Prototype Design Tool

Borch, Ole; Blanke, M.; Buck, J.; Bagnoli, F.; Lootsma, T.F.

Publication date:
1998

Document Version
Også kaldet Forlagets PDF

ATOMOS II

Prototype Design Tool
Task 2.6.3

Contract number: WA - 95 - SC.205
ID-code: A226.03.03.052.001
Date: 1998.12.04

CLASSIFICATION AND APPROVAL

Classification: Public after Review

DEFINITIONS:

Public after Review:

This document may be freely distributed after successful EC review, given EC permission. Publication is governed by the EC Contract and the ATOMOS II Consortium Agreement.

Confidential for the Duration of the Project:

As for "Confidential", but only for the duration of the Project. After final Project Approval by the EC, status for reports classified "Confidential for the Duration of the Project" is automatically downgraded to "Public".

Confidential:

This document is for use of the ATOMOS II Contractors within the ATOMOS II Consortium as regulated by EC Contract No. WA-95-SC.205, and shall not be used or disclosed to third parties without agreement by the ATOMOS II PMC and subsequent EC approval/agreement.

Authors:

Ole Borch[1], Mogens Blanke[1], Jakob Buck[1], Fabio Bagnoli[2], Tako Lootsma[1]

[1] Department of Control Engineering, Aalborg University
[2] d'Appolonia

Name:	Signature:	Date:
Ole Borch	_____	_____
Mogens Blanke	_____	_____
Jakob Buck	_____	_____
Fabio Bagnoli	_____	_____
Tako Lootsma	_____	_____

Approved by task leader _____ _____

Mogens Blanke

Approved for issue as per above classification

DOCUMENT HISTORY:

Issue:	Date:	Initials:	Revised pages:	Short description of changes
001	21 Oct 98	OB		First issue
002	09 Nov 98	MB	i-iii, chapters. 2,3	Front matter revised, References added in Ch.2 Chapter 3 added
003	10 Nov 98	OB	4,5	Merging, front and caption matter revised. Ch.1 out.
004	11 Nov 98	GM-MB	all	Different amendments
005	30 Nov 98	OB		5.1, 5.2 updates
006	02 Dec 98	OB,TL		Figures updated: 5, 11, 13, 15
007	03 Dec 98	OB		Appendix for component figures added.

CONTENTS

1. ABOUT THIS DOCUMENT

1.1 Global objectives of the research in task 2.6

The overall purpose of the research in task 2.6, *sensor fusion to enhance safety*, is to increase ship safety by preventing sensor faults from developing into failures or emergencies. The specific objectives are to obtain increased availability of machinery and navigation subsystems, increased dependability of sensor information presented to operators. A part of this is achieved if malfunction or unexpected shut down of machinery due to simple sensor faults could be avoided.

1.2 Objectives of subtask 2.6.3

The objective of subtask 2.6.3 is to design and implement a software prototype tool, which can aid the design and implementation of autonomous fault detection and supervision. The tool should automate the analysis of fault propagation in complex systems and support the user in selecting methods for sensor information fusion. In particular, the tool should aid the user by showing where fault accommodation could be applied to stop migration of serious faults.

1.3 Acronyms

ADM	Administration
API	Application Programmer Interface
FDI	Fault detection and isolation
FMEA	Failure modes and effects analysis
FMECA	Failure modes and criticality analysis
FPA	Fault-propagation analysis
FTC	Fault –tolerant control
GUI	Graphical user interface
JDK	Java development tool, Sun Microsystems Inc., version 1.1.7 or later
Swing	The JAVA Swing Kit , Sun Microsystems Inc., version 1.03 or later
PDT	Prototype design tool
UI	User Interface

1.4 References and related documents

Allasia G. (1998): Detailed Design Specification for the FMEA Module. ATOMOS II, Task 2.6.2, Part 2.

Bagnoli, F., Allasia G., (1998): FMEA Software Documentation and Demonstration of Prototype FMEA Software. ATOMOS II, Task 2.6.2, Part 3.

Basseville, M. and I. Nikiforov (1994): Statistical Change Detection. Prentice Hall, 1994.

Bell, T.E. (1989) :*"Managing Murphy's Law: Engineering a Minimum-Risk System"*. IEEE Spectrum, June 1989.

Blanke, M., R.B. Jørgensen, M. Svavarsson (1995): A New Approach to Design of Dependable Control Systems. Proc. 40. KoREMA, Zagreb, Croatia, April, 1995.

Blanke, M. S.B. Nielsen and R.B. Jørgensen (1993) : *Fault Accommodation in Feedback Control Systems,* in Hybrid Systems., Springer Verlag Lecture Notes in Computer Science vol. 736, October 1993, pp. 393-425. (ed.R.L.Grosman, A.Nerode, A.P.Ravn, and H.Rischel).

Blanke, M. and R.B. Jørgensen (1993): Reliability Related to Sensor and Actuator Interface in Machinery Systems. Aalborg University report R93-4016.

Blanke, M. and R.B. Jørgensen (1995): Fault Handling Design for Integrated Marine Systems. Proc. 3.rd IFAC Workshop on Control Applications in Marine Systems, CAMS'95, Trondheim, Norway, 10-12. May 1995.

Blanke, M. Consistent design of dependable control systems. Control Engineering Practice, Vol. 4. No.9, Sep. 1996, pp1305-1312.

Blanke M., R. Izadi-Zamanabadi, S. A. Bøgh, C. P. Lunau: Fault Tolerant Control - A Holistic View. Control Engineering Practice, May, 1997, pp 693-702.

Blanke, M., Izadi-Zamanabadi, R., Nilsen, S.O. (1997): Robust Fault Detection for Cargo Control System – Part 2. ATOMOS II, Task 2.6.1, Part 3.

Blanke, M. R. Izadi-Zamanabadi and T. Lootsma, (1998) Fault monitoring and reconfigurable control of a ship propulsion plant. Journal of Adaptive Control and Signal Processing. Vol 12 (7), 1998.

Buck, J., Borch, O. , Blanke, M. Lootsma, T. (1998). Prototype Design Tool – Overall Design. ATOMOS II, Task 2.6.3, Part 1.

Bøgh, S.A.: Design methodology for fault tolerant control-real life and case study. Ph.D. thesis. Dept. of Control Engineering, Aalborg University, Dec. 1997.

Franksen, O.I. (1978): Group Representations of Finite Polyvalent Logic - a Case Study Using APL Notation. Proc. IFAC World Congress, Helsinki, June 1978, pp. 875-887.

Galluzo M., P.K. Andow (1986): *"Reliability Analysis of Systems Containing Complex Control Loops"*. IFAC Proc. Symp. on Reliability of Instrumentation Systems. 1986, pp 47-52

Herrin S.A (1981).: *"Maintainability Applications Using the Matrix FMEA Technique"*.IFAC's Transactions on reliability, vol. R-30 No. 3, August 1981.

Jørgensen, R.B (1995).: *Development and Test of Methods for Fault Detection and Isolation: Theory and Practice. Ph.D. thesis,* Department of Control Engineering, Aalborg University.

Karnopp, D. and R. Rosenberg (1983): Introduction to Physical System Dynamics, McGraw-Hill.

Lege J.M. (1978): *"Computerized Approach for Matrix-Form FMEA"*. IEEE Trans. on reliability. Vol. R-27, No.1, 154-157.

More, T. (1981): Notes on the Diagrams, Logic and Operations of Array Theory. In: Structures and Operations in Engineering Management Systems, (eds.: Ø. Bjørke and O.I.Franksen) Tapir.

Nilsen, S.O., Blanke, M. (1996): Test Cases for Robust Fault Detection. ATOMOS II, Task 2.6.1, Part 1.

Nilsen, S.O. (1997): Robust Fault Detection for Cargo Control System – Part 1. ATOMOS II, Task 2.6.1, Part 2.

Nilsen, S.O., Blanke, M., Zuccarelli, F., Allasia, G. (1996): Overall Specification FMEA

Module. ATOMOS II, Task 2.6.2, Part 1.

Patton, R.J., P.Frank, D. Clarke (1989) Fault Diagnosis in Dynamic Systems: Theory and Applications.Prentice Hall.

Patton, R.J. (1993) Robustness Issues in Fault-Tolerant Control. Proc. Tooldiag, Toulouse, France, 5-7 April, 1993. Vol. 3.

Ulerich, N. H. and Gary J. Powers (1988): *"On-Line Hazard Aversion and Fault Diagnosis in Chemical Processes: The Digraph + Fault tree Methods"*. IEEE Transactions on Reliability, Vol.37, No.2, 171-177

Vries, Th. J. A. de (1994): Conceptual Design of Controlled Electro-Mechanical Systems. Ph.D. thesis, Universiteit Twente, NL.

Vukic, Z. H. Ozbolt and M. Blanke, (1998): Fault detection for Marine Systems. Brodogradnia (HR), Sep. 1998.

1.5 Standards Used

DIN 28004 Fliessbilder verfahrenstechnischer Anlagen, Teil 3, graphische Symbole, Mai 1998. Beuch-Verlag, Berlin, Germany

1.6 Contractual Aspects

This document is the final deliverable of task 2.6.3 of the ATOMOS II project, Contract No. WA - 95 - SC.205

1.7 Document Maintenance

This document is maintained by Aalborg University.

2. PURPOSE OF THE TOOL

The design tool is made to support novel design methodology that focus at systems availability and overall vessel safety when designing automation and monitoring systems. The method includes analysis of fault-propagation (FPA) as an important ingredient. This chapter introduces the method and provides the theoretical background for the algorithms used in the tool. A consistent method for design is first presented. It is based on analysis of component failure modes and their effects (FMEA). Schemes for fault handling are introduced to show which faults should be accommodated to maintain overall ability to operate the ship and its primary functions, thus maintain overall safety. The schemes are developed into an algebraic representation that is suited for computer aided analysis. The result is a computer-assisted methodology for engineering design, which presents the propagation of component faults and shows where fault handling can be applied to stop migration of a fault. Used consequently, the method offers a way to obtain significantly improved dependability of control and monitoring systems.

2.1 Introduction

Maritime automation technology has changed to become more complex and prime propulsion, ancillary and auxiliary systems have a high degree of automation. This has enhanced quality and efficiency in normal operation, but also made systems more vulnerable to faults. As a consequence, industrial attention has changed towards increased dependability. In the marine field, increased availability has the added benefit of increasing the safety level.

Advances in automation have provided integration of monitoring and control functions to enhance the operator's overview and ability to take remedial actions when faults occur. The manual supervision: to detect a fault, isolate it's cause, and accommodate the system to a new condition, has been much improved. However, system complexity and requirements to fast response to faults make it appealing to move the more basic supervision down from the operator to the automation level. To achieve this, plant supervision needs to be automated and become more autonomous.

This is technically possible with integrated automation systems as platforms, but new design methods are needed to cope efficiently with the complexity and ensure that the functionality of a software-based supervisor is correct and consistent.

Fail-operational systems, known from avionics and similar applications, are expensive in both hardware and development effort, and are prohibitive in cost for general maritime use. In the fault-tolerant approach, additional hardware should not be required and implementation costs be limited. The occurrence of faults can be tolerated but it should be prevented that they develop into failures at a subsystem or plant level. Furthermore, it should be guaranteed that all essential faults are detected and all critical faults are accommodated. The basic philosophy is to use existing sensors and actuators in an integrated system and make systematic use of any hardware and analytical redundancy in the available information. Component based fault analysis is shown to assure a much higher degree of completeness than otherwise achievable.

The central research effort concerning the methodology has been done in parallel with the ATOMOS II project, some machinery specific results were obtained in ATOMOS I, (Blanke and Jørgensen, 1994). The essential methodological results were published in Blanke, 1996 and Blanke, et al., 1997. The Ph.D. theses by Jørgensen, 1995 and Bøgh, 1997 present detailed designs of fault analysis and detection on a diesel engine actuator. Bøgh further makes some extensions to the analysis and design methodology and applies the principles to satellite attitude control.

The following sections detail the design method used as background for the ATOMOS design tool.

2.2 How dependability is obtained

Faults in one subsystem of an automated plant have often undesired effects on other subsystems if remedial actions are not taken after a fault occurs.

In present maritime automation, shut down functions and interlocks are used to prevent failures to dilate from one sub-system to another. The use of such functions has, however, the consequence that plant availability is sometimes reduced without good reason. With the ever-higher degree of automation, this has been the key cause to increased vulnerability to simple faults, particularly in sensors and actuators.

Dependability of a control system can be obtained by giving it ability to detect and isolate faults and react with actions that accommodate the control system to the fault. Fault accommodation is predetermined at the design stage: a control system can freeze to a safe state or the controller can be re-configured. This can be done e.g., by replacing the measured signal from a faulty sensor by an estimate obtained from remaining available signals, together with known analytic relations of the particular part of the plant.

2.2.1 Open and closed loop systems

Handling of faults in open loop systems is technically straightforward, but the reactions used to accommodate a fault need to be designed with careful consideration to safety and availability of the total plant. Optimization at a local level may easily violate an overall safety goal.

Handling of faults in closed loop components is a more difficult and challenging task. Properly designed systems can accommodate the effects of faults whereas less careful designs can let fault effects propagate to other subsystems.

