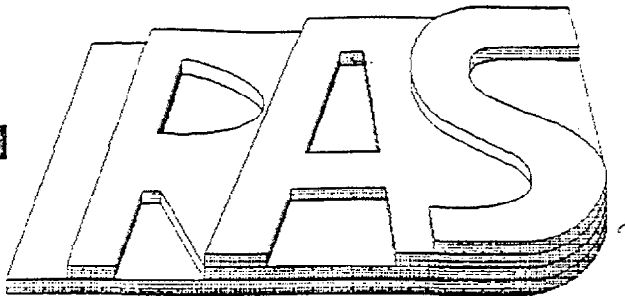
BRITE/EURAM



Document Type:

Document Number: IRAS-DUT-CEC-KG-036b

Project Partner: DELFT

Recipients: CEC, PTA, ALL

Date: June 21st 1995

Authors: A. Bossche, G. Kócza

No. of Pages: 13

Title

IRAS Synthesis Report

February 1st 1992 to May 1st 1995

A. Bossche,
G . Kócza

Abstract

This report is intended to show the results of the IRAS project (BRITE/EURAM 4250), which started on February 1st 1992 and has finished on May 1st 1995. The purpose of this project was to develop fast and reliable tools for risk analysis and real-time fault location in complex systems as they are encountered in different branches of industry. The innovative features of the research project were:

A consistent and compact modelling procedure for creation of component and system models that shows all fault initiation within the system and fault propagation through the system.

An expert system (IRAS: Integrated Reliability Analysis System) that enables designers and engineers m model their systems even when they are not reliability experts.

Using the results of the research on the hierarchical modelling and automated Fault Tree synthesis and analysis, a computer program has been developed for automatic Failure Mode Effect and Criticality Analysis (FMECA) and Real-Time Fault Location (RTFL), using a menu based Model Builder, which allows fast creation of new component and system models and modification of existing models. The prototype software has implemented and applied to a pilot system, a large Rolling Mill Driving System for steel production.

Introduction

Up to the present. although computers have been used for automating certain tasks. Fault Tree Synthesis and Analysis (FTSA) and Failure Mode Effect Analysis (FMEA) have been performed in a manual or (at best) semi automatic manner. This can be a very time consuming and tedious task especially since identical and near identical paths will have to be developed multiple times in the case of reasonably sized industrial applications. Moreover obvious causes are often overlooked. Therefore, it would be convenient if a computer could take over the dull part of the work. In the past several attempts have been made to computerize fault tree construction, however, their effectiveness is strongly limited as a result of insufficient detailed component models. The most important shortcomings in previous modelling were:

Component models describing fault propagation in one direction only. This means that r-he component models depend on the system variable being analyzed (in case of RTFL) or on the basic event being analyzed (in case of FMEA). Furthermore the automatic detection of control loops is also a particularly complicated task.

The development of one single deviation in a process parameter only will make it impossible to take account for the effects of control loops. Although feedback loops can be detected when a variable appears twice in one path. the effect of this loop will be unknown since the loop's active control range is not included in the path. Feedforward loops consisting of two paths introducing deviations of **opposite** sign (shutdown loops) can *not* be detected at all. In order to be able to detect control loops propagation pubs should be developed for any possible deviation of the system variable being analyzed. One draw back of these models is that they show only fault propagation in one direction i.e. directed to the top-event being analyzed. Since a thorough system analysis usually comprises the development of multiple events it also

involves frequent adaptation of component models as well.
Recent studies have shown that compact component models describing hi-directional fault propagation (upstream and downstream) can be developed. This means that the component models do not depend on the system variable which is being analyzed and that they can serve as a basis for automatic development of fault propagation paths for Failure Mode Effect Analysis (FMEA) and (real-time) Fault Location.

In the Brite/Euram project BE-4250 seven partners (3 universities, 2 research institutes and 2 industries) have collaborated in the development of fast and reliable tools for reliability and risk analysis of, and real-time fault location in, production systems as they are encountered in industry.

First a compact and consistent modelling procedure has been developed. for creation of component and system models. Each component is modelled as a black box with in-and outputs reflecting the physical connections to other system components and environment. Each connection might be associated with multiple physical variables, e.g. one connection through which a fluid passed to another component might be associated not only with flow and pressure but with flow temperature and flow mixture as well. The modelling procedure allows hierarchical modelling, i.e. the black-box models reflecting subsystem performance on a high level may be split-up in smaller parts again. System models describe the interconnection of the components. When defined these models allow virtually automatic FMECA and RTFL.

As a practical evidence that the theory is correct, an expert system prototype, IRAS has been designed and implemented, based on the results of the project development. IRAS enables designers and engineers to model their systems even when they are not reliability experts. Computer programs have been developed for automatic FMECA and RTFL.

