 20. The top diagram shows the initial situation where agent A holds several obligations associated with a particular responsibility. Even when an obligation is transferred to agent B (lower diagram) agent A still retains the original responsibility since this is not transferable: we will see in the next section how that responsibility can be fulfilled. Meanwhile agent B has acquired an obligation relating to the state of affairs for which agent A holds responsibility. Agent B must now also hold responsibility for that same state of affairs, as well as agent A, because Agent B will be affected when the obligation is discharged. However agent B's responsibility is to agent A who delegated the obligation; in other words a new responsibility relationship has been created between them. The lower diagram in Figure 20 illustrates how the process of delegation creates a new responsibility relationship between the two agents.

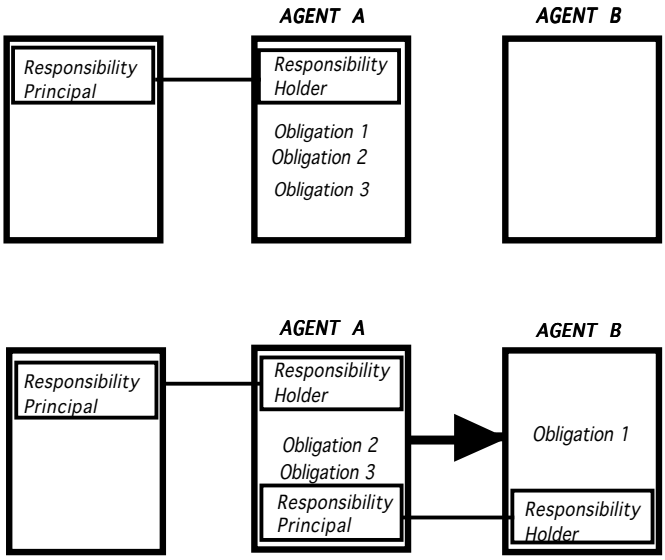

Figure 20 — A Responsibility Relationship Created by the Transfer of an Obligation

An example of this process is where the captain of a ship is responsible to the directors of the company for the safety of their ship. This responsibility to the directors is retained even if the obligations to take safety precautions are transferred to the crew. The crew then acquire responsibility for the state of safety in their respective areas of operation, but their responsibility is to the captain and not directly to the company directors.

A chain of responsibility relationships can thus be created as obligations are passed from one agent to another, with each link in the chain being a responsibility relationship between two agents. Within each individual responsibility relationship both agents have a responsibility for the same state of affairs, although their obligations differ. It should be noted that this delegation process will frequently be implicit rather than explicit, and may be used to explain how the hierarchical organisational structure and distribution of responsibilities has come about over time.

SUMMARY

DELEGATION PROCESS

- When a responsibility holder transfers obligations to another agent he retains his responsibility for the ensuing state of affairs
- The new holder also acquires responsibility for that state of affairs
- The new holder is responsible to the original holder
- i.e. a new responsibility relationship is created between them

4.7.2.4 Functional and Structural Obligations

So far we have only encountered obligations that are functional in nature. They are what agents must do *with respect to a state of affairs* (e.g. execute an activity), in order to fulfil any responsibilities they hold regarding that state of affairs. These we term **functional obligations**. They are indicative of the relationships between the agents and the state of affairs.

We have seen however that when an agent delegates an obligation to another agent, responsibility is still retained for the resulting state of affairs. In order to fulfil this responsibility the agent must ensure that the transferred functional obligation is discharged satisfactorily by the other agent, and thus an agent acquires a new obligation of a structural nature when an obligation is delegated. This is an example of a **structural obligation**. It is to do whatever is appropriate *with respect to another agent* in order to fulfil a responsibility, such as directing, supervising, monitoring and suchlike of the other agent. This other agent also acquires a new structural obligation of a complementary nature to *be* directed, to *be* supervised and suchlike (Figure 21). For example if a director passes an obligation to a manager, the director acquires a structural obligation to direct the manager in the discharging of the transferred obligation, and the manager acquires an obligation to accept direction. In this case the director holds a structural obligation and the manager holds both functional and structural obligations (Figure 22). This can also be expressed as: *Agent A directs (Agent B accepts direction and executes activity)*, where the modal operator "directs/accepts direction" is the structural obligation pair, and "executes" is the functional obligation.