2.2.2 Fault detection and fault-propagation

For the reasons given above, fault analysis needs to incorporate analysis throughout a system. Traditional methods for Fault Detection and Isolation (FDI) do not cover this problem. FDI algorithms can detect the presence of a fault as a difference between actual and expected behavior. Isolation of a particular fault requires a hypothesis about the observed effects from this fault. This is mostly obtained by ad hoc engineering and requires deep process knowledge and engineering skills to make a successful design. It is expensive in terms of both key personnel and time. (Bell, 1989; Galuzo and Andow, 1986).

Analysis of system reliability is mandatory for safety critical systems but is also more and more often used for the common ancillary and auxiliary systems. A major difficulty has been that earlier methods (Patton, 1993) did not ensure a complete description of all possible failure-modes of a system. The development of a consistent method for fault-propagation analysis (Blanke, 1996) helped overcome this problem. The method is component-based and possible component faults are identified at an early stage of design. This approach is feasible since failure mechanisms of industrial components are subject to constant study, and accumulated knowledge is available. The number of principally different components in maritime automation is small enough to make this a manageable exercise.

2.2.3 Fault-propagation analysis (FPA).

A method for fault-propagation analysis was described as follows (Blanke, 1996).

The systematic approach shall provide the following information:

1. *List of faults to detect,*
2. *Mathematical model for use in FDI,*
3. *Basic character and severity of each fault,*
4. *Required reaction to each fault.*

The details are elaborated in the following.

2.3 Component based analysis

2.3.1 The matrix FMEA method

A failure mode and effects analysis (FMEA) (Legg, 1978, Herrin, 1981, Yuan, 1985, Bell, 1989) starts with selection of the lowest level of analysis. In this context, this is sensors, valves, motors and similar components. All potential faults and their effects are determined. An FMEA scheme for each component shows how fault effects out of the component relate to faults at inputs, outputs, or parts within the components. This is illustrated in equation (1).

Using f_{ci} for component faults and e_{ci} for the effects, the FMEA scheme can be expressed as:

$$e_{ci} \leftarrow A_i^f \otimes f_{ci} \tag{1}$$

where A^f_i is a Boolean matrix representing the propagation. The index 'i' is a component identifier and $\otimes$ the inner product disjunction operator. The operation carried out by the operator is equivalent to the scalar Boolean disjunction "$\vee$" and the inner product to the "$\wedge$", i.e., row no. k of (1) is

$$e_{cik} \leftarrow (a_{ik1} \wedge f_{ci1}) \vee (a_{ik2} \wedge f_{ci2}) \vee \ldots \vee (a_{ikn} \wedge f_{cin}) \tag{2}$$

When some faults are effects propagated from other components, we get

$$e_{ci} \leftarrow A_i^f \otimes \begin{bmatrix} f_{ci} \\ e_{c(i-1)} \end{bmatrix} \tag{3}$$

System descriptions are obtained from interconnection of component descriptions. The description of a system with three components and open loop structure is

$$e_{c3} \leftarrow A_3^f \otimes \begin{bmatrix} f_{c3} \\ e_{c2} \end{bmatrix}; e_{c2} \leftarrow A_2^f \otimes \begin{bmatrix} f_{c2} \\ e_{c1} \end{bmatrix};$$

$$e_{c1} \leftarrow A_1^f \otimes [f_{c1}] \tag{4}$$

The combined fault-effect description for this example is constructed in three steps:

$$e_{c3} \leftarrow A_3^f \otimes \begin{bmatrix} f_{c3} \\ e_{c2} \end{bmatrix};$$

$$e_{c3} \leftarrow A_3^f \otimes \begin{bmatrix} I & 0 \\ 0 & A_2^f \end{bmatrix} \otimes \begin{bmatrix} f_{c3} \\ f_{c2} \\ e_{c1} \end{bmatrix};$$

$$e_{c3} \leftarrow A_3^f \otimes \begin{bmatrix} I & & 0 \\ & A_2^f \otimes \begin{bmatrix} I & 0 \\ 0 & A_1^f \end{bmatrix} & \\ 0 & & \end{bmatrix} \otimes \begin{bmatrix} f_{c3} \\ f_{c2} \\ f_{c1} \end{bmatrix} \qquad (5)$$

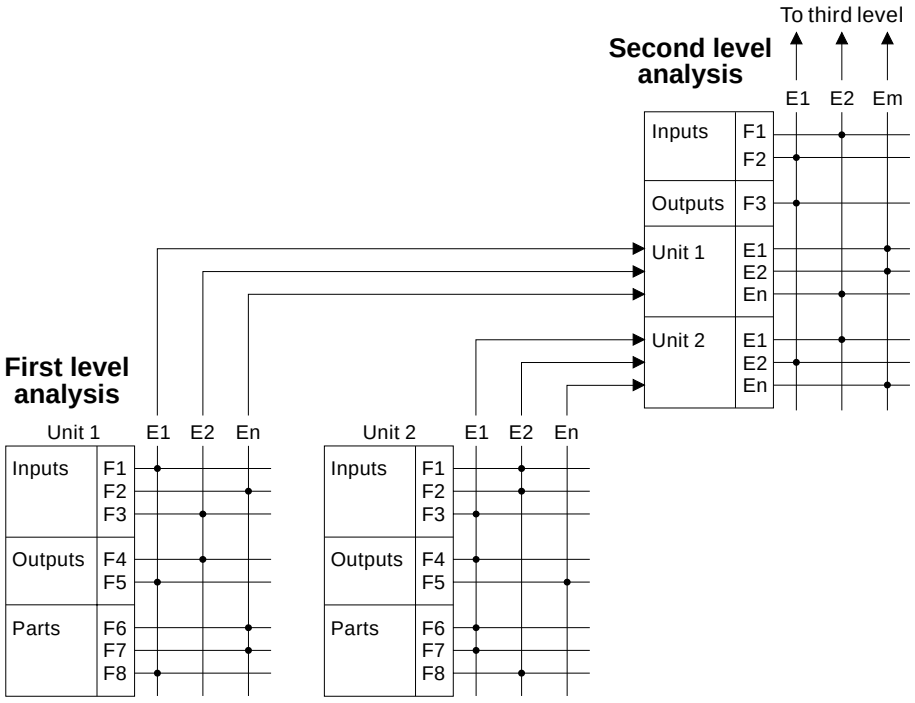

Fig. 1 Failure Mode and Effect Analysis scheme illustrated graphically. Two component LS levels are included.

Effects are seen to be propagated to the next level of analysis and act as parts' faults at that level. This is continued until the system level is reached. The schemes give a subjective mapping from faults to effects: there is a unique path from fault to end effect, but several different faults may cause the same end effect.

Reversal is obtainable through finding the generalized transpose A^b of A^f. The matrices A^f and A^b are each other's pseudoinverse in the Boolean sense. When there is no feedback involved, the result is the capability of isolation of fault effects at any level.

2.3.2 Completeness

Completeness of the fault effect vector is a necessary prerequisite for later fault detection and isolation, because the only faults that can be isolated are those specified in the design. Completeness is obtained if all possible component faults are considered. This is not achievable in a rigorous sense, but engineering experience from risk analysis makes it possible for practical purposes.

It is noted that completeness does not ensure that component fault isolation is possible since several component faults could cause the same effects.

2.3.3 Analysis with closed loop propagation graphs

The FMEA scheme for a set of components connected in a closed loop is principally described as

$$e_{ci} \leftarrow A_i^f \otimes \begin{bmatrix} f_{ci} \\ e_{ci} \end{bmatrix} \tag{6}$$

Looking at the logic operation of this equation, it is obvious that the solution is, if it exists

$$e_{ci} \leftarrow A_i^f \otimes [f_{ci}] \tag{7}$$

The implication is that an automated analysis will need to consider closed loops as special cases. The interpretation of a closed loop in a fault-propagation analysis is merely the observation that closed loop operation may amplify or attenuate the effect of a fault. Which of the two happens depends on the dynamic properties of the control loop. This question can not be answered by the simple Boolean matrix analysis.

In the design tool, it was chosen to automatically locate closed logical loops, let the user cut them open and automatically create additional elements of the faults and effects vectors. Mathematically, this means to replace Eq. (6) by

$$\begin{bmatrix} e_{ci} \\ e_{2ci} \end{bmatrix} \leftarrow A_i^f \otimes \begin{bmatrix} f_{ci} \\ f_{2ci} \end{bmatrix} \tag{8}$$

where e_{2ci} and f_{2ci} are the fault vector and the effects vector elements, respectively, of the signals at the place where a loop has been cut open. It is noted that each of the two vectors can have multiple elements.

The user will manually need to investigate these additional faults and effects, treating them as extra input faults and output effects of the subsystem considered.

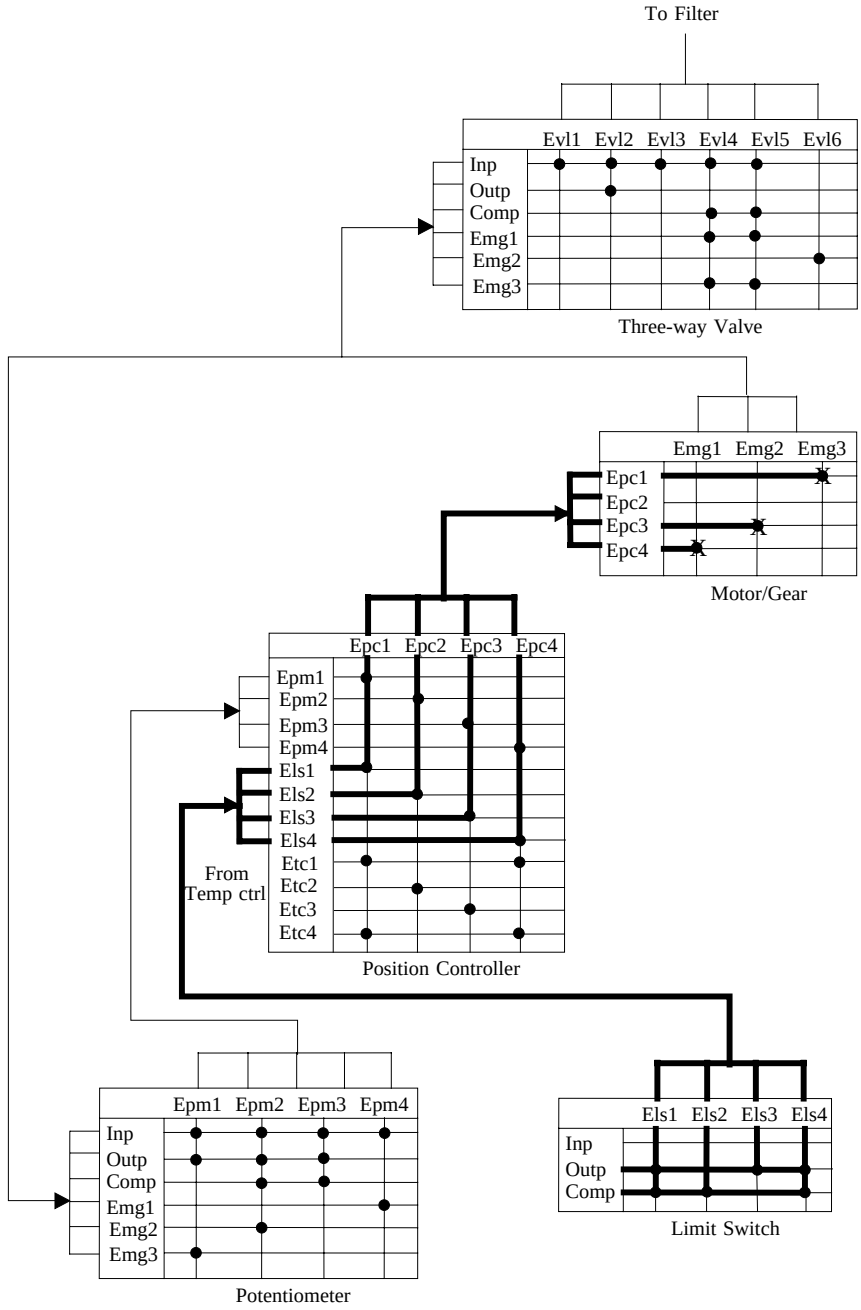

Fig. 2 Propagation of fault effects in closed loop control of a motor driven valve. Solid lines show fault-propagation. Points marked with star show where propagation could be stopped.

Fig. 2 shows the graphical representation of a closed loop FMEA analysis. Bold lines in the scheme show how faults propagate. The important observation is that propagation can be stopped at the points marked with stars. This means that fault handling should be applied exactly at these points.

The component based analysis can thus provide both a list of fault effects and a suggestion of where in a system fault-propagation can be stopped. With this essential information, the user must evaluate the severity of each fault effect and determine, which remedial action(s) shall be

implemented.

When remedial action(s) have been decided, and a component chosen where the action(s) will be implemented, the user modifies the FMEA matrix for the particular component to reflect its new behavior. Re-running the fault-propagation analysis module will then show whether the desired result has been obtained, and documentation for the revised design can be printed.

Various examples have been investigated to illustrate the application of the method, to provide insight in traps, and to highlight features. (Blanke and Jørgensen, 1993), (Jørgensen, 1995), (Blanke et. al, 1997), (Bøgh, 1998), (Blanke and Izadi-Zamanabadi, 1998).

