1961

1981

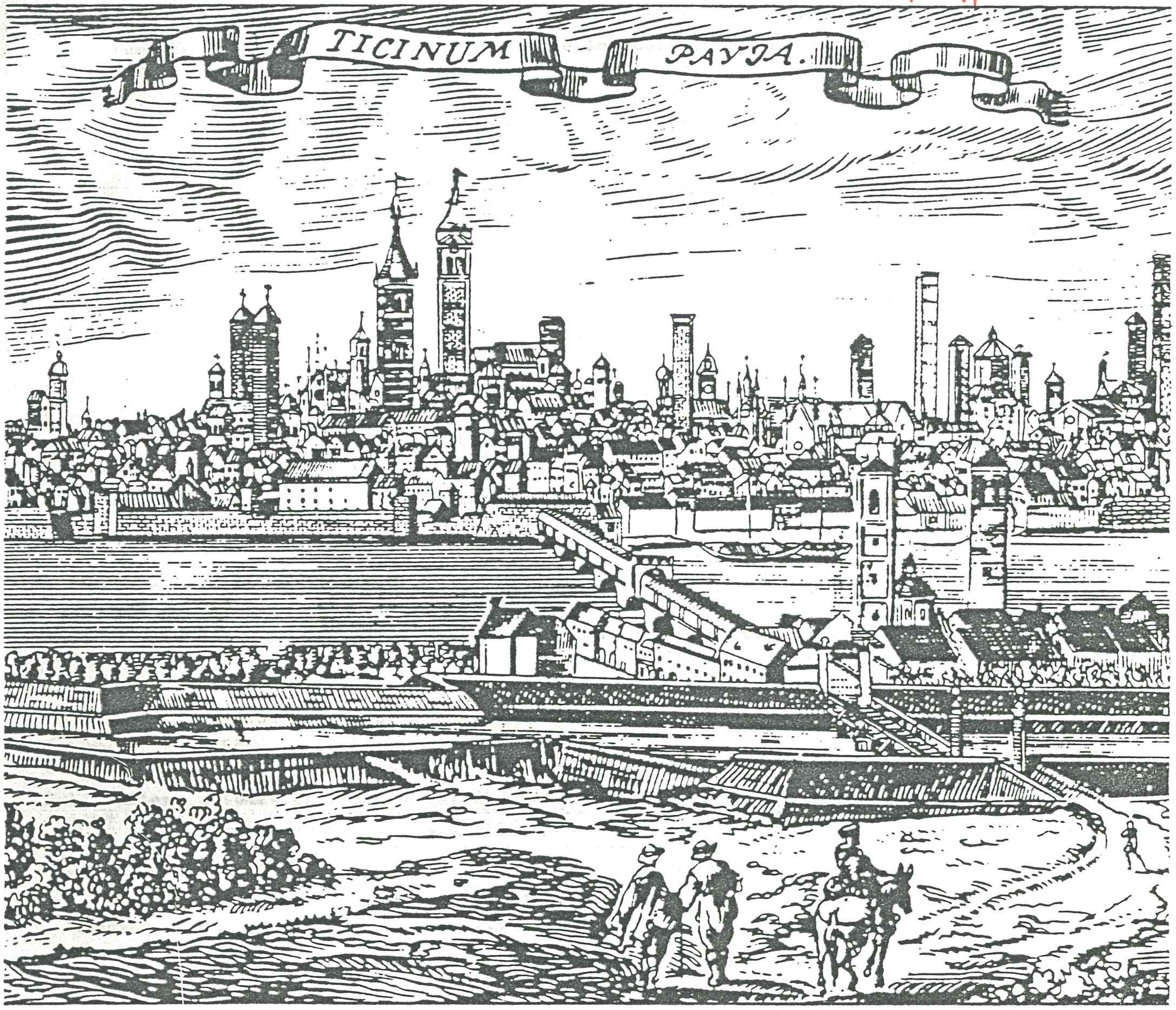
CONGRESSO ANNUALE
ANNUAL CONFERENCE
A.I.C.A.