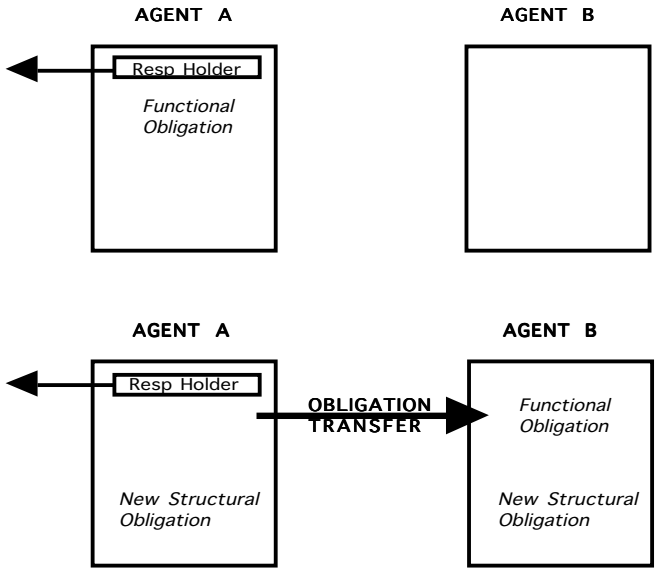

Figure 21 — New Structural Obligations Created by the Transfer of an Obligation

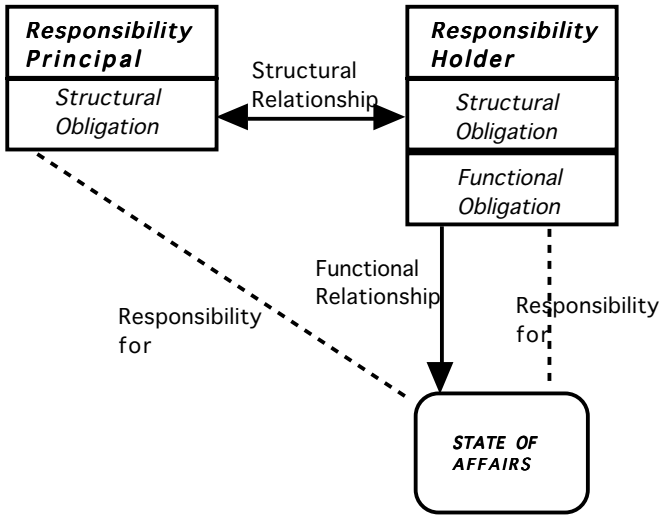

Figure 22 — The Responsibility Relationship in Terms of Obligations

Structural obligations therefore come in pairs and are indicative of the relationships between agents. It is these paired structural obligations that largely determine the flavour of the structural relationships between agents at the role level. Very often these structural obligations are implicit in the hierarchical structure of the organisation rather than arising dynamically from explicit delegation. The distinction between functional and structural obligations is particularly valuable from the point of view of modelling organisational structure, but it should be noted that in reality both types of obligations imply function in that both are realised as activities. Note also that no distinction is made between entities and between relationships; for example, obligations may be regarded as entities linked by a relationship as in the role diagrams, or alternatively as relationships between agents (structural obligations) or between an agent and an action (functional obligations).

SUMMARY

A holder of a responsibility holds TWO TYPES OF OBLIGATION associated with the responsibility:

- FUNCTIONAL OBLIGATIONS are what agents must do with respect to a state of affairs (e.g., execute activity); they are relationships between an agent and a state of affairs;
- STRUCTURAL OBLIGATIONS are what agents must do with respect to other agents (e.g., direct, manage, monitor etc.) to fulfil their responsibilities regarding a state of affairs; they are relationships between agents in the context of a state of affairs.

4.7.2.5 Types Of Structural Relationship

Two categories of structural relationship can be identified on the basis of different patterns of responsibility. Within these categories many types or "flavours" may exist depending on the relative positions of the agents within the organisational structure and their involvement in the particular context.

These two categories are the **contractual** type of relationship between organisations or between distinct organisational units within an organisation, and the **co-worker** type of relationship between agents within an organisation. The distinction is that in a contractual relationship there is no concept of shared responsibility whereas in the co-worker relationship the agents do share responsibility, although their individual responsibilities may be different for the same state of affairs. In a **contractual relationship** the responsibility holder contracts to fulfil the responsibility that is imposed by the responsibility principal (Figure 23). The most typical example of this category is the **service** type of relationship where the server contracts to provide a service to the client. Note that only the server holds responsibility for the provision of the service, and there is no concept of shared responsibility. Of course, the responsibility principal may hold other responsibilities, such as the responsibility for paying for the service provided.

Figure 23 — The Contractual Responsibility Relationship

In this type of structural relationship, the responsibility holder (i.e., the server) apparently holds only functional obligations that must be discharged to fulfil the responsibility, i.e., provide the service. There are however structural obligations on both sides, implicit in the nature of the contract. The client is expected to behave in a "client-like" way to the server: to request, acknowledge and pay for the service, while the server should behave in a "server-like" way by providing the appropriate service under certain mutually agreed conditions.

Most structural relationships between agencies at an inter-organisational level will be of this type. Within a large organisation, relationships between departments will often

be of this type. By looking at the contractual relationships between an agency (organisational unit) and other agencies, we can ascertain what responsibilities are held within the agency of interest and to which agencies it is responsible. At this stage an agency can be regarded as a black box (a container that we do not wish to see inside); we are interested in what responsibilities it holds, and not how they are distributed within the agency. These high level responsibilities define the purpose of the organisation.

The responsibilities held by such an agency are distributed within that agency by the delegation process, either explicitly or implicitly. This process results in a network of responsibility relationships giving rise to structural relationships of the co-worker type. **Co-worker relationships** are distinguished from contractual relationships in that both agents hold responsibility, although in different ways, for the same state of affairs. The main type of co-worker relationship is that which results from the responsibility relationships set up by the delegation process, whether implicit or explicit (Figure 24). The agents are linked by the holding of structural obligations to each other that are created during the delegation process. These paired structural obligations are of the type direct—(accept direction); advise—(request and accept advice); supervise—(be supervised).

Figure 24 — The Co-worker Responsibility Relationship Resulting from Delegation

The nature of these co-worker structural relationships is strongly flavoured by the relative positions of the agents within the organisational power structure insofar as it affects the context in which they are working together. For example, a structural relationship will have a strong element of **power** in it if one agent is senior to another with regard to the specific task and can make and enforce demands on the other. Alternatively it may be what we term a **peer** relationship if the agents are equals and work together as colleagues without any element of enforcement. This power element largely determines the character of the paired structural obligations, i.e., whether the superior agent is directing, managing, supervising or merely collaborating.

A variation on the peer relationship is that of collaboration where there is shared responsibility for a given state of affairs, but no responsibility relationship exists between the two agents, since no element of delegation has taken place. In this case each agent would hold responsibility to a third party but a structural relationship would exist between them by virtue of working together (Figure 25).

