and the actual occurrence of a failure, and (3) **Distinguishability**, which describes the size of the ambiguity sets given a time limit for the observation. Using these metrics, DTOOL provides three kinds of analyses. In *evaluation* mode, the diagnosability characteristics of a design with a predefined sensor allocation are calculated. In *advice* mode, an arbitrary set of requirements can be defined for the diagnosability characteristics, and the tool generates a satisfactory sensor placement. The tool also provides *test tree generation* analysis.

Approved____________________________________ Date________________

ELECTRICAL ENGINEERING

SENSOR-BASED DIAGNOSIS OF DYNAMICAL SYSTEMS

AMIT MISRA

Thesis under the direction of Professor Sztipanovits

In highly automated engineering systems, sensors provide the data needed for control, monitoring and diagnosis. The reliability and cost of these sensors are important issues. The thesis describes a model-based approach to handle sensor failures and to provide optimum sensor allocation in diagnostic systems.

The first part of this work deals with providing reliable diagnosis of failures in a system, given the possibility that sensors might fail. The correctness of diagnostic results depends upon the reliability of observations. The observations can be erroneous because (1) faulty data reading by sensors, (2) modeling error, and, (3) in case of a major fault, loss of model validity. A robust diagnostic system to handle the above possible causes of errors was developed. The diagnostic system uses the physical and temporal constraints imposed on observations in a dynamical system to identify the presence of erroneous observations.

The second part of the work deals with minimizing the costs associated with sensors without sacrificing the diagnosability of the system. A Diagnosability Analysis Tool (DTOOL) was developed, which facilitates the analysis of diagnosability in terms of the sensors in the system. Three metrics to characterize diagnosability were defined : (1) **Detectability**, which gives the longest time that is needed to detect a failure, (2) **Predictability**, which gives the shortest time between the forewarning

14. Gallanti, M. et al., "A Diagnostic Algorithm Based on Models at Different Levels of Abstraction," *Proceedings of 11th IJCAI*, 1989, pp. 1350-1355.

15. J. de Kleer and B. C. Williams, "Diagnosing Multiple Failures," *Artificial Intelligence*, vol. 32, 1987.

16. E. J. McCluskey, "Logic Design Principles," Prentice Hall, Englewood Cliffs, NJ, 1986.

17. Shogo Tanaka, "Diagnosability of Systems and Optimal Sensor Location," Chapter 5 in the book *Fault Diagnosis in Dynamic Systems: Theory and Application*, Prentice Hall International (UK), 1989, pp. 21-45.

18. Ethan Scarl, "Diagnosability and Sensor Reduction," in *Proceedings of the Workshop on AI, Simulation and Planning in High Autonomy Systems*, Cocoa Beach, FL, 1991.

19. S. Chien, R. Doyle and Nicolas Rouquette, "Sensor Placement for Diagnosability in Space-borne systems: A Model-based Reasoning Approach," *Proc. 2nd Int. Workshop on the Principles of Diagnosis*, Milan, Italy, October 1991.

20. Kuipers B., "Common Sense Reasoning about Causality : Deriving Behavior from Structure," *Artificial Intelligence*, 24, 1984, pp. 169-203.

21. Forbus, D. F., "Qualitative Process Theory," *Artificial Intelligence*, 24, 1984, pp. 85-168.

22. Pattipati, Krishna R. and Alexandridis, Mark G., "Application of Heuristic Search and Information Theory to Sequential Fault Diagnosis," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, no. 4, July/August 1990, pp. 872-887.

23. Sheskin, T. J., "Sequencing of Diagnostic Tests for Fault Isolation by Dynamic Programming," *IEEE Trans. Reliability*, vol. 27, no. 5, 1978, pp. 353-358.

24. Šiljak, Dragoslav D., "Decentralized Control of Complex Systems," published by Academic Press, Inc., 1991, pp. 374-378.

25. Horowitz, E. and S. Sahni, "Fundamentals of Data Structures," published by CBS Publishers and Distributors, New Delhi, India, 1983, pp. 306-307.

REFERENCES

1. R. Patton, P. Frank and R. Clark, "Fault Diagnosis in Dynamic Systems: Theory and Application," Prentice Hall International (UK), 1989.
2. F. Hayes-Roth et al., "Building Expert Systems," Addison-Wesley, Reading, Mass. 1983.
3. R. O. Duda and P. E. Hart, "Pattern Classification and Scene Analysis," John Wiley & Sons, 1973, pp. 228-243.
4. Y. Peng and J. Reggia, "A Connectionist Model for Diagnostic Problem Solving," *IEEE Trans. Syst., Man and Cybernetics*, vol. SMC-19, no. 2, March/April 1989, pp. 285-298.
5. T.-H. Guo and J. Nurre, "Sensor Failure Detection and Recovery by Neural Networks," in *Proc. Int. Joint. Conf. on Neural Networks*, July 1991, vol. I, pp. 221-226.
6. M. S. Fox, S. Lowenfield and P. Klienosky, "Techniques for Sensor-Based Diagnosis," in *Proc. 8th Int. Joint. Conf. Artificial Intelligence*, 1983, pp. 158-163.
7. E. Scarl *et al.*, "Diagnosis and Sensor Validation through Knowledge of Structure and Function," *IEEE Trans. Syst., Man and Cybernetics*, vol. SMC-17, no. 3, May./June 1987, pp. 360-368.
8. S. J. Chang, F. DiCesare and G. Goldbogen, *Evaluation of Diagnosability of Failure Knowledge in Manufacturing Systems*, Proceedings, 1990 IEEE International Conference on Robotics and Automation, Vol 1, pp. 696-701.
9. N. H. Narayanan and N. Vishwanadham, "A Methodology for Knowledge Acquisition and Reasoning in Failure Analysis of Systems," *IEEE Trans. Syst., Man and Cybernetics*, vol. SMC-17, no. 2, Mar./Apr. 1987, pp. 274-288.
10. Biegl, C. A., "Design and Implementation of an Execution Environment for Knowledge-Based Systems," Ph.D. Thesis, Department of Electrical Engineering, Vanderbilt University, 1988.
11. S. Padalkar et al., "Real-Time Fault Diagnostics," *IEEE Expert*, Vol. 6, No.3, pp. 75-85, June 1991.
12. Hamscher, W. C., "Modeling Digital Circuits for Troubleshooting," *Artificial Intelligence*, vol. 51, 1991, pp. 223-271.
13. Yu, X. and G. Biswas, "A Method for Diagnosis of Continuous-valued Systems," *Working Papers: Third Intl. Workshop on Principles of Diagnosis*, Rosario, WA, 1992, pp. 57-66.

It also performs the same analysis for all possible combinations of failures in this set and generates a trace.

Consistency Checking: When DTOOL first comes up, it reads in the model files and checks the models for consistency. If it finds something inconsistent, it may (1) list all the inconsistencies in the system and exit or (2) assume an appropriate default and continue on. In either case, the user is given warning messages and enough information about the inconsistencies which can be used to remove the errors in the model.

The analysis algorithms described in Chapter IV used matrices (A^*, A^{min} and A^{max}) for reachability and minimum and maximum time for propagation form one failure to another. Using the matrices made the analysis very fast but had two drawbacks (particularly with large systems, having 500 or more failures) :

1. A considerable amount of memory was used up by the matrices.

2. The time to build these matrices was very large since the algorithms to compute, for example, the A^* matrix is $O(n^3)$. Thus, it took a lot of time to build up these data structures, effectively rendering DTOOL useless for large systems.

To reduce the computational complexity, the analyses algorithms were modified such that :

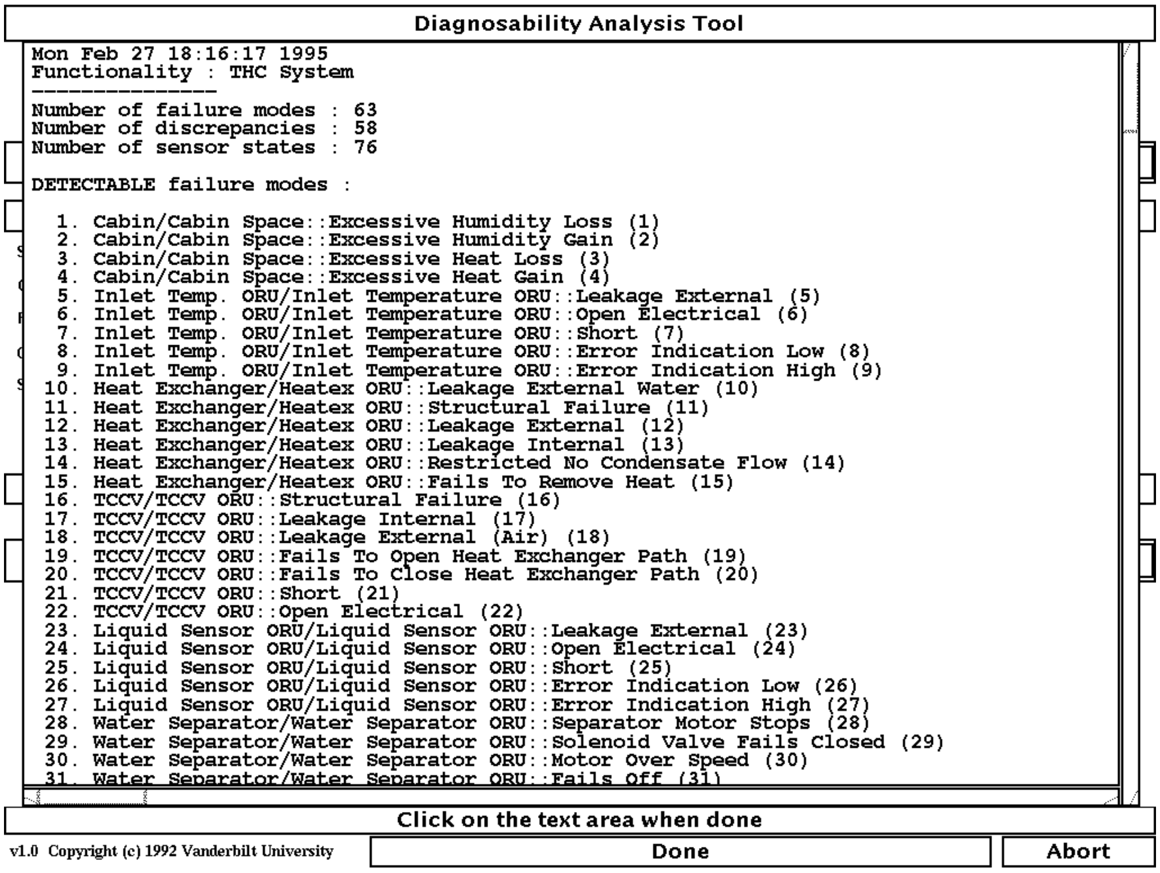
- The calls to determine the reachability and minimum and maximum times for propagation were used as few times as possible.

- Whenever these calls are made, a depth first search of the FPG is initiated, which is only $O(n^2)$.

In addition, some other optimizations were done such that the total time for analysis was reduced by two orders of magnitude, thus making DTOOL usable for very large systems also.

Miscellaneous

Multiple Fault Analysis: The user may want to see how interacting failures effect the propagations of failures in the system. For this, DTOOL allows the user to specify a set of failure modes for multiple failure analysis. DTOOL then assumes that all of these failure modes have occurred and gives a trace of the failure propagations.