ASSOCIAZIONE ITALIANA PER IL CALCOLO AUTOMATICO

PAVIA 23·25 Settembre 1981

atti

volume II



STRUCTURED DESIGN OF FAILURE TOLERANT SYSTEMS

A. Ciuffoletti(*) - L. Simoncini(**)

(*) Selenia S.p.A. - Roma, Italy
(**)Istituto di Elaborazione dell'Informazione - Pisa, Italy

The literature on reliable systems is composed by a very broad range of specific problems and solutions. Very few designs of reliable systems are reported, in which an integrated methodology is taken into account as one of the most important design goals.

This fact makes, in general, difficult to provide a good readability and possibility of evaluation of the proposed solutions for enhancing the reliability of these systems.

The aim of this paper is to provide a structured methodology and several implementative suggestions for the design of failure tolerant systems.

The overall approach does not explicitly differentiate between hard or soft objects and allows the treatment of failure tolerance from the first design phases.

1. INTRODUCTION

Computer Science has been mostly oriented towards the synthesis of stricIty deterministic systems: this means that any component of a system is associated with a specification, which completely describes its behaviour in a definitely predictable way.

In the real world, we have to cope with anomalous behaviours of components and systems. Such behaviours are originated by failures which can be:

a) physical failures or faults, which are largely impredictable or
b) specification or implementation failures like bugs or flaws or malicious attempts, both caused by human errors, which are again impredictable.

The probabilities of such events are relatively low, and this justifies the approach of classical computer science. However the spreading of applications requires high confidence in the computing tools or high reliability for critical control systems. Therefore it is vital to take into careful account the possibility of anomalous behaviour of the system. Should such event happen, the system has to be able to exibit a "positive" and predictable behaviour.

This motivation is the basis of the development of fault tolerant computing which can be seen as a wide and difficult extension of the classical computer science.

The design of systems with well balanced capabilities of coping with failures, requires that robustness characteristics must be considered as initial specifications of the design phase and must constrain each step of the design at any level.

Still the literature on failure tolerant systems is composed by a very broad range of specific problems and solutions. Very few design approaches are reported in which an integrated methodology is considered as one of the most important design goals [1,2,3].

In general this makes difficult to provide a good readability and possibility of evaluation of the proposed solutions for enhancing the robustness of these systems to failures.

In this paper, we present a structured design methodology, by giving the basilar ideas for building up a general framework for the design of failure tolerant systems. Such basilar ideas will be introduced as implementative suggestions; they are useful in that they are relatively simple and consistent. Since they do not differentiate between hard or soft objects they are able to be omogeneously applied to all the design levels of a failure tolerant system.

In Section 2 the general phylosophy is presented and the implementative suggestions are derived and discussed. Most of the ideas discussed in this paper has been previously introduced in [4] and a reduced version of it is in [5].

2. GENERAL PHILOSOPHY FOR STRUCTURED DESIGN

Any system can be decomposed in subsystems of lower complexity; their interaction supports the functional behaviour of the system.

This general statement introduces three central concepts:

- the machine, which represents the basilar abstract component of a system; it exactly corresponds with its specification or with the set of its functional behaviours;

As an example of the previous discussion, let us consider how machines can model data types [9] at running time. The machine can be identified as an instance of a certain data type: this instance will be related statically or dinamically to one or more "users". In the second case we shall say that the instance is a shared resource; but it is to be pointed out that the resource is active, i.e. it is able to judge on the validity of an interaction. Only particularly malicious behaviours could deceive it. Therefore the interactions among the resource and the users are failure tolerant and the expectation will be based on names, codes, lenghts and bounds.

After the error detection has been performed through the symptom analysis it is necessary to identify or diagnose the primary source of the anomalous behaviour.

We shall suppose that the channels are always reliable. In fact, if this is not the case, the diagnosis will be greatly complicated. So we introduce a new implementation suggestion:

- the communication channel is completely transparent.

The implementation of the communication channel will be discussed in Sect. 2.3.

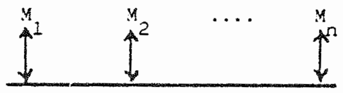
At the interaction object the informations which are necessary for the identification of the source of the anomaly, can a) arrive with the anomalous interaction as a symptom or b) be present in the interaction object as a consequence of previous interactions or by the same specification of the machine.