Figure 25 — The Co-worker Relationship Resulting from Collaboration

A model of co-worker relationships can be of particular value for the identification of role mismatches and in general for checking whether the organisational structure is well-formed. It can help in job design and generation of future scenarios, and in particular can be used to check that every structural obligation held by one role holder is related to an appropriate structural obligation held by another role holder.

SUMMARY

STRUCTURAL RELATIONSHIPS

- Two categories distinguished on the basis of whether or not there is shared responsibility for a given state of affairs:

CONTRACTUAL RELATIONSHIPS

- No shared responsibility
- Usually inter - organisational
- Most common type is the SERVICE relationship: the server holds responsibility for the provision of service to the client

CO-WORKER RELATIONSHIPS

- Shared responsibility
- Usually intra - organisational
- Most common is the DELEGATION relationship
- Strongly flavoured by the agents' relative power in the organisation within the context in which they are working, resulting in a spectrum of relationships ranging from extreme inequality (POWER) to equality (PEER)
- Sub-type is where agents COLLABORATE, but there is no responsibility relationship between them since they are each responsible to a third party

4.8 Inter-agent Communication Model

J. Dobson, University of Newcastle (UK)

In this section, we present a simple model of communicating agents that extends classical models of message transmission in an attempt to capture the underlying purpose and meaning of the communication. We define an agent as an abstract holder of a responsibility. An agent may be an organisation, a work group, an individual or anything else one chooses to assign responsibility to.

The strategy we shall adopt is, in outline, as follows:

i) We shall define the abstract syntax of a message and of a communication. A computer system can be seen as a means of enabling human communication through the passing of messages.

ii) We shall provide a complete enumeration of possible types of failure mode of messages and communications. Security analysis then consists of attempting to identify all the instances of messages and communications in a system (this is the hard part!) and analysing the defined failure mode for each instance.

4.8.1 Messages

The abstract definition of a message is: some **text** passed from a **sender** to a set of **receivers** over a **channel** (maybe more than one simultaneous channel, as in multimedia). No further elaboration of the primitive terms (in bold) will be provided here. We shall leave the definition of communication until Section 4.7.2.

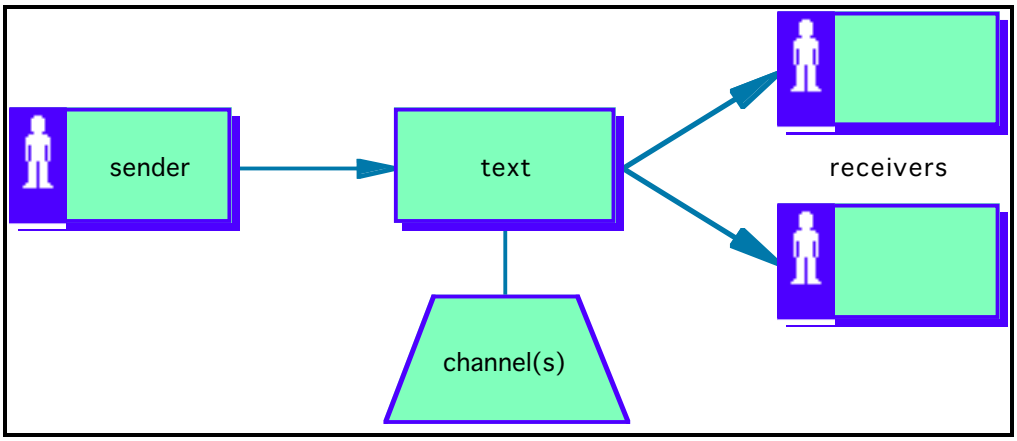

Figure 26 — A Message

This definition allows us to enumerate the possible failure modes of a message:

- The apparent sender (as seen by a particular receiver) might not be the same as the real sender.
- The set of real receivers might not be the same as the set of receivers intended by the sender: some intended receivers might not receive the message and some unintended ones might.
- The text received by a particular receiver might differ from the text intended by the sender.
- There are a number of authorisation failure modes, all of them being some form of the sender not being authorised to send that text to a particular receiver.
- There are a number of sequencing errors over an ordered set of messages: message loss, message duplication, message permutation.
- The communication channel might block or might not have enough capacity.

- The communication channel might suffer from a number of timing faults (messages delivered too late or too early).
- For a message whose text is delivered over a number of channels (e.g., multimedia), there is the possibility of synchronisation errors between the channels.

Our claim is that the above enumeration is complete, in the sense that any failure of an instance of a message (as defined) in a system can usefully be put into one of the above categories. The word "usefully" implies that sometimes there may be a choice of which category to use. A formal theory of messages might be able to prove completeness of the enumeration, though we haven't tried.

SUMMARY

MESSAGES

- Text passed from a sender to a set of receivers over a channel
- Subject to the following failure modes:
- Message loss, message duplication, message permutation
- Mismatch between real and apparent sender
- Mismatch between intended and actual receivers
- Text corruption
- Authorization errors
- Blocked or inadequate channel
- Timing and synchronisation errors

4.8.2 Communication

Communication is a more subtle notion. The basic form of communication is an intention (i.e., a human interest, that which is to be communicated) being mapped by a particular stakeholder (the **speaker**) using a process we shall call **generation** onto a set of messages, which are then sent to a set of other stakeholders (**hearers**) who use individual processes of **interpretation** to reconstruct the original speaker's intention. Again, no further elaboration of the terms in bold will be provided. For example, encryption can be considered one form of generation and decryption as the corresponding interpretation.

