A MODEL FOR FAULT-TOLERANT COMPUTER SYSTEMS

H. A. Goosen[1]
C. H. Hoogendoorn
Department of Computer Science
University of the Witwatersrand
Johannesburg, South Africa

1. INTRODUCTION

Reliability is an important consideration in the design of computer systems. This is a result of the increasing complexity of computer systems and of the greater reliance placed on their continuous and correct operation. In this paper we describe a model which can be used for the analysis and design of fault-tolerant computer systems [1].

Techniques for the achievement of fault tolerance usually consist of the following activities [2],[3]:

- The detection of faults in the system.
- Confining the effects of faults in the system.
- Recovering from the effects of faults.

These activities all rely on a knowledge of the failure modes of the system. The model described in this paper provides a systematic way of progressing from the specification of a system to its failure modes. As a first step in the formulation of the model, we will take a look at the structure of a typical computer system.

2. A MODEL FOR FAULT-TOLERANT COMPUTER SYSTEMS

The designers of a computer system cope with its complexity by decomposing it into simpler parts [4]. There are two ways of doing this. The first is to consider the system as consisting of interacting subsystems (horizontal structuring). The second is to represent the system as a hierarchy of models, each realising a particular level of abstraction of the system's behaviour (this is called vertical structuring).

When viewed at a high level of abstraction, computer systems usually exhibit a hierarchical organisation, as shown in figure 1 [5].

Since a hierarchical structure consists of interacting layers, the migration of effects (e. g. faults) between layers will be governed by the way in which the levels interact. We will study this interaction by isolating a single level and analysing how information moves between this level and the rest of the system.

A level in a true hierarchical system interacts only with the levels immediately above and below it. The nature of this interaction is dependent on the specification of the interfaces between the level under examination and the adjoining two levels. It is therefore possible to restructure the whole system into a three-level hierarchy by isolating the level we are interested in (this level will be referred to as the SYSTEM), grouping all the levels above it into a single level called the ENVIRONMENT, and all those below it into another level called the SUBSTRATE. This is illustrated in figure 2. The specification of the SYSTEM/ENVIRONMENT and SYSTEM/SUBSTRATE interfaces allows us to define a failure. The acceptable behaviour of the SYSTEM is described by the Authoritative System Reference (ASR) [6]. Similarly, the acceptable behaviour of the SUBSTRATE is described by the Authoritative Subsystem Reference (ASuR) [6]. A SYSTEM failure occurs when the ASR is violated, and a SUBSTRATE failure occurs when the ASuR is violated. The ENVIRONMENT is the final authority on the behaviour of the SYSTEM, and can by definition not fail.

[1] *Current address: Center for Reliable Computing, Stanford University, CA94305, USA*

