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. An overview of the ANSA failure model is given.

Chapter Two gives an overview of the ANSA work on dependability. This aims to provide the technology for building open dependable distributed systems on industry standards platforms such as DCE and CORBA. The engineering model provides a choice of mechanisms to enhance the functionality of the basic platform to meet the requirements of applications for dependability. The programming model helps programmers meet the requirements of the chosen engineering. An extended transaction framework and a management model are core components of the engineering and programming models.

APM.1144.00.02 **Draft** 23 February 1994

Technical Report

Distribution:
Supersedes: APM.1062
Superseded by:

Building Dependable Distributed Systems

Building Dependable Distributed Systems

Nigel Edwards

APM.1144.00.02

23 February 1994

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

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

The sponsors of the ANSA Workprogramme have agreed to allow access by companies which have signed an agreement with Bellcore in respect of the Workprogramme of telecommunications research currently known as TINA-C to permit said companies access to and use of certain documents, software, information and deliverables arising from the results of the ANSA Workprogramme. This information will made available by the ANSA sponsors either as paper copies or through the medium of electronic file transfer from the information storage system operated by Architecture Projects Management Limited on behalf of the ANSA sponsors.

This is one such document and access is allowed in strict confidence on the understanding that the user accepts these conditions and on the sole basis that it will be restricted to those persons involved in the DPE work package of the TINA-C Workprogramme and that it will not be disclosed to any other person, firm or corporation.

The use of this information is restricted to its use only for the purposes of the carrying out of the DPE workpackage of the TINA-C Workprogramme and only at the site provided by Bellcore for that Workprogramme. No licence or permission for its use in any other part of the TINA-C Workprogramme or for its subsequent exploitation is granted and the ownership and copyright of all such documents, software, information and deliverables is expressly retained by Architecture Projects Management Limited for and on behalf of the sponsors for the time being of the ANSA Workprogramme. In the event of a company leaving the TINA-C Workprogramme or resigning from its bilateral agreement with Bellcore, then that company shall promptly and without demand return to Architecture Projects Management Limited all copies of any information, documents, software or other IPRs obtained under these provisions.

The access granted by these provision is on the understanding that the TINA-C consortium and the sponsors for the time being of the ANSA Workprogramme intend to and shall promptly enter into a suitable formal agreement for access to information and interavailability of IPRs (including software) for the purposes of the carrying out of the ANSA and TINA- C Workprogrammes.

With regard to any company which is participating in the TINA-C Workprogramme and which is also a sponsor of the ANSA Workprogramme, the obligation of confidentiality and the use restrictions contained in these provisions shall be subject and without prejudice to the obligations undertaken by, and the rights granted to, such company under the ANSA sponsorship agreement.

Contents

1	**1**	**Fundamental concepts**
1	1.1	The aspects of dependability
1	1.1.1	Reliability
2	1.1.2	Availability
2	1.1.3	Safety (or the cost of the failure and the benefit of the service)
2	1.1.4	Security
2	1.1.5	Integrity
3	1.1.6	Measuring Dependability properties
4	1.2	Failures
5	1.2.1	Fault diagnosis
5	1.2.2	Failure models and hierarchies of failure modes
7	1.3	Summary
9	**2**	**Dependability in ANSA**
10	2.1	The programming model
11	2.2	The engineering model
13	2.3	The management model
14	2.4	The extended transactions framework (ETF)
15	2.5	Evaluating dependability
16	2.6	End-To-End arguments in dependability
16	2.7	Summary
17	2.8	Acknowledgement

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 94b] 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 introduces the ANSA failure model and then 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. This aims to provide the technology for building open dependable distributed systems on industry standards platforms such as DCE and CORBA. The engineering model provides a choice of mechanisms to enhance the functionality of the basic platform to meet the requirements of applications for dependability. The programming model helps programmers meet the requirements of the chosen engineering. An extended transaction framework and a management model are core components of the engineering and programming models.

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 = τ. 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 94].