But there is more to communication than that, since we have to explain these unanalysed intentions. Our model is that the speaker's intention arises as a result of an **observation** of states of affairs in the speaker's world, and that a particular hearer's reconstructed intention results in the hearer adjusting the state of affairs in *the*

hearer's model of the speaker's world. This model may be computational, or physical, or cognitive. Sometimes it may be the same as the speaker's world itself, but this is not always the case. We shall call this latter process of adjustment a **deduction** process.

The following figure is a summary of our communication model. Note that it defines three domains (knowledge, data, and information), for later convenience.

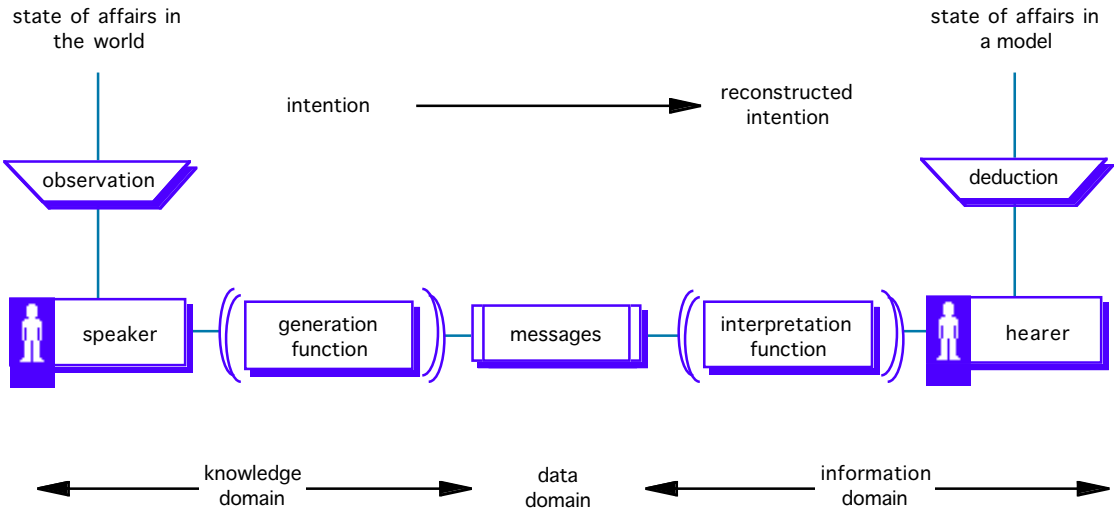

Figure 27 — Communication

This allows us to enumerate the possible failure modes of a communication, other than those that can be categorised as message failures:

- The reconstructed intention might not be the same as the original intention. Sometimes (but not always) this can be identified as a failure in generation (the messages do not carry the intention) or a failure in interpretation (the messages carry the intention but this does not get through to the hearer). Sometimes the generation and interpretation functions might not be mutual inverses.
- The original observation might be incorrect (not correspond to reality in the speaker's world).
- The intention might not capture the original observation correctly ("What I said was … and this was mapped onto the messages correctly, but what I *meant* was …").
- The deduction process might be faulty: the hearer makes inappropriate adjustments in the hearer's model.
- The hearer's model might be inappropriate: the hearer has chosen the wrong selection of state variables, in constructing the model of the speaker's world.

We can now draw up a template that will be used for the categorisation of possible failures in different domains, as follows:

	data domain failures	
message failures	sequence failures	channel failures
real/apparent sender mismatch	lost message	blocked channel
real/intended receiver mismatch	duplicate message	insufficient capacity
sent/received text mismatch	permutation error	synchronisation error
authorisation errors		timing error

knowledge domain failures	**information domain failures**
observation errors	deduction errors
speaker intention errors	hearer reconstruction errors
generation errors	interpretation errors
generation not inverse of interpretation	interpretation not inverse of generation
	modelling errors

We can now indicate how we propose to carry out a vulnerability and threat analysis.

Vulnerability analysis consists in identifying all the instances of messages and communications in the system in the terms we have outlined and using the templates above to identify the various failure modes for each instance.

Threat analysis consists in deciding whether each vulnerability is exploitable, given the domain knowledge or hypothesis that is being assumed, (a threat is related to an assumption.)

At this stage it might be helpful to review the major kinds of error whose possibility might be revealed by our analytical method.

The data domain is basically about *security breaches.* A security policy is a policy over messages, detailing which of the various kinds of data domain failure should not occur. Access control policies govern what counts as authorisation errors, data integrity governs what counts as sent/received text mismatch, lost, duplicated and permutation errors, and denial of service governs what counts as blocked channels, insufficient capacity, synchronisation and timing errors.

But our extension to the knowledge and information domains widens the scope for further classes of possible failure. For example, encryption and decryption can be seen as particular cases of generation and interpretation respectively, and interpretation/generation mismatch means that (for whatever reason) the decryption key is not the one intended by the encryption mechanism. But there are other forms of

generation and interpretation, such as certain operations undertaken on a database application program.

Speaker intention error can occur if “accidental” errors of fact get inserted into a database, and hearer reconstruction errors occur when the information presented is misread from a computer screen (and let no-one say that this does not occur!)

An example of a deduction error is when someone who is perfectly healthy volunteers as a control in the test of an anti-HIV drug. Obviously that person will have been tested at some stage and recorded as having undergone a HIV test. This may later cause someone (e.g., a medical insurance company) to make a misdeduction as to the insurance risk represented by that person.