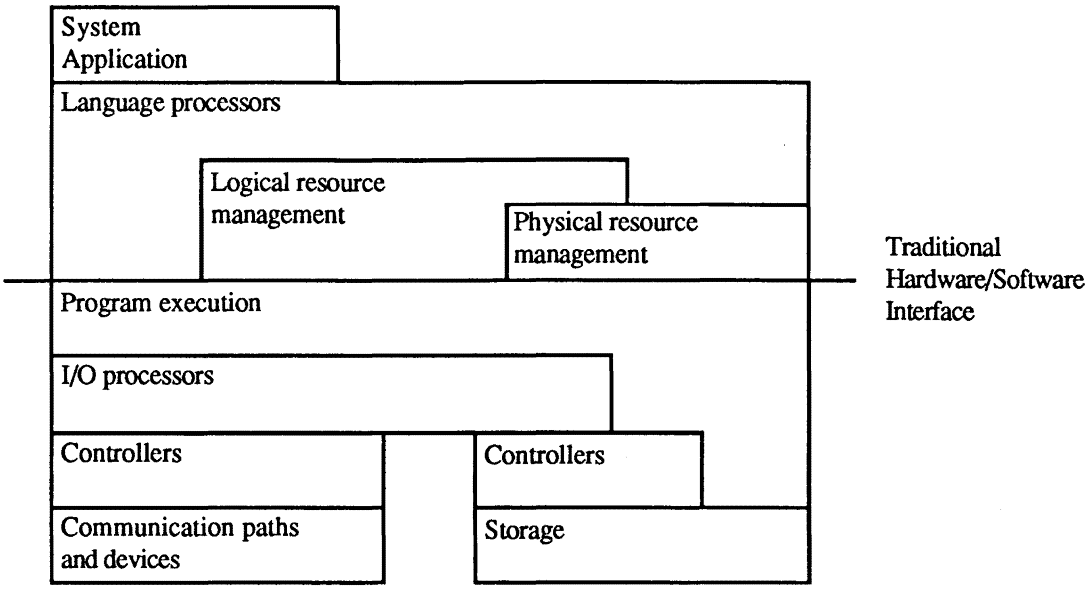

figure 1

Hierarchical Model of a Computer System

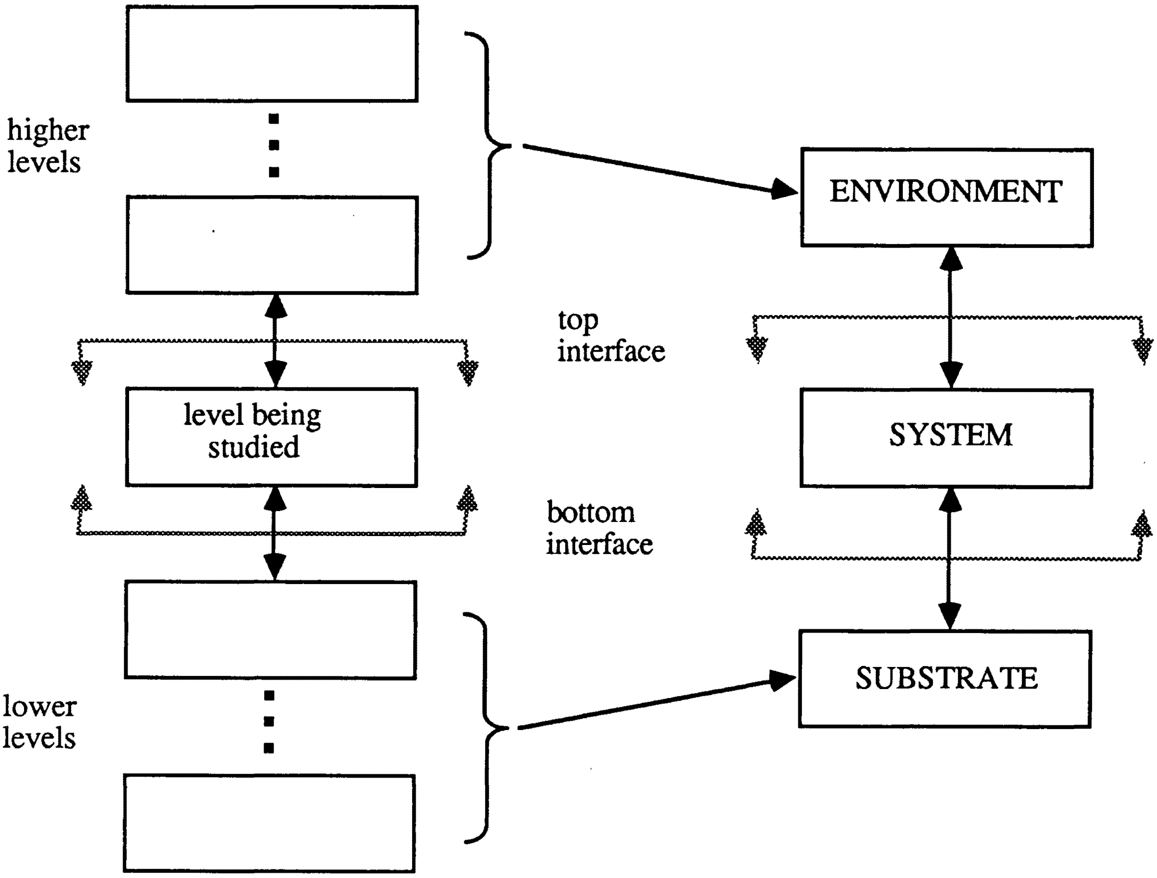

figure 2

Structuring of Computer System into Three-Level Hierarchy

From these definitions it is clear that there are only two possible causes for a failure of the SYSTEM. These are:

- Malfunction of the SYSTEM itself (i. e. an internal fault).
- Violation of the ASuR by the SUBSTRATE.