In the case a), the symptom is always reliable, but it might be difficult and dangerous to try to get a definite diagnosis by associating a specific symptom to a specific machine.

If the interaction object knows the subject, the simplest strategy is to consider it as the responsible of the error. It will be up to the subject to verify if the source of the error has to be ascribed to other machines with which he previously interacted.

With this approach the diagnosis is structured in a step by step way.

This is possible if shared communication channels are not present. In a situation like Fig. 1, which models the existence of a shared communication channel where M_i are machines and the arrow represent the interactions, the knowledge of the subject is rather complex. In this case a machine should be able to identify unambiguously the subj- ect through e.g. an appropriate coding of the subject field in the communicating protocol. This approach may not be completely safe, since failures in the machine which destroy the coding can be thought; in this case it would be necessary to diagnose as failed all the potential subjects and wait for evidence of failure freedom by them.



"Fig 1 Shared Communication Channel"

In case b) let us consider the possibility of determining the subject of an interaction on the basis of informations previous to the interaction. This identification is always possible if a dedicated communication channel exists between two machines.

In the case of a shared communication channel such identification is extremely difficult, even if we consider the existence of previously defined specifications about the use of the communication channel. In fact an anomalous behaviour may not respect such specifications.

As an example let us consider a channel whose utilization is passed explicitely from one user to other with a specific policy; a failed user can attempt an incorrect use of the channel; the machine which detects the irregularity cannot rely its diagnosis on the confidence that the failed machine obeys to such rules.

These analysis determines two other implementative suggestions:

- the diagnosis ascribes the source of an anomalous interaction to the subject of the interaction;
- with common communication channels, the subject must identify itself with a strongly symptomatic code.

The machine which has been judged as failed may be requested to retry the last action or series of actions; this retry request may start a retrospective diagnostic analysis of the interactions of which the requested machine was object, with the aim of finding out possible anomalies.

The first action in diagnosis is:

- retry of interaction is requested to a subject which has exhibit an anomalous behaviour.

As a consequence:

- both subject and object of FTI should be provid- ed with the capability of performing a retry action, that is possibly recovering a previous correct situation.

Moreover it is required that all the diagnostic steps are exercised through FTI.
This allows to state that:

- the diagnosis and the subsequent operative decisions are meaningful only locally to the machine which developed it.

- the interaction, which represents the basilar cooperation act among the machines;
- the implementation relation which maps the set of functional behaviours and of the interactions of lower complexity machines into the functional behaviour of the implemented machine.

We model the system as a set of virtual machines, organized in a hierarchy of successive levels of abstraction; any level of abstraction is composed by machines which cooperate on the basis of interactions; each machine at a given level of abstraction is considered as an implementation of other machines at the next lower level.

We shall assume that any level has its own failure modes.

In the following sections we will deal with the mutual relations among the three concepts.

2.1. The Machine and the Interactions

In this section we shall deal with the relation which exists between machines at the same level of abstraction, and with the treatment of the effects of failures which are detected at that level.

The functional behaviour of a machine depends on the undergoing interactions and on its non deterministic specifications. The non deterministic behaviour of a machine can be better defined by starting that, at any moment, it is very likely to be able to predict its future behaviour, but there is a non zero probability that the machine will exhibit an anomalous unpredictable behaviour.

The definition of anomaly implies that there is an external entity which is able to foresee and expect a given (failure-free) behaviour of the machine, by interacting with it.

In non failure tolerant systems this attitude can be associated esclusively with the user or operator, he can:

a) unconditionally accept the result from the system, in case of complicated, that is unpredictable, computations; or
b) submit the acceptance of the results to their reasonableness; this means that results are in some extent predictable.

In this latter case we shall better say that the user can develop an expectation about the behaviour of the system. This ability needs some redundant knowledge, which is unnecessary if a total confidence is relied on the system.

In the following, interaction object will have the meaning of "user" of the interaction which has been originated by a subject.

In a failure tolerant system, the attitude of expectation shall be assigned to the machines, inside the system.

Such consideration allows to state the first implementation suggestion:

- in failure tolerant systems, any machine which, at a given instant, acts as interaction object, must be able, on the basis of an expectations to judge the behaviour of the subject. If the expectation is not satisfied the behaviour of the subject is judge as anomalous.