Diagnosability Analysis Tool
Click on the text area when done
v1.0 Copyright (c) 1992 Vanderbilt University
Done
Abort

```
Mon Feb 27 18:16:17 1995
Functionality : THC System
---------------
Number of failure modes : 63
Number of discrepancies : 58
Number of sensor states : 76

DETECTABLE failure modes :

  1. Cabin/Cabin Space::Excessive Humidity Loss (1)
  2. Cabin/Cabin Space::Excessive Humidity Gain (2)
  3. Cabin/Cabin Space::Excessive Heat Loss (3)
  4. Cabin/Cabin Space::Excessive Heat Gain (4)
  5. Inlet Temp. ORU/Inlet Temperature ORU::Leakage External (5)
  6. Inlet Temp. ORU/Inlet Temperature ORU::Open Electrical (6)
  7. Inlet Temp. ORU/Inlet Temperature ORU::Short (7)
  8. Inlet Temp. ORU/Inlet Temperature ORU::Error Indication Low (8)
  9. Inlet Temp. ORU/Inlet Temperature ORU::Error Indication High (9)
 10. Heat Exchanger/Heatex ORU::Leakage External Water (10)
 11. Heat Exchanger/Heatex ORU::Structural Failure (11)
 12. Heat Exchanger/Heatex ORU::Leakage External (12)
 13. Heat Exchanger/Heatex ORU::Leakage Internal (13)
 14. Heat Exchanger/Heatex ORU::Restricted No Condensate Flow (14)
 15. Heat Exchanger/Heatex ORU::Fails To Remove Heat (15)
 16. TCCV/TCCV ORU::Structural Failure (16)
 17. TCCV/TCCV ORU::Leakage Internal (17)
 18. TCCV/TCCV ORU::Leakage External (Air) (18)
 19. TCCV/TCCV ORU::Fails To Open Heat Exchanger Path (19)
 20. TCCV/TCCV ORU::Fails To Close Heat Exchanger Path (20)
 21. TCCV/TCCV ORU::Short (21)
 22. TCCV/TCCV ORU::Open Electrical (22)
 23. Liquid Sensor ORU/Liquid Sensor ORU::Leakage External (23)
 24. Liquid Sensor ORU/Liquid Sensor ORU::Open Electrical (24)
 25. Liquid Sensor ORU/Liquid Sensor ORU::Short (25)
 26. Liquid Sensor ORU/Liquid Sensor ORU::Error Indication Low (26)
 27. Liquid Sensor ORU/Liquid Sensor ORU::Error Indication High (27)
 28. Water Separator/Water Separator ORU::Separator Motor Stops (28)
 29. Water Separator/Water Separator ORU::Solenoid Valve Fails Closed (29)
 30. Water Separator/Water Separator ORU::Motor Over Speed (30)
 31. Water Separator/Water Separator ORU::Fails Off (31)
```

Figure 33. Report Panel

The diagnosability scores for the failure modes and discrepancies can be seen by clicking on the corresponding node. DTOOL writes the scores in the text window above the FPG area.

Report Panel

The report panel is popped up when the user asks to generate the report on screen. It contains the evaluation and analysis results in textual form. Clicking on the text area of the report panel pops down the panel. Figure 33 shows the report panel.

Computational Complexity

The first version of DTOOL used

clicking on the corresponding button in the row above the text window. When a button, say, for detectability, is gray text on white, the detectability advice will be shown. When it is white text on gray, the advice won't be shown. When the user asks for results of a specific advice, but no such advice has been generated yet, nothing happens to the FPG. However, if such an advice has been generated, then some of the nodes in the FPG are colored differently to show the results.

The detectability advice results are shown by turning white the nodes for the failure modes that were selected for advice. The discrepancies that need alarms and the sensor states that need to be BIT and enabled to satisfy the detectability criteria are turned yellow. If a failure mode which was selected for advice is not detectable, the text in that node is written in red otherwise it is written in gray. Clicking on a failure mode that is detectable (gray text on white) will make a yellow discrepancy/sensor state to turn black for one second and then yellow again. This way one can find out which discrepancy is needed to detect which failure mode.

The distinguishability advice results are shown in the same manner as detectability advice results except that the failure modes, discrepancies and sensor states that are colored differently correspond to distinguishability advice.

The predictability advice is shown by turning the discrepancies/sensor states that were selected for advice black. If a discrepancy is predictable, the text for that node is written in pink or green depending upon whether it is monitored or not, otherwise the text is written in red. If a sensor state is predictable, the text is written in orange or pale green depending upon whether it is enabled BIT or not. Other the text is written in red. The discrepancies that need alarms to satisfy the predictability criterion are colored yellow.

If the `Statistical` button is turned on, the results shown are those which were generated by statistical analysis.

- All the discrepancies and sensor states that were already "selected-for- predictability" advice are now reverse video, i.e., pink or green text on black for a discrepancy, and orange or pale green text on black for a sensor state.

The user can click on any discrepancy to toggle its selected-for-predictability advice status and correspondingly toggle its colors. Clicking on failure modes won't have any effect.

Changing Alarm and Sensor State Allocation

When the user wants to change alarm allocation, the FPG panel is popped without any special coloring. The discrepancies that are currently monitored are colored pink and those that are not are colored green. The user can change the alarm allocation by clicking on any of these discrepancies. When the user clicks on a discrepancy node, its monitored status is toggled, i.e., if it had an alarm on it, the alarm is removed and vice-versa. The color of the node is also changed accordingly. Similarly, the user can change the sensor state allocation by clicking on sensor state buttons.

Note that changes in alarm and sensor state allocation are local to the current study. Thus, two different studies could be operating on the same functionality and have completely different alarm and sensor state allocations for that functionality. When the user finishes changing the alarm and sensor state allocation, DTOOL automatically recomputes the diagnosability scores for the failure modes and discrepancies in the functionality.

Viewing Results

When viewing results one may choose to view the results of detectability, distinguishability or predictability advice, or any combinations thereof. This is done by

Detectability Advice

When the user wants to select failure modes for detectability advice, the FPG panel is popped with the following changes from the normal coloring scheme:

- The Detectability button is now gray text on white.
- All the failure modes that were already "selected-for-detectability" advice are also gray on white.

The user can click on any failure mode to toggle its selected-for-detectability advice status and correspondingly toggle its colors. Clicking on discrepancies or sensor states won't have any effect.

Distinguishability Advice

When the user wants to select failure modes for distinguishability advice, the FPG panel is popped with the following changes from the normal coloring scheme:

- The Distinguishability button is now gray text on white.
- All the failure modes that were already "selected-for-distinguishability" advice are also gray on white.

The user can click on any failure mode to toggle its selected-for-distinguishability advice status and correspondingly toggle its colors. Clicking on discrepancies or sensor states won't have any effect.

Predictability Advice

When the user wants to select discrepancies/sensor states for predictability advice, the FPG panel is popped with the following changes from the normal coloring scheme:

- The Predictability button is now gray text on white.

display is on, clicking on any node in the graph will make DTOOL draw all the paths leading to that node in red. This way the user can highlight all the failure propagations that can lead to that node.

- **Set Time** : This button is used to set the time parameter for advice.

- **Set Probability** : This button is used to set the probability parameter for advice.

- **All** : This button is used to select *all* the failure modes/discrepancies/sensor states for advice.

- **Statistical** : This is used to turn on the display of the results of statistical analysis.

The colors of some of the buttons on the FPG panel are:

- The `Detectability`, `Distinguishability` and `Predictability` buttons are normally grey with white text.

- Failure modes are gray, with the text in white. The text on failure mode nodes contain the name of the component followed by the name of the failure mode.

- A discrepancy node is colored pink if it is a monitored discrepancy and colored green if it is a non-monitored discrepancy. The text on the discrepancy node is in black and contains the name of the sub-functionality to which this discrepancy belongs followed by the name of the discrepancy.

- A sensor state node is colored orange if it is an enabled BIT sensor state, otherwise it is colored pale green. The text is in black and contains the name of the sensor state.

The large part in the middle of the panel shows the Failure Propagation Graph (FPG) of the functionality (individual or flat hierarchy) that is being operated on. The nodes of the graph represent the failure modes and discrepancies in the FPG. The edges represent the propagations between them. All the nodes corresponding to failure modes are drawn in a column at the left and discrepancies are drawn in columns to the right of the failure mode column. The colors of the nodes and edges are used to convey certain information to the user, which will be discussed in upcoming sections.

Below the FPG area and to its right are scroll bars which can be used to scroll the graph (if the graph is too large to fit in the display area). Above the FPG area are two rows of buttons and one text window. The text window, just above the FPG area, is used to display the diagnosability scores of the failure modes and discrepancies. The `Detectability`, `Distinguishability` and `Predictability` buttons above the text window are used to select the advice whose results are to be shown.

The top row has buttons to zoom in and out of the FPG and to "close" the FPG panel, i.e., to pop down this panel and go back to the analysis panel. There are a few more buttons which do the following:

- **Effects** : When the user clicks on this button, the effects display is toggled (white text on gray – off, gray text on white – on). Normally all the edges in the FPG are drawn with color blue. However, when the effects display is on, clicking on any node in the graph will make DTOOL draw all the paths in the graph originating in that node to be drawn in yellow. This way the user can highlight the failure propagations in the FPG starting from any node.

- **Caused By** : When the user clicks on this button, the caused-by display is toggled (white text on gray – off, gray text on white – on). When the caused-by

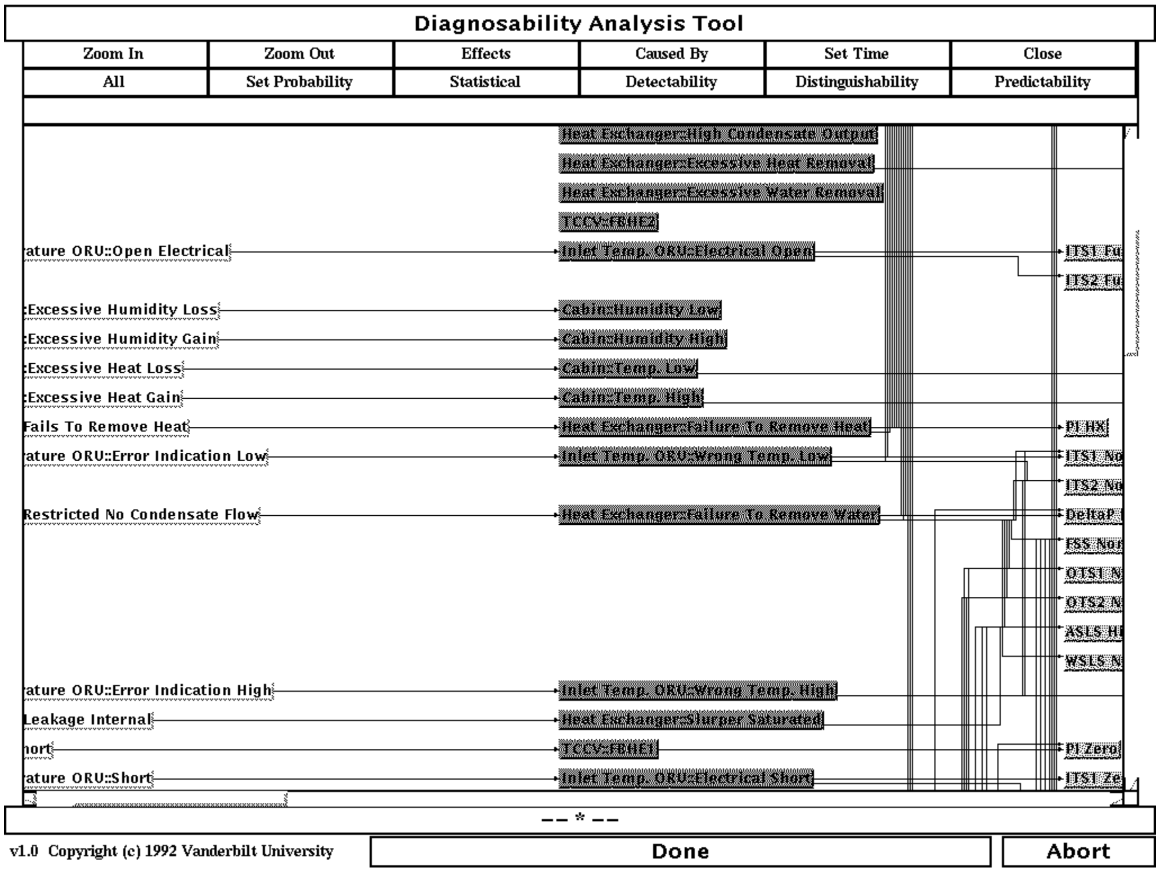

Diagnosability Analysis Tool
Zoom In
Zoom Out
Effects
Caused By
Set Time
Close
All
Set Probability
Statistical
Detectability
Distinguishability
Predictability
Heat Exchanger::High Condensate Output
Heat Exchanger::Excessive Heat Removal
Heat Exchanger::Excessive Water Removal
TCCV::FBHE2
ature ORU::Open Electrical
Inlet Temp. ORU::Electrical Open
ITS1 Fu
ITS2 Fu
:Excessive Humidity Loss
Cabin::Humidity Low
:Excessive Humidity Gain
Cabin::Humidity High
:Excessive Heat Loss
Cabin::Temp. Low
:Excessive Heat Gain
Cabin::Temp. High
Fails To Remove Heat
Heat Exchanger::Failure To Remove Heat
PI HX
ature ORU::Error Indication Low
Inlet Temp. ORU::Wrong Temp. Low
ITS1 No
ITS2 No
Restricted No Condensate Flow
Heat Exchanger::Failure To Remove Water
DeltaP
FSS Nor
OTS1 N
OTS2 N
ASLS HI
WSLS N
ature ORU::Error Indication High
Inlet Temp. ORU::Wrong Temp. High
Leakage Internal
Heat Exchanger::Slurper Saturated
hort
TCCV::FBHE1
PI Zero
ature ORU::Short
Inlet Temp. ORU::Electrical Short
ITS1 Ze
-- * --
v1.0 Copyright (c) 1992 Vanderbilt University
Done
Abort

Figure 32. FPG Panel

Selecting Individual Functionality

When the user clicks on the `Select Functionality` button in the analysis panel, DTOOL pops up the hierarchy panel and prompts the user to select a functionality. The user must then click on one of the functionalities that have at least one child functionality. Clicking on a functionality which is a leaf of the hierarchy will result in an error message from DTOOL. Once the user clicks on a functionality that has children, that functionality becomes the individual functionality for analysis in the current study. If the user tries to close the panel without selecting a functionality, DTOOL gives an error message.