2.4 Mathematical models and simulation of dynamic behavior

Once the list of all component fault-effects are established, dynamic models for FDI need to be generated. To facilitate dynamic analysis and simulation, mathematical models must be described as differential equations. The equations may be non-linear.

The Bond Graph approach (see Karnopp and Rosenberg, 1983) is one classical technique that is well suited for component based dynamic modelling. Other methods could be based on plain use of first physical principles, without the graph technique.

Derivation of dynamic models is not within the scope of the design tool. The models are assumed already available, as part of the component library, and a dynamic model is associated with each component. The format chosen for the dynamic models is the SIMULINK® blocks.

Simulation of dynamic behavior can be carried out using a SIMULINK® program, which is a complete simulation software package (not a part of the prototype), with all necessary facilities, including graphics presentation of results.

2.5 Algorithms for FDI

Analytical methods for detection of faults, and the later isolation (FDI) were investigated in subtask 2.6.1. Computer assisted generation of FDI algorithms would be nice to have, but is a mater of considerable complexity, which is a subject of research in different groups worldwide.

In the design tool, the user is, therefore, requested to select an appropriate FDI method and algorithm in the SIMULINK® environment and use this as part of the individual controllers.

2.6 Implementation of remedial actions

Implementation of the detailed algorithms for controllers and FDI modules, and the supervisory logic needed to switch between normal and not-normal operation, is a manual task and not integrated in the prototype design tool. With the simulation implemented in SIMULINK®, analysis and implementation of simulation function blocks can conveniently be implemented in MATLAB®, which is a de-facto standard for these tasks in the control engineering community.

2.7 Chapter summary

The chapter showed how matrix formulation of the FMEA technique could be used to create a fault-propagation analysis method, which could give a consistent design of fault-tolerance for monitoring and automation systems. The elements of the method were highlighted, and it was explained how the ATOMOS II design tool is implemented in the different elements of the method.

3. OVERALL DESCRIPTION OF THE DESIGN TOOL

3.1 User profile for different tasks

The user will use the design tool when designing new automation systems for use onboard ships. The design tool will assist the user in the analysis of the control system. The user must evaluate the severity of each fault effect and determine which remedial action should be implemented. The user will have a deep knowledge of the components in the control system, and with design of automation systems. Experience with automation system graphical user interfaces is an advantage.

The use is divided in the following actions:

- S**ystem design:** When designing a new automation system, the design tool can assist the user in the analysis of fault-propagation within the system. The user must evaluate the severity of each fault effect and determine which remedial action should be implemented.
- **Component configuration and definition:** A user initially defines a library of components to use in the system design procedure. These components represent available equipment for realizing a particular machinery-automation system. The user should have a deep knowledge of the components and their failure modes to define new components. Normal use does not require deep insight, but a general understanding of process and interface diagrams is needed.
- **Fault-propagation analysis:** The user can initiate a computer-assisted analysis of propagation of faults within the selected subsystem.
- **FDI module (simulation):** The user can select an FDI module, which has a link to a commercial simulation package, where dynamic simulation with and without faults can be carried out. Use of the SIMULINK package requires dedicated knowledge and experience. Note: The SIMULINK package is not a part of the design tool, and component topology is not transferred between the design tool and the SIMULINK program in the ATOMOS II prototype.

3.2 Overview of the Design Tool

The parts in the prototype design tool (PDT) are illustrated in Figure 3. The user communicates interactively with the underlying modules via the user interface (UI). The component library is a database (persistent storage) with components. The user may compose control systems by fetching components from the component library and interconnect them. The user can run a fault-propagation analysis and make (logical) simulations of fault-propagation in the composed system. Producing new components and modifying the component library is also performed via the UI.

The failure modes and effects analysis (FMEA) module and the fault detection and isolation (FDI) module will be designed and implemented separately in other subtasks, but as shown in Figure 3 they will each contribute with UI-components for a composite UI. This way the architecture of the PDT with underlying analysis engines will be transparent to the user.

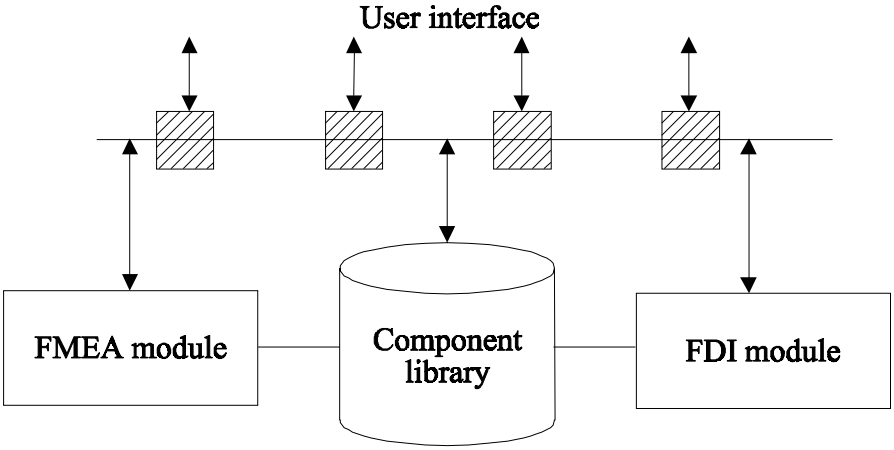

Figure 3: Overview of the PDT.

Through object oriented analysis of the system in Figure 3 an adequate architecture is found. This is finding a model for the component library and the user interface by splitting the system into modules and identifying objects and classes to reach the purpose of the PDT. The graphical user interface is designed along with the object-oriented design of the modules.

3.2.1 The component library

The component library is a database of components representing main. The components include graphical image, attributes, behavior description when failure occurs, and mathematical models. Supporting the user composing control scenarios, the components are implemented with “plugs” equipped with “pins”. A plug corresponds to a physical connection (electrical, fluid…), Pins correspond to the state variables associated with the physical quantity. For fluid, this is temperature, pressure, and mass flow. Different plug types are implemented such connection compatibility can be checked when interconnecting components.

New components can be created, and macros with embedded components can be stored as new components.

Programmable general-purpose components may also be defined. The library is persistent, but at runtime, it is possible to define temporary components, expanding the library or overriding existing components.

3.2.2 The User Interface

In the users’ eyes the design tool is a graphical user interface (GUI) with views and interactive features. Sub-components of this GUI are distributed among the parts of several modules of the PDT. The users are able to interactively change working context.

The users have the possibility to choose between standard components from a component library to build the subsystem for analysis. It is simple to insert components into a design window and choose the connections between the components. The design tool automatically checks the compatibility of components when the user tries to connect them. In case of incompatibility the user is alerted. By clicking a component icon the user gets information about the actual component.

The designer can initiate analysis and the resulting state of the designed control system can be graphically monitored. Further changes to the components failure states can be invoked for analysis.

The designed systems can be stored for later use. These systems can be retrieved again. New components can be built on the basis of designed subsystems for use as black boxes in further system design.

Integrated in the PDT is an editor for construction of the components in the component library.

3.2.3 The Fault Propagation Analysis Module

The Fault Propagation Analysis (FPA) module analyzes open loops of the designed system. The main function of this module is to analyze the fault propagation and present information to the user. It is possible for the user to enter severity for each effect at subsystem level. It is also possible for the user to simulate re-configuration by modifying the propagation of fault-effect through programmable components.

The FPA module includes GUI components for presenting FPA information to the user and to enable interaction.

The basic requirement for the FMEA module is to supply the user of the PDT with the facility to analyze the failure propagation in a particular topology. The main result of the analysis is a matrix (FPA matrix of the system) representing the relationships between failures and end effects of the system.

The failures that can affect a component and, hence, be eventually propagated forward are of two kinds: intrinsic failures and input failures. The intrinsic failures are the failures that arise inside the component due to internal faults. These failures are very important since they can be considered the very *source* of failures within the system. Their propagation through the system will be completely investigated. The input failures represent the failures that can occur to the input ports of a component, which means the failures that are propagated by other components in the topology.

Since the analyses carried on, while investigating the failure propagation, are typically bottom-up based, it is necessary to identify the elements that can be considered as the starting points for these analyses. The components of the topology which have intrinsic failures or input failures external to the system are considered as starting points for the analysis.

On the other hand the components that have output connections with elements not belonging to the topology under analysis are considered as target points. In this case the failures propagated through these connections constitute the effects of failures at topology level, and hence, the target of the failure propagation analysis.

The FMEA matrix of a component is a Boolean matrix that synthesizes the behavior of the component with respect to failure propagation. The matrix represents a model for the relationships between potential failures (both intrinsic and input failures) and their effects at component level. See the document “Overall Specification FMEA Module”, (Nilsen, Blanke Zuccarelli and Allasia, 1996) for further details on the mathematical basis of the FMEA Matrix.

According to the definition of the component model, described in “Prototype Design Tool - Overall Design” (Buck, Borch, Blanke and Lootsma, 1997), input and output ports of a component are represented with *plugs*, where to each type of connection port (hydraulic, mechanical, electrical etc.) is associated a different plug (tube, wire etc.). Every type of plug contains different *pins* that represent physical entities related to the plug (pressure, temperature, voltage etc.). Each pin is modeled with one float variable and four Boolean variables; the latter

indicate the status of the related failure modes (*low pressure* can be TRUE or FALSE, *high pressure* can be TRUE or FALSE etc.).

The Boolean FMEA Matrix describes the propagation of failures, intrinsic failures and input failures (represented at pin level), to the output effects considered at the pins of the output plugs. Hence one element of the matrix defines if an intrinsic failure or a failure at an input pin is propagated as an effect to a given output pin.

The user is able to edit the FMEA Matrix of the components, for which it will be possible to operate on the fault-propagation, and to modify the value of elements of the matrix, simulating an action to stop propagation.

A very critical item is the fault propagation analysis in systems containing closed loops. According to what is stated in document "Overall Specification FMEA Module", (Nilsen, Blanke Zuccarelli and Allasia, 1996), once a closed loops are identified and the effects at one appropriate place in the loop are selected, the loops can be cut at this place and the closed loop effects treated as extra inputs. The result is an analysis, which truncates the closed loop propagation with the benefit that the subsystem FMEA matrix can be obtained. Therefore, the fault-propagation module supports the following actions:

- identify the path that gives rise to a closed loop;
- select one of the components and open the loop at the input side;
- consider the feedback effects as additional inputs in the analysis.

3.2.4 The FDI module

The purpose of the FDI module is to support the control system designer with implementing fault detection and isolation algorithms in the designed control system.

After running the FMEA module the information is available about where it is possible to stop faults from propagating through the system. Their propagation has to be stopped in order to prevent them from causing failures on sub-system or even system level. This can only be achieved if they are detected and identified inside a pre-defined time range. The time range is dependent on the severity of the fault occurring.

The idea is that after starting the FDI module the control system topology will be exported to SIMULINK®. To make this export possible, the component editor offers the possibility to attach a mathematical model file and a description file to each component. In the SIMULINK® environment several FDI methods have been implemented and simulated successfully in the past. After starting the FDI module the control system designer should have the possibility to implement pre-defined or self-defined FDI methods to detect and identify the considered faults.

After successful implementation and simulation of the FDI system the next step necessary to obtain a fault-tolerant control system is the design of the supervisory control. The supervisor chooses between the available re-configuration possibilities and carries them out. Re-configuration means taking actions based on sensor and detector information (sensor information fusion) to stop the faults' propagation with the help of effectors.

Detailed design and implementation of the FDI module is not in the scope of the Prototype Design Tool (PDT).

Some of the problems that have to be solved in the future are e.g. the export of a topology from the PDT to SIMULINK® and the assistance in implementing FDI methods.

Especially the later task is very complicated and perhaps it is not possible to automate it completely. Therefore further research has to be carried out to find out to what extend the

designer can be assisted and guided when implementing FDI systems.

3.3 User interaction and behavior

To give an idea of how the UI works, the following describes the actions and possibilities for the user when working with the PDT.

3.3.1 Starting the PDT

When the application is launched a main window is opened for accommodating all other windows. Now the user is asked to select a component library to use.

3.3.2 Selecting a component library

A file dialog is opened for selecting a component library somewhere on disk. If no library is present a new empty component library is created at the selected path. In any case a component library browser will open with the current library.

3.3.3 Creating or opening a topology

In the main window the user can choose to create a new topology or to open an existing topology. If the user chooses to create a new topology a new one will be created. If the choice is to open an existing topology a file dialog will open for selecting a topology from disk. In any case the topology in question will be displayed in a design window.

3.3.4 Selecting a component from the component library

A component browser will display the library in a window with a tree structure where the desired class of components can be selected (a class is a term for hierarchical grouping of generic material from where component objects can be instantiated). In another window a list will display all the components of the selected class. Clicking one of the components will highlight the selected component. This is now the component selected for any further processing. Whenever another class is selected the component window will be updated and the previous selected component will be deselected.

3.3.5 Creating a new component in the library

In the browser window showing the component classes, the user can select the class of which a new component is to be created. Only the classes with no children in the class structure can be used to create new components (this is called a leaf). Because of this constraint, the creation of new components will not be enabled before a leaf class is selected. When the user chooses to create a component it will be a new component of the selected class. The new component will now have to be configured in a component editor.