SUMMARY

COMMUNICATION

- Human intention mapped on to generation of messages which are then interpreted
- Speaker and hearer have models of each other’s worlds
- Subject to the following failure modes:
- Observation and intention errors
- Generation and interpretation errors
- Generation and interpretation not inversely related
- Modelling and deduction errors

Chapter 5 Conclusion

R. J. Stroud, *University of Newcastle upon Tyne (UK)*

To summarise, this deliverable contains three main contributions:

1. A set of use cases and scenarios illustrating the wide applicability of the concepts and models being developed by the MAFTIA project

2. A detailed discussion of the way in which core dependability concepts can be extended and applied to the problems of intrusion detection and intrusion tolerance

3. A set of conceptual models that together constitute a reference model for the MAFTIA project and will guide the development of the MAFTIA architecture

These models and scenarios will be used to inform the work in the rest of the project and will be refined during the course of the project to take into account feedback from other work packages. In particular, the basic concepts in the reference model will be formalised and used as the basis for some of the validation activity in work package 6. Indeed, work has already begun in collaboration between some of the partners on verifying some of the basic services identified in these reference models and significant progress is being made. Also, the conceptual models will be used to guide the development of the MAFTIA architecture in work package 2. Finally, the analysis of intrusion tolerance and intrusion detection in terms of core dependability concepts will inform the work on constructing an intrusion tolerant intrusion detection system in work package 3.

References

[Abrams *et al.* 1995] M. Abrams, S. Jajodia and H. Podell, *Information Security*, IEEE CS Press, 1995.

[Anderson & Lee 1981] T. A. Anderson and P. A. Lee, *Fault Tolerance — Principles and Practice*, Prentice-Hall, 1981.

[Asokan *et al.* 1997] N. Asokan, M. Schunter and M. Waidner, "Optimistic Protocols for Fair Exchange", in *Proc. 4th ACM Conf. on Computer and Communications Security*, pp.6, 8-17, 1997.

[Asokan *et al.* 2000] N. Asokan, V. Shoup and M. Waidner, "Optimistic Fair Exchange of Digital Signatures", *IEEE Journal on Selected Areas in Communications*, 18 (4), pp.591-610, 2000.

[Avizienis 1967] A. Avizienis, "Design of Fault-Tolerant Computers", in *Fall Joint Computer Conference*, AFIPS Conf. Proc., 31, pp.733-43, Washington D.C.: Thompson Books, 1967.

[Avizienis 1978] A. Avizienis, "Fault Tolerance, the Survival Attribute of Digital Systems", *Proc. of the IEEE*, 66 (10), pp.1109-25, 1978.

[Babaõglu 1987] O. Babaõglu, "On the Reliability of Consensus-Based Fault-Tolerant Distributed Computing Systems", *ACM Transactions on Computer Systems*, 5 (3), pp.394-416, 1987.

[Birman & Joseph 1987] K. Birman and T. Joseph, "Reliable Communication in the Presence of Failures", *ACM Transactions on Computer Systems*, 5 (1), pp.46-76, 1987.

[Birman *et al.* 1991] K. Birman, A. Schiper and P. Stephenson, "Lightweight Causal and Atomic Group Multicast", *ACM Transactions on Computer Systems*, 9 (3), pp.272-314, 1991.

[Birrell & Nelson 1984] A. D. Birrell and B. J. Nelson, "Implementing Remote Procedure Calls", *ACM Transactions on Computer Systems*, 2 (1), pp.39-59, 1984.

[Biskup 1993] J. Biskup, "Sicherheit von IT-Systemen als "sogar wenn – sonst nichts – Eigenschaft"", in *Proceedings der GI-Fachtagung VIS '93*, (G. Weck and P. Horster, Eds.), pp.239-54, Vieweg, 1993.

[Brüggemann 1993] H. H. Brüggemann, "Prioritäten für eine verteilte, objekt-orientierte Zugriffskontrolle", in *Proceedings der GI-Fachtagung VIS '93*, (G. Weck and P. Horster, Eds.), pp.51-66, Vieweg, 1993.

[Campbell & Randell 1986] R. H. Campbell and B. Randell, "Error Recovery in Asynchronous Systems", *IEEE Trans. Software Engineering*, SE-12 (8), pp.811-26, 1986.

[Carter & Schneider 1968] W. C. Carter and P. R. Schneider, "Design of Dynamically Checked Computers", in *IFIP'68 Cong.*, (Amsterdam, The Netherlands), pp.878-83, 1968.

[CEN 13608-1] *Health Informatics - Security for Healthcare Communication - Part 1: Concepts and Terminology*, CEN.

[Chandra & Toueg 1996] R. D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems", *Journal of the ACM*, 43 (2), pp.225-67, 1996.

[Chaum 1981] D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms", *Communications of the ACM*, 24 (2), pp.84-8, 1981.

[Cheriton & Zwaenepoel 1985] D. Cheriton and W. Zwaenepoel, "Distributed Process Groups in the V-Kernel", *ACM Transactions on Computer Systems*, 3 (2), pp.77-107, 1985.

[Cristian 1980] F. Cristian, "Exception Handling and Software Fault Tolerance", in *10th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-10)*, (Kyoto, Japan), pp.97-103, IEEE Computer Society Press, 1980.

[Cristian 1988] F. Cristian, "Agreeing on Who is Present and Who is Absent in a Synchronous Distributed System", in *18th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-18)*, (Tokyo, Japan), pp.206-11, IEEE Computer Society Press, 1988.

[Cristian *et al.* 1985] F. Cristian, H. Aghili, R. Strong and D. Dolev, "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement", in *15th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, (Ann Arbor, MI, USA), pp.200-6, IEEE Computer Society Press, 1985.

[Cristian & Fetzer 1998] F. Cristian and C. Fetzer, "The Timed Asynchronous System Model", in *28th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-28)*, (Munich, Germany), pp.140-9, IEEE Computer Society Press, 1998.