The expectation can be either statically assigned at the interaction object or the interaction object can dinamically create such expectation as a consequence of different situations.

The influence of the communication support on the interaction will be analyzed in Section 2.3.

We define a failure tolerant interaction (FTI) as an interaction which can be submitted to a conditional acceptance by the interaction object.

The condition is based on an expectation about certain characteristic of the interaction, which we call interaction symptons. In addition to the symptoms analysis, the interaction object will perform the current computations on the information.

Information and symptoms are the two components not necessarily separated of an interaction (e.g. a watchdog timer (incorporated in a machine) controls a symptom generally not dependent on the carried information, while a routine which controls if a data is in a given range heavily depends on it). We point out that symptom analysis is not necessary in normal operation, so it can be seen as "redundant" operation. Consequently we have two types of redundancy:

- behavioural redundancies, which determine the generation of symptoms
- computing redundancies, which determine the generation of expectations.

Usually, symptoms are generated even if nothing is specifically provided with this aim in the design of the machine (e.g. locality for a program or status sequence for a processor), while the expectation attitude is generally to be added to the normal implementation of a machine. This requirement shall be inserted by exploiting the characteristics of the machine with minimum impact on it.

Another aspect which derives by the previous discussion is that the interactions shall be known and verifiable by the interaction object. This requirement defines the second implementative suggestion:

- all FTIs are definitely explicit.

This statement determines severe constraints for FTIs being supported by global environments [6]. If an "implicit" channel of interaction exists, it would be possible an uncontrolled spreading of errors. Therefore an environment, which is more oriented to support failure tolerance, is that based on message passing [7,8].

a) a transient failure; in this case the machine goes back to a normal behavioural condition and the system can recover a regular functioning;
b) a permanent failure; in this case the machine cannot go back to a normal behavioural condition. If we want the system to recover, we need some other machine, with analogous functional capabilities, to be able to substitute it.

In the case b), "spares" shall be provided to allow the system to go back to a normal functioning. Two kinds of "spares" can be identified:
a) passive spares: those machines which were not active, that is "stand by" before their insertion;
b) active spares: those machines which, before their insertion, were performing:
   - the same operation which was previously performed by the failing machine; a typical example is the duplication with check;
   - different operations from those performed by the failing machine; in this case the spare has to be sufficiently flexible to fulfill the operation previously carried out by the failed machine; this is the basis of graceful degradation.

The cost in terms of performance and percentage of resources utilization which is associated to the techniques for managing these two types of spares is different. It is optimized in case of active spares performing different operations, since the percentage of resource utilization is te highest and the performance is reduced only in case of failures.

In any case the two type fo spares are compatible, their use depending by the requirements of application of the system.

As regards to machines which exhibit only transient failures, it is sufficient to keep copies or traces which are necessary to recover their normal functional behaviour; we can say that these machines have embedded "self repairing" characteristics.

We define:
- a machine is failure tolerant (FTM) if it exhibits only transient failures.
An implementative suggestion can be stated:
- "spares" shall be provided for non failure tolerant machines (NFTM).

Obviously an object has to be able to start communication with the substitute of the failed machine as well as refuse any interaction by a definitely failed subject.

When a failure tolerant system is implemented, the starting building blocks are non FTMs. At an intermediate abstraction level, between the lowest and the user level, FTIs among NFTMs are introduced (by definition, FTIs do not take place necessarily among FTMs). At thislevel we should necessarily among FTMs). At this level we should provide "spares" for the NFTMs, which are subjects of FTIs, in order to allow the substitution of a definitely failedmachine. It is obvious that all failures, exhibited by NFTMs which are part of the implementation of NFTMs among which FTIs are introduced, will be assimilated to failures in the NFTMs, which are provided with spares.

A better characterization of a FTM is the following:
- a machine is a FTM if all the interactions among the machines, which are part of its implementation, are FTIs.

In fact should an anomalous behaviour of one of the implementing machine have effects at a higher level, these effects will influence the machine at the higher level only in a transient mode. They will disappear when the failure is fixed at the right level. Thus:
- "spares" are not necessary for FTMs, since only the recovery action is needed.

