

ISTIT. EL. INF.
BIBLIOTECA
Post. *ARCA WLO*

ČESKOSLOVENSKÁ VĚDECKOTECHNICKÁ SPOLEČNOST

LSI-06



DIAGNOSTIKA A ZABEZPEČENÍ ČÍSLICOVÝCH SYSTÉMŮ

1981

INTEGRATED DESIGN METHODOLOGY
OF FAILURE TOLERANT SYSTEMS

A. Ciuffoletti*, L. Simoncini**

*Selenia S.p.A., Roma, Italy

**Istituto di Elaborazione della Informazione, C.N.R., Pisa, Italy

Introduction

The treatment of malfunctioning machines has usually been considered as relevant only for the design of critical application systems, supposing that a careful design and debug were sufficient in most cases for a correct working of a system /1/.

The spreading of applications of computer systems and their growing hardware and software complexity has shown the need of coping with residual bugs and failures.

Nevertheless, the failure tolerance attributes are still taken into account only once the functional design of a system has been completed. In our opinion this attitude is wrong: in fact in this way the attention of the designer is concentrated on specific levels of abstraction of the system (typically the hardware level), while the correct implementation of failure tolerance derives only from the integrated design of the several abstraction levels of the system /2/, /3/, /4/.

Of course such integration of different levels designs may be too difficult, if the design is not adequately structured. But if any level is realized according to specific and general rules, the duty of integration will be greatly simplified, since it will be easy to have an intuitive outline of the whole system.

General philosophy for structured design

We model a system as a hierarchy of successive levels of abstraction. To structure the composition of any level, two concepts are introduced: the machine and the interaction; together they model the functionality of a parallel processing environment, since any machine cooperates with the others through interactions.

At a given level any machine is considered as an atomic functionality, whose response is checkable through its partial "predictability". In this way any machine at a given level will check the correct working of any other interacting machine through expectation criteria. Expectation can effectively model both the redundancy (a concept very bound to the hardware) or the assertions (a typical software concept). If the expectation is not satisfied, a failure is detected and a diagnosis is performed by the machine which has detected the anomaly. A failure tolerant interaction (FTI) is

defined as an interaction which can be submitted to conditional acceptance.

Two main characteristics are outlined by the previous discussion; any FTI will distinguish subjects and objects; that is, the one whose behaviour is controlled and which produces the information and the other which uses the information, performing a control on it through an expectation.

Both the objects and the subjects of an FTI have their own kind of activity; we exclude the existence of passive visible items; in fact their existence always poses serious problems of safety.

For the same reasons, it is necessary that any FTI is definitely explicit, since the control of the interaction object can be performed only on completely visible interactions. This constraint does not exclude the possibility of anonymous interactions, even if this circumstance can make difficult the diagnosis. Anonymity is needed in case of interaction channels shared among several machines. In this case we cannot rely on the "subject field" eventually present in the information record.

The first phase of the diagnosis may be a preliminary identification of the subject of the anomalous interaction. Then a retry phase will be executed between both the object and the subject of the anomalous interaction. To this aim they must be able to perform a retry action, that is to recover a possibly correct previous situation. If the retry is not sufficient to the resumption of a correct working, as the failure appears permanent, a replacement can be operated.

What is relevant in the whole diagnosis phase, is that every machine interested in it acts independently, with no possibility of actions stronger than an interaction as previously defined. The reconfiguration too must be developed under the condition that a wrong protective activity cannot damage any function but the one of the machine which executes such action. So the highest protective action a machine can realize is to cut the logical channel connecting it with the machine it considers as failed.

As previously specified, spares are necessary only if permanent failures are present. The spares can be either previously passive machines, or machines which were performing the same work of the failed one (as in the TMR) or previously active machines which are able to carry on the function of the failed machine. In general in the case of a machine which will experience only transient failures, spares can be obviously avoided.

Among different levels of abstraction, an implementation relation exists, so that machines at higher level of abstraction "use" the functionalities which the machines at lower level make available through their interactions.

Let's consider the behaviour of a machine which is implemented on machines at lower level which interact through FTIs. Should an anomalous behaviour of one of the implementing machines, have effect at higher level, these effects will alter the machine at higher level only

in a transient mode and these effects will disappear when the failure is fixed at the right level. We will call failure tolerant machine (FTM) a machine which is implemented starting from FTIs. Its main feature is that it exhibits a sort of "self-repairing" ability, that is, at most transient failures may be present. Nevertheless the ability of retry must be provided for this kind of machines too, to recover a correct state after a failure, but spares are no more necessary.

Since FTMs can be introduced through FTIs we can outline the following bottom up structure of the whole system:

- 1) NFTMs cooperating through NFTIs;
- 2) NFTMs cooperating through FTIs; spares are needed;
- 3) FTMs cooperating through FTIs; if the implementation is reliable no spare is needed.

These conclusions move the attention of the design to the realization of the interactions, more precisely to the implementation of their support. The most relevant features of a FTI are:

- a) transparence to the interaction: if the channel has to be passive it must be completely transparent: otherwise, it should be considered as a machine, someway responsible of the carried information;
- b) the correct working of the whole system relies on the consistence of the above specifications with the real working. At most, transient anomalies can be tolerated without serious drawbacks.

The need of transparence is quite straight, while the need of non-permanent failures of the channel must be discussed in short.

First, let's consider the behaviour of a distributed channel connecting several NFTMS, provided with spares, through FTIs. If both a machine and its spares interface the same channel, then a permanent failure of the channel will probably collapse the system. So, in this case, non permanently failing channels are needed. The same arguments can be used to show that non permanently failing channels are needed to support FTIs among FTMs. In fact, a FTM has, in general, no spares; so the definite failure of the channel accessing it can destroy a necessary function. If the distributed channel supports FTIs between NFTMs, but not all the machines which can perform certain functions are connected at this channel, than its failure can be tolerated and assimilated with the failure of all connected machines. In this case the channel can definitely fail, without catastrophic damage for the system.