Technical description

The IRAS system architecture

The basic structure of IRAS is shown in Figure 1. It shows the different IRAS functions in relation to the IRAS environment. As can be seen the IRAS system offers the user three main functions, a Model Builder, an FMECA tool and a RTFL tool. The Tree Generator is a facility function that constructs Cause Consequence Diagrams, Fault Trees and Causal Trees on request of the FMECA and RTFL functions.
The "Model Builder" is a menu-based computer programm that allows fast creation of new component and system models and modification of existing models. The questions asked to the user are straight forward and simple, requiring only a minimal expertise from the user. The user should have just enough knowledge about the physical behaviour of the component in order to be able to predict the fault propagation within the component. While only a shallow system knowledge is required to decide which variables are important for neighbour components and which are not. The normal procedure is to develop the component models first. Here, the user will be asked to define:

* the number of input/output connections of the component including undesired connections with other components due to environmental coupling.

* the number and type of all important physical variables associated with each connection.
* the fault propagation relation describing the way in which the physical variables affect each other or, in case of induced component failure, the effects on the, component state.
* the fault initiation relations that describe the effect of component failures upon variables and relations between variables.

The questions asked to the user are straight forward and simple requiring only a minimal " expertise from the user. The user should have just enough knowledge about the physical behaviour of the component in order to be able to predict the fault propagation within the component. While only a very shallow system knowledge is required to decide which variables are important for neighbour components and which are not.

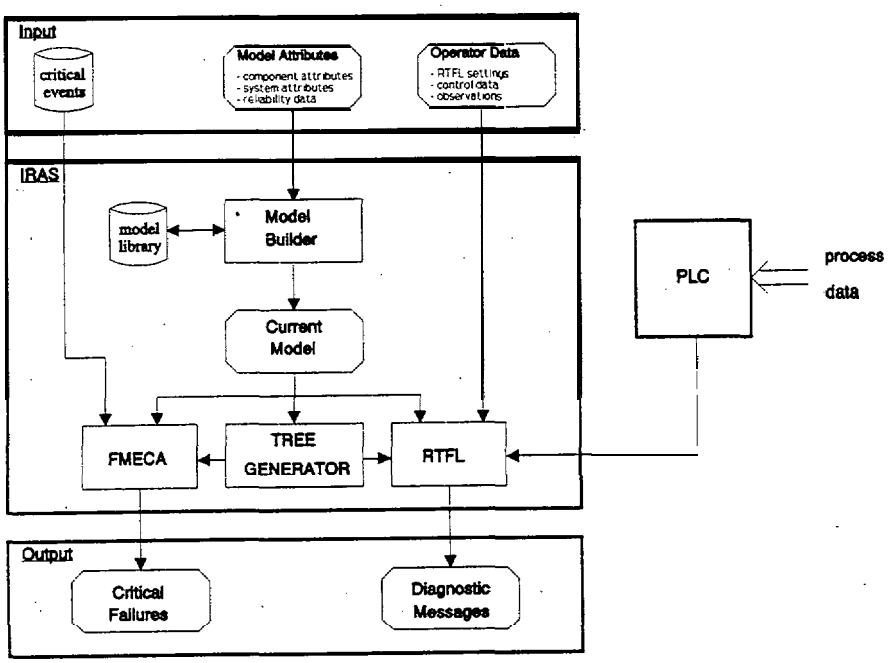


***Figure 1**:The IRAS architecture*

Creation of a system model is even more simple and comprises the definition of the system's infrastructure (component interconnection scheme) and the definition of the systems input and output connections to other systems and environment.

Using the fore-mentioned component and system models the FMECA program is able to develop cause-consequence paths for all component failures or other basic events. Starting at an initial event all propagation paths of the corresponding cause-consequence diagram developed simply by connecting the relations from component and system models in the right manner and for the appropriate parameter values. Subsequently the FMECA module searches the cause-consequence diagram for the presence of control loops (feedback and feedforward loops) that might prevent the initial event from further unconditioned propagation along the paths effected by these loops. Then, when the cause-consequence "diagram is adapted for the compensating actions of these loops, it contains all possible effects that the initial event might

have on the system and the FMECA module presents a list of affected system variables to the user. Furthermore, when the program is also fed with the essential deviations in system parameters and their respective criticalities the FMECA module may perform a Criticality analysis as well. As an output the user gets information on critical component failures, which have to be prevented by appropriate design measures. Finally, the FMECA program allows the user to manipulate the presentation and ordering of the effect and criticality data according to individual demands.