We can outline the hierarchy of abstraction levels in a failure tolerant system as follows:
a) at lower levels of abstraction NFTMs and NFTIs will be present,
b) at intermediate levels NFTMs and FTIs will be present; "spares" shall be provided;
c) at higher levels FTMs and FTIs will be present; "spares" will not be necessary.

At levels of abstraction generally defined as intermediate, FTMs and FTIs will be present together with non failure tolerant ones.

Let us consider the following example, outlined in Fig. 2.

Let M_{13} be failed, and an error be detected by M_{14}. M_{13} will be substited by M_{13} and M_{14} will start again interacting with M_{13}. This reconfiguration will be notified to M_{12} by M_{13} and possibly accepted. In the worst case M_{21} will exhibit a transient failure.

In case that M_{14} is failed and that M_{13} detects the failure, M_{14} could not be substituted immediately since it is not provided with "spares" This failure will be evidenced at the higher level, when M_{21} will detect some anomaly in the behaviour of M_{22}. Then M_{22} will be substituted by M_{22}. The interruption of interactions by M_{13} will be the correct action, after which M_{13} can continue its normal computation.

By this discussion we have pointed out the relation among machines at different level of abstraction.

The relevant deriving characterizations are:
- a machine is FTM if it is well implemented having as implementation basis FTIs and if all the NFTMs, which are part of its implementation, are provided with spares;

This means that any operative decision subsequent to the diagnosis is under the responsability of the machine which developed it. It is also confined in its effects to condition the behaviour of that machine towards the other machines with which it interacts. The only thing a machine can do is notifying an not forcing the other machines with its decision. Situations like "master-slave" will be very dangerous since possible malfunctions could make a machine, which acts as master, to try to deceive the other machines with unknown an uncontrollable effects.

In this sense a machine which has diagnosed another one as definitely failed, closes the communications with it by refusing any other interaction with it. This action, which is the most protective one a machine can undertake, can be performed through the knowledge of the subject of the subsequent interactions; in case of dedicate channel, simply by refusing the interactions coming on that channel; otherwise in case of shared channels by a preliminary analysis and decoding of the subject field of the information. Therefore:

- the operative action successive to the diagnosis of a definitely failed machine consists in the interruption of communications, by refusing any other interaction with it.

With the previous discussion, a characterization of the constraints of an interaction has been derived, when the subject is a possibly failing machine. Such kind of interaction has been defined as FTI.

The several relevant features of a FTI are:

a) as concerning the implementation of the interaction object:
   - creation of expectation on the behaviour of the subject
   - capability of developing a local diagnosis, tentatively ascribing the cause of the error to the subject
   - capability of requesting a retry of operation and of maintaining an internal consistent situation while waiting for the retried interaction
   - capability of undertaking operative actions, by refusing interactions with machines which it has judged definitely failed.

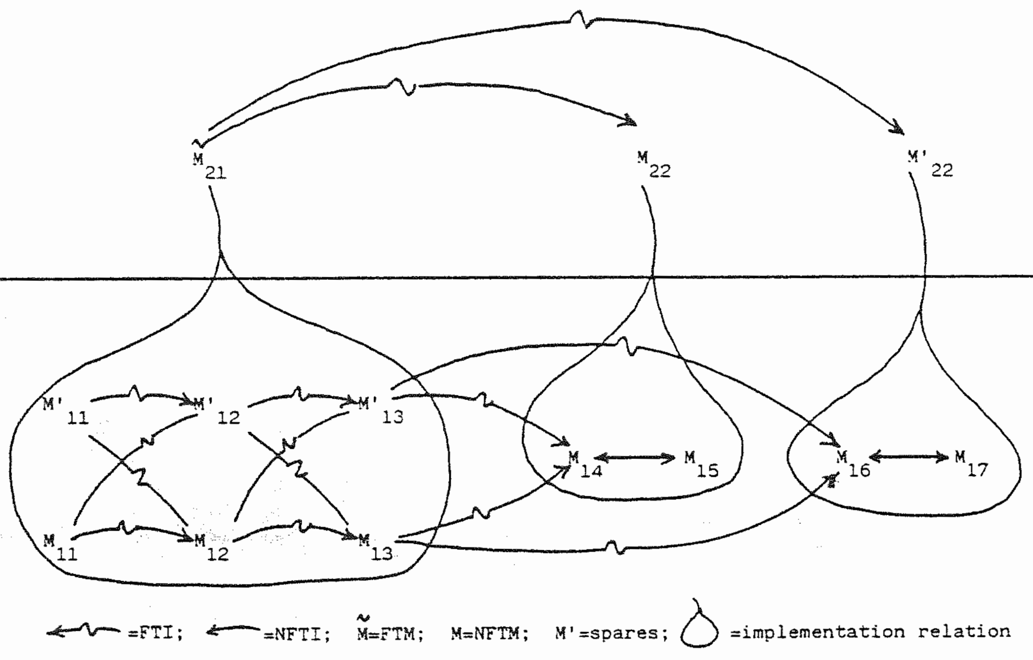
b) as to the subject:
   - capability of supporting the retry action
   - name identification by a strongly symptomatic code, in case of shared communication channel

