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

Generic Hazards for IPS

Nigel Edwards, Ed Oskiewicz

Abstract

IPS (Information Publishing System) is being studied to help understand how to build open dependable distributed systems.

A hazard is something which could cause the failure of the system. One of the problems which needs to be solved is to identify which hazards are important (they have an unacceptably high risk of occurring) and which can be ignored (because the risk of the hazard occurring is acceptably low). Then the engineering and design of the system can be focused on tolerating or avoiding the important hazards. This involves understanding and making explicit many underlying assumptions about the system.

Because the hazards are generic, the paper points to generic solutions. These solutions will be explore in more detail in future papers. The main contribution of the paper is: an increased understanding of what the potential hazards are; a statement of how these hazards could be manifested as failures; and a brief discussion of technology which can be used to tolerate these failures. A number of underlying assumptions are also made explicit. Although these statements are made in the context of IPS, they are likely to relevant to many open dependable distributed systems. Ultimately the aim is to produce a checklist of hazards which need to be considered when building such systems.

APM.1182.00.04 **Draft** 7 April 1994

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

Distribution:
Supersedes:
Superseded by:

1 Generic Hazards for IPS

1.1 Overview

IPS (Information Publishing System) is being studied to help understand how to build open dependable distributed systems.

A hazard is something which could cause the failure of the system. Any fault is a hazard; hazards also include situations which may arise in the normal operation of a system which would not necessarily be considered a fault (e.g. extreme weather). However, in this paper all the hazards identified are faults.

One of the problems which needs to be solved is to identify those hazards which are important (they have an unacceptably high risk of occurring) and those which can be ignored (because the risk of the hazard occurring is acceptably low). Then the engineering and design of the system can be focused on tolerating or avoiding the important hazards. This involves understanding and making explicit many underlying assumptions about the system.

This paper explores the generic hazards which need to be considered and what can be done to make sure that they do not cause the failure of the system. Generic hazards exist independently of the semantics or design of IPS. Other hazards will arise because of the semantics of the application and will be explore in other papers.

Because the hazards are generic, the paper points to generic solutions. These solutions will be explore in more detail in future papers. The main contribution of the paper is:

- An increased understanding of what the potential hazards are
- A statement of how these hazards could be manifested as failures
- A brief discussion of the technology that can be used to prevent these hazards causing failure of IPS
- A number of underlying assumptions are made explicit

Although none of the generic hazards which are identified are ignored because they constitute an acceptably low risk, a number of assumptions are stated. These assumptions can be regarded as an implicit statement of ignored hazards: the hazards arising if these assumptions were reversed are ignored.

Although the paper is written in the context of IPS, much of it is likely to be relevant to many open dependable distributed systems. Ultimately the aim is to produce a checklist of hazards which need to be considered when building such systems.

The remainder of this paper is structured as follows: §1.2 states the generic hazards and §1.3 states those hazards not addressed; §1.4 states the system model and discusses some assumptions about the infrastructure supporting IPS; §1.5 gives a brief overview of the ANSA failure model; §1.6 examines each hazard in turn and discusses how it could lead to faults propagating through IPS; §1.7 discusses failure detection.

1.2 Generic hazards to an object in IPS

Note: We would welcome additions to the lists in §1.2 and §1.3. Please tell us if there is something you think we have missed.

Consider some application object, X in IPS. The following could cause X to fail (what precisely is meant by a failure is discussed in §1.5 and §1.6).

- Software bugs in X or in an object in the same capsule (unit of encapsulation) as X
- Software bugs in objects which X invokes directly or indirectly
- Failure of the underlying nucleus or node (local operating system or host, including host crashing)
- Failure of communications
- Failure of remote nucleus or node (e.g. powercuts, software bugs)
- Failure of remote object: (bad response, no response, not recognising arguments, bad invocation, bad arguments)
- Remote object not meeting QoS requirements
- Faulty installation or reconfiguration of X (this could leave orphan references)
- Attempts to flood the network (a remote object or its infrastructure/ platform flooding the network could starve X)
- Breach of trust by remote object (e.g revealing secrets, abusing credit card, over charging, attempts at free access)
- Attempts to corrupt information held by X (e.g. somebody tries to corrupt the host operating system)

1.3 Generic hazards which are not addressed in IPS

The following security hazards could also cause X to fail. These are not addressed in IPS, although some solutions to these problems are discussed in [ANSA 93].