Flat Hierarchy

The hierarchy panel is also used to select a part of the hierarchy to flatten and to display this information. Normally the buttons corresponding to functionalities in the hierarchy are colored green unless they are part of the flat hierarchy, in which case they are colored brown. Thus, when the user clicks on `Select All` in the analysis panel, all the buttons will be colored brown. When the user `Reset`s the flat hierarchy, all buttons will be colored green.

When the user wants to `Specify` a part of the hierarchy to flatten, DTOOL prompts the user to first select a functionality that is the root of the subtree of interest and then a functionality which is a leaf of the subtree of information. Then the subtree is displayed using color brown.

FPG Panel

The FPG panel is popped up whenever the user wants to select failure modes and/or discrepancies/sensor states for advice, to view the results or to change alarm allocation. Figure 32 shows the FPG panel.

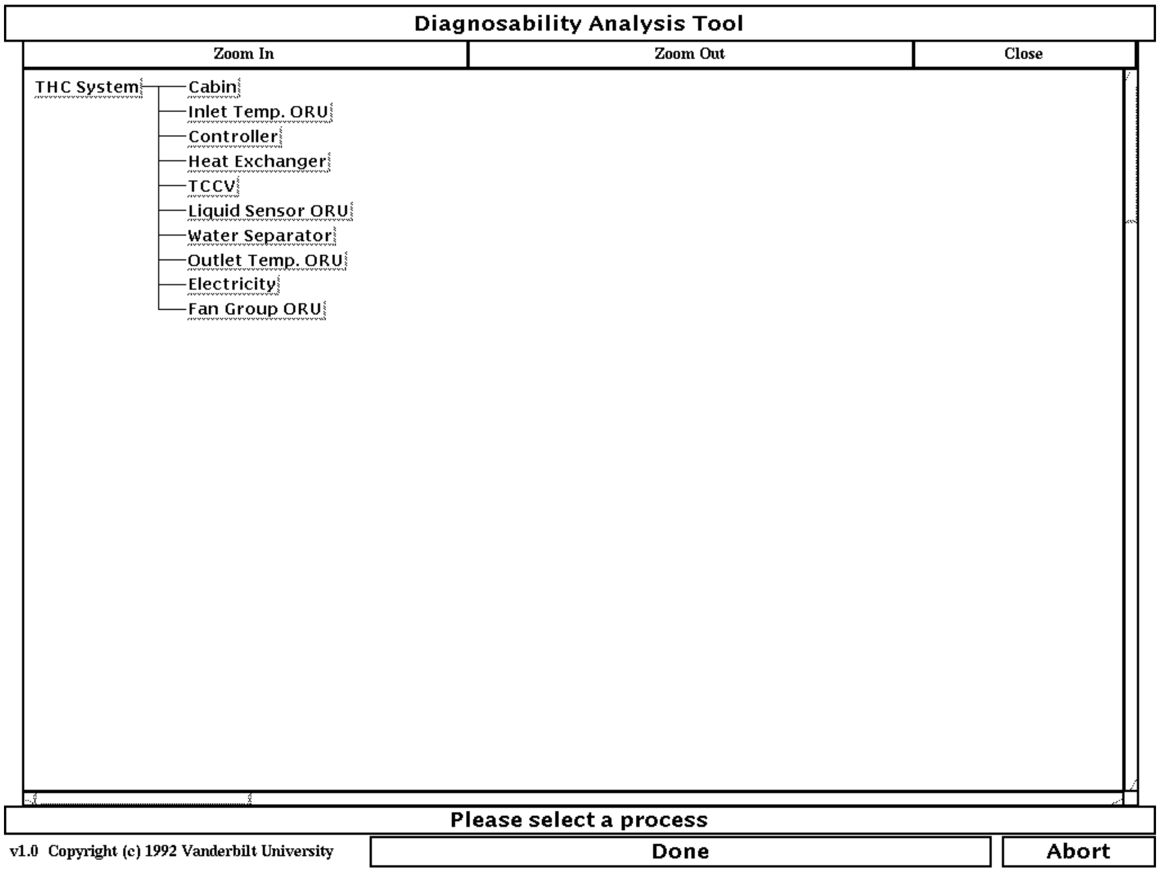


Figure 31. Hierarchy Panel

Hierarchy Panel

The hierarchy panel is popped up whenever the user wants to select an individual functionality or to specify a part of the hierarchy to flatten. Figure 31 shows the hierarchy panel. In the middle of the panel, the functional hierarchy is displayed. At the bottom and right of the hierarchy are scroll bars to scroll the hierarchy (this is useful if the hierarchy is so big that it does not fit in the display area). At the top are buttons to zoom the hierarchy figure in or out (again, useful if the hierarchy does not fit into the display area). There is also a `Close` button which will "close" this panel, i.e, pop down this panel and go back to the analysis panel.

characters) with extension `.sdy`. The file will be written in the study directory. Any already existing file with the same name will be overwritten. Note that any study not explicitly saved will be lost once DTOOL is exited.

Clicking on the `Load` button will pop up a scrollable list of all the files with `.sdy` extension that DTOOL could find in the study directory. By clicking on one of the filenames, the user can ask DTOOL to load the file. If the study in the selected file has the same name as an already existing study, the file is not loaded. Otherwise the file is loaded and the study contained in that file becomes the current study.

The study directory can be changed by clicking on the `Change Directory` button. DTOOL will prompt the user to type in the path of the directory. If the directory exists, it will become the current study directory, else DTOOL will give an error message.

- **Copy Study** : This can be used to create a study with some (perhaps all) of parameters already set. When the user clicks on this button, DTOOL prompts the user for a new study name. After getting the name, DTOOL creates the new study, copies the parameters of the current study into the new study and makes the new study the current study. This is useful if the user wants to create a new study which differs from the current study in only a parameter or two, but does not want to go through the process of specifying every parameter.

- **Change Attributes** : This is reserved for future use.

browse through the report. When the user asks that the report be saved on the disk, the report is saved in the study directory in a file named by appending ".rep.fmt1" or ".rep.fmt2" (corresponding to the format) to the name of the study.

Manipulating Studies

The row of buttons titled `Study` just below the analysis buttons described in the previous section is used for manipulating studies. DTOOL maintains a list of studies with all of their parameters that have been specified. At any given time only one of these studies is "open" and is called the current study. These studies can be written to disks for retrieval and further experimentations later. The manipulation of studies is done by functions offered by the "study" buttons, which are:

- **New Study** : This is used to create a new study. When the user clicks on this button, DTOOL prompts the user for the name of the study. After getting the name, DTOOL create an empty study, i.e., a study for which none of the parameters have been specified. The new study becomes the current study. The user can then select the parameters for the study.

- **Select Study** : Clicking on this button pops up a scrollable list of names of the studies present. The user can then select a study by clicking on its name and that study will become the current study, with all of its parameters restored.

- **File** : Clicking on this button will pop up a menu which will allow the user to save or load a study and to change the study directory.

  Clicking on the `Save` button will make DTOOL write the information about the current study in a file whose name is the name of the study (first eight

Performing Analyses

The buttons in the top part of the analysis panel allow user to perform analysis on the functionality chosen for study. At the top of the panel, just below the `Analysis Panel` label, is a row of five buttons whose functions are as follows:

- **Evaluate** : When the user clicks on this button, DTOOL evaluates the diagnosability scores for the failure modes, discrepancies and sensor states in the functionality being operated on (individual or flat hierarchy).
- **Statistical Analysis** : This button toggles the statistical analysis switch. When the button is drawn in grey with white text, statistical analysis is off. When the button is drawn in white with grey text, statistical analysis is on. This controls whether evaluation and advice done with statistical analysis or not.
- **Advise** : This button is used to ask DTOOL to generate advice for the criteria selected for advice as described in a previous section.
- **Results** : Clicking on this button will pop up the FPG panel and then the user can view the results of evaluation and advice. This is discussed in more detail later.
- **Change Alarm Allocation** : Clicking on this button will pop up the FPG panel and the user can change the alarm allocation. Discussed in more detail later.
- **Generate reports** : Clicking on the `Report` button will pop up a menu from which the user can select the appropriate option to generate format 1 or format 2 report on disk or to view it on the screen. When the user asks the report to be displayed on the screen, a scrollable text window pops up and the user can

suggestion than when advice for this and another criteria is asked for.

The advice selection is made by clicking on any of the `Detectability`, `Distinguishability` and `Predictability` buttons in the row of buttons titled `Select Advice`, at the top of the lower part of the analysis panel. Clicking on these buttons toggles the state for the advice corresponding to that criteria – from generate advice to do not generate advice and vice versa. The state is shown by coloring the button differently. White text on gray means that no advice will be generated to satisfy that criteria. Gray text on white means advice will be generated.

Current Parameters of a Study

The parameters of the study currently open are displayed in the middle part of the analysis panel. The information displayed is the following (from the top line to the bottom):

- Name of the study.
- The name of the selected individual functionality, if any.
- Whether there is a current selection for flat hierarchy functionality.
- Whether DTOOL is operating on individual functionality, flat hierarchy or nothing. When the user wants to operate on individual functionality but has not selected the individual functionality, or when the user wants to operate on flat hierarchy but has not specified the flat hierarchy, DTOOL is "operating on nothing". *Note that if DTOOL is "Operating on nothing", then trying to select parameters for advice or to change alarm allocation etc. will result in an error message from DTOOL.*
- The directory where study files will be retrieved from and stored.

hierarchy will be lost. It will just remain hidden until the user selects to operate on the flat hierarchy.

The selection is done by clicking on either the `Flattened Hierarchy` or the `Individual Functionality` button. DTOOL will change its state accordingly.

Selecting parameters for advice: The user can set the parameters to ask for alarm allocation suggestion for satisfying detectability, distinguishability or predictability criteria. The matrix of buttons titled `Parameters for Advice` in the bottom center part of the panel are used to specify these parameters. Each of the row of buttons can be used to select the time, probabilities and failure modes or discrepancies/sensor states for the above three criteria.

The top row is used to specify the parameters for detectability advice. The user can specify the time limit for detectability by clicking on the `Time = hh:mm:ss` button. This button displays the present selection for the time. When the user clicks on this button, another panel will be popped up which will allow the user to set the hour, minute and second values by clicking on up and/or down arrows. When the user clicks on the `Probability = x.xxx` button, DTOOL prompts the user to type in the desired probability. This probability value is then used for statistical analysis.

To select the failure modes of interest, the user should click on the `Failure Modes` button. The FPG panel will be popped up and the user can select the failure modes. This is described in a later section.

Similarly, the user can select the time, probability and failure modes and discrepancies/sensor states parameters for distinguishability and predictability advices.

Selecting advice: Even if the user selects the parameters for the three types of advice, it doesn't mean that DTOOL will generate alarm allocation advice for those criteria. The DTOOL needs to be told explicitly the criteria for which the user wants advice. This is because asking for advice for one criteria may generate a different

Selecting Parameters for a Study

Selecting individual functionality: The functionality to operate upon can be selected by clicking on the `Select Functionality` button in the lower right corner of the analysis panel. When the user clicks on this panel, the hierarchy panel will be displayed and DTOOL will prompt the user to select a functionality.