[Debar *et al.* 1999] H. Debar, M. Dacier and A. Wespi, "Towards a Taxonomy of Intrusion-Detection Systems", *Computer Networks*, 31 (8), pp.805-22, 1999.

[Denning 1982] D. E. Denning, *Cryptography and Data Security*, Addison-Wesley, 1982.

[Deswarte *et al.* 1991] Y. Deswarte, L. Blain and J.-C. Fabre, "Intrusion Tolerance in Distributed Systems", in *Symp. on Research in Security and Privacy*, (Oakland, CA, USA), pp.110-21, IEEE Computer Society Press, 1991.

[Dierstein 1991] R. Dierstein, "The Concept of Secure Information Processing Systems and Their Basic Functions", in *IFIP/Sec'90*, (Helsinki), pp.133-49, North-Holland, Amsterdam, 1991.

[Dolev *et al.* 1987] D. Dolev, C. Dwork and L. Stockmeyer, "On the Minimal Synchronism needed for Distributed Consensus", *Journal of the ACM*, 34 (1), pp.77-97, 1987.

[Dwork *et al.* 1988] C. Dwork, N. Lynch and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony", *Journal of the ACM*, 35 (2), pp.288-323, 1988.

[ECMA TR/46] *Security in Open Systems — A Security Framework*, ECMA.

[Elmendorf 1972] W. R. Elmendorf, "Fault-Tolerant Programming", in *2nd IEEE Int. Symp. on Fault Tolerant Computing (FTCS-2)*, (Newton, MA, USA), pp.79-83, IEEE Computer Society Press, 1972.

[Fabre *et al.* 1994] J.-C. Fabre, Y. Deswarte and B. Randell, "Designing Secure and Reliable Applications using FRS: an Object-Oriented Approach", in *1st European Dependable Computing Conference (EDCC-1)*, (Berlin), Lectures Notes in Computer Science, 852, pp.21-38, Springer-Verlag, 1994.

[Fischer *et al.* 1985] M. J. Fischer, N. A. Lynch and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process", *Journal of the ACM*, 32 (2), pp.374-82, 1985.

[Fraga & Powell 1985] J. Fraga and D. Powell, "A Fault and Intrusion-Tolerant File System", in *IFIP 3rd Int. Conf. on Computer Security*, (J. B. Grimson and H.-J. Kugler, Eds.), (Dublin, Ireland), Computer Security, pp.203-18, Elsevier Science Publishers B.V. (North-Holland), 1985.

[Fray *et al.* 1986] J.-M. Fray, Y. Deswarte and D. Powell, "Intrusion-Tolerance using Fine-Grain Fragmentation-Scattering", in *Symp. on Security and Privacy*, (Oakland, CA, USA), pp.194-201, IEEE Computer Society Press, 1986.

[Frison & Wensley 1982] S. G. Frison and J. H. Wensley, "Interactive Consistency and its Impact on the Design of TMR Systems", in *12th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-12)*, (Santa Monica, CA, USA), pp.228-33, IEEE Computer Society Press, 1982.

[Garber 2000] L. Garber, "Denial-of-Service Attacks Rip the Internet", *Computer*, 33 (4), pp.12-7, 2000.

[Gong 1992] L. Gong, "A Security Risk of Depending on Synchronized Clocks", *Operating Systems Review*, 26 (1), pp.49-53, 1992.

[Gray 1986] J. Gray, "Why do Computers Stop and What can be done about it?", in *5th Symp. on Reliability in Distributed Software and Database Systems*, (Los Angeles, CA, USA), pp.3-12, IEEE Computer Society Press, 1986.

[Haber & Stornetta 1991] S. Haber and W. S. Stornetta, "How to Time Stamp a Digital Document", *Journal of Cryptology*, 3, pp.99-111, 1991.

[Halme & Bauer] L. R. Halme and R. K. Bauer, "AINT Misbehaving: A Taxonomy of Anti-Intrusion Techniques", http://www.sans.org/newlook/resources/IDFAQ/aint.htm, (accessed: April 10, 2000).

[Huang & Abraham 1982] K. K. Huang and J. A. Abraham, "Low Cost Schemes for Fault Tolerance in Matrix Operations with Processor Arrays", in *12th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-12)*, (Santa Monica, CA, USA), pp.330-7, IEEE Computer Society Press, 1982.

[ISO 7498-2] *Basic Reference Model, Part 2: Security Architecture*, ISO.

[ITSEC] *Information Technology Security Evaluation Criteria*, Commission of the European Communities.

[Joseph & Avizienis 1988] M. K. Joseph and A. Avizienis, "A Fault Tolerance Approach to Computer Viruses", in *1988 Symp. on Security and Privacy*, (Oakland, CA, USA), pp.52-8, IEEE Computer Society Press, 1988.

[Kent & Atkinson 1998] S. Kent and R. Atkinson, *Security Architecture for the Internet Protocol*, Technical Report Request for Comments 2401 IETF, November 1998.

[Kopetz & Ochsenreiter 1987] H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems", *IEEE Transactions on Computers*, C-36 (8), pp.933-40, 1987.

[Ladin *et al.* 1990] R. Ladin, B. Liskov, L. Shrira and S. Ghemawat, *Lazy Replication: Exploiting the Semantics of Distributed Services*, Technical Report N°MIT/LCS/TR-84, MIT Laboratory for Computer Science, 1990.

[Lala 1986] J. H. Lala, "A Byzantine Resilient Fault-Tolerant Computer for Nuclear Power Plant Applications", in *16th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-16)*, (Vienna, Austria), pp.338-43, IEEE Computer Society Press, 1986.

[Lamport & Melliar-Smith 1985] L. Lamport and P. M. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults", *Journal of the ACM*, 32 (1), pp.52-78, 1985.

