Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: Cambridge (0223) 323010
INTERNATIONAL: +44 223 323010
FAX: +44 223 359779
E-MAIL: apm@ansa.co.uk

ANSA Phase III

Building dependable distributed systems

Nigel Edwards

Abstract

This document describes the basic concepts and technologies used in building open dependable distributed systems and how they relate to the ANSA work on dependability.

Chapter One discusses the basic concepts used to describe dependability. The notion of failure is fundamental to these concepts. Hence it looks at the pathology of failures (how they propagate), assigning fault, and the role of failure models and hierarchies.

There are two basic techniques used to build dependable systems: fault tolerance and fault avoidance (sometimes called fault intolerance). Chapter two gives an overview of the ANSA work on dependability. It then looks at the work from the point of view of fault avoidance and finally looks at the work from the point of view of fault tolerance.

APM.1062.00.04 **Draft** 30 November 1993
Request for Comments (confidential to ANSA consortium for 2 years)

Distribution:
Supersedes:
Superseded by:

Contents

1 Fundamental concepts

This document describes the basic concepts and technologies used in building open dependable distributed systems and how they relate to the ANSA work on dependability. The intended audience is system designers and engineers. A companion document [EDWARDS 93c] looks at the need for dependability in open distributed computing and the ANSA vision for the development of open dependable distributed systems.

The remainder of this chapter discusses the basic concepts used to describe dependability. The notion of failure is fundamental to these concepts. Hence §1.2 looks at assigning fault, fault propagation, and the roles of failure models and hierarchies.

Chapter 2 gives an overview of the ANSA work on dependability. It then discusses the means of delivering dependability and how the ANSA work contributes to this.

1.1 The aspects of dependability

The dependability of a service is described by various non-functional properties of that service such as, reliability, availability, safety and security [LAPRIE 92]. The aim of quantifying these properties is to justify the reliance of an enterprise (or business) on that service. If a service is critical to a business then high values of reliability, availability, safety and security may be required. The more critical the service (or the more severe the consequences of failure), the higher these values are likely to be.

Building and deploying a dependable service involves:

1. Understanding how critical the service is to the business
2. Mapping this onto values of properties which can be measured
3. Configuring the service and the engineering mechanisms supporting it so that the measured values of the properties are at least those required.

This section discusses some of the properties which are often considered when building a service. Precisely which properties and what measurements are of interest will depend on the role the service serves in the business; §1.1.6 looks at this issue.

1.1.1 Reliability

Reliability is a measure of the continuity of service; a common measure of reliability is Mean Time To Failure (MTTF). The reliability of a service can also be described as R(t), a function of time. R(τ) is the probability of uninterrupted service from time $t = 0$ to $t = \tau$. This leads to measures of reliability such as **mission-time** which is the time taken for the reliability of a service to drop below a certain level (e.g. t, such that R(t) 0.85).

1.1.2 Availability

Availability is a measure of how often the system is ready for use; it can be expressed as (MTTF/(MTTF + MTTR)) where MTTR is the Mean Time To Repair the system once it has failed.

MTTR may itself be considered to be a measure of dependability. This is sometimes called **maintainability** [LAPRIE 92].

1.1.3 Safety (or the cost of the failure and the benefit of the service)

In general failures of different services will have different consequences for the enterprise which the system is serving. Some measure of the benefit of delivering the service and the cost of failure of the service is needed. The most appropriate measure will depend on the enterprise which the system is serving.

One very simple way of measuring the cost of failure is to classify different failures according to the severity of their consequences for the enterprise. The severity of a failure cannot be defined in terms of system services and failures: it is defined in terms of external consequences. The most severe failure is one which causes catastrophe. The enterprise which the system is serving decides whether or not a failure is a catastrophe based on its codes of practice, rules, ethics and morals. Safety is measure of the systems ability to avoid catastrophe. The least severe failures are said to be benign.

1.1.4 Security

Security is a measure of the system's ability to prevent unauthorised disclosure or handling of information.

1.1.5 Integrity

Often integrity is discussed as being important in the context of service dependability [ANSA 91]: this can be expressed as consistency constraints on data which is maintained by the service[1]. The correctness of the service depends on maintaining the integrity of the data: if it cannot maintain the consistency constraint on the data the service will fail. Hence providing a service of appropriate reliability, availability, safety and security, neccessarily involves satisfying any integrity constraints.

In some senses integrity is the fifth attribute of dependability. However, it cuts across the other four and cannot be directly observed in the same sense as they can: reliability, availability, safety and security can be measured by the results of interacting with the interface at which the service is delivered. The aim of the ANSA work on transaction models is to provide a set of mechanisms which can be used to ensure integrity is maintained [WARNE 93].

