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

A Model for Failures in Dependable Systems

Nigel Edwards, Owen Rees

Abstract

This document describes a model for failures in dependable systems. A general failure model is described in terms of a system consisting of interacting components. This model is then applied to an object-based interaction model.

The model is based on events which occur with some value at some time. Components in the system observe events and have expectations which define regions in a value, time space. A failure is detected when what is observed does not match what is expected.

The concepts in the model can be used to analyse a given configuration of engineering mechanisms, application components and infrastructure to determine what failures can and cannot be tolerated by this configuration. This can then be mapped onto an application-level statement: what failures the applications can and cannot tolerate.

The intention is that the model should provide the underlying framework for further work on dependable distributed computing. Some familiarity with basic principles of object-based distributed computing is assumed.

APM.1027.00.07 **Draft** 26 November 1993

Request for Comments (confidential to ANSA consortium for 2 years)

Distribution:
Supersedes:
Superseded by:

Copyright © 1993 Architecture Projects Management Limited
The copyright is held on behalf of the sponsors for the time being of the ANSA Workprogramme.

Contents

1	**1**	**A Model for Failures in Dependable Systems**
2	1.1	Audience
2	1.2	Relationship to other work
3	1.3	System structure and activity
3	1.4	A model for failure
4	1.4.1	Expectations
5	1.4.2	Occurrences
5	1.4.3	Correctness
6	1.4.4	Failures
7	1.4.5	Consistency between multiple events
8	1.4.6	Understanding the value domain
8	1.5	Failures in object based interaction models
8	1.5.1	Events in an interaction
9	1.5.2	Constraints on expectations
10	1.5.3	A server’s expectations of clients
11	1.5.4	A client’s expectations of servers
11	1.5.5	The engineering components’ expectations of clients and servers
11	1.6	Crash, Byzantine and arbitrary failures
12	1.7	Applying the failure model
13	1.8	Summary and Conclusions
14	1.9	Acknowledgements

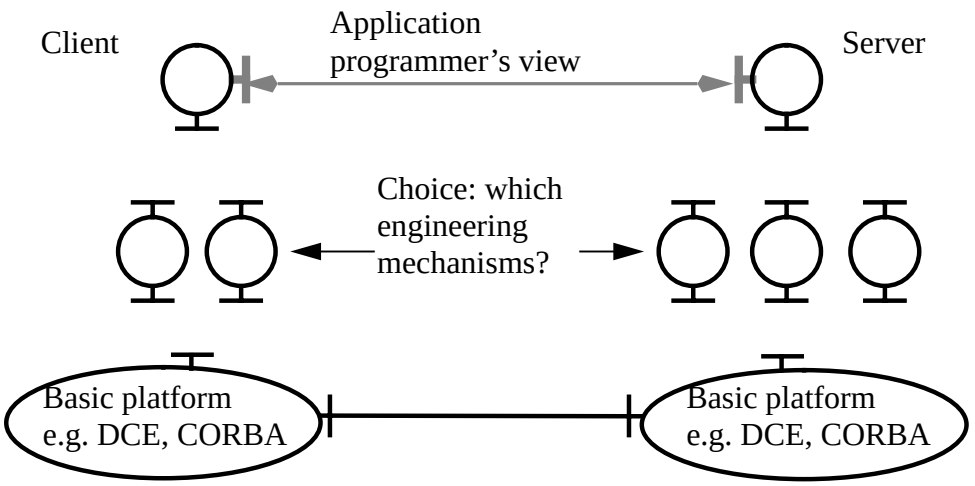
1 A Model for Failures in Dependable Systems

This paper describes a model for failures in dependable systems. The motivation for the model comes from the need for dependability in Open Distributed Systems [EDWARDS 93b].

From an application programmer's point of view such a system consists of a set of interacting application components (clients and servers). Sometimes these components will fail. The model describes such failures in terms of values and time. These are concepts which can be measured directly by components to detect failure.

From the infrastructure designer's point of view an open distributed system will consist of a set of interacting platforms (e.g. DCE, CORBA). On top of these platforms will be a set of engineering mechanisms which support a set of interacting application components. The purpose of the engineering mechanisms is to provide the application components with more guarantees (e.g. dependability) than are offered by the basic platform. This arrangement is shown in figure 1.1. Any mechanism, platform or application component may fail. The same modelling concepts can be used to describe all such failures.

Figure 1.1: An infrastructure designer's view of open distributed systems



The concepts in the model can be used to analyse a given configuration of engineering mechanisms, application components and infrastructure to determine what failures can and cannot be tolerated by this configuration. This can then be mapped onto an application-level statement: what failures the applications can and cannot tolerate. The methodology to do this is outlined in §1.7, a detailed example is given in [EDWARDS 93a]. One of the aims of the ANSA work on dependability is to develop a set of tools and abstractions which use this failure model to help application programmers match their

dependability requirements onto a specific configuration of engineering mechanisms and platforms (further details are given in [EDWARDS 93c]). In general each application will have a different set of requirements which will lead to different configurations.