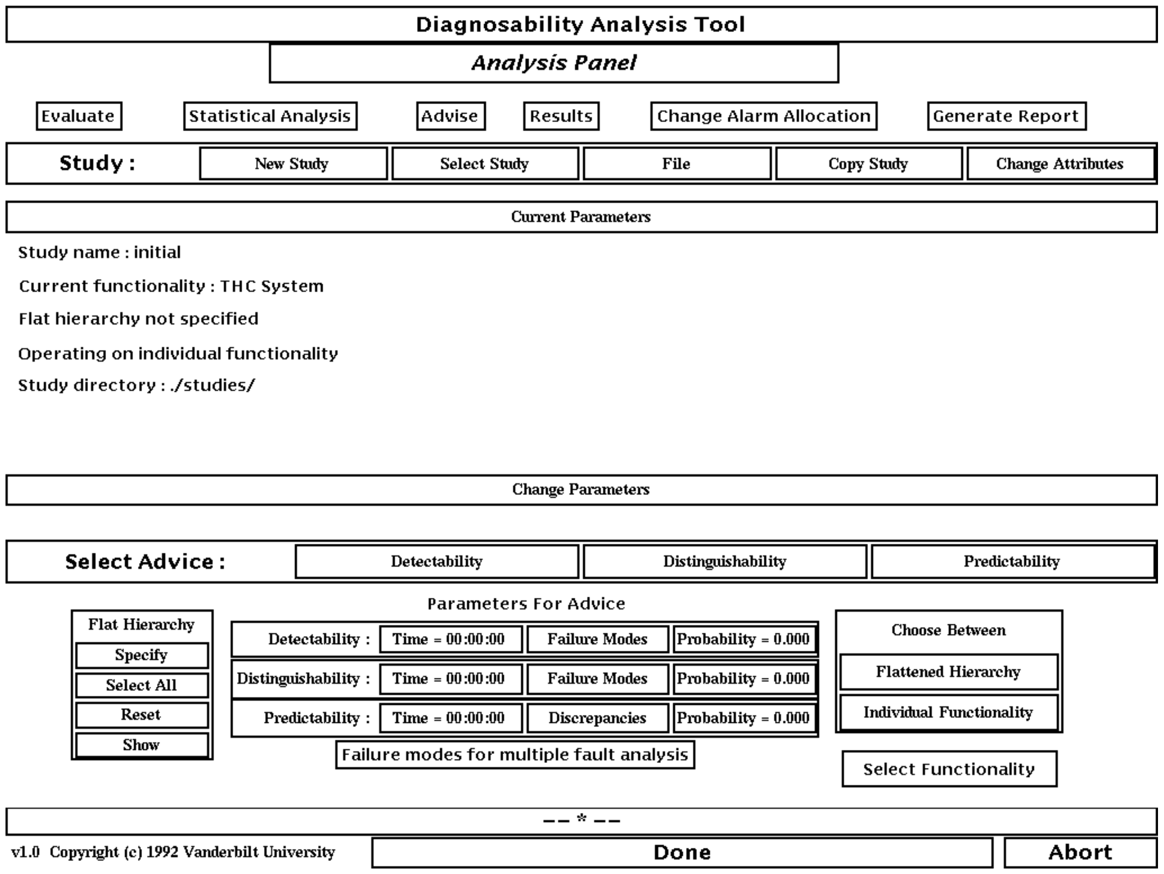
Selecting flat hierarchy: The column of buttons titled `Flat Hierarchy` at the lower left corner of the analysis panel allows the user to manipulate the flat hierarchy. By clicking on the `Specify` button, the user can select the part of the hierarchy that he wants to flatten. The hierarchy panel will be popped up and the user will be prompted to select the root and leaf of the subtree of interest.

By clicking on the `Select All` button the user can ask DTOOL to flatten the whole hierarchy. By clicking on the `Reset` button the user can ask DTOOL to discard any previous selection for flat hierarchy. The `Show` button will pop up the hierarchy panel so that the user can see the part of the hierarchy that has been flattened.

When a part of the hierarchy is chosen for flattening, DTOOL creates a pseudo-functionality for that subtree. This functionality is referred to as the flat hierarchy functionality.

Switching between individual and flat hierarchy functionalities: The column of buttons titled `Choose Between` just above the `Select Functionality` button allows the user to switch between an individual functionality and the flat hierarchy functionality.

In any study, one may specify an individual functionality and a flat hierarchy functionality to analyze. But only one of these may be operated on at one time. Thus, the analysis will be performed for and the results will be shown for the functionality that is currently selected. This does not mean tha when one operates on, say, an individual functionality, the information about any previous analysis done on the flat



Study name : initial
Current functionality : THC System
Flat hierarchy not specified
Operating on individual functionality
Study directory : ./studies/

Detectability : Time = 00:00:00 Probability = 0.000
Distinguishability : Time = 00:00:00 Probability = 0.000
Predictability : Time = 00:00:00 Probability = 0.000

Figure 30. Analysis Panel

asks for a service, some other panel may be popped up. When the user has finished using that service, that panel is popped down and the user interface goes back to the analysis panel. In the subsequent sections, we'll discuss these panels and the services they provide.

Analysis Panel

Figure 30 shows the analysis panel of DTOOL. The analysis panel provides access to the evaluation and advice services, to study management etc. The top part of panel has buttons to ask for advice, to change alarm allocation, to generate report etc. The center part shows some of the current parameters of a study. The bottom part allows the user to select the parameters for a study.

```
            [criticality :
              NONE !! (assigned) NONE !! (inherited)]
                Alarms generated :
                - NONE
           |
           |--> Inlet Temp. ORU::Temp. Low
                (secondary effect)
                [criticality :
                  NONE !! (assigned) NONE !! (inherited)]
                    Alarms generated :
                    - NONE
```

After this comes a trace of the multiple fault analysis that user may ask for (discussed in a later section).

User Interface

A graphical user interface (USINT) has been implemented for DTOOL. The USINT, described below, has been implemented using the Simple User Interface Toolkit (SUIT) and runs under X windows. USINT consists of a number of "panels" which provide different services. At the top of the screen is the title `Diagnosability Analysis Tool`. At the bottom are two buttons – `Done` and `Abort` that can be used to quit DTOOL by clicking on any one of them. Just above these buttons is a text window (yellow background) which is used to display messages and prompts to the user. Usually it just contains the string "`-- * --`". Between the title and the text window, the various panels used by DTOOL are displayed.

When DTOOL comes up, the panel shown is the "analysis" panel. As the user

```
- ITS1 Over StPt

- ITS2 Over StPt

- PI Nominal

- OTS1 High

- OTS2 High
```

Next is a list of discrepancies with their predictability and list of sensor states reached. Following that is a list of sensor states with their predictability and a list of failures that reach them.

The second report format consists of a list of failure modes with their assigned and inherited criticality and a trace of the failure propagations. Following is an example of one such trace :

```
19. TCCV/TCCV ORU::Fails To Open Heat Exchanger Path
    [criticality : NONE !! (assigned) NONE !! (inherited)]


    FAILURE PROPAGATION (reachability analysis):
      |
      |--> TCCV::Fails To Connect
             (primary effect)
             [criticality :
               NONE !! (assigned) NONE !! (inherited)]
                 Alarms generated :
                 - NONE
          |
          |--> Outlet Temp. ORU::Temp. Low
                 (secondary effect)
```

```
        (AND

            |OTS1 High|

            |OTS2 High|

            |PI Nominal|

            |ITS1 Over StPt|

            |ITS2 Over StPt|)


    DISCREPANCY |TCCV::Fails To Connect| propagates to

    DISCREPANCY |Outlet Temp. ORU::Temp. Low|

     SENSOR STATES impacted by this discrepancy :

        (AND

            |OTS1 Low|

            |OTS2 Low|)


    DISCREPANCY |Outlet Temp. ORU::Temp. Low| propagates to

    DISCREPANCY |Inlet Temp. ORU::Temp. Low|

     SENSOR STATES impacted by this discrepancy :

        (AND

            |ITS1 Low|

            |ITS2 Low|)


-- SENSOR STATES reached :

   - ITS1 Low

   - ITS2 Low

   - OTS1 Low

   - OTS2 Low
```

DTOOL Reports

Reports are a way to present the result of analyses in a textual form. Even though DTOOL provides a graphical user interface and shows the result in a graphical form, navigating through all the graphics can be quite tedious, particularly when analyzing large systems. Reports provide the results of analyses in formats that are as concise or detailed as the user wants and can be tailored to suit the user's needs. The reports can be saved in a disk file and printed out. Currently, DTOOL generates reports in two formats.

In the first format report, The failure modes are first listed according to whether they are detectable or not and according to whether they are distinguishable or not. Next, all the failure modes are listed along with there detectability, distinguishability, list of sensor states reached and a trace of the failure propagation. Following is an example of one such trace :

```
 19. TCCV/TCCV ORU::Fails To Open Heat Exchanger Path


      -- Detectability = <0.000000,0.000000>

      -- Distinguishable


      -- Propagations of this failure mode :


            FAILURE MODE

               |TCCV/TCCV ORU::Fails To Open Heat Exchanger Path|

               propagates to

            DISCREPANCY |TCCV::Fails To Connect|

             SENSOR STATES impacted by this discrepancy :
```

concept that DTOOL uses to organize these analyses is called a "Study". A study captures the following pieces of information:

- The name of the study.
- The functionality being analyzed (the functionality could be an individual functionality or flat hierarchy).
- For the above two functionalities
  - Current alarm and sensor state allocation.
  - The diagnosability scores with the current alarm and sensor state allocation.
  - Alarm and sensor state allocation suggestions with information about which alarm/sensor state is needed to satisfy which criteria.

The information contained in a study can be stored on disk in the study directory (which is specified in the configuration file and can also be changed when DTOOL is running). The name of the file in which a study is stored is derived from the name of the study (up to first eight characters), with an extension of `.sdy`.

When DTOOL comes up, it looks for a file named `initial.sdy` in the study directory and loads that study if it finds the file. If the file does not exist, it opens a study with the name `initial`, which is "empty", in the sense that it contains none of the above information except the name of study. Also, whenever the user creates a new study, the study is initially empty. The user must then select the functionalities, alarm allocations etc.

DTOOL provides such facilities as creating, saving, loading, copying the studies. This is described in a later section.

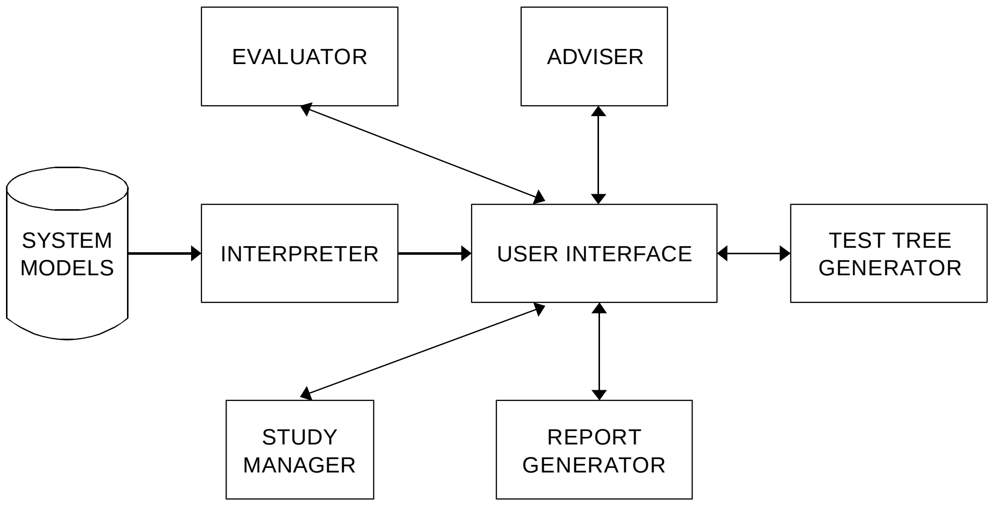


Figure 29. Diagnosability Analysis Tool

- **Test Tree Generator** : This component constructs a test tree which can be used for sequential diagnosis.

- **Study Manager** : The study manager allows user to perform many different diagnosability studies and save and retrieve them from the disk.

- **Report Generator** : This component gernerates reports about the diagnosability of the system in textual form.

- **User Interface (USINT)** : USINT provides a graphical interface to the above analysis services.

Study Management

DTOOL allows the user to experiment with the system, to change alarm and sensor state allocations and see how the diagnosability of the system changes. It also allows the user to ask for advice to satisfy a variety of diagnosability criteria. All these analyses can become very unmanageable unless they are organized. The

APPENDIX C

DIAGNOSABILITY ANALYSIS TOOL

A Diagnosability Analysis Tool (DTOOL) is described. DTOOL provides a graphical user interface to the analyses discussed in chapter IV. DTOOL operates on the fault models of a system and presents the results of analyses graphically and in the form of text reports. Using the DTOOL, a design engineer can answer a number of diagnosis related questions (including the following) :

- How will a failure in a specific component manifest itself?
- How long will it take to detect a fault in a specific component?
- Can a component failure be predicted?
- Does adequate sensor coverage exist to isolate faults to the components ?