1.1.6 Which property and measurement when?

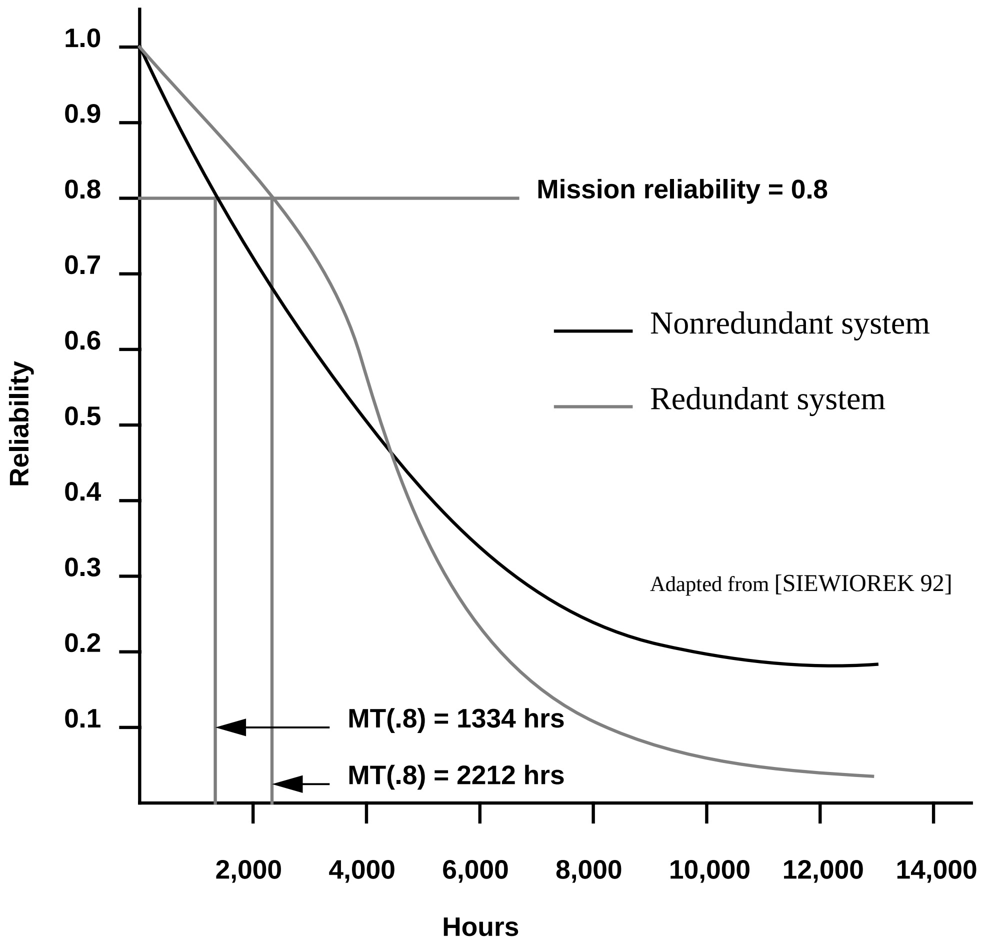
Precisely which properties and what measurements are of interest will depend on the role the service serves in the business. For example, reliability is often used to describe services in which repairs cannot take place (e.g. because

1. Within the security community integrity and consistency are not the same thing. If data is consistent, then it is in some valid form. Integrity adds the condition that the data should be a true reflection of history. This distinction between integrity and consistency is not always made in the fault tolerance community.

access to the system is impossible) or where the service cannot be lost even for the duration of repair (e.g. a flight control system) [SIEWIOREK 92]. Availability is typically used as a figure of merit if the service can be delayed or denied for short periods of time without serious consequences [SIEWIOREK 92] (e.g. an automatic teller machine).

Once it has been decided which properties are appropriate characterisations of the service's dependability, the appropriate measurements for these properties must be chosen. For example consider figure 1.1 which shows the reliability function R(t) for a non-redundant system and a redundant system (triple modular redundancy without repair). The MTTF (area under of the curve) for the non-redundant system is longer than the MTTF of the redundant system. This is because once the redundant system has exhausted its redundancy there is merely more hardware to fail. However, suppose the mission reliability is defined to be 0.8 (i.e. the time for which the system has an 80% chance of running without failing); by this measurement the redundant system is better.