The remainder of this paper is structured as follows. §1.1 describes the target audience and prerequisites for understanding the rest of this paper. §1.2 describes the relationship to other work, including other ANSA work. §1.3 states the assumed system model. §1.4 describes a model for failure in terms of abstract components. §1.5 uses the model to analyse the ODP and ANSA object-based interaction models. §1.6 looks at some of the failure modes often discussed in the literature and considers how they are related to the model. §1.7 outlines how the model can be applied to analyse a configuration of engineering mechanisms, infrastructure and application components. Finally §1.8 states some conclusions and summarises the main points of the paper.

1.1 Audience

Some familiarity with the basic principles of object-based distributed computing is assumed (e.g. ODP [ODP 93], CORBA [OMG 91] or ANSA [LINDEN 93a]). The audience is assumed to be those who are interested in how to characterise the failure of a system component and relate this to system dependability. This includes designers of dependability infrastructures and reference models for distributed dependability, as well as system implementors.

1.2 Relationship to other work

Modern computer systems are enormously complex; distribution introduces further complexity [LINDEN 93a]. 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 [LINDEN 93a] (ODP calls these viewpoints). Each of these projections describes a different aspect of a distributed system. The failure model presented in this paper can be used to relate failures in the engineering and computational projections.

The paper does not present a hierarchical failure mode classification (i.e. a partial ordering on failure modes); there are many in the literature (e.g. [BARBORAK 93], [SHRIVASTAVA 90], [CRISTIAN 90]). Such orderings are useful, because they say when one engineering mechanism can replace another. For example suppose we have an ordering $\subseteq$ on failure modes, and suppose we have 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. The failure model presented in this paper can be used to show that a particular engineering model does or does not implement a particular ordering.

Taken together with the basic concepts of dependability, such as availability and reliability (discussed in [EDWARDS 93c]) this failure model provides a means of describing dependability requirements. This is deliverable D1 in [LINDEN 93b].

Previous ANSA work on dependability has concentrated on groups [OSKIEWICZ 93a] and transactions [WARNE 92]. The model provides a framework within which to discuss these technologies, enabling us to understand what aspects of dependability they address and how they can be used to build dependable systems.

The intention is that the model should provide the underlying framework for further work on dependable distributed computing. This work is summarised in [EDWARDS 93b], which also investigates why dependability is important in open distributed computing. [EDWARDS 93c] gives a more detailed overview as well as describing the basic concepts for dependability in open distributed systems.

1.3 System structure and activity

It is assumed that a system is composed of components. These components can engage in events which can be observed by other components. For example a client makes an invocation which is later observed by a server when it receives this invocation (see §1.5.1). The structure of the system limits which components may observe which other components.

Each component makes its own observations and determines which events it considers to have occurred. An event is considered to occur with some value at some time, by the observer. The model does not require any notion of a global observer, a global ordering on events, nor does it require any notion of global time.

The model looks at only single occurrences of events. A similar assumption is made in the failures/divergence model of CSP [HOARE 85] which considers the ability of a process to refuse to engage in a single event, as opposed to sequences of events. Sequences of events can be built up from individual occurrences; §1.4.5 considers this issue.

1.4 A model for failure

Prompted by [SHRIVASTAVA 90] this section takes a set theoretical approach to modelling failure. Here the aim is to understand what constitutes a failure rather than to build up a hierarchical class of failures as in [SHRIVASTAVA 90].

The formal definition of a failure is given in §1.4.4. Informally, a failure occurs when the event which is observed does not match what is expected or when some event is expected and none is observed. An example of the former is a server receiving an invocation of an operation which it does not support. An example of the latter is a client failing to observe a reply after it has made an invocation. Not all events are failures, for example a server may observe that a client has invoked an operation which it supports.

This definition of failure does not assign fault to either the component which engages (or does not engage) in the event or the component which observes (or does not observe) the event. To resolve this issue, the parameters used to determine correctness must be made explicit [BARWISE 87]. This can be done by using a specification.