3.3.6 Selecting an existing component in the library

The user can select any existing component in the component library and choose to edit the component. In this case the component is loaded from the library and it can now be edited in the component editor.

3.3.7 Editing a component in the library

A component editor will display the attributes of the component to be edited. Some of the

attributes will have values that are assigned by default while the user in text fields can type other attributes in. An image to represent the component must be selected from disk using a file dialog. The user can also choose to add plugs for connection to other components.

3.3.8 Adding plugs to a component in the library

The user must select the (x,y) position on the component image to be associated with the new plug. This is the point from where the connection to other components is drawn when the component is displayed in a topology. The user is prompted to name the plug and to select the type of the plug. The type is important when checking if a connection to the plug is possible.

3.3.9 Saving a component in the library

After editing a component the user can choose to save it in the library. If it is a new component, the user is prompted to name the component. The component does not have to be all finished before it is saved. Some attributes must be filled before the component is ready for use in a topology. If the component is not ready it will be marked/low lighted in the list of components in the component browser.

3.3.10 Cloning a component in the library

In some cases there will be a need for a component very similar to an existing component. This can easily be done by editing an existing component and then choose to save it under a new name.

3.3.11 Working in a topology

Working with a topology in the topology design window is divided into modes that is changed by the user in a toolbar or menu or when a component is selected in the library. This mode decides how the components and connections are viewed and what is possible for the user to do in the topology design window.

3.3.12 Placing a component in a topology

This requires that a component have been selected in the component browser. Changing the mouse pointer type to show that a component has been selected can indicate this. Placing this pointer somewhere in the topology design window and pressing the mouse button can then place the component symbol at the pointer position and insert a copy of the selected component in the topology model.

3.3.13 Connecting components in a topology

If the user chooses to establish connections between component the plugs of the components will be displayed and individually highlighted when the mouse pointer is moved over a plug. If the user presses the mouse button over a plug on a component the first point of a line will be drawn to where ever the pointer is moved to. Clicking again will set the next point of the line until the mouse button is pressed over another component plug. The two components will now be interconnected. This is only if the plugs in the requested connection have similar plug types. Otherwise the connection is not possible.

3.3.14 Moving a component in a topology

If the user chooses to move components around the component images can now be dragged

around with normal mouse dragging. The endpoints of the connection lines will follow when a component is moved. If the connection lines have more than two points the rest of them must be moved manually.

3.3.15 Deleting a component from a topology

In this mode pressing the mouse button on the desired component will delete the component from the topology. The connections and connection lines associated to the component will also be deleted.

3.3.16 View/edit properties of a component in a topology

In any mode pressing the right mouse button will display a pop-up menu. Now the users can chose between normal properties and changing the FPA properties. This will launch one of two component editors according to the selection.

3.3.17 Rotating the image of a component in a topology

In this mode pressing a component will rotate the image of the component by 90 degrees. This affects the end-points of the connection lines on the component to follow.

3.3.18 Editing a connection line in a topology

In this mode each point in the connection lines is highlighted and normal mouse dragging can then drag them.

3.3.19 Deleting a connection line in a topology

In this mode each point in the connection lines will be highlighted and pressing one of these points will delete the connection.

3.3.20 Setting input/output of a topology

In this mode the plugs of the components will be displayed and individually highlighted when the mouse pointer is moved over a plug. If the user presses the mouse button over a plug on a component this plug will also be the plug for the topology. The user is then prompted for the name of the plug. While working in this mode all plugs for the topology is highlighted in another way than the plugs for the components. Pressing an existing plug for the topology will delete it.

3.3.21 Saving a topology

The user can choose to save the topology in the design window. Before it is saved the user will be prompted for the name to save the topology with.

3.3.22 Scrolling the topology design window

The topology editor window will hold the entire visual picture of the topology. If the topology is too big to be shown in the editor window, vertical and horizontal scroll panels will be activated to enable the user to scroll around the picture of the topology.

3.3.23 Zooming in the topology design window

In a future version of the PDT zooming could be included. Choosing the scale factor could then affect the scaling of everything drawn in the design window.

3.3.24 Executing a Fault Propagation on a topology

The user can choose to run Fault Propagation analysis on a topology in the design window. If the topology is ready with input/output defined a FPA analyzer is requested to work with the topology and the topology design window will be inactive while running FMEA analysis.

4. USE OF THE PROTOTYPE TOOL

4.1 Tool environment

For the installation, see the appendix. The source files are written in the Java programming language.

4.1.1 Platform

Minimum for acceptable performance is:

PC Pentium 266 MHz.

32Mbyte RAM and a 17" inch screen.

The Java package jdk 1.1.7 or later and the swing 1.0.3

4.1.2 Disk and file structure in the computer

The root directory for the project is called Pdt, reflecting the tool abbreviation, PDT. All Java compiled codes are stored in the Pdt/classes directory, structured according to the modules in the PDT. As an example, the directory holding the classes for the component module are stored in the library Pdt/classes/Component Module and as shown in Figure 4 to the left. Compiled component classes are stored in Pdt/classes/component library. Libraries for component object instantiated from the respective component classes are stored in the library Pdt/objects/ as shown to the right in Figure 4. A "Dc" component object made in the component editor from the "Dc" class is stored in the "Dc" library. The file name for every component has the extension ".cmp".

The javadoc folder contains the API documentation for the whole tool. The image folder holds the images from where the component creator is able to choose interactively and thereby assign an image to a component.

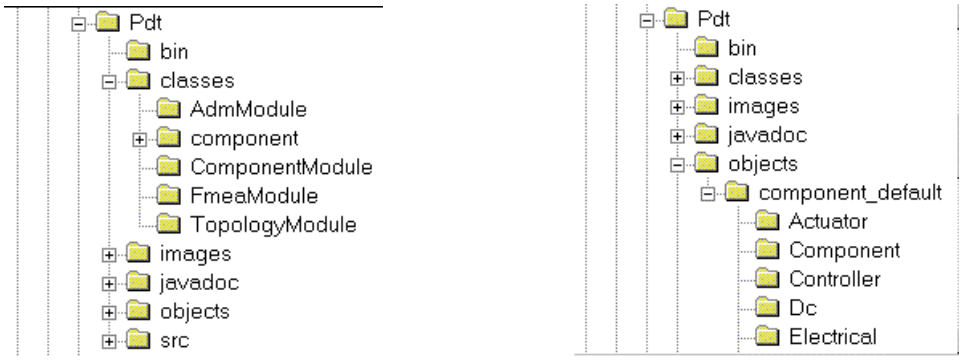

Figure 4: Two snapshots of the directory structure on the platform when the software is installed

4.2 Using the Component Module

When the PDT tool is started the ADM frame and the component browser frame are shown on the screen as seen in Figure 5. To work with components, the browser is needed since the *Properties* button and the *Create* buttons are the only ways to invoke the component editor.

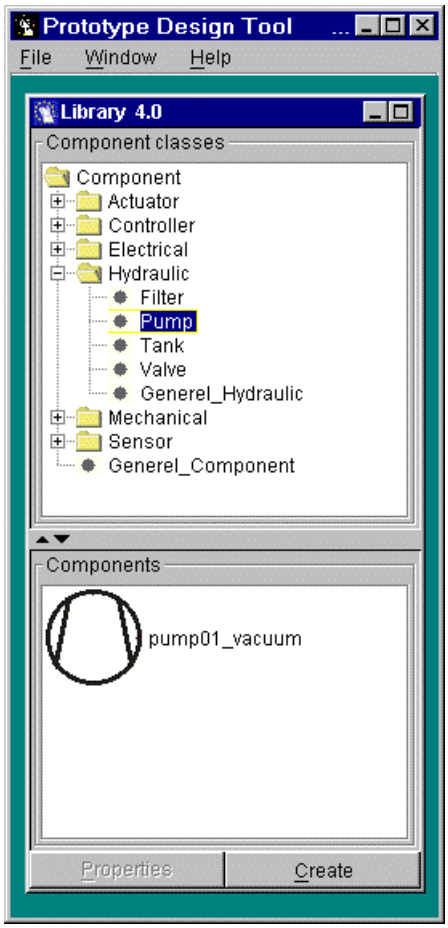

Figure 5: Picture of the PDT on start-up. The component class Pump is selected, which is a subclass of the class Hydraulic, which again is a subclass of the class Component. The outer frame with the menu line belongs to the ADM module.

4.2.1 Building a component

Components from outside must not be inserted in the component library by using a file manager. Only if a copy of this tool - with the same version - has made them, it can be done. This is due to the Java serialization process, where versioning and class definitions are essential.

A component made with the tool has a fingerprint from the tool version, and can only be used in future versions if the component classes are left unchanged.

Creating a component the following sequence must me performed:

1. In the browser top window the component class is selected. When the system allows one to create a component, the *Create* button highlights se Figure 5.
2. Press the button, and the component editor comes up as shown in Figure 6.

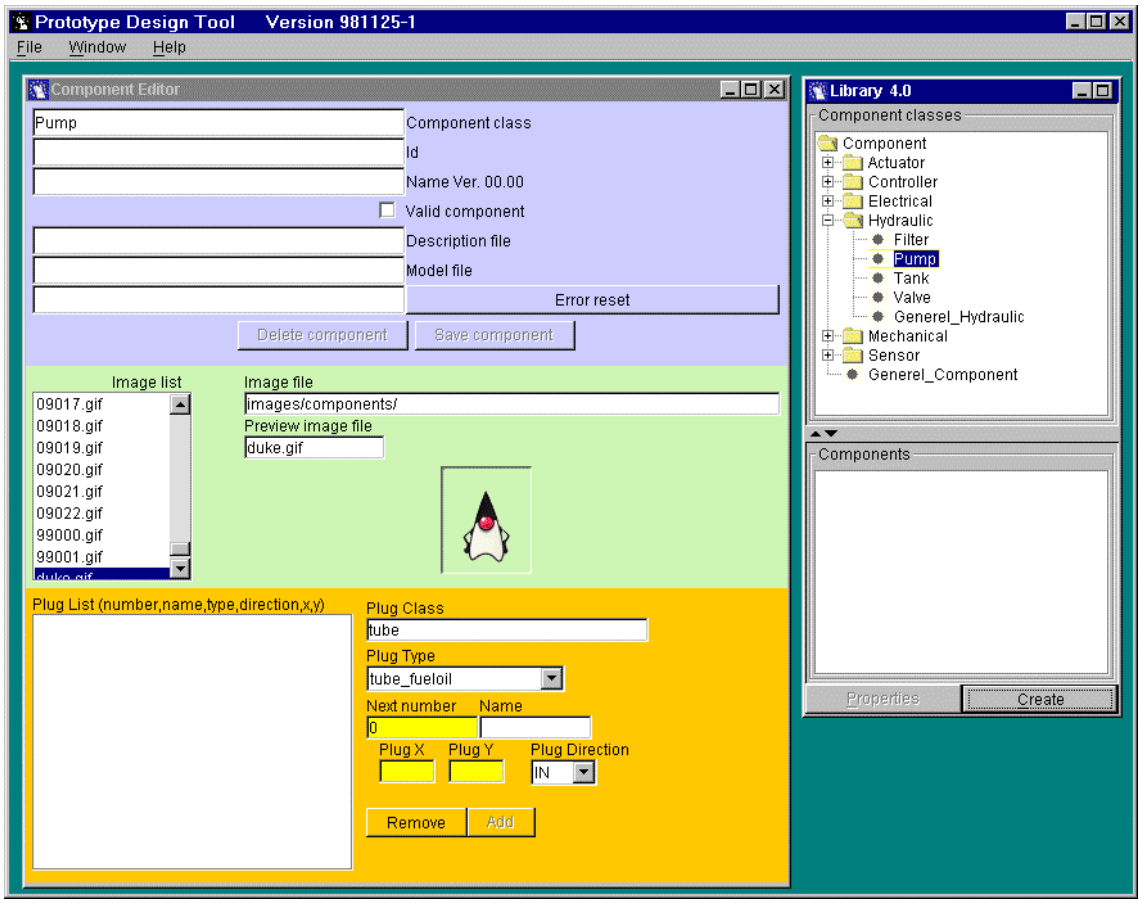

Figure 6: To the left, the editor frame came up after pressing the *create* button.

Information can now be inserted. The image list is used to select an image for the component, which is displayed. The *Image file* name and the *Image library* are shown. The *Delete component* button and the *Save component* button are disabled in Figure 6 due to missing information. The *Plug list* is also empty. The default *Plug class*, the default *Plug direction* and the default *Plug type* are shown. The *Next number* field is the auto plug numbering. It will always show the lowest free pin number. When the mouse is clicked inside the image, a read square is shown on the image, and the co-ordinate for the spot - relative to the top left corner of the image - is displayed in the Plug X/Y field. When all information is given for a plug, the *Add* button is enabled and can be pressed. The plug square will turn into a blue spot. A plug is loaded when the mouse button is released, so releasing the button when the mouse is outside the image area can reject a plug spot operation. The *Plug Name* field is the logical plug name. After an *add* operation all plug information can be seen in the *Plug list* as a comma separated string.