Figure 1.1: The reliability function and mission time



Some of the dependability attributes are in tension with each other (e.g. availability and safety, availability and security). In addition it is unlikely to be possible to maximise all aspects of a service's dependability at the same time: a change made to improve one aspect may well make another aspect of the service's dependability worse. One of the aims of the ANSA work on dependability is to understand the trade-offs involved and how to make them so that the dependability of the service matches what is required

1.2 Failures

From the above it will be seen that understanding the concept of failure is crucial to building dependable systems. The ANSA failure model is described in [EDWARDS 93a]. This section looks at failures: how to understand who is responsible for a failures and the role of failure models.

A failure occurs when what happens does not match what is expected: it is an unexpected event or the non-occurrence of an event when one is expected [EDWARDS 93a]. The failure may be a failure of the component which engaged in the event; or a failure of the component which observed or expected to observe the event.

1.2.1 Fault diagnosis

Fault diagnosis is the process of identifying the faulty component which is responsible for a failure. This can be difficult because the fault may have propagated from the original faulty component by causing other components to fail. If a fault is wrongly attributed to a particular component, erroneous reconfigurations are sure to follow [SCHNIEDER 93a]. In particular good components may be decommissioned while the faulty component is left in the system.

The relevant components in an ANSA system are clients and interfaces. Interfaces are the place where contracts are in place (between client and server), and where reconfiguration is possible. They are also the place where federation boundaries may exist. Fault diagnosis tries to isolate the fault to the particular client or interface from which the fault originated. Sometimes fault diagnosis may have to stop at a federation boundary: beyond that boundary diagnosis is the responsibility of another organisation.

The traditional concept of a failure focuses on service: a failure is said to occur when a service deviates from its specification [LAPRIE 92], [SIEWIOREK 92]. In ANSA the consequences of federation and separation mean that the consequences of mutual suspicion are extremely important. One should not take a client's word for it that a service has failed — it may be that the client itself has failed. The ANSA failure model [EDWARDS 93a] captures this: it does not prejudge whether the faulty component is the one which engages in the event or the one which observes or expects to observe the event.

Detecting a failure means detecting the unexpected. Once a failure has been detected the faulty component needs to be identified: this is the role of fault diagnosis. Whatever does the fault diagnosis requires an unambiguous statement of what the behaviour should be (i.e. what event should or should not have occurred) to determine fault. The ANSA work on federation investigates contracts between clients and servers [HOFFNER 93]. These contracts can be used in fault diagnosis.

1.2.2 Failure models and hierarchies of failure modes

Modern computer systems are enormously complex; distribution introduces further complexity [LINDEN 93]. Managing and understanding these complexities within a single computer requires the introduction of a hierarchy of levels (e.g. circuit level, logic level, program level). A hierarchy of failure models are needed so that dependability at lower levels can be related to dependability at higher levels (e.g. relating circuit level faults to logic functions) [SIEWIOREK 92].

ANSA uses a set of projections for managing the complexities introduced by distribution: the enterprise, information, computational, engineering and technology projections [ANSA 91]. Each of these projections describes a different aspect of a distributed system.

- The enterprise projection is concerned with the purpose of an information processing system within an organisation.
- The information projection is concerned with the meaning and value of information within a system.
- The computational projection is concerned with the decomposition into distributable units which are manipulated by application programmer. These components generate and manipulate the information in the system.
- The engineering projection is concerned with describing the mechanisms and structures needed to support the components in the computational projection.
- The technology projection is concerned with the specific technology used (e.g. specific products and which standards they must conform to).

The engineering projection describes the mechanisms supporting the computational components. In general there will be many different mechanisms and configurations of those mechanisms which satisfy the dependability requirements of a computational component. The ANSA failure model allows the dependability of computational components to be related to the dependability of the supporting engineering mechanisms. Hence it is possible to demonstrate whether an infrastructure can adequately support the dependability requirements of a computational component [EDWARDS 93b].

Configuring the engineering mechanisms to satisfy specific dependability requirements is likely to be a complex an error prone task. Unless proper support is provided to help programmers select the right configuration of mechanisms they are likely to be tempted to ignore what is provided and build their own. The engineering model uses the failure model to provide rules recipes and guidelines for configuring engineering mechanisms to satisfy given dependability requirements. Selecting and analysing the behaviour of a given configuration of components needs to be supported by tools which use the engineering and failure models.