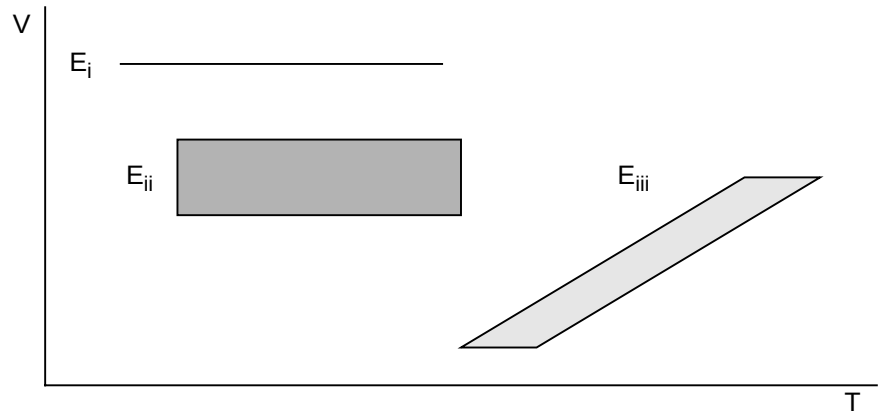
A specification sets expectations: it forms a contract between the component engaging in the event and the component which observes it. However, a specification may not be complete. In such a case a component may engage in an event which is allowed by the specification, but unexpected by the observer. Arbitration may be necessary to determine what to do about this and who is responsible for the failure. The notions of contracts between clients and servers and arbitration is being studied within the ANSA project.

§1.5 looks at other ways of setting expectations which do not involve specifications. This section considers expectations, occurrences, and the kind of mismatches that can arise. The various kinds of mismatch are the failure modes of the system.

1.4.1 Expectations

The event is usually expected to occur within some time interval, and with one of a restricted set of values. In the case of a server, the time interval within which it expects to receive an event may be very long (see §1.5.3). Figure 1.2 illustrates three expectation sets that will be used to illustrate the various failure modes.

Figure 1.2: 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.

All of these cases can be expressed as a set of (value, time) pairs at which the occurrence is expected. The expectation set is a subset of the set of all (value, time) pairs.

$$E \subseteq V \times T \tag{1}$$

This formulation can be used to express not only the cases illustrated above, but also many others, for example E could consist of several disjoint regions.

1.4.2 Occurrences

Occurrences of the event can also be considered as a set of value, time pairs:

$$O \subseteq V \times T \tag{2}$$

In the ANSA model, an individual event can occur at most once. Issues such as retransmission and replay can be dealt with by sets of distinct but related events, as discussed in §1.4.5.

This assumption means that the set O contains either zero or one element.

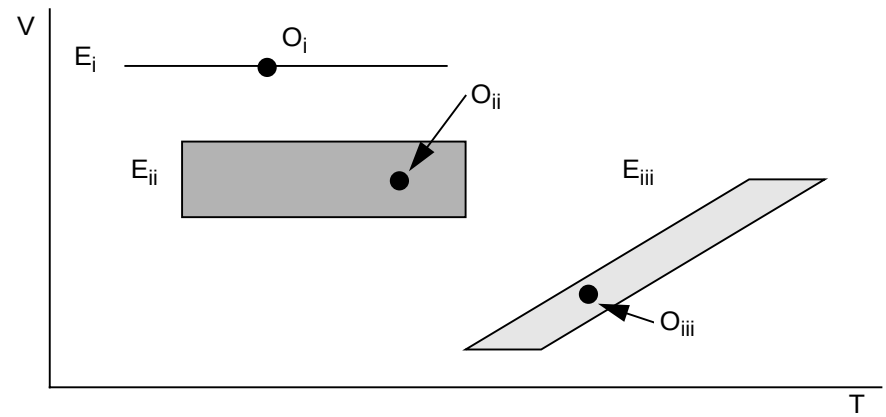
1.4.3 Correctness

The occurrence of an event is considered correct if what happens matches what is expected. Similarly if an event is not expected to occur then it is correct that it does not occur.

1.4.3.1 Correct occurrence of an event

An occurrence is correct if its value and time match one of the expected value, time pairs. Figure 1.3 adds an example of a correct occurrence for each of the three expectation sets of figure 1.2.

Figure 1.3: Correct Events



In terms of the expectation and occurrence sets, the correctness of the occurrence is given by the proposition

$$O \cap E \neq \varnothing \tag{3}$$

The informal definition at the beginning of this section is captured more directly by the equivalent proposition, which says that some event occurs and its value and time are expected

$$\exists v, t \bullet (v, t) \in O \wedge (v, t) \in E \tag{4}$$

1.4.3.2 Correct non-occurrence of an event

If an event is not expected to occur then it is correct that it does not occur.

It is important to consider this case since we wish to consider unexpected occurrences as failures. This means that we must consider the cases where an event is not expected to occur.

A correct non-occurrence satisfies the proposition

$$O \cup E = \varnothing \tag{5}$$

The equivalent proposition that captures the informal statement more directly is

$$\neg(\exists v, t \bullet (v, t) \in O) \wedge \neg(\exists v, t \bullet (v, t) \in E) \tag{6}$$

This says that the event does not occur and it is not expected.

1.4.3.3 *Formal definition of correctness*

Since an event occurs either zero times or once, the event is correct if it has either a correct occurrence or a correct non-occurrence.