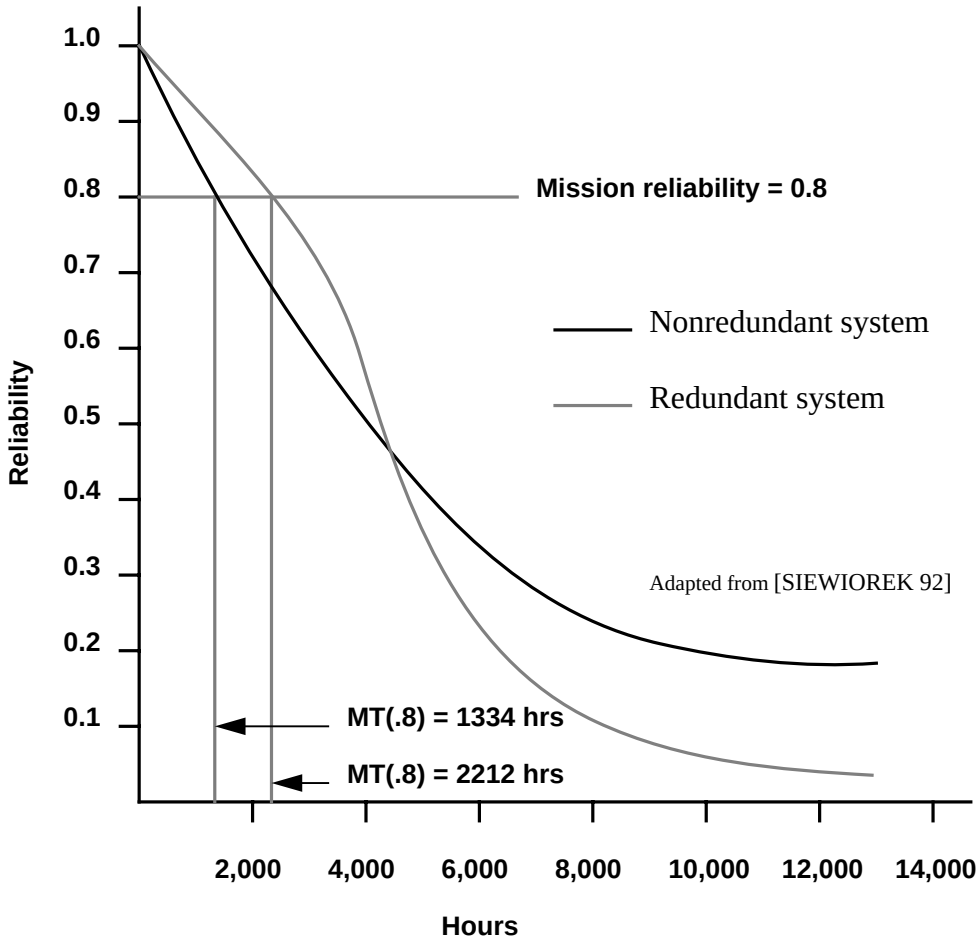
1. Within the security community integrity and consistency are not the same thing. If data is consistent, then it is in some valid form. Integrity adds the condition that the data should be a true reflection of history. This distinction between integrity and consistency is not always made in the fault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

1.1.6 Measuring Dependability properties

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 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