A failure mode describes the characteristics of a class of failures. There are many hierarchies of failure modes in the literature (e.g. [BARBORAK 93], [SHRIVASTAVA 90], [CRISTIAN 90]). These hierarchies are partial orders on failure modes; they are useful, because they say when one engineering mechanism can replace another. For example suppose there is an ordering $\subseteq$ on failure modes, and suppose there are two failure modes x and y such that $x \subseteq y$. Then any mechanism which can detect and tolerate y will also detect and tolerate x. Whether or not a mechanism actually detects and tolerates both x and y will depend on the implementation of that mechanism. Hence it is the engineering model (which describes the engineering objects and the mechanisms which they contain) which will determine the ordering on failure modes. Different engineering models will give rise to different orderings;

within an engineering model different arrangements of components may produce different orderings. The ANSA engineering model for dependability is described in [OSKIEWICZ 93b].

1.3 Summary

Dependability involves considering various attributes, including: reliability, availability, safety, security and integrity. A crucial concept underlying these aspects is the concept of the failure. This chapter discusses the roles of failure models and hierarchies, and discusses fault diagnosis.

2 The means of delivering dependability

There are two basic techniques used to build dependable systems: fault tolerance and fault avoidance (sometimes called fault intolerance). Fault avoidance involves using good engineering practice to minimise the occurrence of faults. Fault tolerance exploits redundancy to negate the effects of faults.

It is important to realise that these two techniques are complementary and not alternatives. Good engineering practice reduces the occurrence of faults, unless the rate at which faults occur is reduced to an acceptable level any redundancy (fault tolerance) will be quickly overwhelmed.

This chapter gives an overview of the ANSA work on dependability. It then looks at the work from the point of view of fault avoidance and finally looks at the work from the point of view of fault tolerance. Fault avoidance and fault tolerance are not mutually exclusive: some technology has aspects of both (e.g. transactions).

2.1 Overview of the ANSA Work on dependability

The aim of the ANSA work on dependability is to develop technology which allows application programmers to use a set of simple concepts to declare their dependability requirements. These requirements will be mapped quickly and efficiently onto a rich set of engineering mechanisms which exploit various redundancy and consistency techniques to deliver the required dependability. The component technologies are:

- A **failure model** which provides the concepts needed by the remainder of the work especially the tools
- A **programming model** which provides programmers with abstractions for expressing requirements and tools for configuring engineering mechanisms
- An **engineering model** which provides a set of engineering mechanisms and sets of standard configurations of these mechanisms — the properties of the latter have been evaluated using the failure model
- An **advanced transaction** framework providing a programming model and set of engineering mechanisms based on transactions
- A **management model** which provides the mechanisms and concepts for fault diagnosis and reconfiguration to maintain dependability

2.1.1 The failure model

The failure model [EDWARDS 93a] together with the basic concepts described in chapter 1 provides the concepts needed to state dependability requirements. These concepts are used by the rest of the work described in this section.

2.1.2 Programming model

The programming model aims to provide programmers with tools and abstractions for expressing dependability requirements. Dependability is an application level concept: what constitutes "dependable" will be determined by the application semantics (see also §2.6). Programmers have an application-level view of dependability: they express requirements for the dependability of a new application or service which they are building and possibly also existing services which it will use.

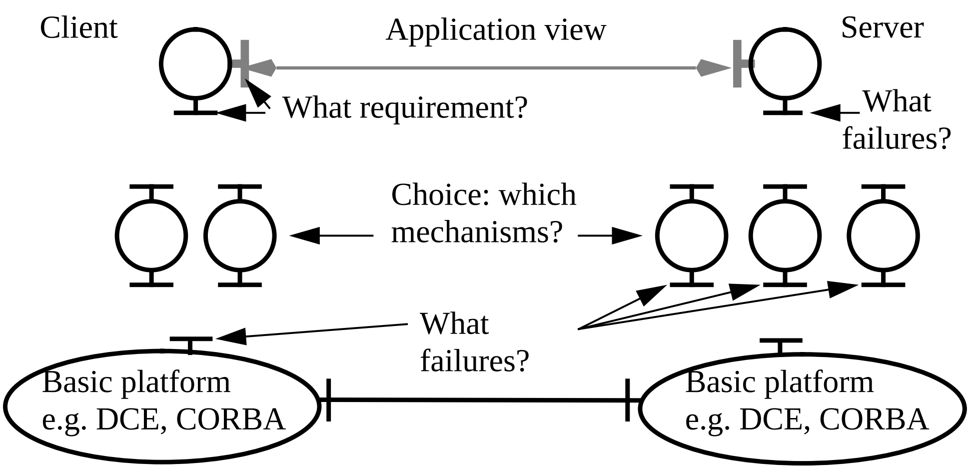
The tools use the failure model to help programmers match the requirements of a new application or service to what is provided by the engineering and other services with which it needs to interact. This is done by choosing and configuring the appropriate engineering level objects; the tools provided can evaluate the dependability of a given configuration.