The *Id* field is a user field, which is the component name used in the PDT. The *Name* field is equal to the file name for the component (don't type the ".cmp" extension), and the version for the serialization is shown after a save operation. The *Component class* reflects the class from where the component instance was made.

When the *Save component* button is enabled and used, a warning is given if the component already exist. For the *Delete component* button an extra choice must be made before the delete operation is completed. In both cases the result might cause an update in the browser component object window (the lower part).

If something goes wrong, a message comes up in the error field, which is erasable by clicking

the *Error reset* button.

After inserting some information for a new component the screen could look like the one in Figure 7.

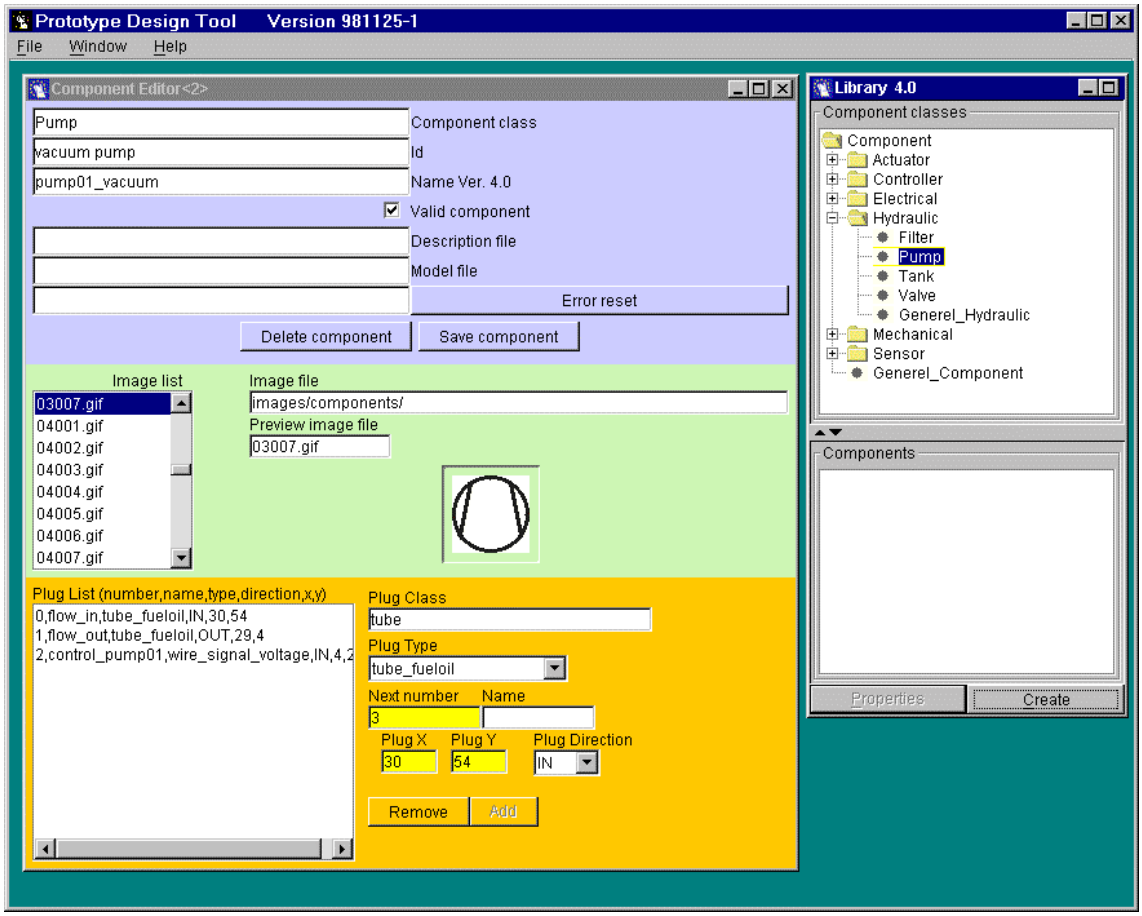

Figure 7: The component is now ready to be saved.

The *Valid component* field is not marked, and a save operation will display the component in the lower browser window with a read component name as shown in Figure 8. It means that it can not be used in a topology.

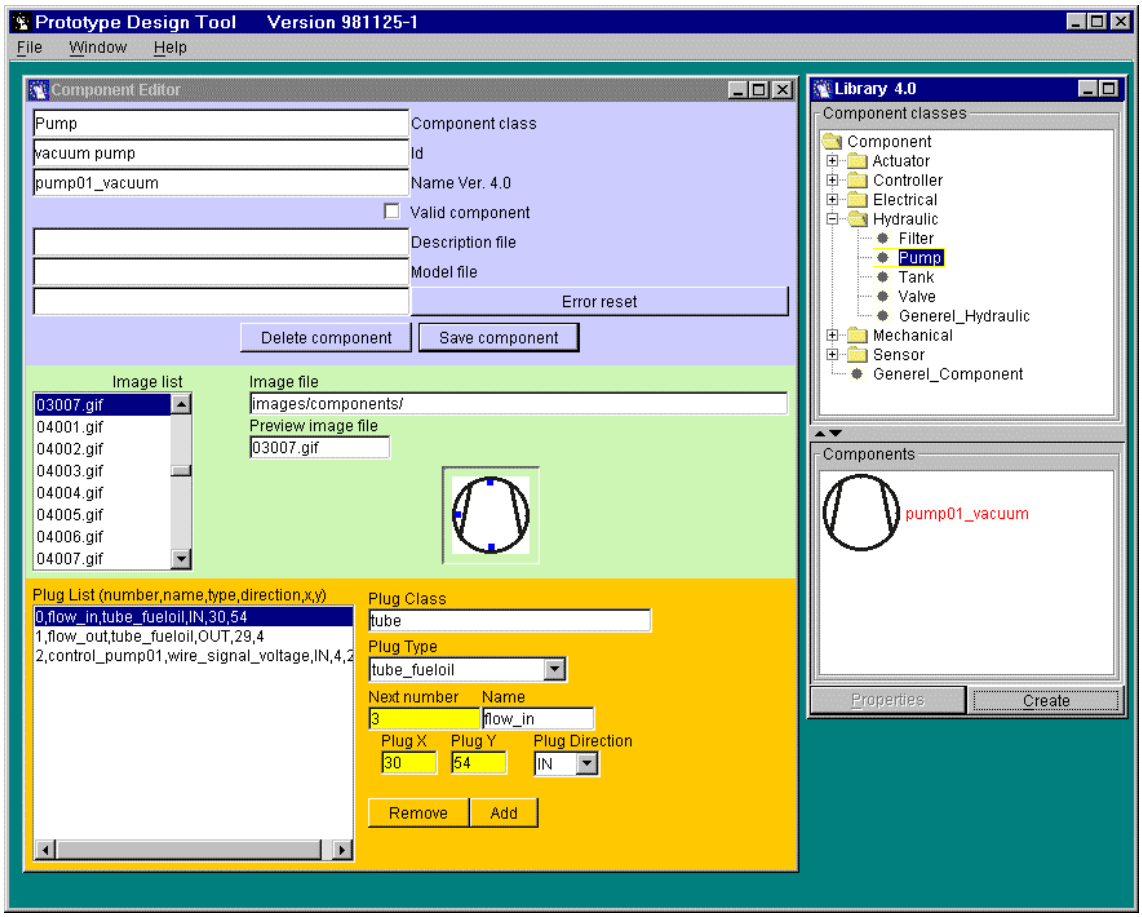

Figure 8: The component saved but not ready for use in a topology

The *Description file* and the *Model file* are not filled in Figure 8. The fields are path references to files in the file hierarchy describing the component and the mathematical description of the component respectively. The last one is used for simulation later on in the FDI module.

4.2.2 Editing a component

After selecting a class (double click) in the browser upper window, component objects will be shown in the lower window if any available. An object is selected by a single click with the mouse. Pressing the *Property* button the component will pop up in the component editor frame. The plug positions will be visible on the image by clicking any plug in the plug list and the plug information will float into the plug fields.

A plug can be moved! Mark the plug in the plug list, click it to bring the information to the plug fields and remove it by pressing the *Remove* button. The plug is then removed from the plug list. The new plug position can now be marked on the image and the plug x,y will reflect the new position. Press the *Add* button and the new plug will show up in the plug list.

If the component name is changed, a save operation will store a new component otherwise a warned overwrite is performed.

4.2.3 Deleting a component

A file manager can be used to delete a component, but it is safer to use the tool. On the other hand a corrupted component can only be deleted by using the file manager.

To delete a component, select it and load it into the editor by pressing the *Property* button, then press the *Delete component* button. There will be a warning dialog. If successful, the component will be removed from the browser window and of course from the component library on disk. The only and last chance to make an immediate recovery is to perform a save operation because all the component data are still in all the editor fields.

4.3 Using the Topology Module

The topology module is used to design and edit topologies using the components available in the component library. The first access to the module is through the main frame controlled by the ADM module.

4.3.1 **Creating a new topology**

Selecting the menu entry *File/New Topology* (see left side of Figure 9) will open a topology editor with a new topology (see right side of Figure 9). By default it will be named “unnamed” until saved with another name. All buttons in the topology editor toolbar are disabled and the only thing to do is inserting components selected in the component browser.

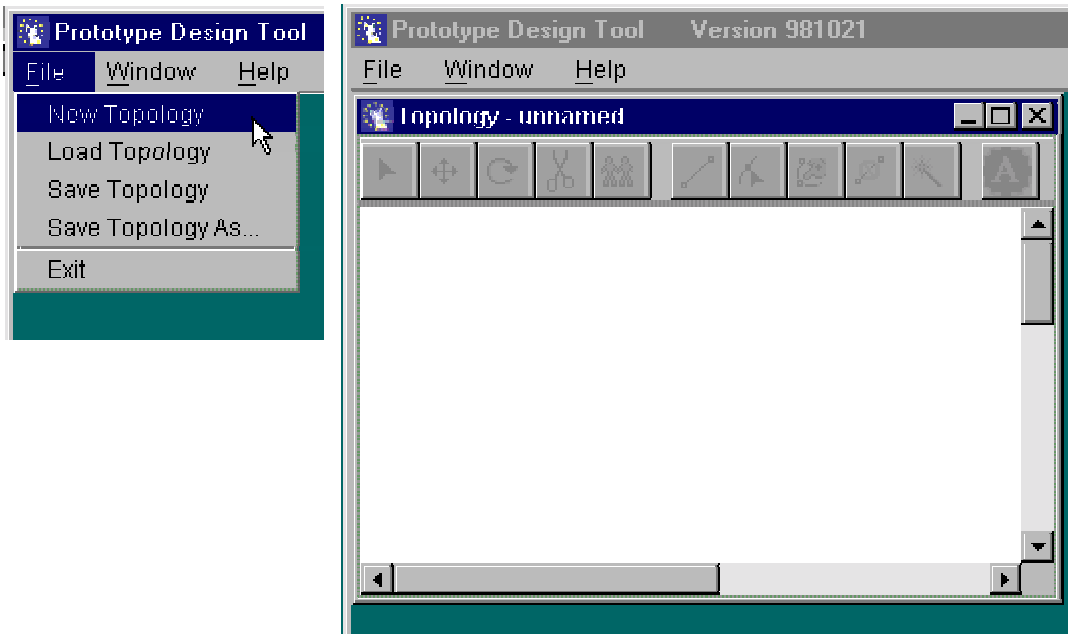

Figure 9: Creating a new topology.

4.3.2 **The Topology Editor**

The topology editor window (see right side of Figure 9) consists of a toolbar and a canvas for drawing the topology. If the topology is larger than the canvas view, the scroll bars can be used to move around the view. The window can be resized to view more or less of the topology. In future development of the PDT scaling and zooming of the topology view can be useful with large topologies.

Pressing any of the toolbar buttons that are enabled will set the topology editor into the mode associated with that button. Figure 10 illustrates these relations. The mouse pointer will change to indicate the current mode. The interaction with the components and the wires will depend on this mode. Some modes (like the Delete component mode) only make sense when the mouse pointer points on a component as they can only affect components. Therefore, the mouse

pointer symbol only changes its symbol to the one of that mode (like the Delete mode) when the pointer points on a component to indicate this restriction.

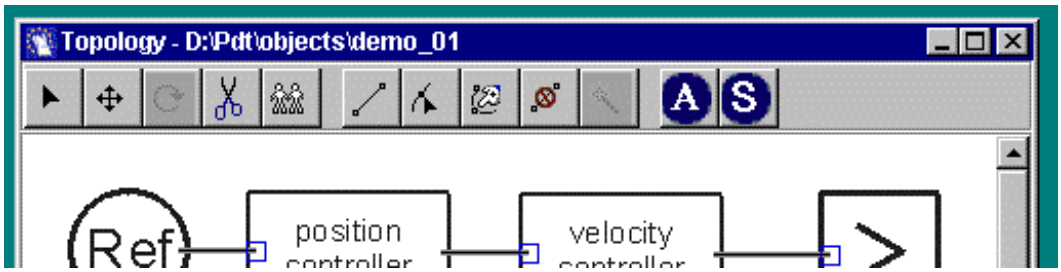

Figure 10: The Topology Editor Toolbar.