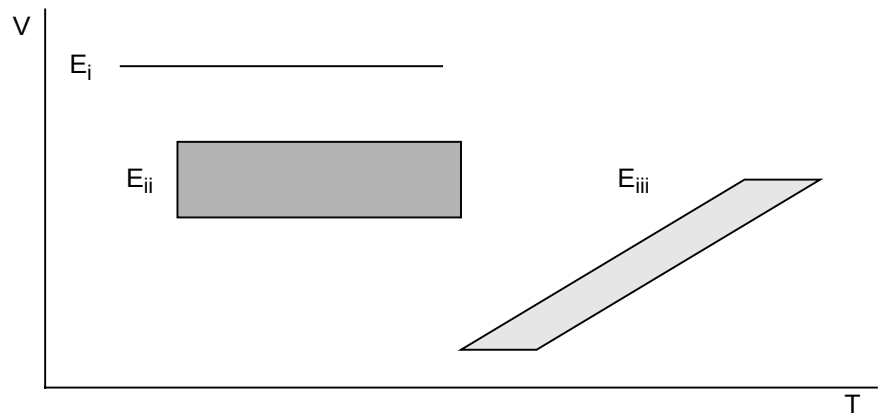
1.2 Failures

From the above it will be seen that understanding the concept of failure is crucial to building dependable systems. This section looks at failures: how to understand who is responsible for a failures and the role of failure models. It begins with a summary of the ANSA failure model [EDWARDS 94a].

The ANSA failure model assumes that a system is composed of components which can engage in events which are observed by other components in the system. An **event** is considered to occur with some value at some time, by the observer; there is no notion of a global observer, a global ordering on events or a global time. An event which occurs is called an **occurrence**.

The model defines **expectation regions** (a region in the $value \times time$ space) which define a time interval and a restricted set of values within which a component expects to observe an event. Boundaries can be drawn around an object's expectation region by determining an objects **expectations**: what it expects from the objects with which it interacts. Three example are shown in figure 1.

Figure 1: Expectations



The three examples are:

- E_i: a particular value is expected in some time interval
- E_{ii}: one of a range of values is expected in some time interval
- E_{iii}: one of a range of values is expected in some time interval but the value is related to the time at which the event occurs.

A **failure** is a mismatch between an occurrence and an expectation. For example, an occurrence which does not match what is expected: any occurrence not in regions E_i, E_{ii}, or E_{iii} would be a failure. The absence of an occurrence when there is an expectation of one is also a failure; this kind of failure is conventionally called an omission failure [LAPRIE 92].

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 fault 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 94a] 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 94a].

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 (e.g. omission failures, value failures, crash failures, fail-stop [LAPRIE 92]). There are many hierarchies of failure modes in the literature (e.g. [BARBORAK 93], [SHRIVASTAVA 90b], [CRISTIAN 90]). Such hierarchies are sometimes referred to as “failure models”. In contrast the ANSA failure model is not a failure hierarchy; it provides a set of concepts for understanding the semantics of failure.

Failure hierarchies arise from 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. Suppose x is omission failures and y is value failures, 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 prescribes the configuration and implementation of engineering mechanisms) 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 discussed further in §2.2.

Failure modes are useful in the engineering of dependable systems. For example, if the failure behaviour of a server is known to be restricted to a well understood mode (e.g. fail-stop), the engineering mechanisms used by its clients only need to be able to deal with this behaviour.

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 Dependability in ANSA

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.

The aim of the ANSA work on dependability is to provide technology which allows application programmers and system designers 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:

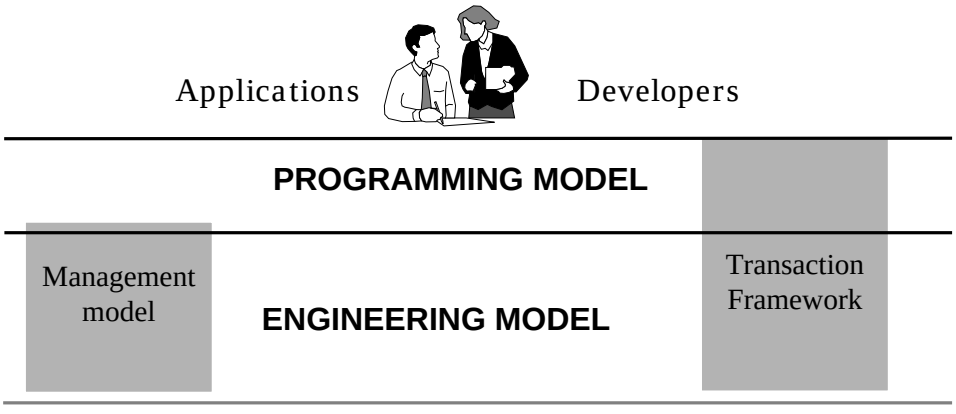
- A **failure model** which provides the underlying concepts;
- A **programming model** which provides programmers with abstractions for building dependable applications;
- An **engineering model** which provides a set of engineering mechanisms and sets of standard configurations of these mechanisms;
- An **extended 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.