2.1.3 Engineering model

The engineering model provides a set of basic engineering mechanisms and a set of standard configurations of those mechanisms to meet specific dependability requirements. It is impossible to predict the requirements of all applications, so the engineering model must allow the (dynamic) composition and configuration of mechanisms in addition to providing a set of standard configurations.

The tools and abstractions provided by the programming model are used to choose between different engineering mechanisms and configure the chosen mechanisms. The relationship between the engineering and programming models is shown in figure 2.1. Note that failures can occur anywhere: in the applications, in the engineering mechanisms or in the basic platform.

Figure 2.1: The relationship between the programming and engineering models



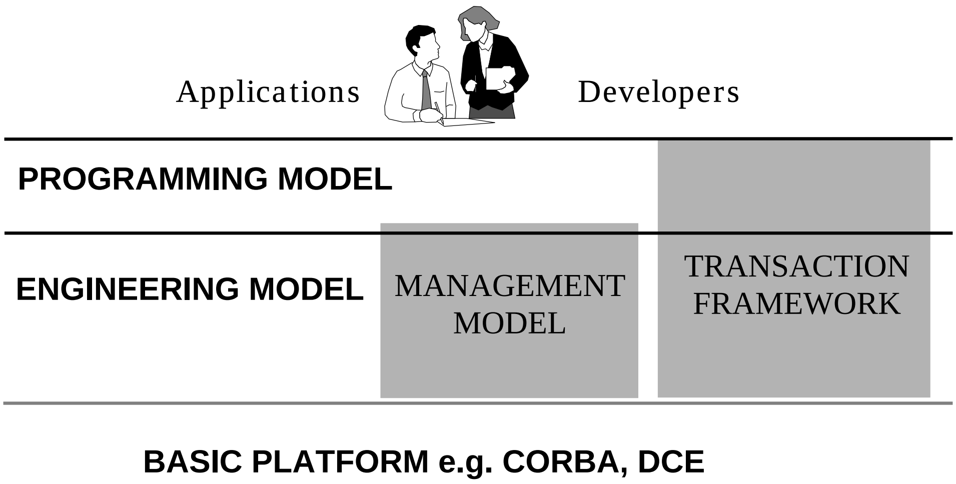
2.1.4 Advanced transactions framework

The advanced transaction framework provides a set of concepts based on transactions which allows programmers to specify different dependability requirements. These requirements are then mapped onto specific engineering mechanisms. The relationship of this work to the programming and engineering model work is shown in figure 2.2.

2.1.5 Management model for dependability

The management model for dependability provides the concepts and mechanisms need to manage the configurations of engineering mechanism. It is concerned largely with how to manage the redundancy mechanisms used to provide fault tolerance. It is specifically concerned with issues of fault diagnosis and dynamic reconfiguration to maintain levels of dependability. This includes on-line upgrading of a service. The relationship between the management model and the work on programming and engineering models is shown in figure 2.2. Much of the management model forms part of the engineering model.

Figure 2.2: How the ANSA dependability work fits together



2.2 Fault avoidance

This section looks at ANSA from the point of view of fault avoidance.

2.2.1 The programming model for dependability

Using tools to choose and configure the engineering objects helps to avoid faults being introduced by incorrect configurations of objects (e.g. invocation of an unsupported operation). The amount of checking possible is limited by technologies available which can express the behaviour of objects.

It is usual to perform type checking based on information provided in an interface definition. The interface definition is a very limited specification of the expected or allowed behaviour. Unfortunately the theory for relating behavioural descriptions is not well developed (e.g. timing constraints, concurrency constraints or constraints on ordering of invocations). One of the aims of the ANSA work on federation is to develop the concepts needed to specify contracts between objects, this will allow more sophisticated checking of interactions between objects.

This checking should be applied to both engineering objects and application objects. Currently checking tends to be limited to application level objects, because the configuration of supporting engineering objects tends to be rather static and uniform. Well understood, standard configurations do not require constant checking and validation. In the future the aim is to have dynamic configurations of engineering objects configured specifically to match client and server requirements, each new configuration will require checking.

2.2.2 The engineering model for dependability

Much of the engineering model is concerned with the provision of redundancy to give fault tolerance. However, it also provides some mechanisms which enforce requirements; these can be regarded as fault avoidance mechanisms. An example of such a mechanism is a protocol which enforces ordering between messages to ensure a group of servers see messages in the same order. This avoids the state of the servers becoming inconsistent.