c) as to the interaction channel:
   - transparency to the interaction,
   - access to a small subset of the object machine state.

2.2. The Machine and the Implementation

In this section we will deal with the relation between machines at different level of abstraction and with the treatment of the effects of failures which are detected in levels different from that in which the failure has been originated.

An anomaly which is detected in the behaviour of a machine, can be originated by:



"Fig 2 Example"

- the effects of failures at lower level are modelled, in failure tolerant machines, as transient failures; therefore only a recovery action is necessary in FMTs and "spares" are not needed.

2.3. The Implementation and the Interactions

In the previous section the problem of implementing FTMs from NFTMs has been dealt with. The solution of this problem requires the introduction of FTIs.

The constraints for an interaction being a FTI concern the interacting machines and the functionality of the communication channel.

In this section we will discuss the characteristics for a correct implementation of a communication channel.

The constraints for an interaction being a FTI require that a communication channel can:
i) be transparent to the interaction,
ii) exhibits only transient anomalous behaviours.

It is quite straight by the point ii) that communication channels, which are able to support FTIs, have to be implemented on FTIs at the lower level of abstraction. Therefore let us first consider communication channels among NFTMs which support FTIs.

A channel is usually accessed by two or more users: if both a machine and its "spares" are accessed by the same channel (i.e. they are objects of interactions supported by this channel), then the failure of the channel will imply the unavailability of both the machine and its spares. The situation is exactly the same if both a machine and its "spares" can access the channel (i.e. they are subject of interactions supported by this channel). In fact, particular malfunctionings may exist for which a machine can hold indefinitely the channel. Therefore:

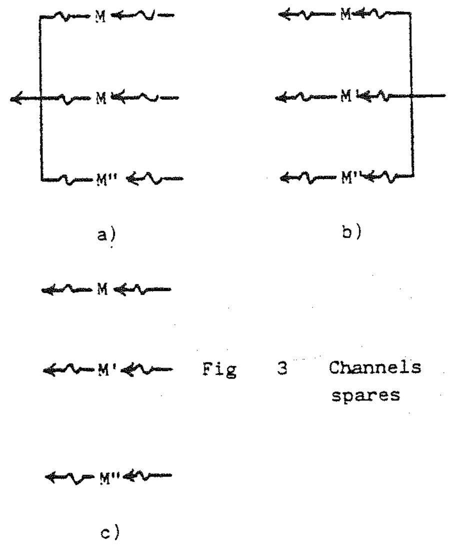


Fig 3 Channels to-from spares

- not all the machines which can support a certain function have to access or be accessed through the same channel.

Therefore the situation in Fig. 3a) or 3b) are not advisable, while that one in Fig. 3c) would be safe.

In other words, we have in general that a correct implementation of a communication channel for supporting FTIs among NFTMs, should rely on a point to point basis and not on shared communication channels (e.g. a bus with arbitration) as outlined in Fig. 4. The approach in Fig. 4, which is almost usual, is completely unsafe, since, as previously said, the failure either of a processor or of the channel can determine the loss of all the system.

Nevertheless a complete point to point approach is very expensive and difficultly expandable.