Figure 2.1 shows the relationship between the programming model, the engineering model, the management model and the transaction framework. The engineering model provides a set of services to enhance the functionality provided by basic platforms for distribution such as DCE and CORBA. Application programmers build dependable applications using concepts provided by the programming model and services provided in the engineering model.

The system designer needs to elucidate the requirements of the application components for dependability (from the supporting engineering) and also the requirements which the engineering components impose on the behaviour of the application components. Designers need to be familiar with both the engineering and programming models to make the trade-offs between what (dependability) can be provided by the engineering and what can be provided by the application components.

The role of the engineering model is to provide a choice of services. It helps the system designer to choose the service which satisfy the application components requirements (for dependability). The role of the programming model is to ensure that the application components meet the requirements of

Figure 2.1: How the ANSA dependability work fits together



the engineering components. The role of the failure model is to provide the concepts for stating requirements.

The management model is concerned with such issues as the maintenance of dependability in the presence of faults (fault diagnosis and reconfiguration), how to install new applications and how to upgrade existing ones. The services required for management form part of the engineering model. In addition some concepts in the programming model may deal specifically with management issues.

The use of different kinds of transactions to build dependable systems is being studied. The services provided by the advance transaction framework forms part of the engineering model. The framework also provides abstractions to help programmers use these services. These abstractions are part of the programming model.

The remainder of this paper describes the above work in more detail, with the exception of the failure model which is described in §1.2.

2.1 The programming model

This section describes the requirements for the programming model. The role of the programming model is to provide the concepts to assist programmers in making sure that application components meet the expectations (requirements) of their supporting engineering components. Together with the engineering model, it enables system designers to make trade-offs between what is provided by the engineering components and what is provided by the application components. As an example suppose there is a choice of replication mechanisms including one that consumes fewer resources, but can only be used if the state of a service is immutable (does not change when the service is invoked). There are a variety of possible tools and techniques that can be used to exploit this opportunity.

1. A tool to analyse the application code automatically and report whether or not it has the immutability property.
2. Guidelines and rules for the programmer which explain how to write the code so that it has the appropriate characteristics.

3. A tool that transforms the code so that the most appropriate mechanism is used. This approach has already been investigated for an atomic activity infrastructure: code transformation was used to insert calls to appropriate lock mechanisms whenever mutable state was accessed [WARNE 94].

The scope of this technology will be set in part by the programming language which is used. Some languages are less amenable to transformation and automatic analysis than others.

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. Part of the intended work on the programming model will be to extend this technology by adding information about the expected behaviour of the interface. For example:

- Whether an operation updates or observes mutable state.
- How an operation affects the outside world: does it read information in the outside world (sensor) or does it make changes to the outside world (actuator).

If type checking tools can be enhanced to check these attributes, then at least some expectations can be checked automatically.

Programmers may choose to exploit application level redundancy — 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])
- The use of stability and self stabilisation [SCHNIEDER 93b]