Note that the above mechanism may be itself part of a replication protocol which is intended for fault tolerance.

2.2.3 The advanced transaction framework

The advanced transaction framework includes the aspects of fault avoidance described in §2.2.1: tools which can check configurations of objects. In addition the transaction framework provides programmers with a set of abstractions which can be used to enforce consistency requirements [WARNE 93]. For example computations enclosed within “flat” transactions have well defined relationships with each other (described by the so-called ACID properties) [GRAY 93]. This helps to avoid faults which can be introduced by concurrent interfering computations.

The tools provided to support the advanced transactions framework will assist the programmer in configuring the engineering mechanisms (see §2.2.1) which will help to avoid faulty configurations.

2.2.4 The management model for dependability

The management mechanisms themselves need to be dependable — system recovery mechanisms can be responsible for 35% of system failures [TOY 93]. The tools provided to support the management model will help to configure the management mechanisms (see §2.2.1), reducing the likelihood of faulty configurations.

2.3 Fault tolerance

Fault tolerance is concerned with the provision and management of redundancy to negate the effects of faulty components. This section looks at the ANSA work on dependability from the point of view of fault tolerance.

2.3.1 The programming model for dependability

The programming model provides programmers with the concepts needed to express the need for redundancy. The tools help to map these requirements onto what is provided by the engineering model.

The programming model also provides support for capturing application level redundancy. This redundancy is supported and supplemented by the underlying engineering mechanisms to help provide fault tolerance. Application level redundancy is redundancy which is associated with the application semantics. For example:

- Knowledge which restricts the time or value of a result, e.g. time should not run backwards; this kind of redundancy could be captured by behavioural descriptions of objects
- Alternative algorithms for achieving a particular aim (this is sometimes called resourcefulness [ABBOTT 93], recovery blocks is one example which exploits this idea [RANDELL 75])
- Auditing and self checking routines
- The use of stability and self stabilisation [SCHNIEDER 93b]

2.3.2 The engineering model for dependability

The engineering model provides a set of mechanisms to supplement and support application redundancy. The engineering model is not only concerned with the provision of redundancy, it is also concerned with the management of redundancy, the latter is considered separately in §2.3.3. Redundancy can be provided in the form of extra storage, processing or communications, further it can be provided in space or time (e.g. doing the same thing twice simultaneously or sequentially) [OSKIEWICZ 93b]. Examples include:

- Replication [OSKIEWICZ 93a]
- Checkpointing [BIRRELL 87]
- Replicated stable store

Comprehensive lists of redundancy technology are given in [SIEWIOREK 92] and [SMETHURST 93].

2.3.3 The management model for dependability

The management model for dependability is concerned with the management and configuration of redundancy mechanisms. Redundancy mechanisms need configuring to deliver some given level of dependability and managing to ensure that a service is delivered in spite of the presence of faulty components.

Often redundancy management will be embedded into the mechanisms which provide redundancy. However, simpler more robust designs can often result by separating the redundancy mechanisms from the controlling mechanisms. For example separating the adjudicator from the variant controller [RANDELL 93].

A redundant system may go through as many as eight stages when a failure occurs [SIEWIOREK 92].

1. **Fault confinement** is concerned with limiting propagation of the fault. This involves liberal use of detection mechanisms to try and detect a fault as soon as possible.
2. **Fault detection** measuring value and time and comparing what is observed to what is expected. Redundancy provides the information needed to set expectations.
3. **Fault diagnosis** is used if fault detection does not identify the faulty component. Redundancy can provide the information need for diagnosis. Two important approaches are specification based diagnosis and symptom based diagnosis. Specification diagnosis uses specifications to determine the expected behaviour and develop diagnostic tests from this. Symptom based diagnosis recognises that failures are often preceded by a period of instability and tries to recognise trends which indicate the faulty

component. Recent uses of symptom based diagnosis have shown it can be more effective than specification based techniques [SIEWIOREK 92].

4. **Reconfiguration** takes place once the faulty component has been identified. The aim is either to isolate the system from the faulty component or to replace it with a spare.
5. **Recovery** attempts to remove the effect of the fault. Redundant information can be used to correct the erroneous state (space redundancy). Alternatively the system can roll (backwards or forwards) and either retry or try an alternative strategy (time redundancy).
6. **Restart** takes place once all the damaged state has been removed. In extreme cases large parts of the system may need to be restarted from its initial state.
7. **Repair** restores the faulty component to an undamaged state. Redundancy might be used to correct erroneous state.
8. **Reintegration** involves reconfiguring the system to introduce the repaired component.