Combining propositions (3) and (5) gives a proposition for a correct event

$$(O \cap E \neq \varnothing) \vee (O \cup E = \varnothing) \tag{7}$$

1.4.4 Failures

A failure occurs when what happens does not match what was expected. This could be because the observer is faulty (its expectations are wrong) or the event itself is not correct. It is usual to assume that the observer is correct and that a failure occurs when an event is not correct, but this is not fundamental to the model.

The proposition for failure is therefore the negation of (7), the proposition for correctness

$$\neg((O \cap E \neq \varnothing) \vee (O \cup E = \varnothing)) \tag{8}$$

This can be simplified to

$$(O \cup E \neq \varnothing) \wedge (O \cap E = \varnothing) \tag{9}$$

The failure modes which correspond to different mismatches of expectation and occurrence are described in the rest of this section.

1.4.4.1 *Unexpected occurrence*

If the event occurs and is not expected to occur, then this is a failure.

The proposition which captures this case is

$$O \neq \varnothing \wedge E = \varnothing \tag{10}$$

1.4.4.2 *Omission failure*

The event is expected to occur and does not occur.

$$E \neq \varnothing \wedge O = \varnothing \tag{11}$$

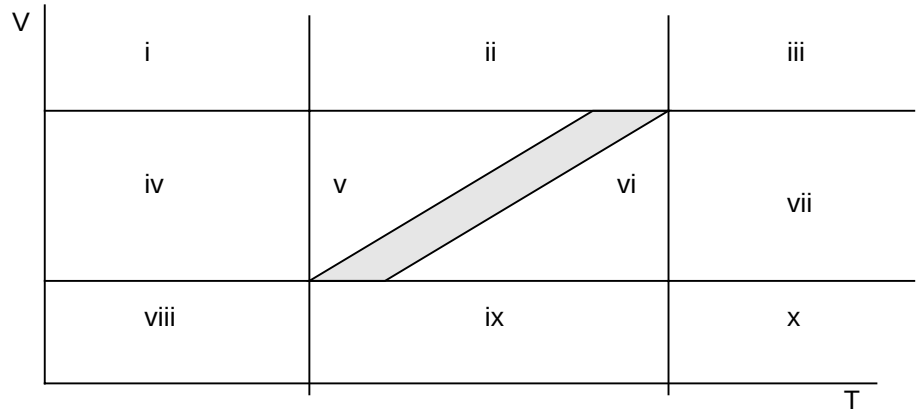
1.4.4.3 *Incorrect occurrence*

This leaves those cases when an event occurs, an event is expected, and the occurrence does not match the expectation.

$$O \neq \varnothing \wedge E \neq \varnothing \wedge (O \cap E = \varnothing) \tag{12}$$

Since both the expectation and occurrence sets are non-empty, the various cases can be illustrated graphically. Figure 1.5 shows a shaded region for the expectation and divides the space outside the expected region into ten areas.

Figure 1.4: Failure occurrence regions



The following regions are value failures: i, ii, iii, viii, ix and x.

The following regions are timing failures (early or late): i, iv, viii, iii, vii and x.

Regions v and vi are not covered in many failure mode classifications: they represent an occurrence of an event with a value and a time which are not expected together. In [SHRIVASTAVA 90], failures of this kind are considered in the discussion of clock faults, and they are classified as emission failures.

1.4.5 Consistency between multiple events

Consistency relations between events describe how the expectations of components vary over time.The simplest consistency relations are between events which are local to a single component. These are relations which constrain the region of expectation using the events which have so far been observed by the component and events in which the component has engaged. For example the relation may constrain the value of the next event expected using the sequence of events which have been observed so far. Hence an event which breaks the consistency relation will be an incorrect occurrence failure (or possibly an unexpected occurrence failure, if the expectation region collapses to zero). Such incorrect occurrences can be detected by mechanisms which are local to the component observing the event.

More complicated consistency relations may constrain the expectation regions using events which are observed by different components. For example the relation may require that the next events observed by a group of components each have the same value. Again events which break the consistency relation will be incorrect occurrence failures. However, these failures cannot be detected by mechanisms which are local to individual components. Detecting such failures is a distributed consensus problem, requiring collaboration between mechanisms at each of the observing components.

Often, rather than employing mechanisms which detect deviations from the consistency relation, mechanisms are used which enforce the consistency relation. For example [OSKIEWICZ 93b] describes a mechanism which runs a distributed consensus algorithm to ensure that all members of a group receive the same invocations and terminations.

Sometimes it is possible to express a global property as a set of local ones. This makes the job of enforcing the consistency relation or detecting deviations

easier, since the mechanisms which do this can be made local to individual components.