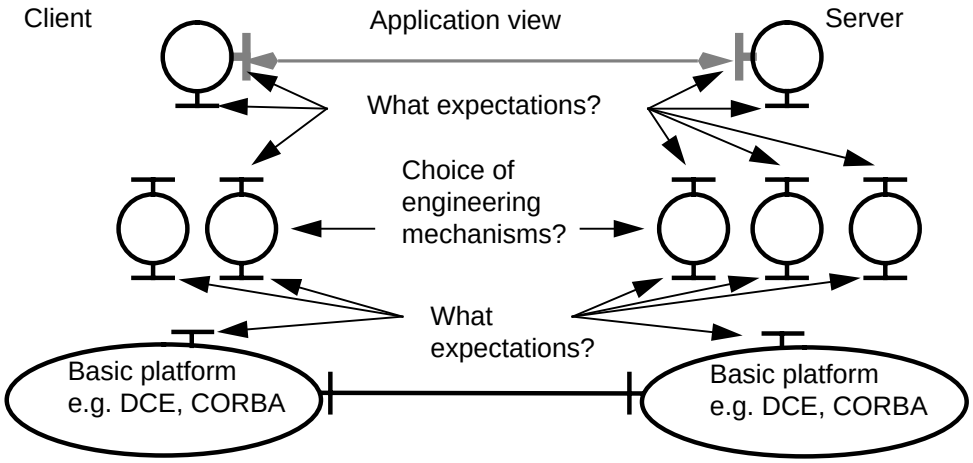
2.2 The engineering model

This section describes the requirements for the engineering model. The engineering mechanisms are positioned between the underlying platform and the application components as shown in figure 2.2. Failures can occur in the application components, the underlying basic platform or the engineering mechanisms themselves.

As shown in figure 2.2 the application components and engineering will have requirements (expectations of each other). The role of the engineering model is to help the designer select a configuration of engineering mechanisms which ensure that the expectations of the application components are satisfied even when failures occur. The designer also uses the programming model to makes trade-offs between what is provided by the engineering and what is provided by the application components.The engineering mechanisms enhance the functionality of the basic platform so that it meets the requirements for dependability. The engineering model also need to ensure that the expectations of the engineering mechanisms and underlying basic platform are matched and vice-versa.

The engineering model will consist of a set of basic services which are useful for building dependable systems, for example: reliable multicast, state transfer mechanisms, audit services, persistent storage services.

Figure 2.2: The relationship between the programming and engineering models



Each service will have a typed interface (just like any service which is visible to the application). This will allow type checking to be applied to engineering objects (as well as 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 allow dynamic configuration of engineering objects specifically to match client and server requirements, each new configuration will require checking.

The engineering model will prescribe standard configurations of mechanisms which will have well understood behaviours (and expectations). These configurations must themselves be dependable. Examples of such configurations could be:

- a configuration of mechanisms to ensure fail-stop behaviour;
- replication for a non-mutable service;
- replication for a mutable service.

Initially the engineering model is likely to consist of a set of mechanisms and a set of standard configurations for those mechanisms. However, structuring the mechanisms as services will allow recombination of these mechanisms in application specific ways. Application specific configuring of the engineering mechanisms 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. This suggests that eventually it will be necessary to provide tool support for configuration.

The opportunities to build completely new systems are becoming fewer and fewer. In general new applications and systems will have to interwork with what already exists. If a new application is to be dependable, the dependability of the existing services with which it interworks 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 application needs to be protected from them. The engineering model needs to provide mechanisms and configurations of mechanisms to do this.

Much of the engineering model is concerned with the provision of redundancy to give fault tolerance. 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). Examples include:

- Replication [OSKIEWICZ 93]
- Checkpointing [BIRRELL 87]

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

The engineering model will 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).

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.

2.3 The management model

This section describes the requirements for the management model. The part of the engineering model discussed in §2.2 is concerned mostly with the provision of mechanisms which provide redundancy. The management model includes the part of the engineering model which is concerned with how to manage this redundancy to tolerate faults, how to maintain dependability, how to install new applications and how to upgrade existing ones. For example consider an active replica group; managing the group involves providing mechanisms which can detect failures, reconfigure the group to remove the faulty member, and adding new members to maintain the level of dependability when existing ones fail. 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** is measuring value and time and comparing what is observed to what is expected.
3. **Fault diagnosis** is used if fault detection does not identify the faulty component. Fault diagnosis is discussed in §1.2.1.
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”.

Management will usually be embedded into the redundancy mechanisms which they manage. However, often it is necessary and convenient to have separate management and service interfaces for the redundancy mechanisms e.g. [OSKIEWICZ 93], even if the interfaces are onto the same object.

Just like all the engineering mechanisms, the management mechanisms themselves need to be dependable — system recovery mechanisms can be responsible for 35% of system failures [TOY 93].