4.3.3 Placing components

When a component, valid for topology use, is selected in the component browser the mouse pointer indicates that the component can be inserted (see Figure 11). A mouse click will insert a copy of the selected component and show the image for that component in the topology as shown in the left window of Figure 11. The user is prompted for a component reference name. Since there are now components in the topology the buttons for moving, removing, cloning and connecting components are enabled (the rotation is always disabled because this feature is not available in the current version of the PDT).

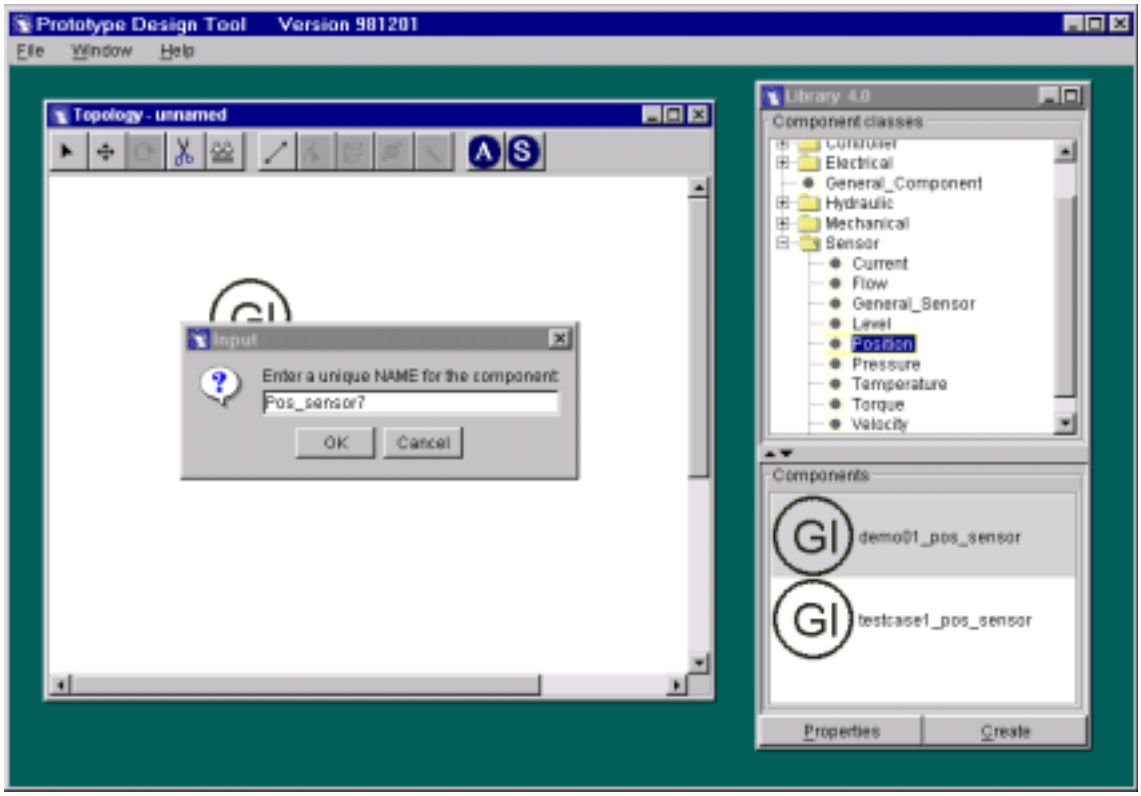

Figure 11: Place selected components in the topology.

4.3.4 Moving components

Pressing the *Move component* button in the toolbar (see Figure 10) will enable the *move* mode. Any component can now be moved/dragged around with the mouse. Any connection made to

this component will follow graphically.

4.3.5 **Deleting components**

Pressing the Delete component button in the toolbar (see Figure 10) will enable the delete mode. When moving over a component the mouse pointer changes indicating that the component can be deleted. Clicking on a component will delete this component and all connections to this component.

4.3.6 Cloning components

Pressing the Clone component button in the toolbar (see Figure 10) will enable the clone mode. When moving over a component the mouse pointer changes indicating that the component can be cloned. Clicking on a component will create a clone of the component and place it a bit offset from the original. The clone will not inherit any connections.

4.3.7 Connecting components

Pressing the Connect components button in the toolbar (see Figure 10) will enable the connection mode. When moving the mouse pointer over a component the plug points will be indicated with red spots. The spot closest to the pointer will be larger than the others as shown in the left part of Figure 12. When keeping the pointer still for a while over/near a plug point, the direction and type of the plug will be shown in a small box. Clicking the left mouse button over/near a plug point will initiate a connection. When moving the mouse pointer a line (wire) will be drawn from that plug point and to tip of the mouse pointer. When moving the mouse pointer over the component candidate for connection, all compatible plug points on that component are indicated with red spots (only plug points of the correct type and opposite direction of the first chosen components plug will be available). The closest spot will be larger than the other spots and the wire will be drawn to that plug point as shown in the right part of Figure 12. Clicking the left mouse button will close the connection between the selected plugs of the selected components. Currently the only way to cancel the connection in progress is to press another button on the tool bar.

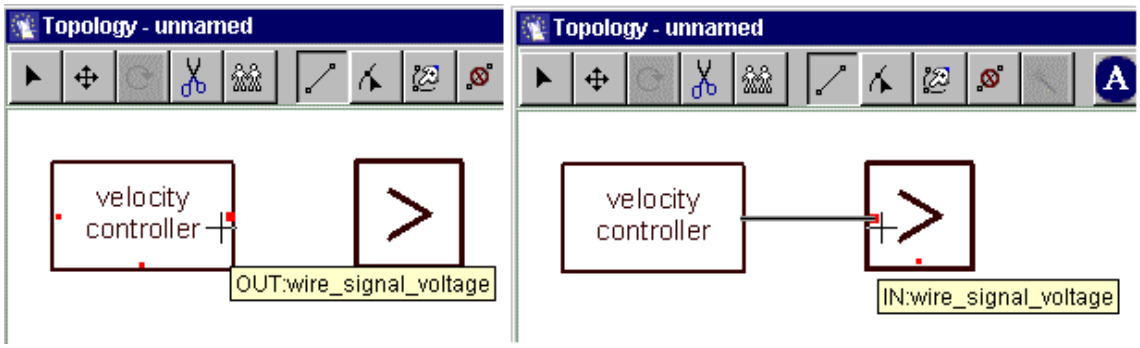

Figure 12: Connecting two components.

4.3.8 Disconnecting components

Pressing the *Delete connections* button in the toolbar (see Figure 10) will enable the *disconnection* mode. Moving the mouse pointer over a wire (or very close to it) and clicking the left mouse button will delete the connection.

4.3.9 Loading a topology

Selecting the menu entry *File/Load Topology* (see left side of Figure 9) will open a file dialog where a topology can be chosen. This will then load the selected topology and view it in a topology editor window. An attempt to open a non-existing topology will just create a new topology.

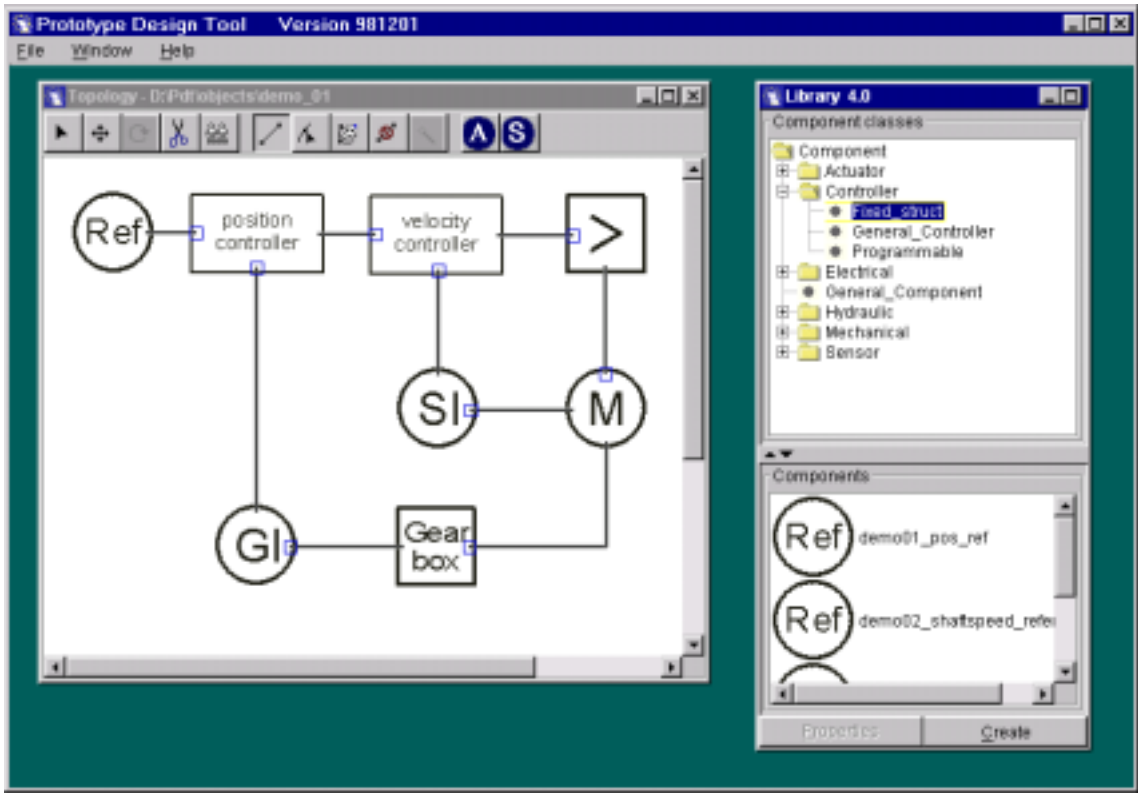

Figure 13: A topology loaded into the topology editor.
The small box on one end of the wire marks an IN-plug

4.3.10 Saving a topology

Selecting the menu entry *File/Save Topology* (see left side of Figure 9) will save the topology of the topology editor window currently active. It will overwrite the file it was loaded from. Selecting *File/Save Topology As...* will open a file dialog where the filename can be specified before saving. In the attempt of saving with the name of an existing file, a prompt for accepting overwriting will pop up.

4.3.11 Working with multiple topologies

Each time a topology is loaded or a new one is created it will be handled and viewed by a topology editor window. If two topologies have the same name, the windows will be distinguishable by a postfix number according to the order of opening the windows. To bring

any of the windows to the front can be done by selecting it in the *Window* menu as shown in Figure 14.

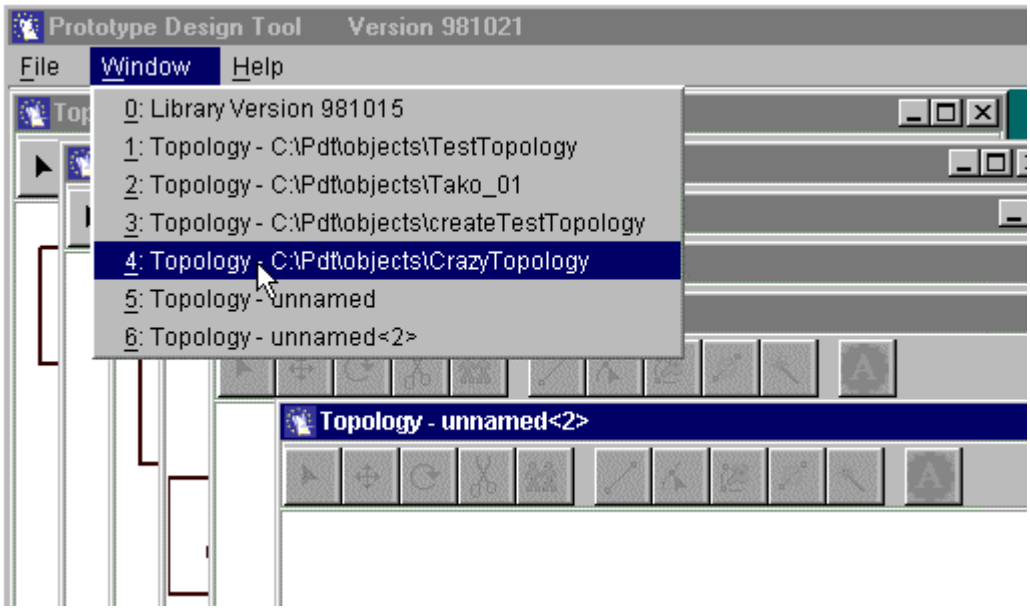

Figure 14: Activation of a topology window, bringing it to the front.

4.4 Using the FPA Module

When the user launches the analysis, a GUI appears, which is similar to the one of the topology editor. This GUI is simply a window in which the topology, which was formerly edited, is displayed, without the possibility of changing it.

The GUI of the FPA module has a toolbar with four buttons, corresponding to the four principal operations the FPA module performs:

- identification of closed loops;
- cut of closed loops (the user can decide where to cut);
- restoration of the cut connections;
- Calculus of FMEA matrix of the system and performing reverse analysis.