1.4.6 Understanding the value domain

This presentation of the model has only associated a single value with an event. This allows an event to be represented in two dimensional space. In a real system it may be convenient to regard an event as having many values associated with it: for example each parameter of an invocation could be regarded as being a separate value.

The collection of values can be treated as a single value for the purposes of the model described above. Alternatively, the components of the value can be considered separately, making it possible to have an event that is correct with respect to some components of the value, but a failure with respect to others.

Some components of the value may be more significant than others in terms of value failures. In the safety community, the severity of the potential mishap is sometimes referred to as “hazard criticality” [LEVESON 86], and failures in different components of the value may correspond to hazards with different criticalities. The idea that some parts of a message are more sensitive than others is well established in the security community. The OSI Security Architecture [ISO 89] defines “selective field protection” to support this requirement.

If an event has n values then the expectation set occupies a region in n+1 dimensional space, and figures 1.2, 1.3 and 1.5 can be used to represent a two dimensional cut through this space. An occurrence occupies a point in this space. The same idea is used to model the program input space in [BISHOP 93].

1.5 Failures in object based interaction models

This section applies the failure model to an object-based interaction model, specifically the ANSA and ODP interaction model. This analysis is also applicable to other object-based interaction models such as CORBA [OMG 91], and RPC interaction models such as that supported by DCE.

The failure model itself does not make any assumptions about local or remote interaction. Hence it can be applied to objects which are interacting remotely and to objects which are interacting within the same address space.

The components of an ANSA and an ODP system are objects which interact through interfaces. The interaction allowed is described in [REES 93a] and [ODP 93] and is shown in figure 1.5.

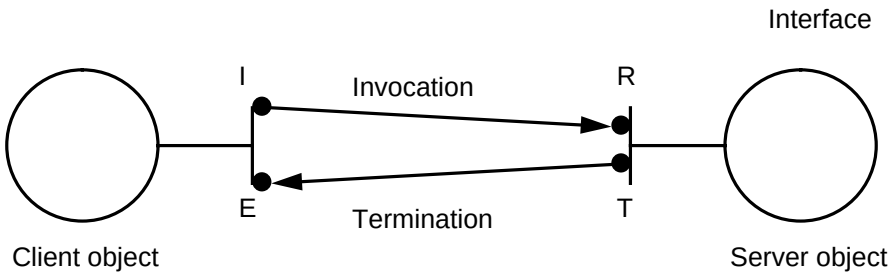
A client object interacts with a server object by invoking operations on interfaces provided by the server.

1.5.1 Events in an interaction

Four events that occur in an interaction are illustrated in figure 1.5. Of these, two are events in which components engage, and the other two are observations of events made by components.

The event labelled I is the event in which the client engages to initiate the invocation. The value for this event consists of an operation name and zero or more parameters. The client’s expectation will change when it engages in this

Figure 1.5: The interaction model



event; it will be expecting a response that corresponds to the requested invocation.

The event labelled R is the server's observation of the invocation request. This is the "request event" described in [REES 93b] that informs the server that it has been invoked. The value of this event is expected to be the same as the value of the event I.

The event T is the "terminate event" described in [REES 93b] that occurs when the server has completed the evaluation of the operation. Its value consists of a termination name followed by zero or more parameters. The particular name and parameters are expected to be the ones defined by the behaviour of the server given the value of the event R, the state of the server at the time of the event R, and any other events that may have been observed by the server.[1]

The event E is the client's observation of the termination. It is expected to have the same value as the event T.

The expectations above are that the engineering mechanisms (or objects) and platforms involved in the binding between client and server do not corrupt values. However, failures can occur in these objects. Identifying four separate events in the interaction model allows a link to be made between expectations and failures at the application-level (or in the computational model) and expectations and failures in the engineering. It would be harder to introduce this link if invocation and termination were regarded as atomic events, as these atomic events would need to be subdivided when studying the role of engineering objects in the interaction.

1.5.2 Constraints on expectations

An interface type is defined by a set of signatures. Each signature specifies:

- the name of an operation
- the number and types of argument parameters
- the termination names and result parameters

1. The "activate event" described in [REES 93b] has not been discussed. An elaboration on the behaviour of the server would be the right place to introduce it. There is scope for a discussion of how concurrency control can cause timing faults in its efforts to preserve consistency.

Both ODP and ANSA allow for types to specify more properties, e.g.: concurrency constraints and environmental constraints. DCE and CORBA have no such facilities.