From the discussion so far it should be clear that the SYSTEM is a purely abstract machine. It depends entirely on the SUBSTRATE to provide the primitives it uses to produce the behaviour specified by the ASR. Any internal malfunction of the SYSTEM must therefore be the result of faults in its logical implementation.

A SUBSTRATE failure can also cause a SYSTEM failure. By definition, the SUBSTRATE consists of all levels in the computer system below the SYSTEM level, including the underlying physical world. A fault in the SUBSTRATE (a logic gate stuck-at-1, for example) might result in the violation of the ASuR, which, in turn, might cause the SYSTEM to fail.

The tolerance of faults in the logical implementation of systems has been extensively researched elsewhere [3]. This paper will therefore avoid this area, and will instead concentrate on interaction accross the SYSTEM/SUBSTRATE interface.

3. SYSTEM/SUBSTRATE INTERACTION

In this section we will develop a theory about the interaction between levels in hierarchically structured systems in general, and apply that theory to computer system design. The purpose of this is to predict the failure modes of the SUBSTRATE, as this will provide clues for the design of fault detection and confinement structures in the SYSTEM.

3.1 Definition of a System

A system exists in an environment, as shown in figure 3. The environment can be modelled as a set of state variables. Each state variable can assume one value (state) out of a finite set of states. Some of these variables (the inputs) have effects on the system, while the values of others (the outputs) are determined by the system. The system itself can be seen as a description of the relationship between the inputs and the outputs. Those parts of the system where the inputs enter the system and the outputs emerge from it, are called the interfaces of the system.

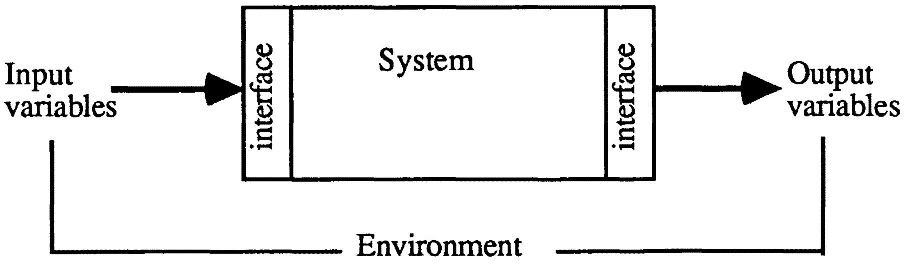

figure 3

System Model

It is obvious that this way of viewing a system gives rise to a hierarchical structure, since the environment only observes an abstraction of the behaviour of the system. The environment is the layer above the system, and the interaction between system and environment can be represented by the interaction between the SYSTEM and ENVIRONMENT in our three-level hierarchy.

3.2 Internal Structure of a System

A system consists of a collection of component subsystems connected in such a way that their combined action will satisfy the specified input/output relation of the system. This is illustrated in Figure 4.

Since the system "observes" the subsystems only in terms of their respective input/output relationships, it follows that the system is one level of abstraction higher than its component subsystems. The interaction between the system and its component subsystems can be represented by the interaction between the SYSTEM and SUBSTRATE in our three-level hierarchy.