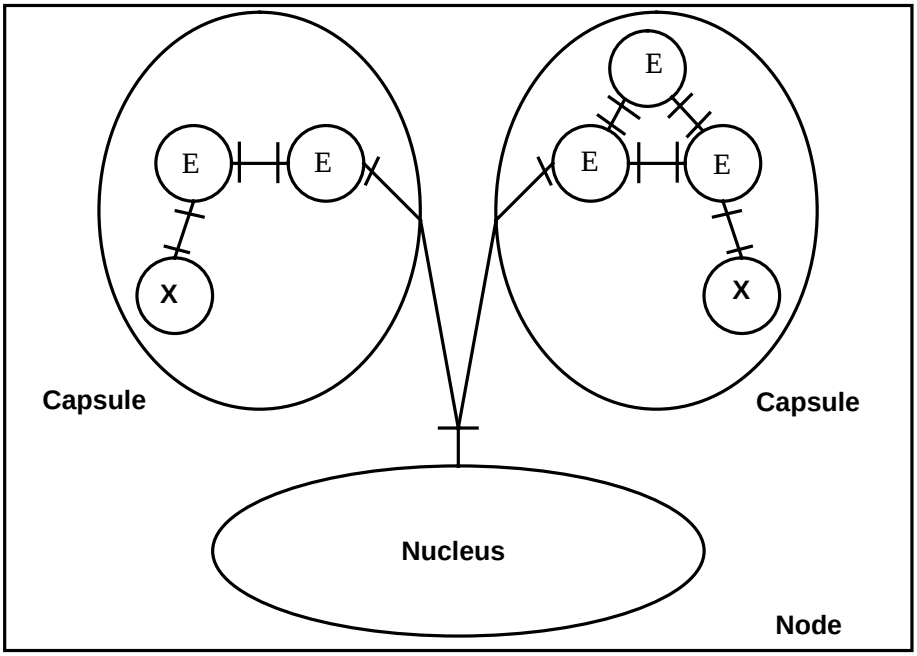
- Bogus (illegal) remote objects (e.g. an object masquerading as a bona fide object)
- Remote object not recognising X (and therefore refusing to talk to it)

1.4 Assumptions about the infrastructure

The system model is based on that of ODP [ODP 93]. The system consists of a number of nodes which can communicate using a network. Each node contains one nucleus object and a number of capsules. Each capsule consists of a number of objects. An object X may interact directly with other objects in the same capsule; interaction with remote objects (in different capsules) only takes place via the nucleus. In general X's interaction with the nucleus will take place via engineering objects: these are objects which enhance the functionality of the nucleus (in this paper) collaborating with X to tolerate and mask failures.[1] A single node is shown in figure 1.1.

The following assumptions concern the nucleus, the node and network, collectively called the infrastructure. These assumptions should not be taken

Figure 1.1: A node in IPS



E: Engineering Object
X: Application Object

for granted; it is important to make sure that the behaviour of the infrastructure does not invalidate them. The consequences of the assumptions being reversed can be thought of as ignored hazards.

Assume that the infrastructure will suppresses messages which it does not understand. It enforces type safe interaction: an object in a capsule will only get a message which it understands (i.e it is allowed by the interface definition). Further there are no unsolicited messages: replies are only sent in response to invocations.

Assume that the infrastructure suppresses late messages. So an object in a capsule either receives a message within some bounded time, or it never receives it.

Note: Do we need to add any assumptions about the engineering objects here?

Two sets of communication assumptions are set out in the following sections.

1.4.1 Strong communication (synchronous) assumptions

Assume bounded communication delays. It is hoped that modern networking technology like ATM will be able to provide statistically bounded communication delay. The work of the ANSA performance group will show how to build a binding between objects with bounded communication delays and message delivery (in particular timed RPC) [WAI 94]. The precise bounds on communication delay is something which cannot be determined until the binding is made; calculating it will require knowledge of the binding structure.