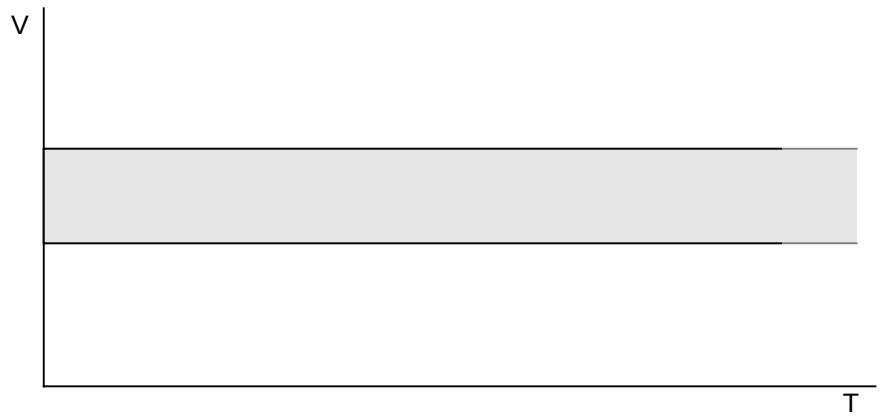
The interface type limits the expectations of both the client and the server. It may be thought of as part of a contract between the client and the server which sets bounds on the expectation region. The stronger the contract between the client and server, the smaller the expectation region.

The technology for stating timing constraints in ANSA and ODP like systems is not well understood. ANSA does not specify an means of constraining time within the interface type, although a proposal has been made in [STEFANI 93] for using Quality of Service to express timing constraints. ODP states that timing constraints are part of the environment contract in an interface type, but no technology is specified for stating these constraints. In the model presented here, expressing frequency or throughput failures can only be done by looking at sequences of occurrences and defining consistency relations over those occurrences.

1.5.3 A server's expectations of clients

A server offering a service through an interface must expect operation invocations to select any of the operations defined by the interface type. The expected value space is constrained by the operation name and the parameter types associated with that operation; figure 1.5 shows the two dimensional representation for this simple case.

Figure 1.6: A servers expectations



A server's expectations may be further constrained if the invocation is received from an active replicated client group [OSKIEWICZ 93a]: it expects the values and times of invocations from members of the replica group to agree.

A server may expect a client to invoke operations in a certain sequence and within certain time constraints.

Security constraints may restrict from whom the server is expecting to receive invocations. The observed event must include information that allows the server to check that the invocation is authorised. This may be an explicit part of the value (e.g. a Privilege Attribute Certificate as defined in [ECMA 89]). Alternatively, the server may use a previously negotiated key either to

decipher the value, or to check a cryptographic checkvalue. These techniques allow unauthorised invocations to be detected as value failures.

1.5.4 A client's expectations of servers

A client's expectation of a server is also constrained by the interface type: this constrains the termination a client expects when it invokes one of the set of operations supported by the interface[1]. Typically a client will expect some response within a given time of making the invocation. This simple situation corresponds to the sets shown in figure 1.2. A client does not expect a termination from a server until it has made the invocation, this is an example of a consistency relation between events which constrain expectations (§1.4.5).

A client's expectations will be constrained further if it invokes an active replica group (i.e. it expects the values and times of the terminations to agree) or if it has some semantic knowledge of the service (e.g. time should not run backwards).

The process of trading [ANSA 93], [ODP 93] will also constrain the client's expectation region. A client's expectation of the trading service is that it will get back an interface which matches its request. If it is unable to obtain something which satisfies its request, negotiation may take place which will affect the client's expectations of the service it subsequently decides to use.

1.5.5 The engineering components' expectations of clients and servers

In the engineering model, a platform provides services to computational objects (i.e. client and server objects) through typed interfaces [ODP 93], [OSKIEWICZ 93b]. Hence within the engineering model we can use exactly the same ideas of expectations as within the computational model.

For example most platforms have an implicit expectation that computational objects will transmit below a certain frequency, f. If they exceed this frequency of transmission they are regarded as flooding the network. Detecting this kind of failure requires the ability to look a sequences of occurrences. Although it could in principle be detected anywhere, it is probably be easier to take remedial action if it is detected as close as possible to the client or server object responsible.

Trading sets the expectations of clients and servers in the computational model: similarly binding sets expectations in the engineering model. When binding occurs it establishes the expectations of the engineering objects which support interaction between the client and server object. Binding may also be visible in the computational model. If it is, it will also set client and server expectations.

At federation boundaries interceptors [ANSA 93] may set and enforce expectations.

1.6 Crash, Byzantine and arbitrary failures

This section considers how some of the failure modes discussed in the literature relate to the model described in §1.4.

1. ODP takes the view that a binding occurs between client and server interfaces to capture the polarity and duality of the interface concept.

The model described in §1.4 says nothing about crash failures (or fail-silence). This is because a crash failure usually involves multiple non-occurrences. For example [CRISTIAN 90] states that a server is suffering a crash failure "if, after a first omission to respond, ... [it]... omits to respond to all subsequent inputs, until its restart". Hence a crash failure involves observing multiple non-occurrences, deciding that the component has suffered a crash failure and repairing it. It is something which cannot be demonstrated by testing.