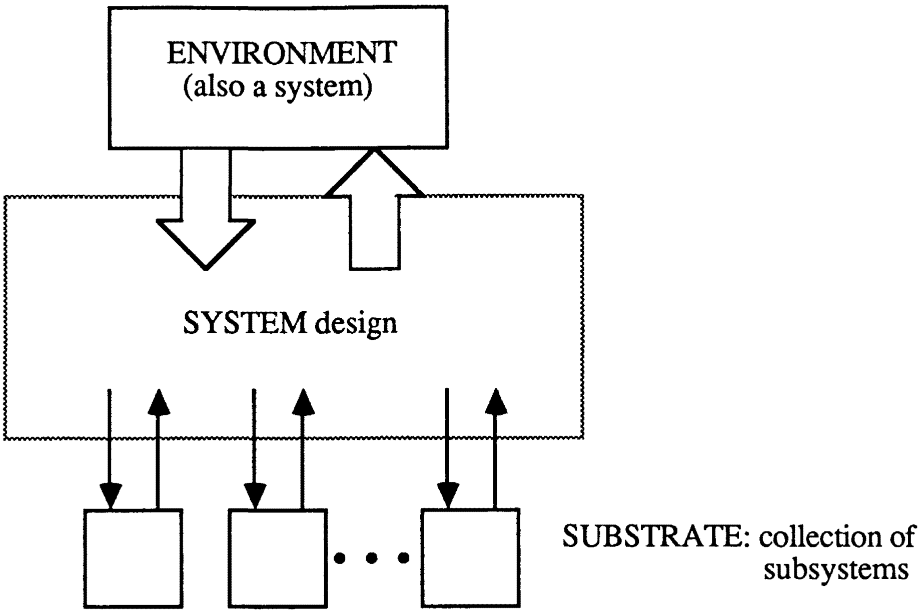

figure 4
Composition of a System

3.3 Application to Computer Systems

In section 2 it was shown that computer systems are hierarchically structured. Figure 2 illustrates the restructuring of a typical computer system to allow for the analysis of one level in the hierarchy in terms of the proposed model. The behaviour of the layer is represented by the SYSTEM/ENVIRONMENT interaction, while the structure of the layer is represented by the SYSTEM/SUBSTRATE interaction.

4. FUNCTIONAL MODULARISATION

The concept of functional modularisation will be used to formulate a more specific model applicable to computer systems. This is done by isolating the functions we expect a "general" computer system to perform, and by providing the model of the system with a separate module for each of these functions. This is intended to serve as an example when this analysis method is applied to a specific computer system.

The functions performed by the SUBSTRATE are identified by studying the specification of the SYSTEM/SUBSTRATE interface. This enables us to ignore the details of the implementation of the functions. This simplified view of the SUBSTRATE should enable us to think clearly about the SYSTEM/SUBSTRATE interactions. SUBSTRATE failures will manifest themselves in terms of the abstractions existing at the SYSTEM/SUBSTRATE interface, and the SYSTEM will have to make use of these same abstractions to detect and tolerate faults.

5. GENERAL SUBSTRATE MODEL

Every interface in a hierarchical computer system down to the microcode interpreter presents the abstraction of an information-processing machine to its higher level. This is also true of the SUBSTRATE. Any SUBSTRATE, when viewed from a higher level of abstraction, is therefore a "virtual" information-processing machine (IPM). In this section we will develop a model of a general IPM.

Every level in a computer system can be seen as a SUBSTRATE by a suitable choice of the SYSTEM layer. All these SUBSTRATEs present interpretive interfaces to their higher levels. A general IPM will therefore be modelled as an interpreter. This interpreter will consist of a collection of subsystems representing the functions we expect the IPM to perform, as well as a subsystem which interprets instructions and controls the other subsystems.

Most current machines exhibit Von Neumann organisation. This implies passive storage, a processing unit which performs state changes in this storage and a control unit which sends the processing unit sequentially through an instruction stream [5].

Functions which are performed only as the result of an instruction being interpreted are called explicit functions. These functions are described in the architecture of the IPM and they define the characteristics of the instruction interface of the IPM. Functions which are performed implicitly or automatically (i.e. without the interpretation of an instruction), are called implicit functions. In the case of an interpretive IPM these implicit functions include the storage of instructions, the reading of the next instruction to be executed from program memory, the interpretation of instructions, and the scheduling of events which implement the execution of an instruction. The IPM model will be provided with a function module for every function, both explicit and implicit, which it performs.

5.1 Explicit Function Modules

There are a few basic functions which include all operations that can be performed on information. Information can be created (acquired) or destroyed (disposed of). It can be stored or retrieved. Furthermore it can be transformed into another item of information by the application of a transformation function. Every explicit function an IPM performs can therefore be reduced to one or a combination of the following:

- The creation and destuction of information. In practice this means communication with the external world.
- Storage/retrieval of information.
- Transformation of information.

The model will contain a module for each of these functions.

5.2 Implicit Function Modules

The implicit functions support the explicit functions and allow them to be combined to satisfy a design specification. These functions allow for the representation, storage and implementation of design descriptions (programs). The execution of a specific implicit function is automatically triggered by the occurrence of some internal state.