2.4 The extended transactions framework (ETF)

This section describes the requirements for the extended transaction framework (ETF). Transactions exploit both fault avoidance and fault tolerance. For example computations enclosed within traditional transactions have well defined relationships with each other: they are constrained by the fundamental properties of atomicity, consistency, isolation, and durability (collectively known as the ACID properties [BERNSTEIN 87]). This helps to avoid faults which can be introduced by concurrent interfering computations.

Transactions use redundancy to undo their effects should they need to abort (fault tolerance). 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].

The traditional transaction model, with its strict ACID properties, is highly effective in some application areas such as conventional databases. However, it frequently found lacking in functionality, flexibility, and performance when used in other applications areas, especially those involving collaborative or long-lived activities. Such applications typically require some, but not all of the ACID properties. This has led to the development of many different kinds of transaction models, for example: Split Transactions [PU 88], Coloured Transactions [SHRIVASTAVA 90a] and Transaction Groups [SKARRA 89].

As observed in [CHRYSANTHIS 90], irrespective of how successful these extended transaction models are in supporting their intended application domains, they merely represent points within the spectrum of interactions possible within competitive and cooperative environments. Therefore, they each capture only a subset of the interactions to be found in any complex information system.

ETF is intended to address these concerns; it is intended to support a wide range of telecommunications and other business applications. Inevitably, different classes of applications will require different transaction models. These model will have: differing concurrency control methods; differing recovery procedures; differing resource placement, migration and replication

strategies; and differing timeliness (execution responsiveness) guarantees. ETF must enable such application diversity to interoperate effectively.

ETF identifies a number of new primitives for controlling the behaviour of different transaction models. The inspiration for these primitives stems from the abstract concepts of the ACTA meta-model [CHRYSANTHIS 92]. The transaction model is characterised by controlling four basic attributes of a transaction. ETF provides primitives to control these attributes (VPCR).

- **Visibility**: the degree with which members of the extended transaction are able to observe each others effects before the transaction as a whole terminates its execution and commits or aborts.
- **Correctness**: the acceptable effects on system state that members are permitted to produce.
- **Permanence**: the rules by which members are allowed to record their results in the stable state of the system.
- **Recoverability**: the capability of an extended transaction as a whole, or its members in part, in the event of failure, to recover and take the system to some state that is considered correct.

Hence ETF allows programmers and designers to construct transaction models which match the requirements of the application object (providing a service). It is intended to use the concept of transaction-based work flows (e.g. [DAYAL 93], [SHETH 93]) to describe how different application objects supporting different transactions models fit together. A workflow will consists of several related transactions which interwork to achieve a specific goal. Each transaction in a work flow may be different when characterised in terms of VPCR.

From figure 2.1, it can be seen that EFT spans both the programming and engineering models. It is intended to build engineering mechanisms for EFT which support the control of the VPCR attributes. Additional mechanisms will be needed to support the concept of workflows. The programming model will contain the concepts needed to drive the engineering mechanisms supporting EFT.

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 possible piece of future work is to analyse and compare the configurations of engineering mechanisms in the engineering model using techniques such as Markov Modeling [SIEWIOREK 92]. This will give relative measures of the dependability of different engineering options.

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: it is not sufficient to consider dependability purely in terms of protocols provided by the underlying engineering and platforms. 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]. Consequently dependability needs to be considered at the system design stage and throughout the development of the system.

Some techniques have a minimal effect on the structure of the system. For example, within ANSA, technology for transparent replication has been developed [OSKIEWICZ 93]. Provided the client or server satisfied certain (well understood) assumptions it is possible to hide replication from the application programmer, so that the decision to replicate or not can be made after the code has been written. This dramatically increases the complexity of the underlying engineering required to support the application components with consequent loss of performance (compared with non-replicated code). Hence, the scope and potential of technology for transparent dependability is limited.

Other techniques for dependability require more participation from the designer and programmer, but may result in applications which need less complicated engineering to support them and have better performance. For example, suppose a service needs to replicated. If a service can be designed so that it is immutable (does not change its state when invoked) the underlying support for replication can be made much simpler, since no mechanisms is required to coordinate state changes between the replicas (it never changes). To make a service immutable may require the designer to take special steps, for example, to ensure that any state concerning the interaction between client and server is held by the client. In addition the programmer of the service needs to avoid code which makes changes to state.