The RTFL function is able to extract all faults and fault combinations which are most consistent with the set of measured variables even when some sensor circuits provide faulty information. For this reason it has a build-in monitoring function that monitors the values of important process parameters which subsequently are compared with a vector of expected values. When the process information shows intolerable deviations an alarm signal is given to the operator and, a fault location procedure can be started either automatically or on operator command. Such a fault location action proceeds as follows. First this algorithm uses the above mentioned component and system models to create causal trees which describe both, all basic events (and the corresponding propagation paths) that influence the top variable and the resulting effect on this variable. This in contradiction to fault trees which show the basic events for one effect only. The causal trees comprise the propagation of both positive and negative deviations and, therefore, allow the automatic detection of feedback and feedforward loops that might prevent some faults from reaching the top-event. The RTFL programm adapts the causal trees so that the compensating effects of control loops are included,
Since the system and component models describe fault propagation by indication of cause-consequence relations and their priorities (probabilities), the corresponding diagnostic (anti-causal) probabilities can be calculated. For each potential failure cause inferred by the system the number of measured parameter values inconsistent with this cause is determined. Failure causes having too many inconsistencies are considered as probably not being responsible for the fault and consequently are deleted. Sensors may have different reliabilities which can be represented by probabilities for malfunction of these sensor circuits. In case the data available from control sensors is" insufficient for adequate fault location, additional diagnostically relevant data may be collected either by placing additional, auxiliary sensors for diagnosis or by direct observation of human operators.

The Tree Generator is a processes, which is not directly connected with the user. It is connected wit FMECA and RTFL, and it gets the commands from those two processes and gives respond to them. However, this so-called slave process has its importance to the IRAS, since it generates the Trees, which are the basis of" each reliability analysis. The Tree Generator should work very fast in case of a RTFL request, because the on-line fault detection would not be possible otherwise. The main task of TG is to generate Trees upon request of the different processes, which it is connected with. TG has 3 subprocess:

- Causal Tree Generator
  Fault Tree Generator
  Cause Consequence Diagram Generator

Each subprocess is controlled by a central TG main process, which is communicating with the RTFL and FMECA, and always scanning a corresponding socket channels, waiting for a Tree Generation request, If a request is detected, the main process, reads the actual Model ""

Data Base to memory, and initialises the requested tree with the root event, and activates the proper subprocess. When tie Tree is ready, We TG controller sends a ready message to tie caller process, and sends the generated Tree to it. After that, it retires to the background, and scans the socket channels for new requests.

General model description

In designing the model structures and modelling procedure several considerations were made such as: how systems are usually described (hierarchy of main system, subsystems and components), how faults propagate through a system and how fault propagation is affected by different operational modes.

Hierarchical modelling

Since fault propagation usually occurs through the 'normal' physical interconnections between components (or between subsystems) the fictional block diagrams of a system, which split up the system in clear subfunctions and interconnecting flow paths, could be used as a basis for the development of the IRAS fault propagation models. In these functional diagram items usually are depicted as black boxes with its in-and outputs reflecting the physical connections to other components and to the environment. The black boxes may represent both basic components as well as complete subsystems. Complex systems usually are described by multiple functional block diagrams i.e. one for the overall system function and others describing the functions of subsystems to more detail.