1. Other engineering objects may perform different functions. For example they may provide location transparency. Such functions are not the concern of this paper.

Assume no partitions. This means the network needs to offer multiple routes between objects. This could be implemented by redundant networks (e.g. dual LANS) or multiple point to point links with intelligent routing.

Assume that there are two levels of service. One in which a single message is sent out; this can fail. A second in which the message is sent by all known routes, duplicates are suppressed, but the message is guaranteed to arrive with some known bounds (greater than the first method). This is very similar to the assumption made by Cristian for his FAA work [CRISTIAN 91].

These assumptions do not preclude an object (e.g. the browser) disconnecting itself from the network. It just means that this should be regarded as a failure of the object, rather than a failure of communications.

1.4.2 Weak communication (asynchronous) assumptions

Assume the network has unbounded communication delays and may be subject to partitions. Assume that the infrastructure ignores late messages (turning them into omission failures).

1.4.3 The weak versus the strong assumption in IPS

The advantage of the strong assumption is that simpler membership service protocols can be built (such as one of those described in [CRISTIAN 91]). The disadvantage is that it is non-trivial to realise such assumptions. The advantage of the weak assumption is that most networks have this or stronger characteristics. The disadvantage is that more complex membership protocols are needed e.g. [MACEDO 94], because it is harder to reach agreement with these weaker guarantees.[1]

It is anticipated that services in IPS will be replicated to achieve high availability. However, the replica groups will have different consistency constraints and hence will use different consistency protocols. A general purpose server group membership protocol is needed which makes no assumptions about how the members of the group are related.

We choose to make the strong communication assumption. This is because IPS could be deployed over a modern telecommunication network and such networks are evolving to the point where it is becoming possible to give the strong communication guarantees described in §1.4.1.

However, such communication networks do occasionally fail [COAN 94]. Thus IPS should be designed to be fail safe in this respect. Hence it would seem prudent to know exactly where the assumptions are used (e.g. only the membership service) and to design the system to limit damage when those assumptions become invalid.

One possible option is to allow the infrastructure to report to an object when the network is failing to meet its communication guarantees. The question of what such an object should do requires careful thought — it is easy to get into a situation where the entire system shuts itself down. For example, if the failure is because the object is overloaded, then killing it could overload other

1. Fischer Lynch and Paterson [FISCHER 85] have shown it is impossible to reach consensus in an asynchronous system with crash failures, because it is impossible to distinguish a failed process from a slow one. This makes building a membership protocol extremely hard and complex. Making this distinction is not a problem in synchronous systems, so building a membership protocol is much easier.

objects as demand is switched to alternative services. If the failure is due to congestion in the network, then requiring the object to kill itself may only increase the congestion in other parts of the network, endangering other objects.

1.5 A brief reminder of the failure model

This section informally describes the ANSA failure model which is used in the remainder of this paper. For a more detailed description of the failure model see [EDWARDS 94].

1.5.1 Omission failure

An event (invocation/reply) is expected, none is observed.

1.5.2 Incorrect Occurrence

An event is expected (invocation/reply), but the occurrence does not match what is expected. This is because the value is wrong or the time is wrong (value could be wrong argument, or parameter, or invocation of the wrong operation). As discussed in [EDWARDS 94] such failures include both timing and value failures.

1.5.3 Unexpected Occurrence

No event is expected (invocation/reply), but one occurs. Server's always expect an invocation, so an unexpected occurrence means replies sent when no invocation was made; it is assumed that these are suppressed by the infrastructure.

Hence in this paper, which is only concerned with generic hazards, unexpected occurrences are not an issue. Section 1.7 discusses how application semantics may mean there are times when servers are not expecting invocations, in which case unexpected occurrence could arise.

1.5.4 Consistency failures

The ANSA failure model does not have an explicit notion of consistency failures (including Byzantine failures). In general, in any system, there will be consistency relations constraining sets of events. Any occurrence of an event violating such a consistency relation will be an "incorrect occurrence". This is because such failures can be detected (or masked) only by mechanisms which manipulate their value or time.