A block diagram of DTOOL is shown in Figure 29. The different components of DTOOL perform the following tasks :

- **Interpreter** : The interpreter takes the models in the model database, interprets them and converts them into data structures used by the other components of DTOOL.
- **Evaluator** : This component analyzes the system and computes the diagnosability metrics with a predefined sensor allocation.
- **Adviser** : The adviser generates suggestions about alarm and sensor allocation that can meet the diagnosability criteria specified by the designer.

i. Set $T_{j,i} = 1$ if $A^*_{j+n,l} = 1$ and $A^{min}_{j+n,l} \geq \hat{t}$ else set $T_{j,i} = 0$.

3. Find the minimum cover for the discrepancies in D_1 by using the Quine-McClusky [16] algorithm on T. The alarm allocation suggestion is to put an alarm on the discrepancies that form the cover.

Algorithm to Compute $A^{p,t}$

The algorithm for computing $A^{p,t}$ uses a $(m+n+o) \times (m+n+o)$ matrix, T, which contains the values for propagation times. $T_{i,j}$ is the time a failure will take to propagate from vertex i to vertex j in G'. The matrix T is similar to A^{min} and A^{max} matrices. The algorithm is :

1. Initialize $A^{p,t}_{i,j} = 0, \ 1 \leq i \leq n, \ \ 1 \leq j \leq m + o$
2. Do $NUM\_SAMPLES$ times :
   (a) Generate a random propagation time for each edge in G', using the time interval and the probability distribution on the edge.
   (b) Compute values for T, by applying the shortest path algorithm on G' using the random propagation times generated above as the weights.
   (c) For $1 \leq i \leq n, \ \ 1 \leq j \leq m + o,$ if $T_{i,n+j} \leq t, \ \ A^{p,t}_{i,j} = A^{p,t}_{i,j} + 1$.
3. For $1 \leq i \leq n, \ \ 1 \leq j \leq m + o, \ \ A^{p,t}_{i,j} = A^{p,t}_{i,j}/NUM\_SAMPLES$.
4. For $1 \leq i \leq n, \ \ 1 \leq j \leq m + o,$ if $A^{p,t}_{i,j} \geq p, \ \ A^{p,t}_{i,j} = 1$ else $A^{p,t}_{i,j} = 0$.

following condition – an alarm is allocated to a discrepancy only if it makes a new failure mode distinguishable.

Algorithm for *BreakIntoClusters(T)*

```
1. Find the largest edge $\hat{e} \in T$.
2. Remove $\hat{e}$, thus creating two trees $T_1$ and $T_2$ corresponding to two failure mode
   clusters $C_1$ and $C_2$.
3. For all $d \in NSD_c^{C_1,C_2,\hat{t}}$ increment $P(d)$ by 1.
4. $BreakIntoClusters(T_1)$.
5. $BreakIntoClusters(T_2)$.
```

Algorithm For Predictability Advice

For predictability of discrepancies in the system, the advisory task is :

- Given a partial alarm allocation, a specified set of discrepancies $D_1 \subseteq D$, and a time interval of length $\hat{t}$, determine the optimal allocation of new alarms such that $t_{min} \geq \hat{t} \; \forall \; d \in D_1$.

The algorithm is based on finding minimum covering for a bipartite graph :

```
1. Form a $(m + o) \times k$ table $T$, where $k = |D_1|$. Let the discrepancies in the set
   $D_1$ be $\hat{d}_1, \hat{d}_2, ..., \hat{d}_k$.
2. For $i = 1$ to $k$ do
   (a) $l = Node(\hat{d}_i)$.
   (b) For $j = 1$ to $m + o$ do
```

$NSD^{\hat{f}_i,\hat{f}_j,\hat{t}}$, is a set of all discrepancies reachable by either $\hat{f}_i$ or $\hat{f}_j$ but not by both, in time $\hat{t}$. Consider a discrepancy $d \in D', Node(d) = k$. Then $d \in NSD^{\hat{f}_i,\hat{f}_i,\hat{t}}$ iff

$$A^*_{ni,k} = 0 \text{ and } A^*_{nj,k} = 1 \text{ and } A^{max}_{nj,k} \leq \hat{t}$$

or

$$A^*_{ni,k} = 1 \text{ and } A^*_{nj,k} = 0 \text{ and } A^{max}_{nj,k} \leq \hat{t}.$$

The distance function between $\hat{f}_i$ and $\hat{f}_j$ is defined as

$$\gamma(\hat{f}_i, \hat{f}_j, \hat{t}) = \frac{1}{|NSD^{\hat{f}_i,\hat{f}_j,\hat{t}}|} \tag{4}$$

The distance function defined above gives the weights used on the edges in $\hat{G}$.

During the divisive clustering, many clusters will be generated, each containing some failure modes. Consider two clusters C_i and C_j. Then the *non*-shared discrepancy set for the two clusters is defined as

$$NSD_c^{C_i,C_j,\hat{t}} = \bigcap_{\hat{f}_k \in C_i, \hat{f}_l \in C_j} NSD^{\hat{f}_k,\hat{f}_l,\hat{t}} \tag{5}$$

The algorithm assigns points to the discrepancies in order to rank them according to their importance. These points are denoted by $P(d)$. The algorithm to find an alarm allocation for distinguishability is :

1. Create $\hat{G}$ as defined above.

2. Find the minimal spanning tree T for the graph $\hat{G}$.

3. $BreakIntoClusters(T)$. The algorithm for $BreakIntoClusters$ is described below. It also allocates points to the discrepancies in the failure propagation graph.

4. Sort the discrepancies by the points given to them in non-increasing order. Then, one by one, allocate alarms for the discrepancies in this list in their non-decreasing order of points until the distinguishability criteria is met, with the

3. Find the minimum cover for the failure modes in F_1 by using the Quine-McClusky [16] algorithm on T. The alarm allocation suggestion is to put an alarm on the discrepancies that form the cover.

Algorithm For Distinguishability Advice

For distinguishability of failure modes in the system, the advisory task is :

- Given a partial alarm allocation and a set of failure modes $F_1 \subseteq F$, and a time interval of length $\hat{t}$, determine the optimal allocation of new alarms such that all $f \in F_1$ are distinguishable in time $\hat{t}$.

The algorithm for advising alarm allocation for distinguishability is based on hierarchical clustering. It uses the minimal spanning tree method for divisive clustering [3] of failure modes. The motivation for using hierarchical clustering comes from the fact that it captures the topology of the FPM in the division of failures into clusters and sub-clusters. For distinguishability advice, the failure modes are divided into clusters that share the least number of discrepancies. This gives an approximate idea of how "far" the failure modes are from each other. Examination of discrepancies reached by these clusters gives an idea of the relative importance of the discrepancies for distinguishing the failure modes.

The algorithm begins with building a complete graph, $\hat{G} = (V, E)$. The vertices in V represent the failure modes $f \in F_1$, i.e., vertex v_i represents $\hat{f}_i \in F_1$. The edge set E includes edges from every vertex to every other vertex, i.e., $e_{i,j} =< v_i, v_j >\in E \ \forall \ 1 \leq i \leq k, \ 1 \leq j \leq k, k = |F_1|$. Each edge $e_{i,j}$ in $\hat{G}$ is weighted with the "distance" between failure modes $\hat{f}_i$ and $\hat{f}_j$.

For defining the distance between failure modes, consider two failure modes $\hat{f}_i, \hat{f}_j \in F_1$. Let $Node(\hat{f}_i) = ni$ and $Node(\hat{f}_j) = nj$. The non-shared discrepancy set,

```
        i. Let $i = Node(\hat{f})$ and $< t_{erl}, t_{lat} >$ be the detectability of $\hat{f}$.
        ii. If $t_{erl} < A^{min}_{i,j}$ and $t_{lat} < A^{min}_{i,j}$ then
            • if $A^{min}_{i,j} - t_{erl} > t_{max}$ then $t_{max} = A^{min}_{i,j} - t_{erl}$.
            • if $A^{min}_{i,j} - t_{lat} > t_{min}$ then $t_{min} = A^{min}_{i,j} - t_{lat}$.
    (d) The final values of $t_{min}$ and $t_{max}$ give the predictability of $d$.
```

Advice Algorithms

Algorithm For Detectability Advice

For detectability of failure modes in the system, the advisory task is :

- Given a partial alarm allocation, a specified set of failure modes $F_1 \subseteq F$, and a time interval of length $\hat{t}$, determine the optimal allocation of new alarms such that $t_{lat}(f) \leq \hat{t} \;\; \forall \, f \in F_1$.

The algorithm for generating detectability advice is based on finding minimum covering for a bipartite graph :

```
1. Form a $(m + o) \times k$ table $T$, where $k = |F_1|$. Let the failure modes in the set
   $F_1$ be $\hat{f}_1, \hat{f}_2, ..., \hat{f}_k$.
2. For $i = 1$ to $k$ do
    (a) $l = Node(\hat{f}_i)$.
    (b) For $j = 1$ to $(m + o)$ do
        i. Set $T_{j,i} = 1$ if $A^{*}_{l,j+n} = 1$ and $A^{max}_{l,j+n} \leq \hat{t}$ else set $T_{j,i} = 0$.
```

Algorithm For Evaluating Distinguishability

The following algorithm for evaluating distinguishability is similar to the covering analysis presented in [8].

```
1. Form a $n \times (m+o)$ matrix $\Omega$ such that for $1 \leq i \leq n,\ \ 1 \leq j \leq m+o,\ \ \Omega_{i,j} = 1$
   iff
     • $d_j$ is monitored and
     • $A^*_{i,j+n} = 1$ and
     • $A^{max}_{i,j+n} \leq \hat{t}$.
2. Compute $\Lambda = \Omega\Omega^T$.
3. For $i = 1$ to $n$, do
   (a) If $\Lambda_{i,i} = 0,\ DIST(f,t) = FALSE$, else
       i. $DIST(f,t) = TRUE$.
       ii. For $j = 1$ to $n$, do
           A. if $j \neq i$, then if $\Lambda_{i,i} = \Lambda_{i,j} \wedge \Lambda_{j,j} = \Lambda_{j,i},\ DIST(f,t) = FALSE$.
```

Algorithm For Evaluating Predictability

The algorithm to find the predictability of discrepancies is :

```
1. For all $d \in D'$ do
   (a) $j = Node(d)$; $t_{min} = t_{max} = 0$.
   (b) Find the set $F_1 \subseteq F$of failure modes that reach $d$, i.e.,
       $F_1 = \left\{\hat{f} \mid A^*_{Node(\hat{f}),j} = 1\right\}$.
   (c) For all $\hat{f} \in F_1$,
```

APPENDIX B

DIAGNOSABILITY ALGORITHMS

Evaluation Algorithms

Algorithm For Evaluating Detectability

The algorithm to assess the detectability of failure modes is:

```
1. For all $f \in F$ do
    (a) $i = Node(f)$; $list = EMPTY$;
    (b) For all $d \in D'$, do
        i. $j = Node(d)$
        ii. if $\underline{\mathrm{m}}_{j-n}$ then if $A^*_{i,j} = 1$ then add $j$ to $list$ and associate $A^{min}_{i,j}$ and $A^{max}_{i,j}$
            as keys for this item.
    (c) If $list = EMPTY$, $f$ is not detectable.
    (d) If $list \neq EMPTY$ then
        i. $f$ is detectable.
        ii. sort $list$ on key $A^{min}_{i,j}$ in increasing order. The first item on the list
            gives the earliest possible time of detection.
        iii. sort $list$ on key $A^{max}_{i,j}$ in increasing order. The first item on the list
            gives the latest possible time of detection.
    (a) for all $s$ such that $< s, \hat{a} > \in SA$ add $h^s =< s, \hat{a}, false >$ to $r$.
```

<u>Algorithm for *FindNextAlarm*</u>

```
1. $nextAlarmTime = \infty$, $nextAlarm = NULL$.
2. for all ringing alarms $a$ do
    (a) Let $t$ be the time when $a$ started to ring.
    (b) for all alarms $\hat{a}$ that are descendants of $a$
        i. Set $time = t + A^{max}_{Node(d(a)),Node(d(\hat{a}))}$.
        ii. if $time > CURRENT\_TIME$, then if $time < nextAlarmTime$,
            set $nextAlarmTime = time$ and $nextAlarm = \hat{a}$.
3. if $nextAlarmTime \neq \infty$, return $ev =< \hat{a}, nextAlarmTime, Ringing >$ else
   return $NULL$.
```

not have yet expired. The procedure $IsOnlyPath()$ then checks whether the path to go from f to a ringing descendant of f *must* go through node $\hat{a}$. If so, then $\hat{a}$ should have rung and is a missing alarm of h^f.