[Lamport *et al.* 1982] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems*, 4 (3), pp.382-401, 1982.

[Lampson 1981] B. W. Lampson, "Atomic Transactions", in *Distributed Systems — Architecture and Implementation*, (B. W. Lampson, Ed.), Lecture Notes in Computer Science, 105, pp.246-65, Springer-Verlag, Berlin, Germany, 1981.

[Landwehr *et al.* 1994] C. E. Landwehr, A. R. Bull, J. P. McDermott and W. S. Choi, "A Taxonomy of Computer Program Security Flaws", *ACM Computing Surveys*, 26 (3), pp.211-54, 1994.

[Laprie 1992] J.-C. Laprie (Ed.), *Dependability: Basic Concepts and Terminology*, Dependable Computing and Fault-Tolerance, 5, 265p., Springer-Verlag, Vienna, Austria, 1992.

[Laprie 1995] J.-C. Laprie, "Dependable Computing: Concepts, Limits, Challenges", in *Special Issue, 25th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-25)*, (Pasadena, CA, USA), pp.42-54, IEEE Computer Society Press, 1995.

[Laprie *et al.* 1990] J.-C. Laprie, J. Arlat, C. Béounes and K. Kanoun, "Definition and Analysis of Hardware-and-Software Fault-Tolerant Architectures", *Computer*, 23 (7), pp.39-51, 1990.

[Laprie *et al.* 1998] J.-C. Laprie, J. Arlat, J.-P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J.-C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac and P. Thévenod, *Dependability Handbook*, Report N°98346, LAAS-CNRS, 1998 (draft).

[Lee & Anderson 1990] P. A. Lee and T. Anderson, *Fault Tolerance — Principles and Practice*, Dependable Computing and Fault-Tolerant Systems, 3, Springer-Verlag, Vienna, Austria, 1990.

[LMED 1976] *Longman Modern English Dictionary*, Longman Group Limited, 1976 (O. Watson, Ed.).

[Melliar-Smith & Randell 1977] P. M. Melliar-Smith and B. Randell, "Software Reliability: The Role of Programmed Exception Handling", *ACM SIGPLAN Notices*, 12 (3), pp.95-100, 1977.

[Meyer & Pradhan 1987] F. Meyer and D. Pradhan, "Consensus with Dual Failure Modes", in *Digest of Papers, The 17th Int. Symp. on Fault-Tolerant Computing Systems*, (Pittsburgh, USA), pp.214-22, IEEE CS, 1987.

[Neuman & Ts'o 1994] B. C. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks", *IEEE Communications Magazine*, 32 (9), pp.33-8, 1994.

[Nicolaïdis *et al.* 1989] M. Nicolaïdis, S. Noraz and B. Courtois, "A Generalized Theory of Fail-Safe Systems", in *19th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-19)*, (Chicago, MI, USA), pp.398-406, IEEE Computer Society Press, 1989.

[Norman 1983] D. A. Norman, "Design Rules Based on Analyses of Human Error", *Communications of the ACM*, 26 (4), pp.254-8, 1983.

[OMED 1992] *The Oxford Modern English Dictionary*, Oxford University Press, 1992 (J. Swannel, Ed.).

[Peterson *et al.* 1989] L. L. Peterson, N. C. Buchholz and R. D. Schlichting, "Preserving and Using Context Information in Interprocess Communication", *ACM Transactions on Computer Systems*, 7 (3), pp.217-46, 1989.

[Peterson & Weldon 1972] W. W. Peterson and E. J. Weldon, *Error-Correcting Codes*, MIT Press, 1972.

[Pfitzmann *et al.* 2000a] B. Pfitzmann, M. Schunter and M. Waidner, "Cryptographic Security of Reactive Systems", in *Workshop on Secure Architectures and Information Flow*, (Royal Holloway, University of London), Electronic Notes in Theoretical Computer Science (ENTCS), 2000a.

[Pfitzmann *et al.* 2000b] B. Pfitzmann, M. Schunter and M. Waidner, *Secure Reactive Systems*, IBM Research Report N°RZ 3206 (#93252), IBM Research, June 2000b.

[Pfitzmann & Waidner 1994] B. Pfitzmann and M. Waidner, *A General Framework for Formal Notions of 'Secure' Systems*, Report N°11/94, Universität Hildesheim, April 1994 (ISSN 0941-3014).

[Pfitzmann & Waidner 2000] B. Pfitzmann and M. Waidner, *Composition and Integrity Preservation of Secure Reactive Systems*, IBM Research Report N°RZ 3234 (#93280), IBM Research, June 2000.

[Powell 1991] D. Powell (Ed.), *Delta-4: a Generic Architecture for Dependable Distributed Computing*, Research Reports ESPRIT, 484p., Springer-Verlag, Berlin, Germany, 1991.

[Powell *et al.* 1988] D. Powell, G. Bonn, D. Seaton, P. Veríssimo and F. Waeselynck, "The Delta-4 Approach to Dependability in Open Distributed Computing Systems", in *18th IEEE Int. Symp. on Fault-Tolerant Computing Systems (FTCS-18)*, (Tokyo, Japan), pp.246-51, IEEE Computer Society Press, 1988.

[Pradhan 1986] D. K. Pradhan, "Fault-Tolerant Multiprocessor and VLSI-Based System Communication Architectures", in *Fault-Tolerant Computing, Theory and Techniques*, pp.467-576, Prentice Hall, Englewood Cliffs, 1986.

[Rabin 1989] M. O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance", *Journal of the ACM*, 36 (2), pp.335-48, 1989.