It is only possible to discuss the details of many consistency relations in the context of a detailed analysis of a design (e.g. once the need for a particular replication protocol is identified, and the application semantics is understood). Such an analysis is beyond the scope of this paper which considers only generic hazards.

However, one of the generic hazards does define a consistency relation: the constraint that the network should not be flooded by a remote object or infrastructure can be expressed only in terms of (the times of occurrences of) multiple events.

1.6 Fault propagation

This section discusses how the hazards identified in §1.2 can cause failures which need to be tolerated by IPS, if the faults are not to be propagated. Each hazard is considered separately and the nature of the failure it can cause is considered. The following assumptions are made about the nature of software bugs and QoS.

- Assume software bugs in an application object X, or in an object in the same capsule as X, either have no effect on X or else cause X to fail in which case it is a hazard to the rest of the system.
- Assume software bugs in objects which X invokes directly or indirectly cause those objects to fail or else have no effect on those objects.
- Assume remote object not meeting QoS requirements is a failure of the remote object (omission failure or timing failure).

1.6.1 Software Bugs in an application object or its capsule

Software bugs in an application object, X, or in objects in the same capsule as X, cause the failure of X: either omission or incorrect occurrence failure; the infrastructure suppresses unexpected occurrences. Nothing can be done for X, but remote users of X need to be protected (see discussion of how failures of remote objects can affect X in §1.6.2 and §1.6.6).

1.6.2 Software Bugs in remote objects invoked by an application object

Software bugs in remote objects which are invoked by an application object X will either have no effect or else cause the remote object to suffer an omission failure or an incorrect occurence failure. The nucleus will suppress late messages, so the only failures which need to be tolerated, or masked by X and its engineering objects are omission failures, value failures and early messages (because the object is sending messages too frequently).

1.6.3 Failure of an application object's infrastructure

This will cause the failure of X; nothing can be done for X, but users of X need to be protected (see discussion of how failures of remote infrastructure can affect X in §1.6.5).

1.6.4 Failure of communications (synchronous guarantees)

Messages are either delivered to a capsule within the bounds of communication delay or lost (omission failure). A delivered message may have the wrong value (be corrupt). Hence X and its engineering objects need to detect or mask omission and value failures.

1.6.5 Failure of remote infrastructure

It will appear to X and its engineering objects as though the remote objects on the infrastructure are suffering omission failures or incorrect occurrence failures; unexpected occurrences are suppressed by X's nucleus.

The assumption about message delivery mean that late messages will be suppressed, so the only incorrect occurrences are value failures or early messages (because the remote infrastructure is sending messages too frequently, flooding the network).

If there is unacceptably high probability of remote infrastructures flooding the network or sending messages which would be difficult to transform into the failure modes discussed, then a strategy such as that used in [SHRIVASTAVA 92] could be employed to implement fail silent nodes.

Alternatively the infrastructure could negotiate with the network for resources. Thereafter the network could throttle the infrastructure to prevent it consuming more resources than allocated in the negotiation. This approach is used to control congestion in networks [COAN 94].

1.6.6 Failure of remote object

Failures of remote objects which are not invoked by X (directly or indirectly), may effect X. This is because these objects may be trying to invoke X or in failing send messages to X. In addition an object which X invokes directly or indirectly may fail.

Many kinds of failures will be transformed into omission failures by the infrastructure, others will appear to X or its engineering objects as incorrect occurrence failures: wrong value or too early (because the object is sending messages too frequently, flooding the network).

Certain kinds of software bugs may cause the object to try to flood the network (e.g. it enters an infinite loop) with apparently correct messages. This is a similar problem to the infrastructure flooding the network; similar solutions can be used. The object's infrastructure could require the object to negotiate with it for (communication) resources and thereafter refuse to allow it to exceed its resource limit. An object would be prevented acquiring sufficient resources to flood the network.

1.6.7 Remote object not meeting QoS requirements

See failure of remote object in §1.6.2.