The model does not describe Byzantine (arbitrary) failures. Such failures cannot be detected; they can only be masked through redundancy. The classic Byzantine failure is sending different values to two or more receivers (e.g. in a multicast), when consistency constraints demand that these should be the same value [DOLEV 87].

It is known that using authentication techniques substantially simplifies the handling of Byzantine failures [BARBORAK 93], [DOLEV 87]. The authentication techniques can range from simple checksums to complicated cryptographic techniques. These kinds of techniques operate in the value domain, making the value space as sparse as possible. Time domain techniques which look at sequences of occurrences have to be used to detect some other kinds of malicious behaviour, such as attempts to flood the network (§1.5.5).

1.7 Applying the failure model

Failure detection is the detection of deviations from what is expected. The failure model provides a way of describing expectations. Hence we can also describe deviations from expected behaviour. This allows us to state the behaviour of failure detection mechanisms and also the behaviour of mechanisms which enforce expectations.

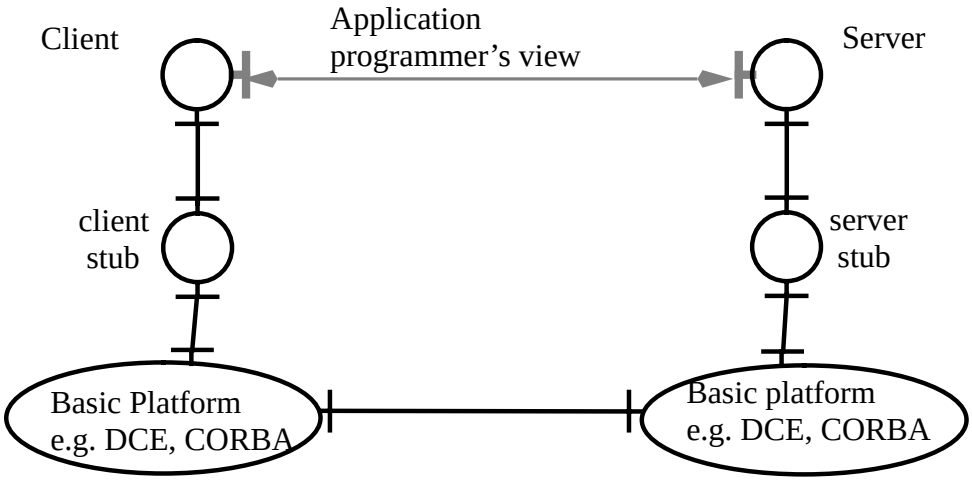
As a very simple illustration of how the model can be used consider figure 1.1. This shows the objects supporting a client and server interacting with each other. Suppose the client invokes the server. After receiving the invocation the server's node crashes. This means that the server, its stub and platform will not send any more messages. Now the client's platform is expecting to receive a message from the server's platform within a certain time in response to its invocation. The client's stub is expecting to receive a response from its platform and the client is expecting to receive a response from its stub.

After some fixed time, the client's platform times-out on the server's platform and returns a response indicating this to the client stub which in turn returns it to the client. Hence the failure is detected (although not tolerated unless the client has a mechanism for dealing with it).

[EDWARDS 93a] uses the failure model presented here to analyse the implementation of active replica groups described in [OSKIEWICZ 93a]. This involves analysing a set of engineering objects which implemented active replication over a basic platform supporting RPC interaction. The following is a summary of the method used, readers requiring further details are referred to [EDWARDS 93a].

1. The expectations of engineering objects and platform involved in the binding between a client group and a server group are stated.
2. The expectations of the application objects are derived from 1 (i.e. the expectations which the clients and servers involved in the interaction have of each other).

Figure 1.7: Infrastructure designers view of client/server interaction



3. Each failure mode of a server object (listed in §1.4) is considered in turn to see if the engineering objects or platform can detect and tolerate this failure. If the failure cannot be tolerated, the expectations of the client object will not be met.
4. A similar analysis is conducted for each client object

This analysis results in a statement within the model of the failures which can be tolerated and those which cannot be tolerated: something which could not be done in [OSKIEWICZ 93a] because the technology to perform this analysis was not available. The analysis also gives considerable insight into how the implementation might be improved to tolerate more kinds of failures.

The notion of a contract between components is crucial to setting expectations. Present technology allows only very weak contracts (e.g. interface definition languages only define signatures). Stronger contracts will allow better failure detection mechanisms to be built. Currently, active replication is one of the best forms of failure detection: the members of active replica groups have an implicit contract to agree (of course, they may all be wrong). However, active replication is not appropriate for all applications.

The model described does not explicitly cover events which report detected failures. In one sense these are correct since they are anticipated. In another sense they are failures since usually it is expected that the system will operate correctly, so events reporting failures are unexpected. In applying the model experience has shown that these events should be treated as unexpected events (and therefore failures) [EDWARDS 93a].