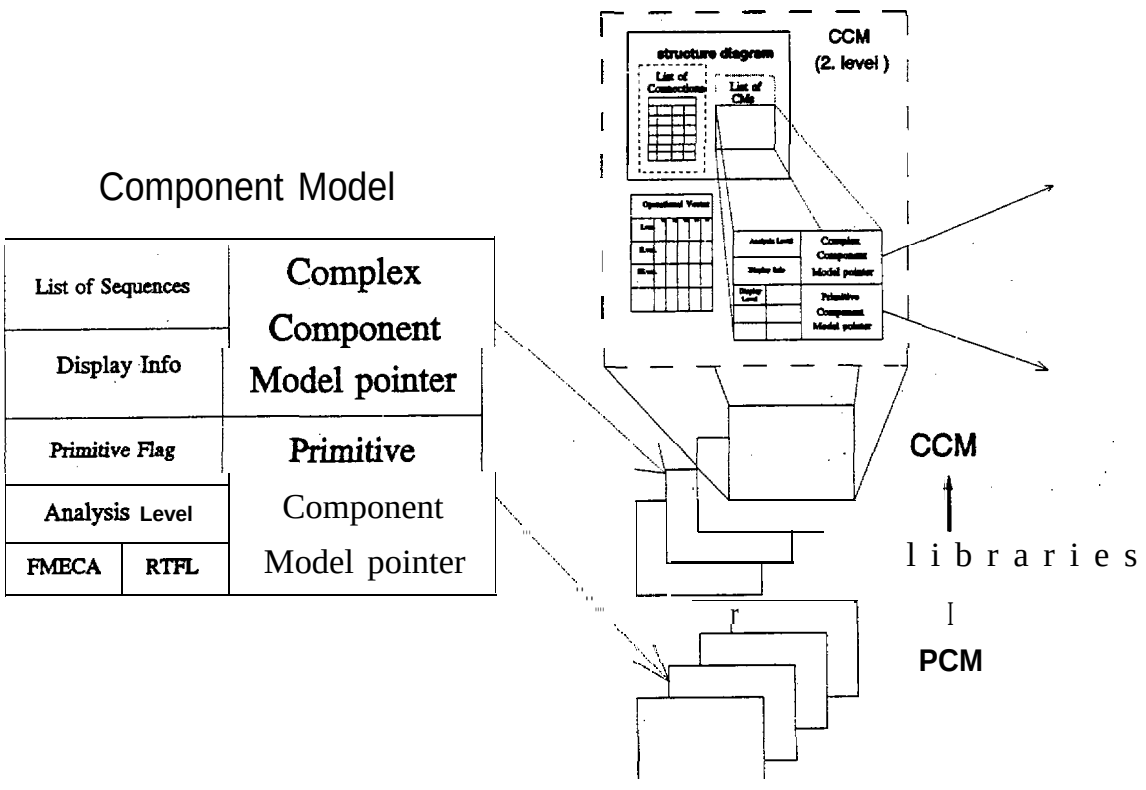


Figure 2: *System Model Structure*

The overall functional diagram of the system that is to be modelled usually is a good basis for the overall Fault Propagation diagram. Then in subsequent steps the subsystems in these models can be modelled to more detail using the lower level functional diagrams as a basis for this refinement. The created IRAS models support this type of hierarchical modelling.
The Model Builder allows the user to create a *Component Model (CM),* and define gates at

the border of each *CM* (black boxes on the screen). Each gate can contain multiple Variables, and those gates, which have common Variables can be connected in the system diagram. This component hierarchy allows a logic partitioning of systems. The *Component Models can* not only be connected with each other, but with *Environments* as well. The *Environments* represent either environmental interactions (desired or undesired) with the System model at the given level, or connections with the system level above. This hierarchical way of modelling simplifies the reuse of (subsystem models that were defined in a previous model.

Component Model (CM)

Each *Component Model can* be associated with two models (see figure 2):

A *'Primitive Component Model'* which describes fault and level propagation between in- and outputs of the Component Model.

A 'Complex *Component Model'* which describes the Component Models as a compound structure of smaller items.

Each *Component Model (CM)* may have both models associated, however, in analysis only one can be active at a time. In addition to the pointers to the respective *PCM* and *CCM* libraries each *CM* contains information on which model will be used for analysis and which will be displayed and how it will be displayed:

The *Analysis Level* shows which pointer is valid for the current analysis.'

The *Display Info* contains information on size, orientation, position and gates of the CM.

Main system models usually have only a *Complex Component Model* (*CCM*) associated while the smallest replaceable units of a system usually will have only a *Primitive Component Model* (*PCM*) associated. The intermediate sized units may have both representations whereby the *Primitive Component Model* (*PCM*) *can* be activated for high level (less detailed) analyses and the *Complex Component Model* (*CCM*) can be chosen for lower level (more detailed) analyses,

Complex Component Model (CCM)

A *CCM* consists of a *List of CMS* and a *List of Connections. The* model is completed with a *List of Operational Vectors (see* figure *2). The List of Connections* describes the:

interconnections between the CMS in the *CCM*

the CM connections with the *Environment. Three* different types of environmental variables can be distinguished;

-*I*: describing desired interaction with the environment such as process inputs and outputs,

-*E:* describing undesired environmental interactions such as leakages or excessive ambient temperatures,

- *U*: describing the *CCM* connections with the upper system level.

The *Structure Diagram* is a graphical representation of a *CCM*. The user can *see on* the screen how the system looks like and which *CMS are connected* to each other. In this manner it is easy to follow signal propagation through the system. The general structure of a *Structure Diagram* can be seen on Figure 3. The *List of Operational Vectors* describes the sequence of expected (abstract) parameter levels and the corresponding expected (real) values, their deviation tolerances, optional initial (transient) value and the time constant

during normal operation required by the variable to settle at its expected value. Since not all transitions of parameter values occur instantaneously the *Operational vectors* can contain timing information as well. Each vector is either time driven& event driven. In the "first case the time-interval in' which the transition to the next vector is expected must be specified.