If the system has requirements for high availability all these stages may have to take place "on-line".

2.3.4 The advanced transaction framework

The advanced transaction framework includes many of the aspects of fault tolerance discussed in §2.3.1, §2.3.2 and §2.3.3. The aim is to deliver the constraints specified by the programmer in spite of the effects of faults. Transactions use redundancy to undo their effects should they need to abort. For example, the Tandem transaction processing monitor (Pathway) distributes work to available processors. Should any of this work be lost or compromised by failure it is automatically restarted after being rolled back to its initial state [BARTLETT 92].

2.4 Resolving "Who guards the guardians?"

Resolving the "Who guards the guardians?" dilemma means making sure that the engineering mechanisms introduced for dependability are themselves dependable. As shown in figure 2.1 these mechanisms themselves may fail. The engineering mechanisms need to be configured so that these failures are tolerated and do not compromise the dependability of the applications which they support. In particular it is important to avoid introducing single points of failure in the engineering. (This can happen when adding in the fault detection mechanisms such as comparators.)

2.5 Evaluating dependability

It is often hard to measure all the parameters which affect the dependability of a design. Thus in general it is difficult to make absolute measurements of dependability; it is easier to perform evaluations which compare one design to another. For example a statement such as: "Design A has an MTTF 1.6 times that of Design B" is a comparative statement. Given a comparative measurement it may be possible to turn it into an absolute measurement. It may be known that design B has an MTTF of 8000 hours, so A's MTTF is predicted to be 12,800 hours. Thus comparative measurements allow the effects of changes in the engineering to be measured.

One of the aims of the work on the programming model is to develop tools which will support comparative evaluations.

2.6 End-To-End arguments in dependability

Dependability is an end-to-end concept: what is dependable and what constitutes a failure to an application can only be understood by understanding the application semantics. For example, a file transfer is completed successfully when all the file data has been safely and correctly stored in the file system of the recipient machine, not just when the data has been delivered by the network to the machine (it may crash before storing the data) [SALTZER 81]. Hence dependability techniques which do not take account of application semantics will always be insufficient [CHERITON 93]. This is why the redundancy techniques discussed in §2.3 all impinge to varying degrees on the programming model.

Techniques such as transparent replication will have only a small effect on the programming model: programmers or installers of the service need to be aware of the degree of replication. Techniques such as recovery blocks [RANDELL 75] require much greater involvement from the programmer. Understanding which technique is appropriate can only be done by understanding the application semantics.

2.7 Existing Systems

The opportunities to engineer systems from scratch are becoming fewer and fewer. In general new applications and systems will have to interwork with what already exists. If the new applications are to be dependable the dependability of the existing services it interworks with needs to be evaluated. If they do not provide sufficient dependability they need to be enhanced in some way, or at the very least the new service needs to be protected from them.

The engineering model [OSKIEWICZ 93b] needs to provide mechanisms to do this.

2.8 Summary

There are two basic techniques in building dependable systems: fault tolerance and fault avoidance. Fault tolerance uses redundancy to negate the effects of faults. Fault avoidance uses good engineering practice to minimise the occurrence of faults.

This chapter has described, from the point of view of fault tolerance and avoidance, the work on the programming model for dependability, the engineering model for dependability, the advanced transaction framework and the management model for dependability.

The tools which support the configuration of engineering mechanisms carry out fault avoidance checks. The programming model can be viewed as providing programmers with technology to express the needs of the application for redundancy. The engineering model can be viewed as providing and managing that redundancy. The advanced transaction framework is part of the engineering and programming models. The management model is part of the engineering model.

2.9 Acknowledgement

The author would like acknowledge the contribution to this document of Ed Oskiewicz, seconded to the ANSA team by BT, Owen Rees of APM Ltd., and John Warne, seconded to the ANSA team by BNR-Europe. The comments of Andrew Herbert, of APM Ltd., where also very helpful.

References

[ABBOTT 93]

Abbott, R.J., "Resourceful Systems for Fault-Tolerance, Reliability and Safety", ACM Computing Surveys, Vol. 22, No. 1, March 1990, p35-68.

[ANSA 91]

"An Application Programmer's Introduction to the Architecture", APM Ltd., Cambridge U.K., 1991.

[BARBORAK 93]

Barborak, M., Malek, M., Dahbura, A., "The Consensus Problem in Fault-Tolerant Computing", ACM Computing Surveys, Vol, 25, No. 2, June 1993.