It may seem unusual that the only path to a ringing descendant is through $\hat{a}$ and yet the time for $\hat{a}$ has not expired. Such a seemingly inconsistent situation arises because the time for failures to propagate is modeled as an interval instead of a single value.

Algorithm for *LocateFSC*

```
1. Set $r = \phi$.
2. sort the hypotheses $h^f \in \hat{r}$ in decreasing order of rank.
3. Set $alarmsToExplain$ = number of events $ev \in EV$ such that $type(ev) = Ringing$.
4. while $(alarmsToExplain > 0)$ and $(rank(h^f) > 1)$ (where $h^f$ is the next unexamined hypothesis in the sorted list).
    (a) Add $h^f$ to $r$.
    (b) for all $\hat{a}$ such that $\hat{a} \in P(h^f)$ or $\hat{a} \in S(h^f)$
        i. mark $ev \in EV$ as explained, where $ev = < \hat{a}, t, Ringing >$.
        ii. $alarmsToExplain = alarmsToExplain - 1$.
    (c) for all $\hat{a}$ such that $\hat{a} \in MP(h^f)$ or $\hat{a} \in MS(h^f)$
        i. for all $s$ such that $< s, \hat{a} > \in SA$ add $h^s = < s, \hat{a}, missing >$ to $r$.
    (d) mark $f$ as a fault source.
5. if $alarmsToExplain > 0$, for all ringing alarms $\hat{a}$ that are not explained,
```

Let $Node(f) = i$ and $Node(d(a)) = j$. Then the algorithm for *Recompute-Time()* is :

1. Let $erl = t - A_{i,j}^{max}$, $lat = t - A_{i,j}^{min}$ and $consistent = FALSE$.
2. for all $\hat{h}^f \in \hat{r}$ such that $\hat{h}^f$ and h^f stand for the same f, if a is consistent with $\hat{h}^f$ do :
   (a) Set $consistent = TRUE$.
   (b) if $erl > t_{erl}(\hat{h}^f)$ set $t_{erl}(\hat{h}^f) = erl$.
   (c) if $lat < t_{lat}(\hat{h}^f)$ set $t_{lat}(\hat{h}^f) = lat$.
   (d) Add a to $P(\hat{h}^f)$.

   else Add a to $SP(\hat{h}^f)$.
3. if $consistent = FALSE$, add a new hypothesis h^f_{new} to $\hat{r}$ with $P(h^f_{new}) = \{a\}$, setting $t_{erl}(h^f_{new}) = t - A^{max}_{Node(f),Node(d(a))}$ and $t_{lat}(h^f_{new}) = t - A^{min}_{Node(f),Node(d(a))}$.

<u>Algorithm for *FindMissingAlarms*</u>

1. for all $h^f \in \hat{r}$ do:
   (a) for all $\hat{a}$ such that h^f is ancestor of $\hat{a}$ and $\hat{a}$ is not ringing
      i. if $ShouldHaveRung(\hat{a})$ or $IsOnlyPath(h^f, \hat{a})$
         then $\hat{a}$ is a missing alarm of h^f.
      ii. if $\hat{a}$ is a missing alarm of h^f then if it is a primary alarm of h^f, add it to $MP(h^f)$ else add it to $MS(h^f)$.

The procedure $ShouldHaveRung()$ returns a true value if the alarm $\hat{a}$ should be ringing, given $t_{erl}(h^f)$, $t_{lat}(h^f)$ and the present time. However, the time for $\hat{a}$ may

TABLE 4 Primary Alarm Sets

Hypothesis	Failure Mode	P	SP
h_1^f	f_1	$\{a_1, a_2\}$	$\{a_3\}$
h_2^f	f_1	$\{a_3\}$	$\{a_1, a_2\}$

(c) if a is not a descendant of an alarm $\hat{a} \in P(h^f)$ then if a is a descendant of $\hat{a} \in SP(h^f)$ then put a in $SS(h^f)$.

(d) if a is a primary alarm (of some failure mode(s)) but not a primary alarm or a secondary alarm of f, put a in $SP(h^f)$.

Algorithm for *RecomputeTime()*

Whenever a hypothesis h^f gets a new primary alarm a, its $t_{erl}(h^f)$ and $t_{lat}(h^f)$ needs to be recomputed. Note that one failure mode may have more than one hypotheses in $\hat{r}$. This will happen if two or more primary alarms of the failure mode conflict temporally with each other. In such cases, the primary alarms are divided into subgroups such that the alarms in one group are temporally consistent with each other. Then, there will be as many number of entries in $\hat{r}$ for this failure mode as there are groups. These entries will have the consistent primary alarms in set P and all other primary alarms in set SP.

For example, let there be a failure mode f_1, with primary alarms a_1, a_2, and a_3. Out of these primary alarms, let a_1 and a_2 be temporally consistent with each other, but not with a_3. In this case, there will be two entries in $\hat{r}$, h_1^f and h_2^f, with primary alarm sets shown in Table 4.

APPENDIX A

DIAGNOSTIC ALGORITHMS

Algorithm for *UpdateHypotheses*

```
1. Find all the $f$ such that $a$ is a primary alarm of $f$ and $h^f \notin \hat{r}$ and add $h^f$ to $\hat{r}$,
   setting $t_{erl}(h^f) = t - A^{max}_{Node(f),Node(d(a))}$ and $t_{lat}(h^f) = t - A^{min}_{Node(f),Node(d(a))}$.
2. for all $h^f \in \hat{r}$ do :
   (a) if $a$ is a primary alarm of $h^f$ then $RecomputeTime(h^f)$. The algorithm for
       $RecomputeTime$ is explained later.
3. for all $h^f \in \hat{r}$ do :
   (a) if $a$ is a primary alarm of $h^f$ then
       i. for all $\hat{a} \in P(h^f)$ do : if $\hat{a}$ is a descendant of $a$ then
          A. Remove $\hat{a}$ from $P(h^f)$.
          B. if $\hat{a}$ is consistent with $a$, put $\hat{a}$ in $S(h^f)$,
          C. else put it in $SS(h^f)$.
       ii. for all $\hat{a} \in SP(h^f)$ do : if $\hat{a}$ is a descendant of $a$ then
          A. Remove $\hat{a}$ from $SP(h^f)$.
          B. if $\hat{a}$ is consistent with $a$, put $\hat{a}$ in $S(h^f)$,
          C. else put it in $SS(h^f)$.
       iii. for all $\hat{a} \in MP(h^f)$ do : if $\hat{a}$ is a descendant of $a$, remove $\hat{a}$ from
            $MP(h^f)$ and put it in $MS(h^f)$.
   (b) if $a$ is a descendant of an alarm $\hat{a} \in P(h^f)$ then if $a$ is consistent with $\hat{a}$,
       put $a$ in $S(h^f)$ else put it in $SS(h^f)$.
```

be different. $FM1$ will cause $DY1$, then $DY2$, then $DY3$ and then $DY4$, while $FM2$ will cause $DY3$, then $DY4$, then $DY1$ and then $DY2$. Since the diagnoser looks at the sequence also, the failure modes are actually distinguishable. Thus, the distinguishability analysis algorithm should be modified accordingly.

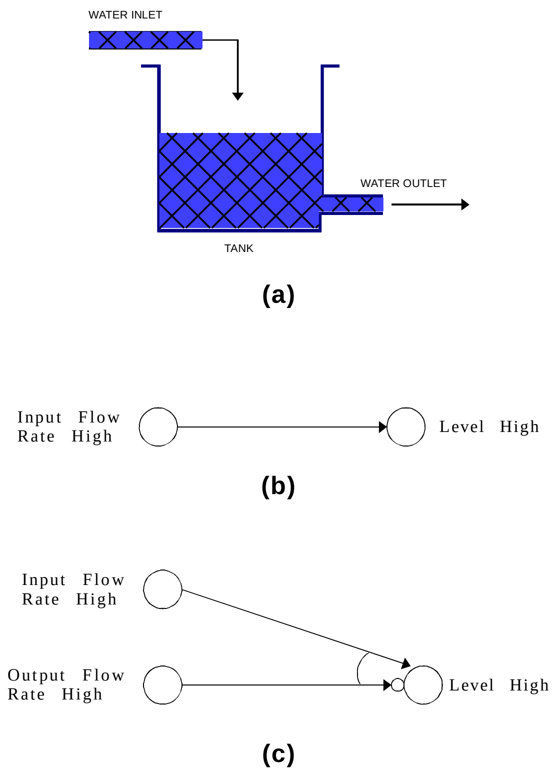


Figure 27. A Simple System

Failure Sequence and Distinguishability

The distinguishability analysis considers the set of discrepancies that are reached after a certain time has passed. If the FPG has loops, the analysis may find that faults are not distinguishable. As an example, consider the FPG shown in Figure 28. In this case, both $FM1$ and $FM2$ will cause all four discrepancies after enough time has passed, and thus, the analysis will say that they are not distinguishable. This is because it does not consider the fact that the *sequence* of the discrepancies will

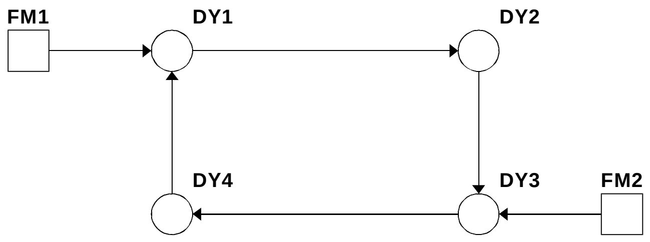


Figure 28. FPG with a Loop

Next, the diagnoser has to get more evidence to confirm or contradict these hypotheses. At this point a decision has to be made as to which sensor value to read. Obviously, the choice will be guided by the cost of the test as well as how useful the sensor is for confirming/contradicting the current hypotheses. At this point distinguishability analysis and test sequencing can be used to determine the sensor and make a request to the monitoring sub-system. This is a dynamic process in which the requests have to be determined by current set of hypotheses which can change as the fault scenario evolves.

Boolean Relationships Between Propagations

The FPM described in this thesis allows only AND and OR nodes in the failure propagation graphs. However, in many situations, it does not suffice.

As an example, consider the simple system shown in Figure 27a. The input to the tank is a stream of water which is let out of the tank through another pipe. If a fault occurs which causes the input flow rate to increase, the tank level will increase if the output flow rate remains the same. This situation can be modeled with the FPG shown in Figure 27b. However, if the output flow rate goes high (because of a leak, perhaps), the level will not rise. Thus, the fact that the level will rise only if the output flow rate is not high should also be modeled.

The FPM should be modified which allows one to express the conditions on which a failure propagation depends. The failures incident on any given failure can have an arbitrary conjunctive/disjunctive relationship between them. Using this method, the above example can be remodeled as shown in Figure 27c. What the failure propagation now expresses is the fact the tank level will rise if the input flow rate increases *and* the output flow rate is *not* high.

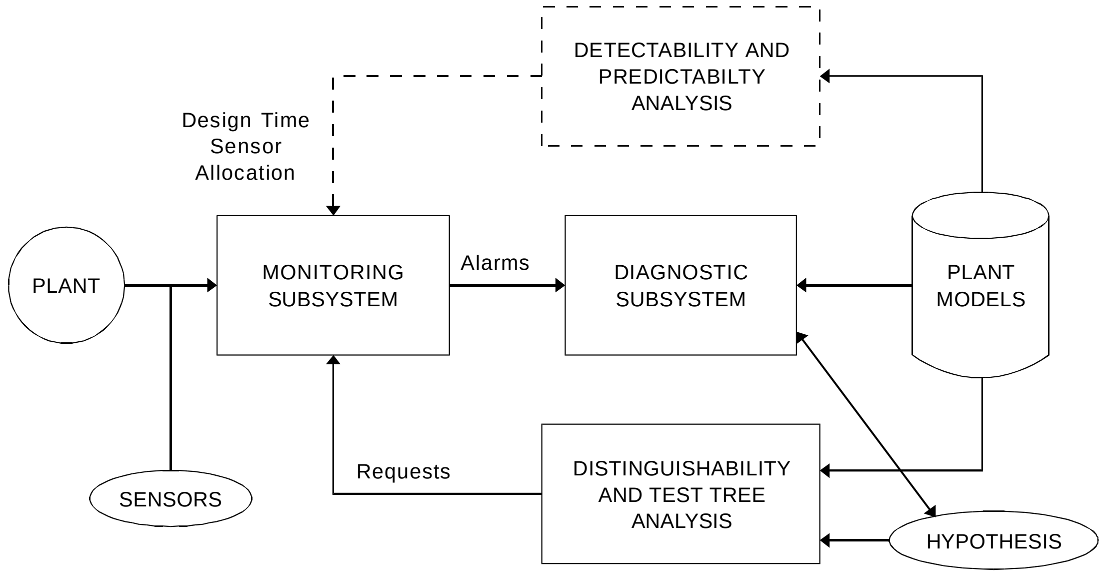

DETECTABILITY AND PREDICTABILTY ANALYSIS
Design Time Sensor Allocation
PLANT
MONITORING SUBSYSTEM
Alarms
DIAGNOSTIC SUBSYSTEM
PLANT MODELS
Requests
DISTINGUISHABILITY AND TEST TREE ANALYSIS
SENSORS
HYPOTHESIS

Figure 26. Diagnosis with Distinguishability/Test Sequence Analysis

Test Sequencing and Diagnosis

Test sequencing analysis needs to be enhanced by considering repair actions, repair cost, probability of spurious tests, test setup cost and time.