1.8 Summary and Conclusions

This document describes an abstract failure model in terms of a system consisting of interacting components. This model is then applied to an object based interaction model.

The model can be used to analyse a given configuration of engineering mechanisms, application components and infrastructure to determine what failures can and cannot be tolerated by this configuration. From this a

statement is generated, within the model, of the failures which can and cannot be tolerated by the configuration [EDWARDS 93a].

More work is needed to investigate the nature of contracts between clients and servers which fix expectations. The stronger these contracts the more powerful the failure detection mechanisms which can be used.

The model has been successfully applied to previous work on active replica group mechanisms [EDWARDS 93a]. More experience of applying the model is needed to identify improvements either in the model itself, or in its use.

1.9 Acknowledgements

The authors acknowledge the valuable contributions made by Ed Oskiewicz, seconded to the ANSA Team by BT, and John Warne, seconded to the ANSA Team by BNR-Europe

The authors are grateful to Santosh Shrivastava of the University of Newcastle upon Tyne: his comments on an earlier versions of this document enabled substantial improvements to be made. The authors are also grateful to Brian Coan of Bellcore, Paul Harry of Hewlett-Packard and Andrew Herbert of APM Ltd. for their comments.

References

[ANSA 93]

"The ANSA model for Trading and Federation", AR.005.00, February 1993, APM Ltd., Cambridge

[BARBORAK 93]

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

[BARWISE 87]

Barwise, J., Etchemendy, J., "The Liar — an Essay on Truth and Circularity", Oxford University Press, 1987.

[BISHOP 93]

Bishop, P.G., The Variation of Software Survival Time for Different Operational Input Profiles, in FTCS 23, the 23rd International Symposium on Fault-Tolerant Computing, Toulouse, France, June 1993, pp98-107.

[OMG 91]

"The Common Object Request Broker: Architecture and Specification", OMG Document number 91.12.1, Revision 1.1, 1991.

[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.

[DOLEV 87]

Dolev, D., Lamport, L., Pease, M., Shostak, R., "The Byzantine Generals", in Concurrency Control and Reliability in Distributed Systems, van Nostrand Reinhold, 1987, Bhargava, B.K., (Ed.), pp348-369.

[ECMA 89]

"Standard ECMA-138, Security in Open Systems, Data Elements and Service Definitions", December 1989, ECMA, Geneva.

[EDWARDS 93a]

Edwards, N.J., "Applying the ANSA failure model to Active Replica Groups", APM.1046, September 1993, APM Ltd., Cambridge, U.K.

[EDWARDS 93b]

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

[EDWARDS 93c]

Edwards, N.J., "Building Dependable Distributed Systems", APM.1062, November, 1993, APM Ltd., Cambridge, U.K.

[HOARE 85]

Hoare, C.A.R , “Communicating Sequential Processes”, Prentice Hall International, 1985

[IGGULDEN 93]

Iggulden, D., Architecture and Frameworks, TR.038.00, February 1993, APM Ltd., Cambridge U.K.

[ISO 89]

Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture”, ISO 7498-2 : 1989 (E)

[LEVESON 86]

Leveson, N. G., “Software Safety: Why, What, and How”, ACM Computing Surveys, June 1986.

[LINDEN 93a]

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

[LINDEN 93b]

van der Linden, R., “Operational Plan: 1/3/93 - 28/2/95”, APM.1031, Cambridge, U.K., June 1993.

[OSKIEWICZ 93a]

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

[OSKIEWICZ 93b]

Oskiewicz, E., Edwards, N.J., “ANSAware 4.1 support for interface groups”, Tr.35, APM Ltd., Cambridge U.K., February 1993.

[OSKIEWICZ 93c]

Oskiewicz, E., “An Engineering Model for Dependability”, APM.1044, APM Ltd., in preparation.

[ODP 93]

Basic Reference model of Open Distributed Processing - Part 3: Prescriptive Model, Secretariat ISO/IEC JTC1/SC21, American National Standards Institute, June 1993.

[REES 93a]

Rees, R.T.O. “The ANSA Computational Model”, AR.001.01, January 1993, APM Ltd., Cambridge, U.K.

[REES 93b]

Rees, R.T.O., “Using path expressions as concurrency guards”, TR.022.00, January 1993, APM Ltd., Cambridge, U.K.

[STEFANI 93]

Stefani, J.B., “Some Computational Aspects of QoS in ANSA”, CNET/ RC.W03.JBS.001, January 1993, (Available from APM Ltd., Cambridge, U.K.).

[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.

[WARNE 92]

Warne, J.P., Rees, R.T.O, “ANSA Atomic Activity Model and Infrastructure”, AR.004, February 1992, APM Ltd., Cambridge U.K.