These functions can be divided into two groups: functions associated with the control of the machine, and functions associated with instruction storage and execution. The model will be provided with a CONTROL and INSTRUCTION module to represent these functions. Since the control of the machine involves some internal state, the CONTROL module is provided with a private memory, the CONTROL MEMORY.

The structure of the SUBSTRATE model is shown in figure 5.

6. SYSTEM/SUBSTRATE INTERFACE

We have now completed the development of the model for a computer system. It consists of three layers: the ENVIRONMENT, the SYSTEM, and the SUBSTRATE. Interaction between the SYSTEM and the SUBSTRATE is studied by representing the SUBSTRATE as an IPM. We will take a closer look at this interaction by considering the execution of a typical intstruction, and then give a definition of the SYSTEM/SUBSTRATE interface.

1. Previous instruction execution completed.
2. The CONTROL module fetches the value of the instruction pointer from storage (exactly where the instruction pointer is stored depends on the architecture, e. g. in the case of the Intel iAPX 432 it is in an internal register, which will be represented

by the CONTROL MEMORY [1]). The value of the instruction pointer is transferred to the INSTRUCTION module, and the INSTRUCTION module is signalled to read the indicated instruction.

3. The INSTRUCTION module reads and interprets the instruction and passes the resulting symbols to the CONTROL module.
4. The CONTROL module decides on the sequence of primitive functions necessary to· execute this instruction, and sequences their operation.

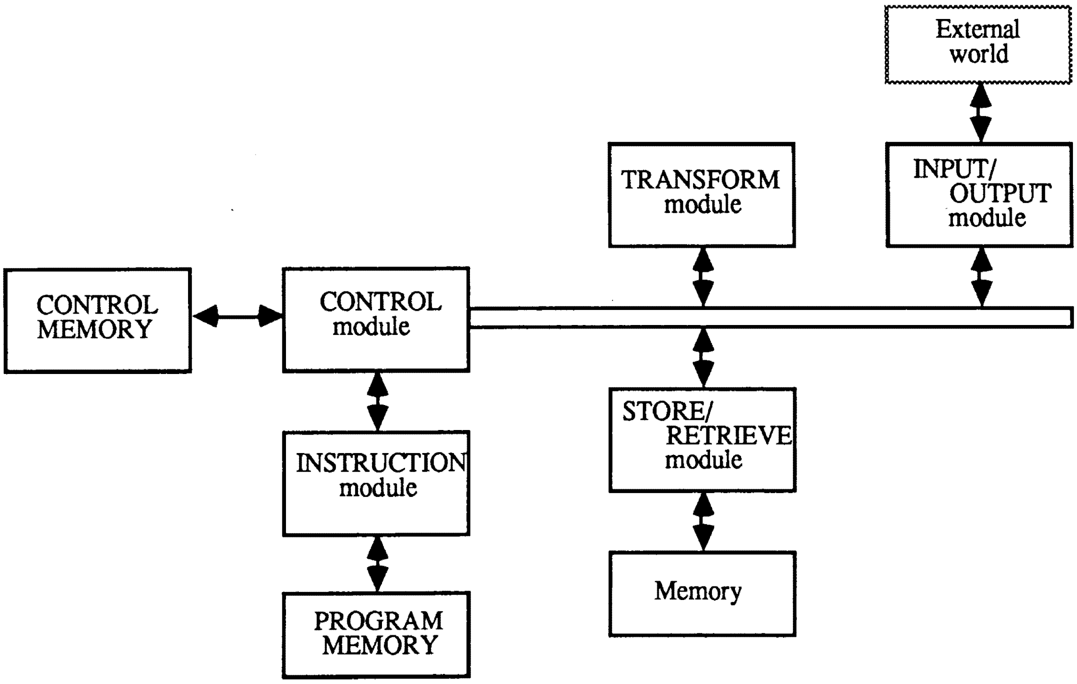

figure 5

The General Substrate Model

As far as the SYSTEM is concerned, the only information which exists is that which is explicitly defined in the instruction sequence. All information crossing the SYSTEM/SUBSTRATE interface has to be defined by the SYSTEM. This information can either be resident in the storage (and therefore in the SUBSTRATE), or else can enter the SUBSTRATE through the INPUT/OUTPUT module. Consider, for example, the following two Pascal statements (assume both **source** and **destination** are integers):

```
read (source);
destination := source;
```