Another interesting direction for research is that of combining diagnosis with the distinguishability and test sequence analysis. In most real systems, there is a subset of sensors that are on-line. This subset may not always remain the same – different sensor values may be needed at different times depending on the diagnostic requirement.

Diagnosing a system, where not all sensors provide data all the time, requires generation of requests for some sensor values. This problem combines on-line diagnosis with test sequencing. The idea is illustrated in Figure 26. When there is an indication of a fault (some fault(s) have been detected by the on-line sensors), the diagnoser hypothesizes about the faults. The set of initial on-line sensors can be determined by using the detectability analysis to ensure that every fault is detectable.

functionalities.

- DTOOL was developed to provide the design engineer with an easy to use tool to analyze the diagnosability. The notable features of DTOOL are :
  - A graphical user interface.
  - Study management.
  - Report generation.
  - Visual parameter selection and result presentation for evaluation, advice and for changing alarm and sensor state allocation.
  - Consistency checking of models.

Future Work

Probabilities

The statistical diagnosability analysis can be enhanced by addressing the following issues :

- Failure rates of components.
- Sensor reliability.
- Non-independent propagation of faults.

Sensor states are currently specified by dividing the continuous range of values read by a sensor into discrete ranges which have sharp boundaries. In real systems, because of sensor drifts, noise and inexact analysis, such sharp boundaries may not be practical. Thus, sensor states should be defined as a probability distribution over these ranges. This will add another aspect to sensor reliability. It will require changes in diagnosability analysis as well as the robust diagnostic algorithms.

- It is robust against a large number of sensor failures and degrades gracefully as the number of sensor failures increase.
- It is event-driven and uses incremental non-monotonic reasoning.
- It predicts future events and uses the predictor-corrector principle to revise its hypotheses.
- The algorithm is of polynomial complexity and hence, is scalable.

The feasibility of this approach has been demonstrated by modeling many different engineering systems and testing it using simulation. It has also been tested against system actually in operation and has worked well.

3. **Diagnosability** : Diagnosability analysis algorithms and a Diagnosability Analysis Tool were developed which allow a design engineer to assess the diagnosability characteristics of the system, to experiment with different sensor assignments and to ask for suggested sensor coverage in order to ensure certain level of diagnosability. The steps involved in developing this tool were :

    - Diagnosability characteristics were identified by analyzing the diagnostic process. Three main diagnostic concepts were identified – fault detection, isolation and prediction.
    - Metrics were defined to quantify the above three diagnosability characteristics. The metrics are – detectability, distinguishability and predictability. The three metrics also capture the dynamics of the system.
    - The tasks of diagnosability analysis were defined. They are – evaluation, advice and test tree generation.
    - Algorithms were developed to perform the above tasks. These algorithms are capable of operating on a single functionality or on a hierarchy of

CHAPTER V

CONCLUSION

In this thesis we have addressed some issues related to sensor-based diagnosis of dynamical systems. Specifically, the objectives that were achieved are :

1. **Fault Modeling** : Paradigms were developed to model the faults in a system using multiple aspect hierarchical modeling. The primary aspects are the physical components and the functionalities. A failure propagation model was developed to describe component faults, discrepancies, sensors and their interactions. The fault models capture the structure and dynamics of the system using parameterized failure propagation graphs. The role of sensors is modeled with alarm allocation, sensor allocation, sensor states and logic operators.

2. **Robust Diagnostics** : A diagnostic algorithm was developed that is robust against observation errors. The notable features of the algorithm are :

   - It operates on a hierarchy of functionalities and restricts the diagnostic search to the those parts of the hierarchy where the fault has occurred.
   - It identifies loss of model validity in case of large faults and restricts its search to those parts of the hierarchy where the model of the system seems to be valid.
   - It diagnoses single and multiple faults in components as well as sensors.
   - It uses the principle of structural redundancy to identify false and missing alarms and from there, the sensors responsible for it.

Diagnosability Analysis Tool

We have developed a Diagnosability Analysis Tool which provides a graphical the user interface to the analyses discussed in this chapter. This tool is described in Appendix C.

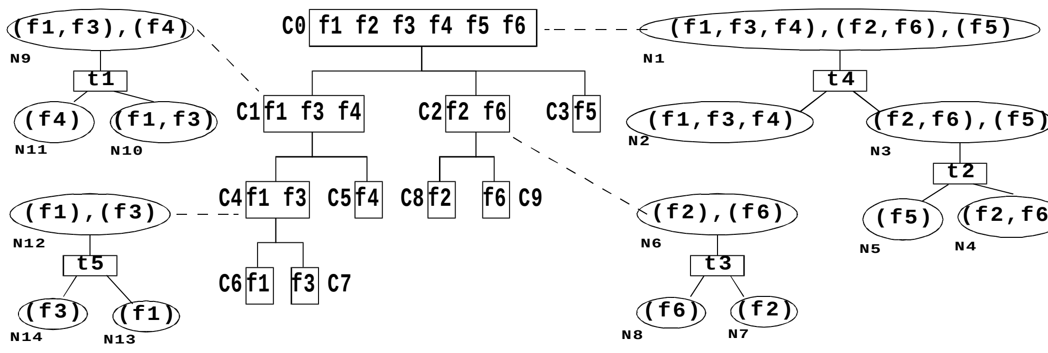
The individual clusters at the leaves of the local test trees might contain more than one failure. The failures contained in such clusters are isolated by replacing the cluster at the leaf of a local tree by the local tree rooted at the node corresponding to this cluster in the cluster hierarchy. For example, OR node N_2 is replaced by N_9, N_4 is replaced by N_6 and so on. This gives the final test tree for the above example which is shown in Figure 25b.

The local test tree construction is done by recursively dividing the the sub-clusters into two groups until we are left with individual clusters at the leaves. At each step in the process of generating the test tree, there might be more than on way to divide the clusters into two groups. For example, consider the cluster C_0. There are many possible ways of dividing the clusters into two groups, e.g., $\{(C_1,\ C_3),(C_2)\}$ or $\{(C_1,\ C_2),(C_3)\}$ etc.

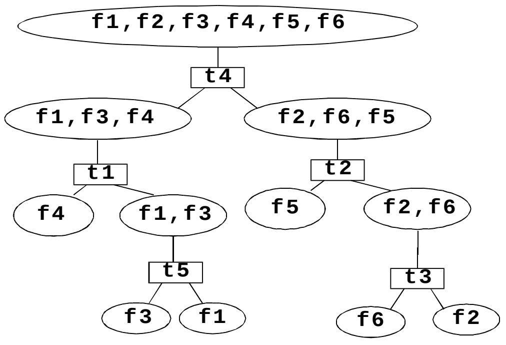
The selection of groups is done by the following method – (1) rank the available tests, (2) pick a the highest ranked test and (3) put the subclusters that are implicated by the test into one group and the rest into the other group. An available test is a test that can be reached by the failures in some of the clusters but not by failures in the other clusters and hence can be used to distinguish one set of clusters from another. The available tests are ranked according to the following criteria :

- If the cost of a test is high, it should have lower rank.

- If a test will split the clusters down the middle, i.e., it will divide the clusters into two equal sized groups, the test should have a high rank. The more lopsided the division that a test will result in, the lesser its rank should be. This heuristic is used to increase the likelihood of construction of a balanced test tree.



(a)



(b)

Figure 25. Generating Test Tree

Local Test Trees

The refinement of the tree is performed starting at the top node and then going down the cluster hierarchy. At each node in the hierarchy, *local test trees* to isolate the subclusters from each other are constructed. As an example, consider the cluster hierarchy in figure 24d which is also shown in Figure 25a. The clusters in the hierarchy are marked C_0, C_1, ..., with cluster C_0 being the top cluster. The failures that are in a cluster are shown inside the box for that cluster. For node C_0, a local test tree is constructed that distinguishes between C_1, C_2 and C_3. Similarly, local test trees are constructed for all cluster nodes (except leaf nodes). These local test trees are shown in Figure 25a with the dotted lines indicating the cluster node that they are associated with.

The above method is generalized by constructing a complete graph G_f whose nodes represent the failures in the system and the weighted edges between the nodes represent the "distance" between the failures. The distance, γ_{ij}, between two failures f_i and f_j is defined as

$$\gamma_{ij} = \sum_{k=1}^{n} R_{ik}.R_{jk} \tag{2}$$

Thus γ_{ij} is the number of tests reached by both f_i and f_j.

Next, edges from G_f removed in the increasing order of their weights. First all the edges with weight 0 are removed, next all the edges with weight 1 are removed and so on until no edges are left. At the beginning, G_f is a complete graph and all the failures are in one cluster which will be at the top of the cluster hierarchy. As edges ar removed, at some point G_f will be broken up into two or more disconnected components. The sets of failures corresponding to the nodes in these components will form the subclusters of the top cluster. Since removing more edges will break these components into further disconnected components, the subclusters will be broken down into further subclusters as we continue removing edges from G_f until we are left with single failures forming the clusters at the leaves of the cluster hierarchy.

The above discussion of clustering ignored the probabilities of failures which can be incorporated by modifying the distance measure such that it now becomes

$$\gamma_{ij} = [(1 - max\,[p(f_i),\ p(f_j)]) \times 10^{n_s}] \sum_{k=1}^{n} R_{ik}.R_{jk} \tag{3}$$

where n_s is the number of significant digits in the probability values of failures. The rest of the algorithm remains the same. Modifying the distance measure in this way has the effect of making the failures with higher probabilities move to near the top of the hierarchical cluster tree. This will result in a test sequence that will tend to test for higher probability failures earlier in the sequence, thus reducing the expected cost of testing.

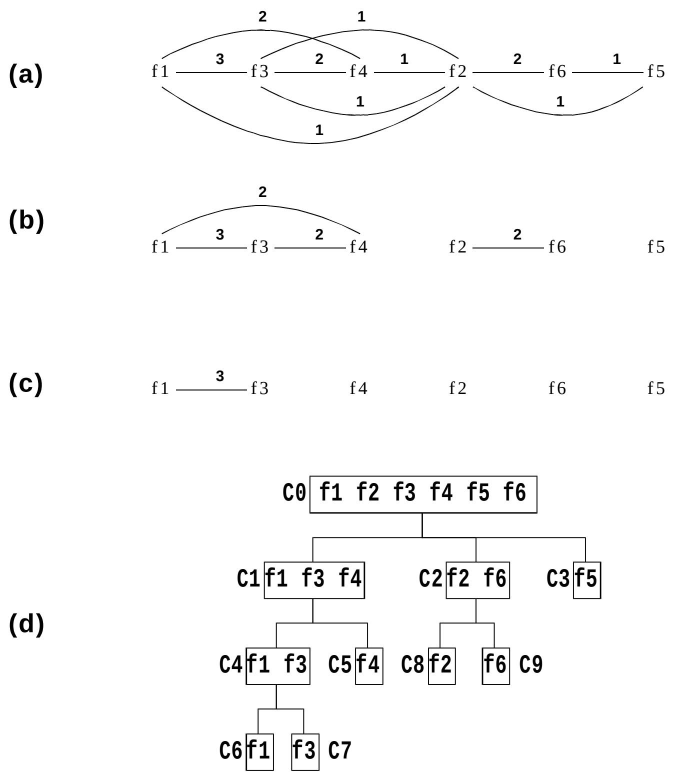


Figure 24. Hierarchical Clustering

The clustering method works by dividing the group of failures into sub-groups that have "weak interactions". The strength of interactions between failures is given by the number of tests they share. For example, Figure 24a shows the failures of the system shown in Figure 23. The lines between failures show the "interaction" (whether they share tests or not) between them. The numbers on the lines show the "strength" (number of shared tests) of their interactions. If the lines with strength 1 are removed, the failures will be divided into the three groups shown in Figure 24b. Next, removing lines with strength 2 will further sub-divide the failure into the groups shown in Figure 24c. This process can be continued until all the groups have a single failure. This breakdown of failures into groups and sub-groups gives the hierarchical clustering which is shown in Figure 24d.