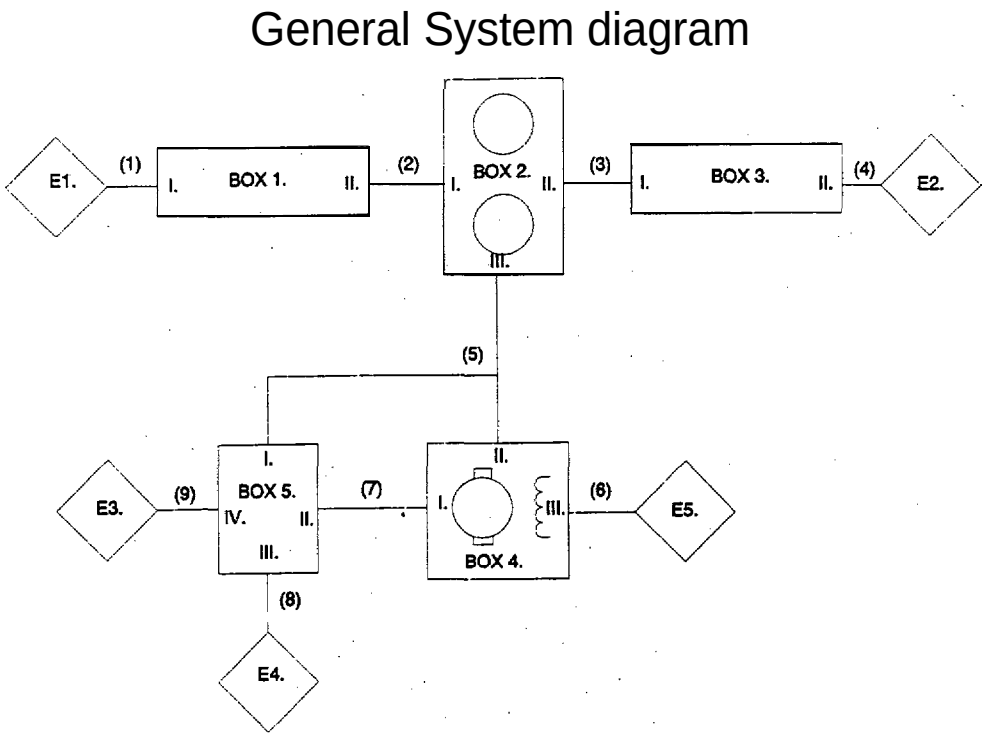


Figure 3: *Main System Structure Diagram*

The structure of a *CCM* is identical to that of the main system level model, i.e. it consists of a *List of Connections,* a *List of CMs* and *Operational Vectors* for the internal parameters of the subsystem. This similarity in structure allows subsystems to contain other subsystems at the more detailed level, etc.

Primitive Component Model (PCM)

A *PCM* contains the:

Level Propagation Model with the List of Gates and variables and their descriptions, and the *Level Matching Relations*

Fault Propagation Model with the Component States and *their Probability Data* and with the *Special events* as can be seen in Figure 4.

Operational Modes and Operational Levels

Usually a production system can be used at different production levels (Full production, reduced production, idle, etc.). This means that the components of the system may be used in multiple modes as well where each mode may imply different conditions for the fault propagation through the component. The effects of input levels on the operational mode of a component and on the propagation of faults must be reflected in the models.

The steady-state operational mode of an item (system, subsystem or component) is defined completely by the desired levels of all its I/O variables (including control signals) or by the sequence of those variables. Often, however, due to delay times or time constants in the items, I/O variables may change again before all item variables have settled at their constant level after the last change. During these transitions in operational mode the vector of internal variable levels, which is used for comparing in RTFL, is subjected to multiple changes until all levels have settled at constant values.

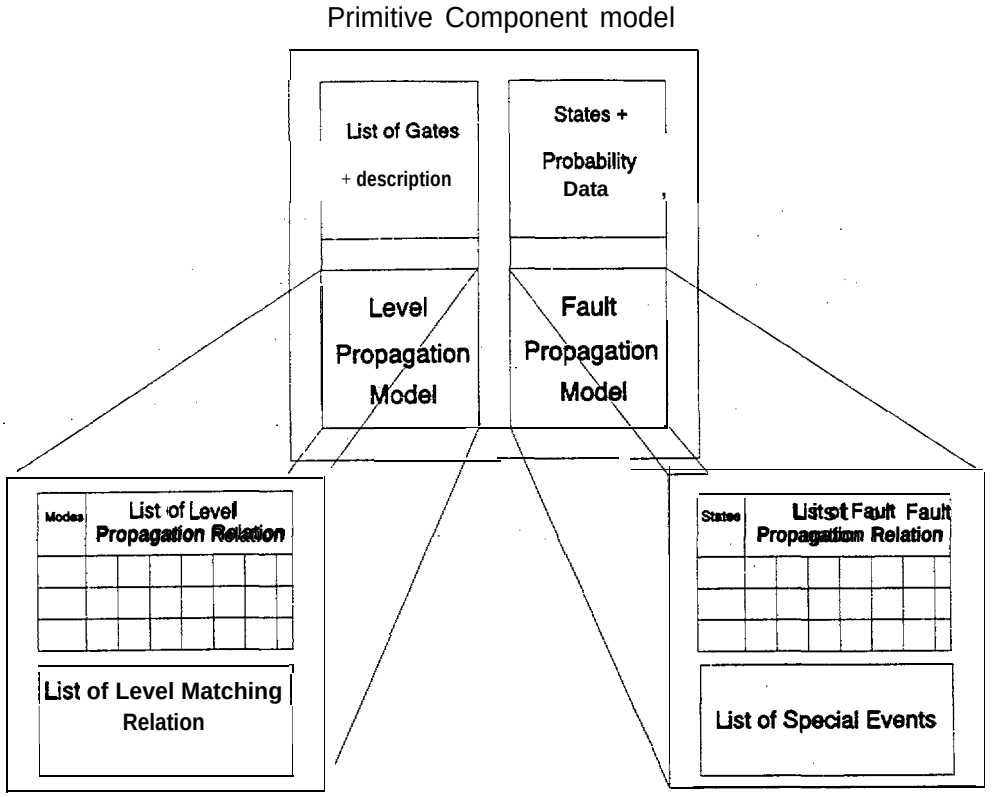


Figure 4: *Primitive Component Model Structure*

For FMECA the requirements are less stiff, however also there it must be able to know the different operational modes of the components in order to predict their reaction on incoming deviations and subsequent fault propagation. The operational modes, operational levels and the level propagation is laid down in so-called Level Propagation Models (LPM) which describe how operational levels propagate from the inputs to the outputs of the black boxes and how the internal state of the corresponding components or subsystems may change as a result of a transient in an input level.