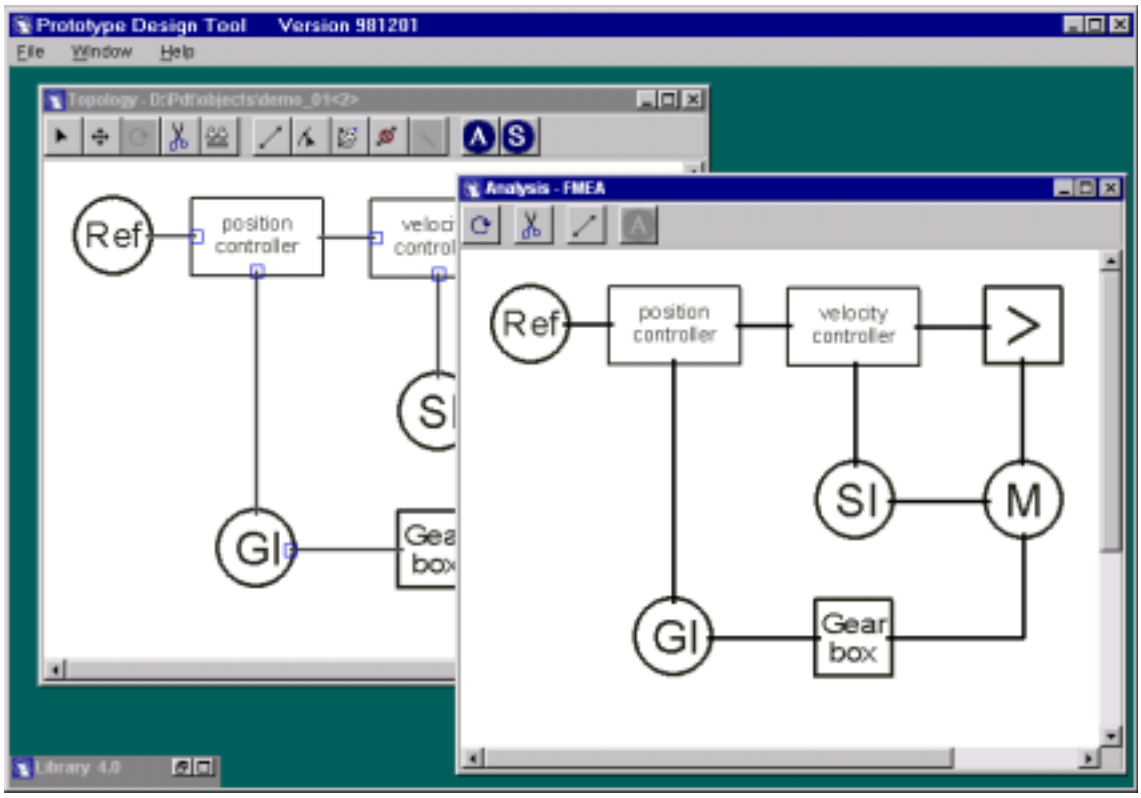

Figure 15: FPA module GUI on top of the topology GUI

4.4.1 Identification and cut of closed loops

When the user clicks on the button of loop identification (first button on the left) the «pre-analysis» is performed.

If loops are present in the topology the analysis cannot be performed. Therefore, the button, which launches the analysis, is disabled, so the user is forced to launch the «pre-analysis» phase to identify the loops.

When loops are found a dialog box appears (Figure 16). In this dialog box the number of loops identified in the topology is pointed out.

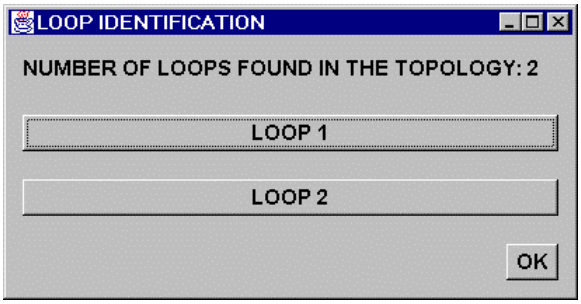

Figure 16: Dialogue box for the loops

Every button of this dialog box corresponds to a different loop, which is found within the topology. The user is able to click on these buttons, making a new window appear for each loop, which specifies which components are involved in the loop itself (Figure 17). At the same time the connections, which are concerned in this loop, appear with a different coloring (blue instead of black). This graphical solution is particularly useful when the topology becomes

complex. In this way the user can easily identify each loop of the topology and then carry on with the following step (i.e. the cutting of the loops).

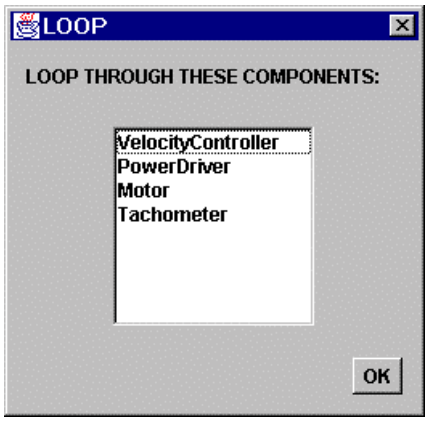

Figure 17: Dialogue box listing the components involved in the loop

When a closed loop is identified the user has to open it before the user can start the analysis. In order to do that the user must press the second button of the toolbar, then performing a click with the mouse on the connection making a cut and the connection will be deleted. Coloring the connection in red shows this graphically. Consequently, the components with this connection in input will have, as new faults, the effects in output at the other component. In the same way the effects propagating from the components which have this connection in output will become new end effects of the whole topology.

The user is also able to restore a connection: in order to perform this action he must press the third button of the toolbar, then he must click with the mouse on the red wire of the connection to be restored. The connection color will then be changed to black, highlighting the successful completion of the operation.

4.4.2 Launching the analysis and reading the results

After having cut all the closed loops, the user is able to launch the real analysis phase, by clicking the relative button of the toolbar.

The results of the analysis are displayed in a window (Figure 18**)**, in which there is a list of all the possible end effects of the system (including the «fictitious end effects» which are created while cutting the loops).

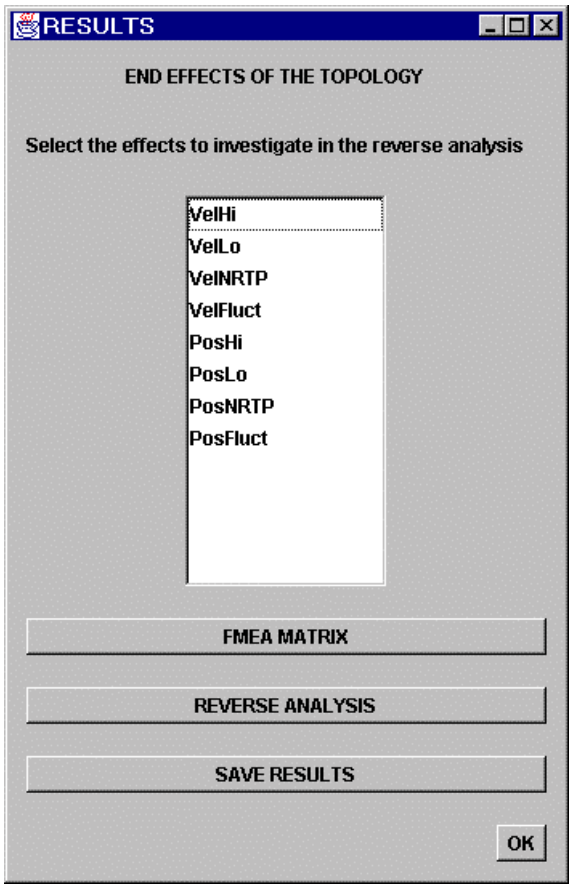

Figure 18: Main GUI displaying the results

The user can decide to see the results in a matrix way (i.e. he can have the whole FMEA matrix shown, including the rows of the «fictitious» faults and the columns of the «fictitious» end effects) by clicking on the button «FMEA MATRIX». He can also see the single relationships between the end effects and all the possible faults of the system (this is done by selecting one or more items from the list of the end effects and then clicking on the button «REVERSE ANALYSIS»).

If the user wants to see the results in the second way, a window with a list of all the faults, which generate the chosen end effects, is shown (Figure 19). It simply means that there is an «OR» logic relationship between the chosen end effects and all the faults which are pointed out in the list (i.e. if one of the faults in the list occurs then the end effects comes true). It means that if the user decides to choose just one effect the list represents the possible faults that can generate the single end effect, if the user chooses more than one end effect the list represents all the possible faults that can generate the chosen end effects together.

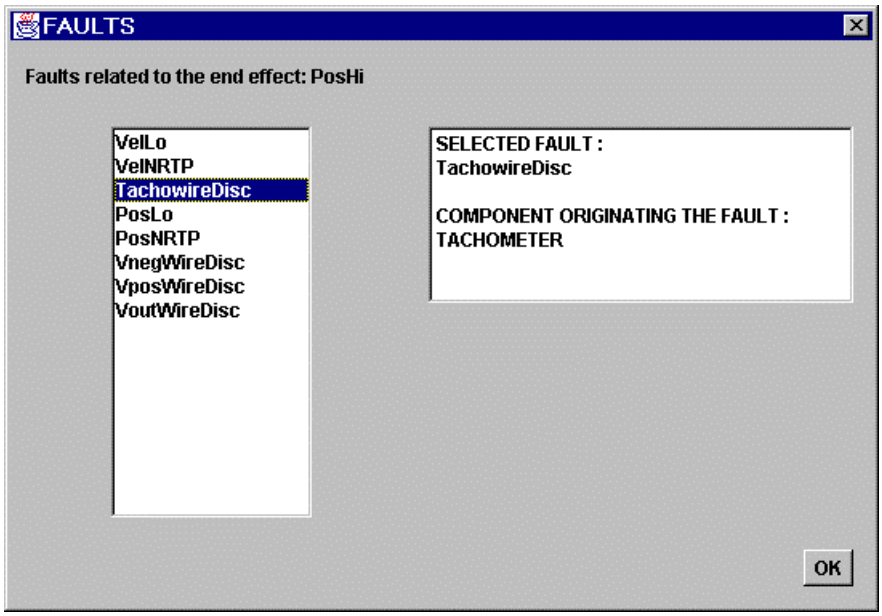

Figure 19: GUI listing all the faults related to a chosen end effect

Every listed fault can be selected, and as a consequence of this action the name of the selected fault and the name of the component, which originates it, are displayed in the box in the right side of the window (see Figure 19).

The user can continue performing the reverse analysis by double clicking on an item in the list of faults, in this way a new window (Figure 20) will appear. In this window a list of the components, which are crossed by the fault under investigation, is shown.

In this way, the user is able to know all the components, which are involved in the propagation of each single fault. This is done in order to locate places where fault accommodation could be implemented, i.e. in a programmable component. In the FPA module, accommodation is equivalent to a change the properties of the associated FMEA matrix of a programmable component.

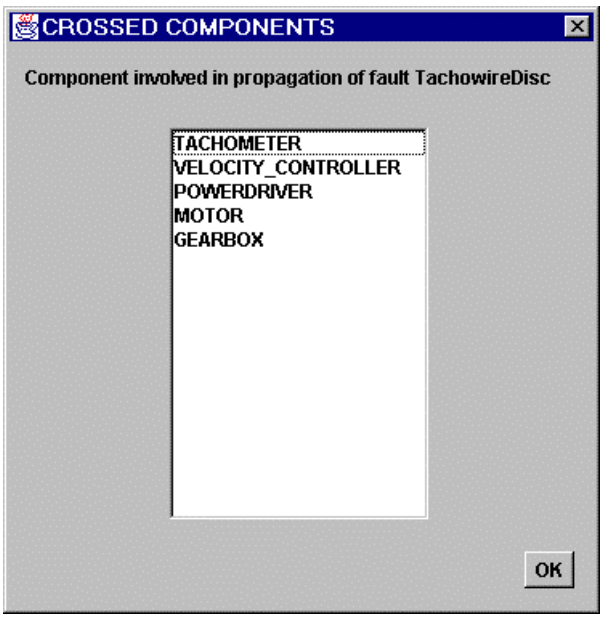

Figure 20: GUI listing all the components crossed by a selected fault

The other way to view the results is to have all the FMEA matrix of the system displayed on the screen. This could be done by clicking on the «FMEA MATRIX» button of Figure 18, which makes a window, with the whole matrix of the system, appear (Figure 21). This way to represent the results of the analysis is easily understandable for simple topologies with few faults and end effects, but it becomes hardly intelligible if the topology has hundreds of faults and end effects.