This paper describes technology which is being developed which allows designers and engineers to consider dependability issues, as well as technology which tries to make dependability transparent. In the former case the concept of selective transparency is important: hiding irrelevant detail from the programmer.

2.7 Summary

The ANSA work on dependability aims to provide the technology for building open dependable distributed systems. It is anticipated that such systems will be built using industry standards such as DCE and CORBA platforms.

A failure model has been developed and its use in the design of dependable systems is being investigated. One of the roles of the designer is to identify the requirements of the application components in terms of what each component expects from the underlying engineering. The notion of expectations in the failure model can be used for this. The engineering model then helps to identify a suitable configuration of mechanisms to meet these expectations. The engineering components enhance the functionality of the basic platform so that it can meet the application's requirements for dependability.

The designer also need to state what constraints the application components must meet, i.e. what the engineering expects of the application components. Again the notion of expectations can be used for this. The programming model helps the programmer to build application components which satisfy these constraints.

Taken together the programming and engineering model enable system designers to make trade-offs between what dependability is provided by the engineering mechanisms and what is provided by the application components.

Designers of dependable systems should take an end-to-end view: careful design and partitioning of functionality can reduce the need for complicated and sophisticated engineering mechanisms to support an application.

As part of the work on programming and engineering models, an extended transaction framework and management model are being developed.

2.8 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., and Paul Vickers of Hewlett-Packard, where 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.

[BERNSTEIN 87]

Bernstein, P.A., Hadzilacos, V., Goodman, N., "Concurrency Control and Recovery in Database Systems", Addison-Wesley Publishing Company Inc., 1989.

[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.

[CHRYSANTHIS 92]

Chrysanthis, P.K., Ramamritham, K., "ACTA: The Saga Continues", Database Transaction Models for Advanced Applications, Edited by Ahmed K. Elmargarmid, Morgan Kaufmann Publishers, 1992.

[CHRYSANTHIS 90]

Chrysanthis, P. K., Ramamritham, K., "ACTA: A Franework for Specifying and Reasoning about Transaction Structure and Behavior", Proceeding of the ACM SIGMOD International Conference on the Management of Data, 1990.

[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.

[DAYAL 93]

Dayal, U., Hsu, M., Ladin R., "Organizing Long-Running Activities and Triggers and Transactions", ACM SIGMOD Proceedings, 1990.

[EDWARDS 94a]

Edwards, N.J., Rees, R.T.O, "A Model for Failures in Dependable Systems", APM.1043, APM Ltd., Cambridge, U.K., 1994.

[EDWARDS 94b]

Edwards, N.J., "Open Dependable Distributed Systems", APM.1045, APM Ltd., Cambridge, U.K., 1994.

[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 93]

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

[PU 88]

Pu, C., Kaiser, G., Hutchinson, N., "Split Transactions for Open-Ended Activities", IEEE Proceedings of the 14th Conference on VLDB, 1988.

[RANDELL 75]

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

[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.

[SHETH 93]

Amit Sheth, Marek Rusinkiewicz, "On Transaction Workflows", Data Engineering Bullitin, June 1993.

[SHRIVASTAVA 90a]

Shrivastava, S.K. , Wheater, S.M., "Implementing Fault-Tolerant Distributed Applications Using Objects and Multi-Coloured Actions", n Proc. 10th International Conference on Distributed Systems, Paris, France, 28th May - June 1st, 1990.

[SHRIVASTAVA 90b]

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.

[SKARRA 89]

Andrea H. Skarra, “Concurrency control for cooperating transactions in an object-oriented database”, SIGPLAN Notices, 24(4), April 89.

[SMETHURST 93]

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

[TOY 93]

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

[WARNE 94]

Warne, J.P., “An Extensible Transaction Framework”, APM.1060, APM Ltd., Cambridge, U.K., 1994.