Level Propagation Model

The *Level Propagation Model* contains of a set of *Level Propagation Relations* describing level propagation between input and output variables and a set of *Level Matching Relations* that match the internal parameter levels with. the external parameter levels of the component. The *Level Propagation Relation* consist of a *cause, an effect* and (optional) *conditions, which* all are either variables with one or more level indicators or substates of the component. These relations can be used to describe:

how (operational) changes of input levels affect the levels of output variables.
how a change in input level may cause the component to switch from one internal mode to another.
how a change in internal mode of the component changes the levels of output variables.

Important *delay times* and *time constants can* be stored in the relations as well.

The *Level Propagation Relation* also contains a list of *Level Matching Relations* that matches internal and external interpretations of parameter levels and their deviations. This because the component model taken from the general library may be created by another person (and even for another system) than the person modelling the current system.

The effects of input Levels on the operational **Mode** of a component usually is limited to a few ranges of Levels. Therefore, only a limited number of ranges of operational Levels is distinguished in modelling:

Zero Level:	Z
±Low Level:	±L
±Moderate Level:	±M
±High Level:	±H
±Very High Level:	±VH

Faults and Fault Propagation

The proper performance of a system may be disturbed by causes from within the system (component failures) and from the outside (failures in input variables or environmental effects). In both cases the original failures will initiate one' or more deviations in system variables which then propagate further through the system. On the other hand extreme deviations in a variable may cause components to fail (secondary failures). Fault initiation and fault propagation is assumed to take place inside the black boxes (component models) and is described by so-called 'fault propagation relations' which may describe:

deviation propagation from one variable to another.

-a component state (failure) initiating a variable deviation (or a special event).

-an extreme variable deviations (or special event) causing a transition' in component state (usually from a good state to a failed state).

Since fault propagation may depend on the operational levels of the input variables, on the internal failure mode of the component and on deviations in other variables, the relations can be conditioned for these dependencies.

Special events are those events that can not be seen as simple parameter deviations, often these are so-called trigger events. Examples of such events are: a lightning stroke that triggers the failure of a component or a component failure that under unfortunate conditions may cause an explosion.

Fault Propagation Model

The ***Fault*** *Propagation Model* contains *Fault Propagation Relations* (FPR's) for the normal and faulty *states* of a component, where the normal states may have substates representing the different operational modes, An ***FPR*** contains a cause and an *effect* which are either component variables or component states or special events, a *condition* which can be also a variable or a component state or a special event, a *propagation factor,* a *time delay,* a *time constant,* and (optional) ***probability*** *data* that can be assigned to a relation when the occurrence of the *effect* is not absolute certain even when all *condition* are met.

In the **IRAS** models limited number of **Deviation Levels are** used to describe the differences from the expected values. **Deviation Levels** are:

very high	$\delta = +100$
high	$\delta = +10$
normal	$\delta = +1$
low	$\delta = +1/10$
very low	$\delta = +1/100$
low reversed value	$\delta = -1$ o
large reversed value	$\delta = -100$

This set allows fault propagation to be modeled in sufficient detail for most applications such as FMECA and RTFL. In some cases even a smaller number of Levels will suffice, e.g. when a **Variable Level is normally** Zero the **Deviation Levels +1/10** and +1/100 can be omitted. **Furthermore,** for **Variables** such as temperature or chemical concentration where a reverse value is physically impossible the reversed values can be neglected. **Deviations** always are defined relative to the desired **operational Level.**

Results

The above described procedures are implemented into a prototype software for Integrated Reliability Analysis System **(IRAS)** and applied to a pilot system of sufficient complexity to demonstrate the essential possibilities of the IRAS system. Therefore, a large Rolling Mill Driving System for steel production with a **(semi-)continuous** product flow has been chosen as the pilot system. This Mill includes many analog and digital signals and multiple control loops and **forms,** from a complexity point of view, an ideal test platform for **IRAS** system.