In conclusion, permanently failing channels are consistent with our structure only if the machines connected to them are replicated elsewhere on different channels. Such an attitude brings to the "privatization" of the channel itself. The trend is to split a common channel into several private channels, as realized in the Pluribus system /5/. The eventual failure of one of these private channels will bring, at most, to the loss of one of the set of connected machines; this loss is tolerated, since we are in the case of FTIs among NFTMs provided with spares.

Further, once the problem of building FTIs on permanently failing

channels has been solved at a given level, it is possible to implement non permanently failing channels at higher levels.

Example of level structuring

To explain the philosophy outlined in the previous section, let us consider a very general example of level structuring. Let us consider a program which implements a routine. In our view, the program can be structured as a machine. In fact other routines at the same level will interact with it, exchanging informations and commands. At run time this program will be present, as any other compiled program, in the program memory as a sequence of words. In this way an entity belonging to an higher level will be mapped in a set of functionalities at a lower one. At this lower level the program consists of a stream of instructions (i.e. commands and informations) sent to machines at the same level. More precisely the program will be present, at this level, as the internal state of a program support machine, namely the program storage. This machine will be the object of interaction from the sequence controller which will send to him the address of the next instruction and it will send this information to the instruction decoder. This machine will interpret the coded instruction and will send commands to the data manipulation-register unit and to the sequence controller. Using a directed graph representation, where the arrows represent directed interaction capabilities, and the nodes are the machines, the situation can be outlined as in Fig. 1.

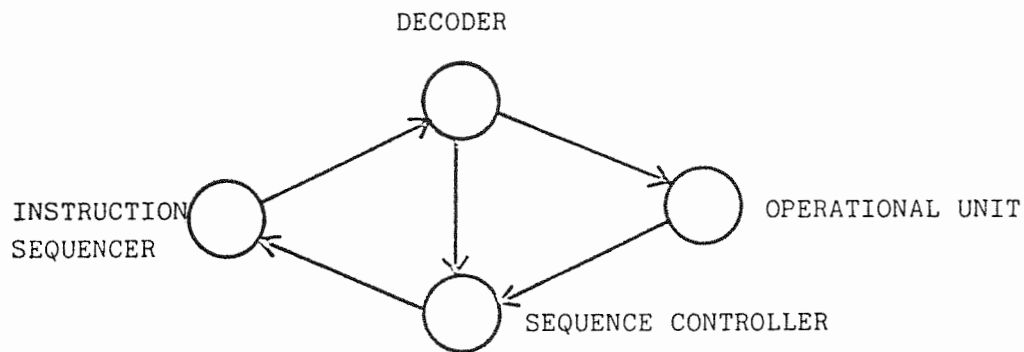


Fig. 1

In this outline all the machine are represented as logical entities and are still independent by their phisical realization. This structuring is therefore strictly correlated with the semantics of the level. It must be pointed out that the implementation relation is not concerned neither with the compilation nor the interpretation. In fact the interpretation always resides at the same level with the "coded" program, while the compilation simply "translates" one coding in another coding, regardless the level where the coded object resides. The only relevant feature to determine the level which a given machine belongs to depends from its specification and implementation basis. In

fact the duty of the programmer and of the compiler, is to map the machine specification in a set of functionalities available as implementation basis. So, from our point of view, levels can be really called "abstraction level" and they are not language dependent: that is, two programs, both written in assembler language can reside at different levels. A typical example of this statement is a kernel primitive and a user program, both written in PLM. Requests of separation between implementations of different levels are inconsistent, since any machine is mapped in a set of lower level machines, in such a way that lower level data implement upper level data. It would be like to forbid the updating of the program counter at the kernel level: such an action would be uncontrollable and unavoidable, as any kernel machine uses the program counter without knowledge of it.

An example of expectation, which the instruction sequence machine can exhibit on the interactions from the sequence controller machine, can be based on the requirement that the program has a bounded locality. If no reference is allowed outside a given range around any instruction, the instruction sequence machine may check that the address coming from the sequence controller is in this range. Otherwise the expectation is not fulfilled and an anomalous behaviour is detected.

Conclusion

In this paper we have presented a structured methodology for the design of failure tolerant systems. This methodology is useful in the structuring of any abstraction level, since it does not differentiate between hard or soft objects. In some way it may be considered recursive, in the sense that its application at a given level helps the designer to respect the same approach at the next level.

References

- /1/ A. Avizienis, "Fault Tolerance: The Survival Attribute of Digital Systems", Proceedings of the IEEE, Vol. 66, n° 10, Oct. 1978, pp. 1109-1125.
- /2/ W.C. Carter, "Fault Detection and Recovery Algorithms for Fault Tolerant Systems", IFIP Working Conference on "Reliable Computing and Fault Tolerance in the '80's", London, Sept. 1979.
- /3/ B. Randell, P.A. Lee, P.C. Treleaven, "Reliability Issues in Computing Systems Design", Comp. Surveys, Vol. 10, n° 2, June 1978, pp. 123-164.
- /4/ A.L. Hopkins, "On Virtual Levels of Fault Processing for Very Reliable Systems", IFIP Working Conference on "Reliable Computing and Fault Tolerance in the '80's", London, Sept. 1979.
- /5/ D. Katsuki et al., "Pluribus - An Operational Fault Tolerant Multiprocessor" Proceedings of the IEEE, Vol. 66, n° 10, Oct. 1978, pp. 1146-1159