In this case we have to consider the channel as no more transparent, and substitute it with an explicit interacting machine, possibly provided with spares. This is outlined in Fig. 5.

The requirement that the communication channel is an explicit interacting machine, determines the need that it is an intelligent unit, able of refusing on the basis of an expectation the interactions with the Ps. With this approach, we actually replace the shared communication channel with a set of dedicated ones and a machine. Therefore the discussion of the previous sections can be applied to this case.

For what concerns higher levels of abstraction, we have already pointed out that it is always possible to implement correctly communication channels able to support FTI's, once this problem has been solved at lower levels. In this way we can meet the requirement that at higher level only transient anomalous behaviours are induced by failures at lower levels.

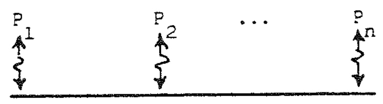


Fig 4 Common bus

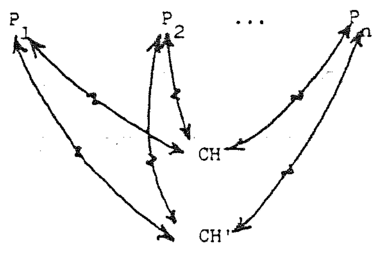


Fig 5 Explicit channel

CONCLUSION

In this paper we have dealt with the problem of implementing a system tolerant to failures in its h/w or s/w support. Some design rules are given which can help in this complex task. Those rules have their validity at any abstraction level, and in a way, are recursive: in fact the application of the rules at a given level helps the designer to respect the same rules at the higher ones. So, the risk of implementing conflicting policies is avoided, and a good readability easily provided. In a word, we give a methodology to structure failure tolerant designs. Of course the drawback of any structure is to constrain the choices the designer can do: but he will not have to deal with problems that the structure will "solve" for him. In our case important problems as:

- which components have to be replicated
- how to build channels which do not affect failure tolerance
- how controls the functioning of a certain component
- which kind of consequence a low level failure can have on higher levels
- what is the meaning of redundancy and how to introduce it effectively
- which concept underlies different failure tolerant policies which are practically implemented (as TMR and "graceful degradation") are treated and solved in a completely general way.

One of the main problems left unsolved by the presented structure is how to support the retry operation. In fact the retry models the recovery action, and related to this is the problem of the spread of recovery actions, known as "domino effect" [1]. In this way we will turn our future efforts.

REFERENCES

[1] B. Randell, P.A. Lee, P.C. Treleven, "Reliability Issues in Computing System Design", Comp. Surv. Vol. 10, n° 2, June 1978, pp. 123-164.

[2] W.C. Carter, "Fault Detection and Recovery Algorithms for Fault Tolerant Systems", IFIP Working Conference, "Reliable Computing and Fault Tolerance in the '80's"; London, Sept. 1980.

[3] A.L. Hopkinks, "On Virtual Levels of Fault Processing for Very Reliable Systems", IFIP Working Conference, "Reliable Computing and Fault Tolerance in the '80's", London, Sept. 1980.

[4] A. Ciuffoletti, "Approccio strutturato alla modellistica di sistemi fault tolerant", ISI, University of Pisa, (Tesi di Laurea), April 1980, in Italian.

[5] A. Ciuffoletti, L. Simoncini, "Integrated Design Methodology of Failure Tolerant Systems", to be published in the Proceed. of FTSD, Intern. Conference on Fault Tolerant Systems and Diagnostics, Brno, Sept. 1981.

[6] C.A.R. Hoare, "Monitors: an Operating System Structuring Concept", Communications of the ACM, Vol. 17, n° 10, Oct. 1974, pp. 540-557.

[7] R.E. Bryant, J.B. Dennis, "Concurrent Programming", MIT Report, 1979.

[8] E. Manning, N.J. Livesey, M. Tokuda, "Interprocessor Communication in Distributed System: One View", IFIP'80, North-Holland, 1980, pp. 513-520.

[9] T.W. Pratt, "Programming Languages: Design & Implementation", Prentice-Hall, Englewood Cliff, N.J.

This work has been supported by the "Progetto Finalizzato Informatica Sottoprogetto P1, Obiettivo MUMICRO" and by the Convention between SELENIA and CNR.