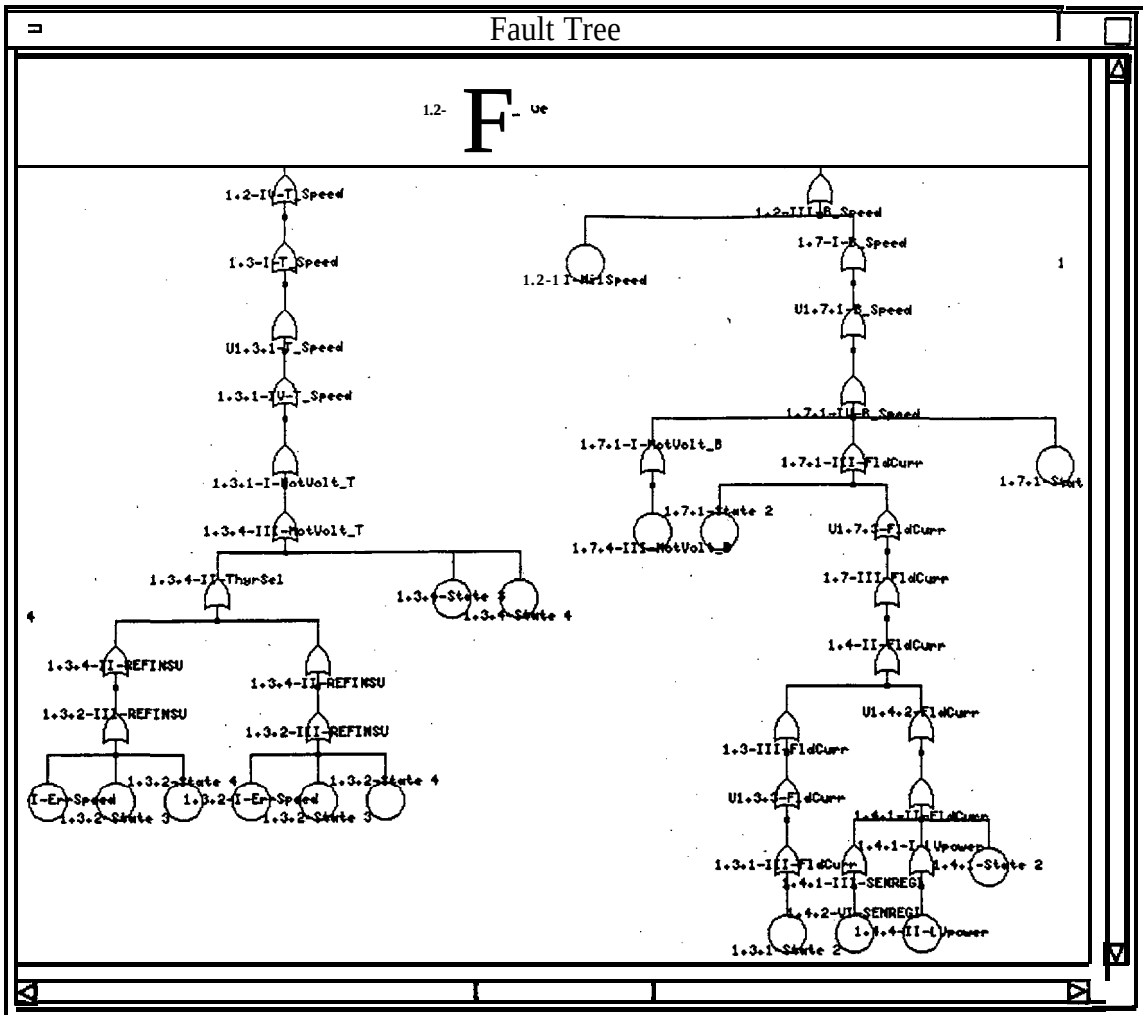


Figure 5: *A part of an IRAS generated Fault Tree*

A Model Library has been developed for this steel mill with several hierarchical levels, and many Primitive and Complex component models. The IRAS has been tested with that Model Data Base, and has generated several Fault Trees from 25 upto 5666 Tree Nodes, depending on the Root Event. Some IRAS generated trees can be seen in figure 5-7.

The prototype is a pre-competitive in nature since the software and library of models have been designed especially for the demonstration projects, however, the underlying modelling procedures and techniques allow a much wider application area. Therefore, since the IRAS prototype has been successfully applied to the pilot system(s), future generalisation and commercialization of IRAS should "not be difficult.

Conclusions

The developed Model Structure and Modelling procedure are suited to model fault initiation in, and fault propagation through, (semi) continuous production systems with a limited number of operational modes.