[BARTLETT 92]

Bartlett, J., Bartlett, W., Carr, R., Garcia, D., Gray, J., Horst, R., Jardine, R., Jewett, D., Lenoski, D., McGuire, D., "Fault Tolerance in Tandem Computer Systems", in [SIEWIOREK 92], p586- 648.

[BIRRELL 87]

Birrell, A.D., Jones, M.B., Wobber, E.P., "A Simple and Efficient Implementation for Small Databases", in Proc 11th ACM Symp on OS Principles, 1987, ACM OS Review, Vol. 21, No. 5 p149-154.

[CAMERON 93]

Cameron, E.J., "Scenario", APM.1064, APM Ltd., Cambridge U.K., 1993.

[CHERITON 93]

Cheriton, D., Skeen, D., "Understanding the Lmitations of Causally and Totally Ordered Communication", in Proc 14th ACM Symposium on Operating System Principles, 1993.

[CRISTIAN 90]

Cristian, F., "Understanding Fault-Tolerant Distributed Systems", IBM Research Report, RJ 6980 (66517) 8/24/89 (revised 4/6/90), Almaden Research Center, California, USA.

[EDWARDS 93a]

Edwards, N.J., Rees, R.T.O, "A Model for Failures in Dependable Systems", APM.1027, APM Ltd., Cambridge, U.K., 1993.

[EDWARDS 93b]

Edwards, N.J., "Applying the ANSA failure model to active replica groups" , APM Ltd., Cambridge, U.K., 1993.

[EDWARDS 93c]

Edwards, N.J., "Open Dependable Distributed Systems", APM.1073, APM Ltd., Cambridge, U.K., 1993.

[GRAY 93]

Gray, J., Reuter, A., “Transaction Processing: Concepts and Techniques”, Morgan Kaufmann, 1993.

[HOFFNER 93]

Hoffner, Y., Beasley, M., Deschrevel, J.P., “Federation topic plan”, APM.1028, APM Ltd., Cambridge U.K.

[LAPRIE 92]

Laprie, J.C. (ed.), “Dependability: Basic Concepts and Terminology”, Springer-Verlag 1992

[LINDEN 93]

van der Linden, R., “An Overview of ANSA”, AR.000.00, APM Ltd., Cambridge U.K., May 1993.

[OSKIEWICZ 93a]

Oskiewicz, E.O., Edwards, N.J., “A Model for Interface Groups”, AR.002.01, Cambridge U.K., May 1993

[OSKIEWICZ 93b]

Oskiewicz, E.O. “The Dependability Engineering Model”, APM.1044, APM Ltd., Cambridge, U.K., in preparation.

[RANDELL 75]

Randell, B., “System Structure for Software Fault Tolerance”, IEEE Trans. on Software Engineering, SE-1, No. 2, June 1975, p220-232.

[RANDELL 93]

Randel, B., Xu, J., “Object-Oriented Software Fault Tolerance: Framework, Reuse and Design Diversity”, in Predictably Dependable Computing Systems (PDCS 2) First Year Report 1993.

[SALTZER 81]

Saltzer, J.H., Reed, D.P., Clark, D.D., “End-To-End Arguments in System Design”, in Proc. 2nd International Conference on Distributed Systems, Paris, France, 8-10th April, 1981, p509-512.

[SCHNIEDER 93a]

Schnieder, F.B., “What Good are Models and What Models are Good?” in Distributed Systems (second edition), Mullender, S., (ed), Addison-Wesley, 1993.

[SCHNIEDER 93b]

Schnieder, M., “Self-Stabiliszation”, ACM Compting Surveys, Vol. 25, No. 1, March 1993, p45-67.

[SHRIVASTAVA 90]

Shrivastava, S.K., Ezhilchelvan, P., Little, M., “Understanding Component Failures and Replication in Distributed Systems”, ISA Project Report: UNT/TR1, University of Newcastle May 1990.

[SIEWIOREK 92]

Siewiorek, D.P., Swarz, R.S., “Reliable Computer Systems — design and evaluation”, Digital Press, 1992.

[SMETHURST 93]

Smethurst, R., Wharton, P., “ OPENFramework Availability”, Prentice-Hall 1993.

[TOY 93]

Toy, W.N., “Fault-Tolerant Deisgn of AT&T Telephone Switching System Processors”, in [SIEWIOREK 92], pp533-574.

[WARNE 93]

Warne, J.P., “An Extensible Transaction Framework”, APM.1060, APM Ltd., Cambridge, U.K., in preparation.