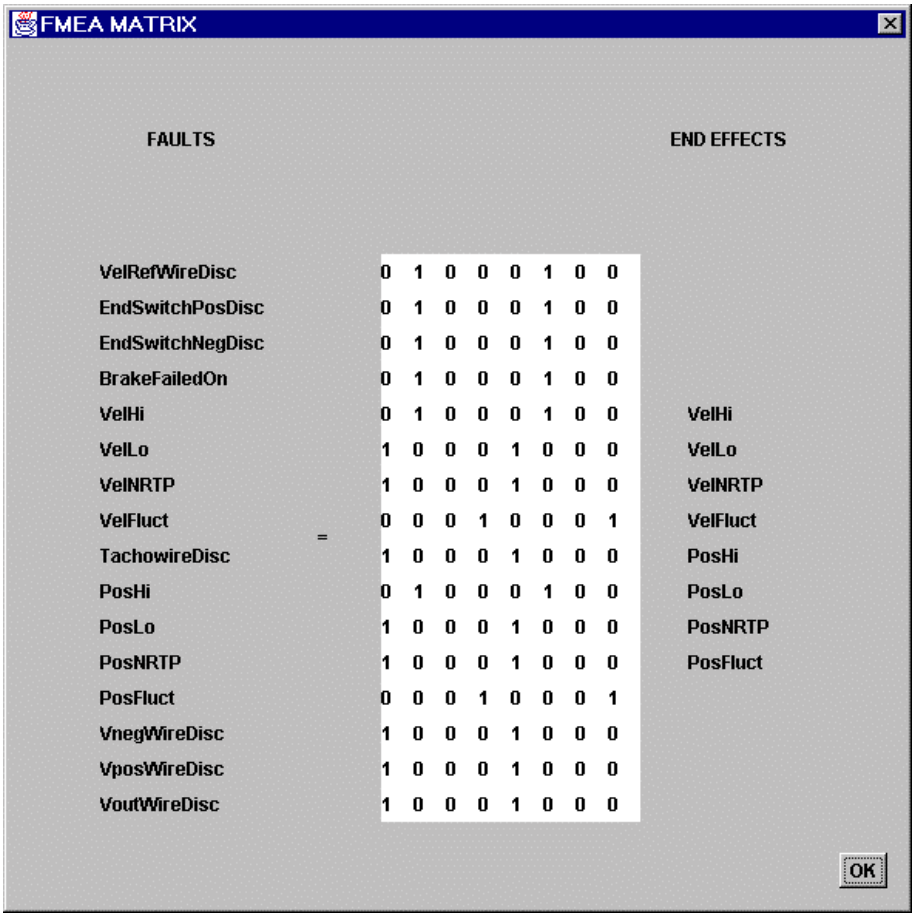

Figure 21: GUI displaying FMEA matrix of the system

4.4.3 Editing characteristics of the components

The user must insert all the information about inputs and outputs of the components, including the relations among them (i.e. the matrices of the components). Essentially he is able to perform, for each component, the following operations:

- insert the names of input variables (for each Boolean *pin variable* of the *plugs* in input);
- insert the names of output variables (for each Boolean *pin variable* of the *plugs* in output);
- insert a new fault for the component, with its name;
- remove one of the faults of the component;
- edit the inner FMEA matrix of the components.

In order to do all these operations the user must click with the right button of the mouse on the component symbol. As a consequence of this action a «pop-up-menu» appears and the user must select the item «Component Editing» in order to make the corresponding dialog box

appear. This dialog box has a set of buttons which let the user open new windows dedicated to all the single operations which have been listed above.

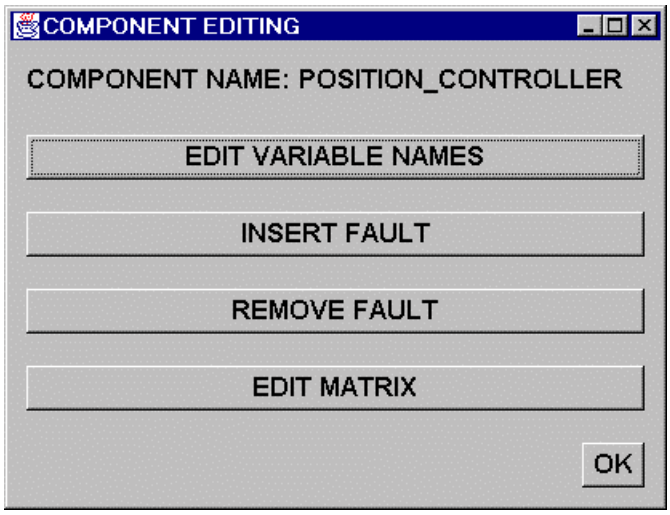

Figure 22: Dialogue box for component editing

The most important operation the user is able to perform by using this GUI is the editing of the inner failure matrix (FMEA matrix) of the selected component. In order to do this there is a dialog box, which has a menu with the inputs variable of the components, the names (which have been previously defined by the user himself) and several checkboxes, corresponding to the output variables of the component.

Since every row of the FMEA matrix corresponds to a relationship between an input variable and the output variables of the component, the user can edit separately each row of this matrix by selecting an item from the menu. (It means that the user chooses an input variable and consequently which row of the matrix he wants to edit). He can then decide which elements of the row must be filled with «1» by simply setting the checkbox of the output variable, corresponding to that element, in an «on» state.

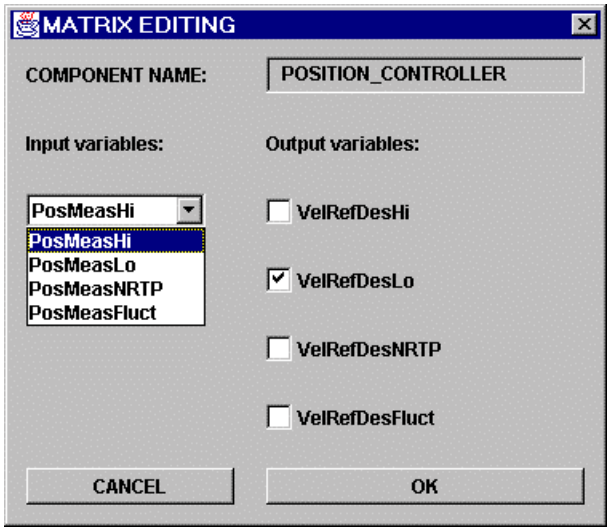

Figure 23: GUI for editing the FMEA matrix of the component

4.4.4 Fault Activating

By clicking on the «Fault Activating» item of the «pop-up-menu» of the component in the

Analysis window a new window appears in which all the faults of the component are listed in checkboxes and the user is able to activate or deactivate them. Deactivating a fault does not mean that the fault is removed from the component, but simply that its propagation it is not taken into consideration for a particular analysis. An example could be a component, which has several types of failure, and each type of failure depends on the working environment of the topology in which the component is placed.

When loops are cut, «fictitious» faults are created in the topology. These new faults are pointed out in the «Fault Activating» window of the components, which have them in input.

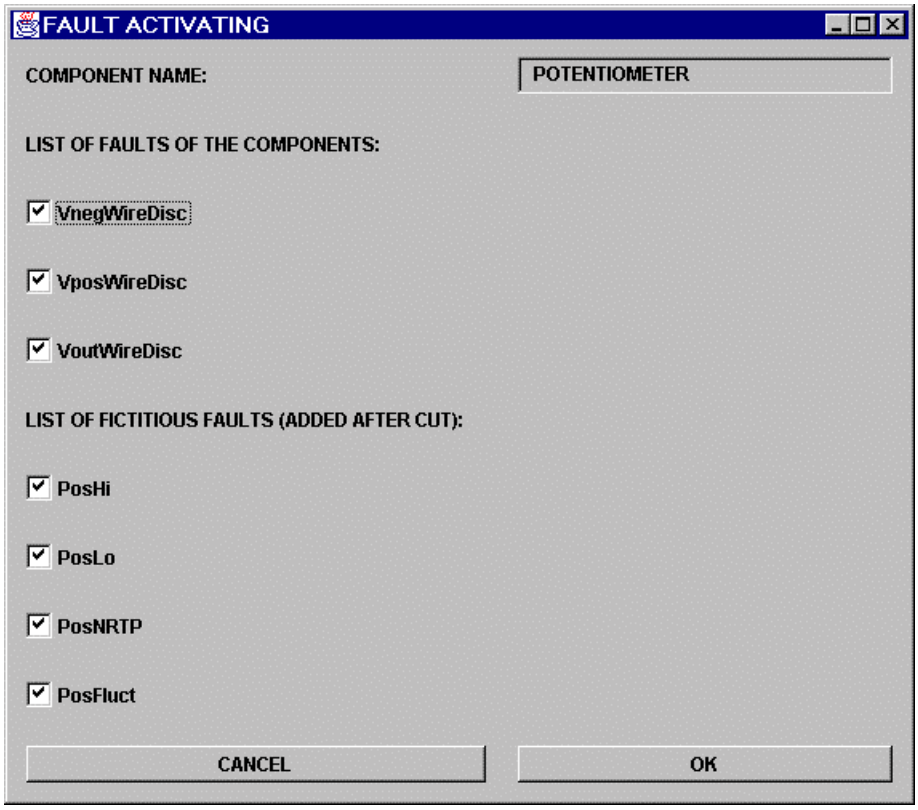

Figure 24: GUI for activating the faults of the component including «fictitious» ones

4.4.5 Launching another analysis

When an analysis finishes the user is allowed to change the settings of the components (i.e. changing variable names, inserting and removing faults and changing the matrix). But, if he wants to change the topology of the system, he must come back to the editor of the topology and then launching another FPA analysis session.

4.5 FDI Module

As described in section 3.2.4, the FDI module could not be implemented in the ATOMOS II prototype design tool. The complexity of implementation is so high that a next version of the prototype tool is required to achieve this.

FDI requires detailed mathematical models and dedicated algorithms for detection and isolation. As an underlying part of components, mathematical models in SIMULINK® format would be needed and the structure of the simulation model generated automatically from parsing the model structure defined by the topology module. From a bank of FDI detectors, a suitable method could be selected, and appropriately connected within the controller

("programmable component") to reflect fault accommodation. Supervisor functionality would be implemented as well. Thereafter, continuous-time simulation could be used to study the detailed performance of the selected detectors for different scenarios of component failure.

The "S" button on the topology toolbar, see Figure 13, will activate the SIMULINK® environment, which enables continuous-time simulation and plot of timehistories.

5. APPENDIX

5.1 Installation

The Pdt directory is copied into the root of a disk drive, eg.

The env.bat file must be edited so three resource pointers are valid:

JDKHOME, SWINGHOME and PDT. (If SWINGHOME is part of the JDK kit, SWINGHOME should be equal to JDKHOME).

Ex:

SET JDKHOME=C:\JDK	points to the java development kit(>= version 1.1.7.)
SET SWINGHOME=c:\swing	points to the root of java swing kit (>version -1.0.3)
SET PDT=d:\Pdt	points to the PDT software package in which the zip file was unpacked.

The SET CLASSPATH should not be changed.

The pdt.bat file must be edited so one resource pointer is true:

SET sim_path	points to the simulation application program. The variable has the fixed name 'sim_path' and must not be changed. The argument is pointing at the input file for the simulation application. A few .mdl files are found in Pdt/src/SimulationModule.

5.2 Starting the application

The simplest way to start the PDT is to double click the pdt.bat file. Shortcuts can be made. The ADM module is thereby started, ready for editing a component or opening a topology.

5.3 Number and description of the figures for the software tool

The next table shows the relation between the figure number for the image assigned to a component in the component editor (Figure 7) and the component type.

Figure number:	**Description:**
00001	General component
00002	Mechanical component
00003	Programable component
00004	Controller
00011	P controller
00012	PI controller
00021	Velocity controller
00022	Position controller
00031	Reference signal
01001	Actuator/drive
01002	Electrical motor
01011	power drive
01021	gear box
02001	sensor, general/unclassified
02002	all electrical variables indicator/sensor
02003	flow rate indicator/sensor
02004	gauging/postion/length indicator/sensor
02005	time indicator
02006	level indicator/sensor
02007	humidity/moisture indicator/sensor
02008	pressure/vacuum indicator/sensor
02009	quality indicator/sensor (e.g. analysis, concentration, conductivity)
02010	radiation indicator/sensor
02011	speed/frequency indicator/sensor
02012	temperature indicator/sensor
02013	weight/force indicator/sensor
03001	tank (shape01)
03002	tank (shape02)
03003	heat exchanger
03004	gas filter
03005	filter, general
Figure number:	**Description:**

03006	pump, general
03007	vacuum pump
04001	globe valve
04002	globe valve, hand-operated, straight through
04003	globe valve, remote controlled, straight through
04004	safety valve, straight through
04005	self-closing valve
04006	quick-closing valve
04007	regulating valve (needle valve)
04008	gate valve
04009	globe valve, angle
04010	globe valve, three-way
04021	non return valve, straight
04022	non return valve, hand-operated, straight
04023	non return valve, remote controlled, straight
04024	flap, straight through
04041	ball valve (cock), hand-operated
04042	ball valve (cock), remote controlled
04061	butterfly valve, hand-operated
04062	butterfly valve, remote controlled
04081	cock, straight through
04082	cock, three-way
04083	cock, angle
04084	cock, three-way, L-port in plug
04085	cock, three-way, T-port in plug
04101	close-off valve, straight
04102	close-off valve, angle
04120	reducing valve

figure number:	**description:**
09001	CO_2 green
09002	CO_2 red
09003	d green
09004	d red
09005	damper green
09006	damper red
09007	em_ge green
09008	em_ge red
09009	fire pump green
09010	fire pump red
09011	heat green
09012	heat red
09013	hose green
09014	hose red
09015	smoke green
09016	smoke red
09017	sprinkler green
09018	sprinkler red
09019	valve green (h)
09020	valve red (h)
09021	valve green (v)
09022	valve red (v)
99000	input from outer world
99001	output to outer world

5.4 Software task distribution

Component Module Ole Borch

Topology Module Jakob Buck

FPA Module Fabio Bagnoli