However the Model Library is only created for the prototype software, it contains all

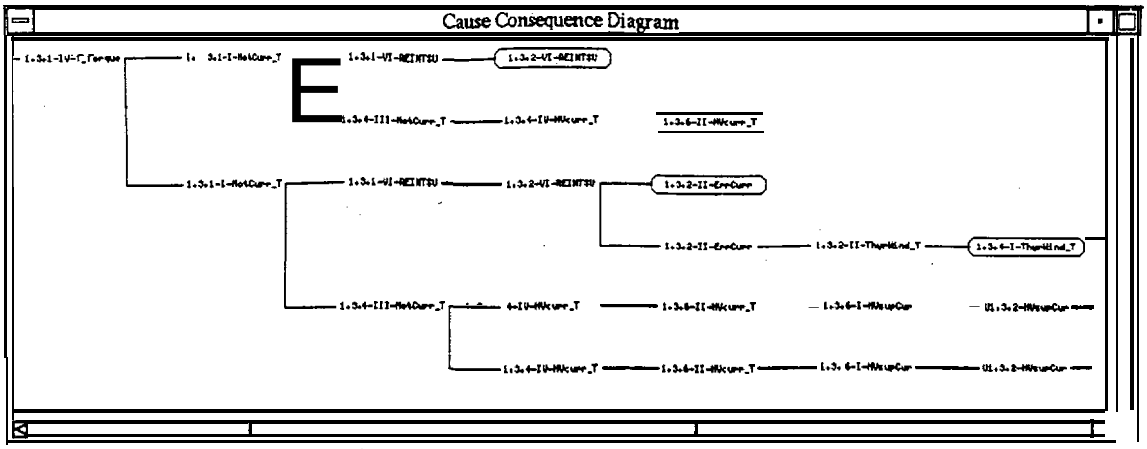


Figure 6: A part of a Cause-Consequence diagram, generated by IRAS

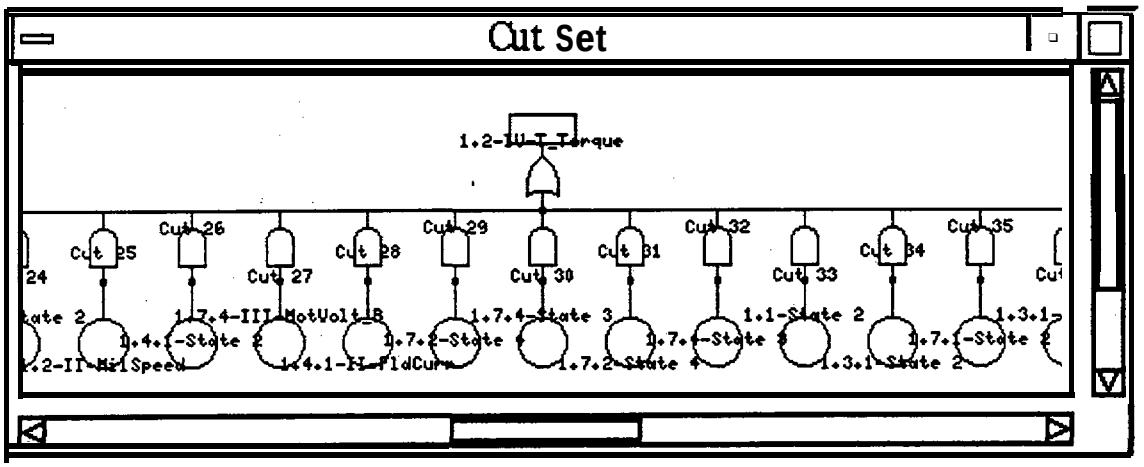


Figure 7: A part of a Cut Set, generated by IRAS

information necessary to build fault trees, causal trees and cause-consequence diagrams thus offering the possibility for computer-aided reliability analysis and real-time fault location. The IRAS prototype has proved, that automated Fault Tree synthesis and computer-aided analysis is possible using the developed Models and Tree Generator routines. The graphical Model Builder is a very useful tool for modelling production systems, as it decreases the time of creation and avoids many inconsistencies of the Model. RTFL contributes to a quick recovery, in case of a failure of complicated technical systems. Other potential applications that were not evaluated within the IRAS project, may be:

- risk assessment for environmental protection,
- quality control,
- preventive maintenance, based on fault diagnosis.

All these applications can use the same type of model and the same RTFL algorithms, and may actually share one computer if applied to the same technical system. The differences lie in the variables observed, in the definition of components and flows, and in what constitutes an 'event'.

Acknowledgement

The authors wish to express their gratitude to the representatives of the other partner organizations in the project. Furthermore the Commission of the European Community is acknowledge for their financial support to this IRAS, Brite/Euram 4250 project.

References

A. Bossche, G. Kócza and J.R. Mollinger (1993), *Fault propagation models for reliability assessment of production systems,* SRE symposium, Reliability: A competitive edge, Arnhem, The Netherlands, pp. 385-397.,

Bossche, A. (1991), *Computer-aided fault tree synthesis III,* Reliability Engineering and System Safety, Elsevier Applied Science; Vol. 33, pp. 161-176.

Bossche, A. (1991), *Computer-aided fault tree synthesis II,* Reliability Engineering and System Safety, Elsevier Applied Science, Vol. 33, pp. 1-21.

Bossche, A. (1991), *Computer-aided fault tree synthesis I,* Reliability Engineering and System Safety, Elsevier Applied Science, Vol. 33, pp. 217-241.

Bossche, A. (1988), *Fault tree analysis and synthesis,* PhD-thesis, Delft University of Technology, The Netherlands