[Randell 1975] B. Randell, "System Structure for Software Fault Tolerance", *IEEE Transactions on Software Engineering*, SE-1 (2), pp.220-32, 1975.

[Rennels 1986] D. A. Rennels, "On Implementing Fault-Tolerance in Binary Hypercubes", in *16th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-16)*, (Vienna, Austria), pp.344-9, IEEE Computer Society Press, 1986.

[Rodrigues *et al.* 1996] L. Rodrigues, K. Guo, A. Sargento, R. v. Renesse, B. Glade, P. Verissimo and K. Birman, "A Transparent Light-Weight Group Service", in

Proc. 15th Int. Conf. on Fault-Tolerant Computing Systems (FTCS-15), (Niagra-on-the-Lake, Canada), pp.130-9, IEEE CS, 1996.

[Rodrigues *et al.* 1994] L. Rodrigues, E. Siegel and P. Verissimo, "A Replication-Transparent Remote Invocation Protocol", in *Proc. 13th Int. Conf. on Fault-Tolerant Computing Systems (FTCS-13)*, (Dana Point, California, USA), IEEE CS, 1994.

[Rodrigues & Verissimo 2000] L. Rodrigues and P. Verissimo, "Topology-aware Algorithms for Large-scale Communication", in *Recent Advances in Distributed Systems*, (S. Krakowiak and S. K. Shrivastava, Eds.), LNCS vol. 1752, Springer-Verlag, 2000.

[Roscoe 1998] A. W. Roscoe, *The Theory and Practice of Concurrency*, Prentice Hall, Hertfordshire, 1998.

[Schulzrinne *et al.* 1996] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, Techical Report Request for Comment 1889 Audio-Video Transport Working Group, January 1996.

[Shamir 1979] A. Shamir, "How to Share a Secret", *Communications of the ACM*, 22 (11), pp.612-3, 1979.

[Siewiorek & Johnson 1982] D. P. Siewiorek and D. Johnson, "A Design Methodology for High Reliability Systems: The Intel 432", in *The Theory and Practice of Reliable System Design*, (D. P. Siewiorek and R. S. Swarz, Eds.), pp.621-36, Digital Press, 1982.

[Smith 1986] T. B. Smith, "High Performance Fault-Tolerant Real-Time Computer Architecture", in *16th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-16)*, (Vienna, Austria), pp.14-9, IEEE Computer Society Press, 1986.

[Steiner *et al.* 1988] J. G. Steiner, C. Neumann and J. I. Schiller, "Kerberos: an authentication service for open network systems", in *USENIX Winter Conference*, (Dallas, TX, USA), pp.191-202, 1988.

[Taylor *et al.* 1980] D. J. Taylor, D. E. Morgan and J. P. Black, "Redundancy in Data Structures: Improving Software Fault Tolerance", *IEEE Transactions on Software Engineering*, SE-6 (6), pp.383-94, 1980.

[Trouessin 2000] G. Trouessin, *Towards Trustworthy Security for Healthcare Information Systems*, Report N°GT/2000.03, CESSI/CNAM, June 2000.

[Veríssimo & Almeida 1995] P. Veríssimo and C. Almeida, "Quasi-Synchronism: a Step Away from the Traditional Fault-Tolerant Real-Time System Models", *Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS)*, 7 (4), pp.35-9, 1995.

[Verissimo *et al.* 2000] P. Verissimo, A. Casimiro and C. Fetzer, "The Timely Computing Base: Timely Actions in the Presence of Uncertain Timeliness", in *Proc. of DSN 2000, the Int. Conf. on Dependable Systems and Networks,* pp.533-52, IEEE/IFIP, 2000.

[Verissimo & Raynal 2000] P. Verissimo and M. Raynal, "Time, Clocks and Temporal Order", in *Recent Advances in Distributed Systems,* (S. Krakowiak and S. K. Shrivastava, Eds.), LNCS vol. 1752, Springer-Verlag, 2000.

[Verissimo *et al.* 1997] P. Verissimo, L. Rodrigues and A. Casimiro, "Cesiumspray: a Precise and Accurate Global Time Service for Large-Scale Systems", *Journal of Real-Time Systems,* 12 (3), pp.243-94, 1997.

[Wakerly 1978] J. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications,* Elsevier North-Holland, New York, 1978.

[Xu *et al.* 1995] J. Xu, B. Randell, A. Romanovsky, C. Rubira, R. J. Stroud and Z. Wu, "Fault Tolerance in Concurrent Object-Oriented Software through Coordinated Error Recovery", in *Proc. 25th Int. Symp. on Fault-Tolerant Computing,* (Pasadena, USA), pp.499-508, 1995.

[Xu *et al.* 1999] J. Xu, B. Randell, A. Romanovsky, R. J. Stroud, A. F. Zorzo, E. Canver and F. v. Henke, "Rigorous Development of a Safety-Critical System Based on Coordinated Atomic Actions", in *Proc. 29th Int. Symp. on Fault-Tolerant Computing,* (Madison, USA), pp.68-75, 1999.

[Xu *et al.* 1998] J. Xu, A. Romanovsky and B. Randell, "Coordinated Exception Handling in Distributed Object Systems: from Model to System Implementation", in *Proc. Int. Conference on Distributed Computing Systems (ICDCS-18),* (Amsterdam, The Netherlands), pp.12-21, IEEE CS, 1998.

[Yau & Cheung 1975] S. S. Yau and R. C. Cheung, "Design of Self-Checking Software", in *1st. Int. Conf. on Reliable Software,* (Los Angeles, CA, USA), pp.450-7, 1975.

[Ziegler 1976] B. P. Ziegler, *Theory of Modeling and Simulation,* John Wiley, New York, 1976.