1.6.8 Faulty installation or reconfiguration of an application object

This will cause interface references (to or from X) to be faulty. Assume this will appear as an omission failure to object's trying to access X or object's trying to traverse links from X. Replication is not going to help here, installation or reconfiguration needs to be automated to ensure it does not happen!

1.6.9 Attempts to flood the network

See discussion of failure of remote objects (§1.6.6) and infrastructures (§1.6.5).

1.6.10 Breach of trust by remote object

There are federation issues involving notions of contract which need to be solved to define trust adequately. Some breaches of trust may be manifested as incorrect occurrence failures within the system (e.g over charging, attempts at free access). We know of few solutions to this problem; having audit processes to check that values like charges are within bounds is one defence against such failures. Other breaches of trust may involve events in the system's environment: revealing secrets, abusing credit cards.

1.6.11 Attempts to corrupt information held by an application object

If the information held by X is corrupted, X may eventually suffer an incorrect occurrence or omission failure, remote users would then need to be protected against this. Auditing (attempting to detect and correct faulty data structures), checkpointing and replication are some of the techniques which can be used to tolerate such faults.

1.7 Failure detection

In terms of the failure model, whether or not something is a failure is determined by an object's expectations. This paper considers only generic expectations that any system of interacting client and servers might have.

The application will further constrain expectations, fixing expected values for arguments and results, and ordering of invocations. Further there will be consistency constraints relating events in the system: (e.g. access of information is accompanied by charging). These will be manifested as omission failures and incorrect occurrence failures which can be detected by application components. Some of these kinds of failures could be detected by having the infrastructure compare outputs of replicated components.

The lifecycle of components in IPS may mean that there are times when a server is not expecting to receive an invocation: for example before it has been installed or after it has been decommissioned. If an invocation were to arrive at such a time it would be an unexpected occurrence failure. Whether or not this is a danger, will be known only once we have refined the design of IPS further and considered how servers are installed and decommissioned.

Note: Should this last paragraph be added to the list of generic hazards?

1.8 Acknowledgement

Discussions with Andrew Herbert, of APM Ltd., were extremely helpful.

References

[ANSA 93]

"A Framework for Federating Secure Systems", AR.008.00, APM Ltd., Cambridge, U.K., May 1993

[COAN 94]

Coan, B.A., Heyman, D., "Reliable Software and Communication III: Congestion Control and Network Reliability", IEEE Journal on Selected Areas in Communication, 12(1), January 1994, pp40-45.

[CRISTIAN 91]

Cristian, F., "Reaching Agreement on Processor Group Membership in Synchronous Distributed Systems", Distributed Computing, 4(4), April 1991, pp175-187.

[EDWARDS 94]

Edwards, N.J., Rees, R.T.O., "A Model for Failure in Dependable Systems", APM1143, APM Ltd., Cambridge, U.K, February 1994.

[FISCHER 85]

Fischer, M.J., Lynch, N.A., Paterson, M.S. "Impossibility of Distributed Consensus with One Faulty Process", Journal of the ACM, 32(2), pp374-382, April 1985.

[MACEDO 94]

Macêdo, R.A., Ezhilchelvan, P.D., Shrivastava, S.K., "Newtop: A Fault-Tolerant Total Order Multicast Protocol Using Causal Blocks", Department of Computer Science, Unversity of Newcastle upon Tyne, 1994.

[ODP 93]

ISO/IEC JTC/SC21 "Draft Recommendation X.903: Basic Reference Model of Opend Distributed Processing - Part 3: Prescriptive Model", June 1993.

[SHRIVASTAVA 92]

Shrivastava, S.K., Ezhilchelvan, P.D., Speirs, N.A., Tao, S., Tully, A., "Principal Features of the Voltan Family of Reliable Node Architectures for Distributed Systems", IEEE Transactions on Computers, 41(5), May 1992, pp542-549,

[WAI 94]

Wai, F., Otway, D., Howarth, N., Herbert, A., "A Performance Framework", APM.1137, APM Ltd., Cambridge, U.K., March 1994.