The first instruction orders the SUBSTRATE to move an integer value from the INPUT/OUTPUT module to the variable **source** in the STORE/RETRIEVE module. The CONTROL module does this by getting the value from the INPUT/OUTPUT module, transferring it to the CONTROL MEMORY, and transferring it from there to the STORE/RETRIEVE module. The value is moved from one substrate module to another through the CONTROL MEMORY. This movement is controlled by the instructions of the SYSTEM, and since the value can be transferred to any other module once it is in the CONTROL MEMORY under indirect SYSTEM control, we consider the CONTROL MEMORY interface as part of the SYSTEM/SUBSTRATE interface.

The execution of the next instruction proceeds in a similar fashion. The CONTROL module transfers the value from the variable **source** in the STORE/RETRIEVE module to the CONTROL MEMORY. This means that information crosses the interface from the SUBSTRATE to the SYSTEM. In the next phase the information is transferred from the CONTROL MEMORY to the variable **destination** in the STORE/RETRIEVE module. This once again represents information moving from the SYSTEM to the SUBSTRATE.

The interface can now be defined:

1. Information is communicated from the SYSTEM to the SUBSTRATE when
   a. instructions and/or other information is read from the PROGRAM MEMORY into the CONTROL module.
   b. information is transferred from the CONTROL MEMORY to any other module.
2. Information is communicated from the SUBSTRATE to the SYSTEM when it is transferred from any module to the CONTROL MEMORY.

7. APPLICATION OF THE MODEL

This model can now be used to find the failure modes of a specific SUBSTRATE, and to investigate the effects of the identified failures on the SYSTEM. The first step in using the model is to organise the computer system so that a particular level is isolated for analysis. This results in the three level structure of ENVIRONMENT, SYSTEM, and SUBSTRATE.

The SUBSTRATE is then fitted into the model for a general SUBSTRATE. This is done by identifying the functional abstractions available at the SUBSTRATE level and matching them to the modules of the general SUBSTRATE model.

The SUBSTRATE modules are then analysed individually. The modules are very simple, and it is possible to exhaustively enumerate the failure modes of each module. The effects of each failure mode on the SYSTEM behaviour is found by looking at the functional permutation introduced by the specific failure mode.

An example of such an analysis performed on the Intel iAPX 432 can be found in [1].

8. SUMMARY AND CONCLUSIONS

Although many models have been advanced for the study of fault-tolerant systems, they are directed at classifying systems, their failures, and the possible causes for these failures. These models have a limited application to the design of fault-tolerant systems.

In this paper we described a model which was developed to assist with the structured design and analysis of multi-level fault-tolerant computer systems. This model provides a step towards a more theoretical approach to the design of these systems.

The model is intended as a qualitative tool for the design and the failure-mode analysis of fault-tolerant systems. It can be used for the prediction of the failure modes of a level in a hierarchical system, and for determining the effects of these failures on the higher level.

REFERENCES

1. Goosen, H. A., [1984], *A Model for Fault-Tolerant Computer Systems*, M.Sc. Dissertation, University of the Witwatersrand, Johannesburg, October.
2. Avizienis, A., [1978], "Fault-Tolerance, The Survival Attribute of Digital Systems," *Proceedings of the IEEE*, **66**, 10, October, pp. 1109 - 1125.
3. Randell, B. et al, [1978], "Reliability Issues in Computing Systems Design," *Computing Surveys*, **10**, 2, June, pp. 123 - 165.
4. Horning, J. J. and B. Randell, [1973], "Process Structuring," *Computing Surveys*, **5**, 1, March 1973, pp. 5 - 30.
5. Myers, G. J., [1982], *Advances in Computer Architecture, 2nd Edition*, Wiley Interscience.
6. Robinson, A. S., [1982], "A User Oriented Perspective of Fault-Tolerant System Models and Terminologies," *Proceedings of the 12th International Symposium on Fault-Tolerant Computing*, pp. 22 - 28.