2. $t1$ will fail and clear $f4$.

3. $t5$ will pass and clear $f1$. Therefore the fault must be $f3$.

If the assumption that the diagnosis is triggered by the detection of a fault is dropped, the the test tree can be modified to detect a fault as shown in Figure 23c. In this case, the test sequence first determines whether the system is fault free (represented by $f0$ in the figure) or whether a fault exists. If a fault exists, it proceeds with the diagnosis.

The problem of constructing the optimal test tree is known to be NP-complete. The test tree construction algorithm presented here addresses this problem by (1) using hierarchical clustering to derive an "approximate" test tree and (2) refining the approximate test tree.

Hierarchical Clustering

At each AND node in the test tree, the group of failures is divided into two sub-groups by the test. There will be different costs associated with different combinations of sub-groups. Which means that the expected cost of testing will depend upon what the division of failures into groups and sub-groups is. Minimizing the cost by trying every possible hierarchical decomposition is computationally prohibitive; instead some heuristic method should be used for generating the hierarchy.

The algorithm described here uses hierarchical clustering to derive an approximate test tree. The motivation for using hierarchical clustering comes from the fact that it captures the topology of the FPG in the division of failures into clusters and sub-clusters. It divides failures into clusters that share the least number of tests. Thus, at any level, the failures are grouped in such a way that they can be distinguished from each other more "easily" than if there was a different grouping.

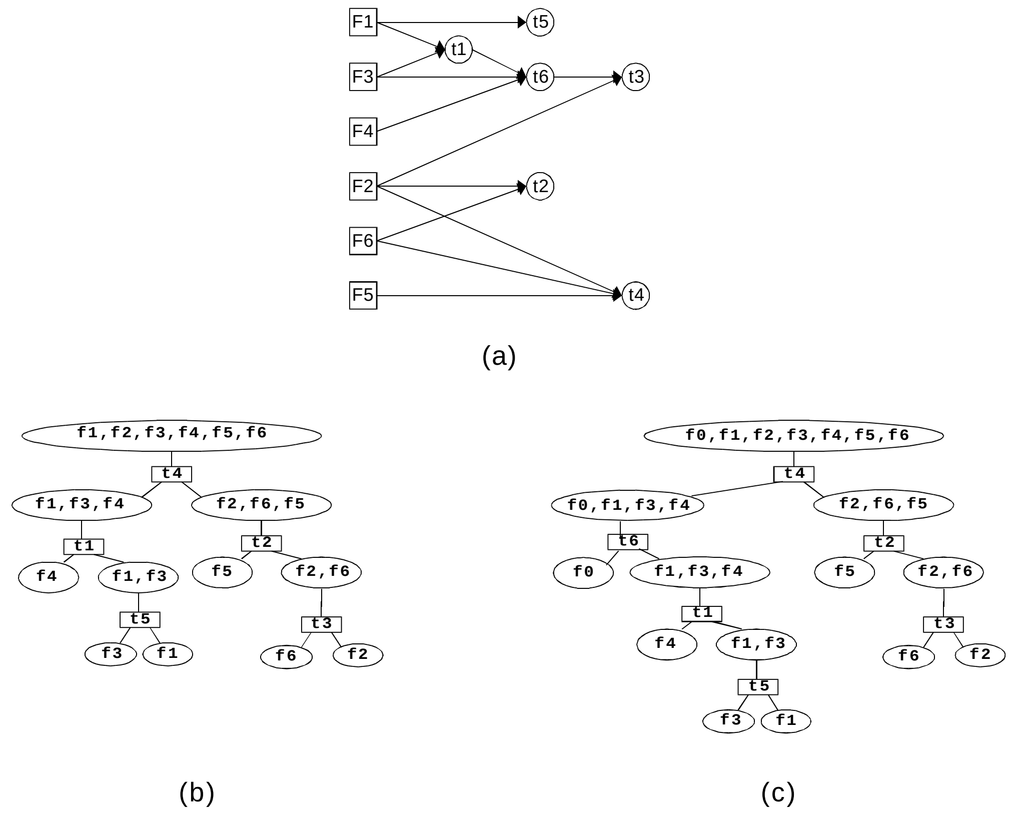


Figure 23. Test Sequencing

- $\underline{c} = [c_1,\ c_2,\ \ldots c_{m+o}]$ is the vector of test costs.

- R, a $n \times m + o$ binary matrix, is the reachability matrix such that $R_{i,j} = 1$ if and only if test t_j detects failure f_i, else it is 0. R can be obtained from DYP quite easily. $R_{i,j} = 1$ iff $A^*_{i,j+n} = 1$.

The problem then is to devise a test algorithm (the binary AND/OR search tree) that is able to isolate the failures in F using the tests in T such that the expected cost of testing is minimized, where the expected cost J is given by :

$$J = \underline{p}^T A \underline{c} = \sum_{i=1}^{m+o} \sum_{j=1}^{n} a_{ij}\ p(f_i)\ c_j \qquad (1)$$

where $A = (a_{ij})$ is an $n \times m + o$ binary matrix such that $a_{ij} = 1$ if test t_j is used in the path leading to the isolation of failure f_i, else it is 0.

Consider the FPG shown in Figure 23a. Figure 23b shows a test tree for the system. The convention is to branch left if the test passes (the corresponding discrepancy is not observed) and to branch right if the test fails. The test tree shown assumes that the diagnosis is triggered when some fault has been detected.

If $f2$ is the fault that has occurred, the diagnosis will proceed in the following manner :

1. Check test $t4$. Since $f2$ has occurred, $t4$ will fail, and $f1, f3, f4$ will be cleared.

2. Check test $t2$. $t2$ also will fail and $f5$ will be cleared.

3. Next, $t3$ will fail and $f2$ will be returned as the faulty component.

If fault $f3$ occurs, these will be the steps :

1. $t4$ will pass, clearing $f2, f5, f6$. However, since it is known that *some* fault is present, it must be one of $f1, f3, f4$.

- does not specify a *sequence* of testing which minimizes the average expected cost of testing.

We have developed a method for generating a test tree from FPM which is described in the next section. The description will use terms commonly used in the literature on test sequencing problem. The correspondence between those terms and FPM is :

- The failure modes are referred to as "failures" in the literature. Associated with each failure is a probability of occurrence.

- A "test" is equivalent to a request for the status of a discrepancy. Associated with each test is the cost of performing the test. The failures in the system are "detected" by these tests. In terms of FPM, this means that the failures can propagate to discrepancies which can be monitored. The detection of a failure is done by generating a request for the statii of alarms associated with some or all of these discrepancies.

Test Tree Construction

The test sequencing problem involves constructing a binary AND/OR decision tree in which the OR nodes represent the ambiguity sets and the AND nodes represent the test to perform [22]. The test sequencing problem can be defined as a five-tuple $< F, \underline{p}, T, \underline{c}, R >$ where

- $F = \{f_1,\ f_2,\ \ldots f_n\}$ is the set of n failures in the system.

- $\underline{p} = [p(f_1),\ p(f_2),\ \ldots p(f_n)]^T$ is the *a priori* probability vector of the failures.

- $T = [t_1,\ t_2,\ \ldots t_{m+o}]^T$ is the set of $m + o$ available tests.

- There is no cost involved in generating the alarms.

- The failure rates of components are same, i.e., all the failure modes can occur with equal probability.

However, this is not the case in many engineering systems. The sensors in the system do not necessarily provide signal values on-line. What this means is that the alarms are not always on-line. Thus, a monitored discrepancy can go unobserved unless the observation is specifically requested. Further, usually there are some costs involved in reading the sensor value, in which case the cost should be taken into account before generating the request for the status of an alarm.

The cost consideration and the fact that the probabilities of failure modes occurring are different, give rise to the problem of *test sequencing* [23]. The goal in test sequencing is to generate a sequence of requests for status of alarms which can be used to distinguish the failure modes, while keeping the average expected cost to a minimum.

There are some similarities and differences between the analysis for distinguishability advice and test sequencing. The result of both the analyses is a set of alarms that can be used to distinguish the failure modes, and these alarms should be incorporated in the system at design time to ensure distinguishability. The crucial difference is that distinguishability advice is generated for alarms that are on-line while test sequencing specifies the sequence of alarm requests to be generated without assuming that any of the alarms are on-line.

The set of alarm requests generated by the test sequence can always be considered as distinguishability advice. The reverse, however, is not true, since distinguishability advice :

- does not take into account alarm costs and failure probabilities.

- Predictability : A discrepancy $d, Node(d) = j$ is predictable time t in advance with probability p if $\exists$ a failure mode $f, Node(f) = i$ such that $A^{*}_{i,j} = 1$ and f is detectable in time $A^{min}_{i,j} - t$ with probability p. Thus, this algorithm requires finding the statistical detectability of all the failure modes that reach d.

Advice

The algorithms for generating advice remain essentially the same as in case of non-statistical analyses, with some minor modifications which are listed below :

- Detectability : In the prime implicant table method, modify the step 2.b.i for constructing the implicant table T, to – Set $T_{j,i} = 1$ if $A^{p,t}_{l,j} = 1$.

- Distinguishability : The non-shared discrepancy set is redefined as
  $d \in NSD^{\hat{f}_i,\hat{f}_j,t,p}$ iff

$$A^{p,t}_{ni,k-n} = 1 \text{ and } A^{p,t}_{nj,k-n} = 0$$

or

$$A^{p,t}_{ni,k-n} = 0 \text{ and } A^{p,t}_{nj,k-n} = 1.$$

  The distance function and $NSD^{C_i,C_j,t}_{c}$ remain the same.

- Predictability : The construction of the implicant table is now defined by $T_{j,i} = 1$ iff $A^{p,t}_{j,l} = 1$

Distinguishability and Test Sequencing

The distinguishability advice analysis described above made a few assumptions, which are :

- The alarms in the system are all on-line (BITs).

The following analysis assumes that all the failure propagations are independent of each other. The statistical detectability and distinguishability analyses use matrix $A^{p,t}$, a $n \times (m+o)$ matrix. $A^{p,t}_{i,j-n} = 1$ iff f can propagate to d in time t with probability p, else it is 0, where $i = Node(f)$ and $j = Node(d)$. The values in $A^{p,t}$ are computed by running a given number of simulations, generating a random number for each of the propagation times and finding the average number of times that the failure modes propagate to discrepancies in time t.

The algorithm for computing $A^{p,t}$ is given in Appendix B. The statistical predictability advice analysis also uses a $A^{p,t}$ which is computed in the same manner except that the comparison in step **2c.** of the algorithm is modified to $T_{i,n+j} \geq t$.

Statistical Analyses Algorithms

Evaluation

The evaluation algorithms are modified for statistical analysis in the following manner :

- Detectability : A failure mode $f, Node(f) = i$ is detectable in time t with probability p if there exists a monitored discrepancy $d, Node(d) = j$ such that $A^{p,t}_{i,j-n} = 1$. If there are more monitored discrepancies which f reaches in time t with probability p, the probability of detection of f increases. However, this analysis only checks if *any* such discrepancy exists since that is enough to find whether f is detectable in time t with probability p.

- Distinguishability : The distinguishability analysis is done the same way as in the case of non-statistical analysis. Only the definition of Ω changes. $\Omega_{i,j} = 1$ iff d_j is monitored and $A^{p,t}_{i,j} = 1$.

Using the above probabilities, additional analyses can be performed :

1. **Detectability** :

   (a) Evaluation – For all failure modes $f \in F$, determine if f is detectable in time t with probability p.

   (b) Advice – Given a partial alarm allocation, a specified set of failure modes $F_1 \subseteq F$, a time interval of length $\hat{t}$ and a probability value p, determine the optimal allocation for new alarms such that every $f \in F_1$ is detectable in time $\hat{t}$ with probability p.

2. **Distinguishability** :

   (a) Evaluation – For a given time t and probability p, for all failure modes $f \in F$, determine if f is distinguishable in time t with probability p.

   (b) Advice – Given a partial alarm allocation and a set of failure modes $F_1 \subseteq F$, a time interval of length $\hat{t}$ and a probability value p, determine the optimal allocation for new alarms such that all $f \in F_1$ are distinguishable in time $\hat{t}$ with probability p.

3. **Predictability** :

   (a) Evaluation – For all discrepancies $d \in D'$, determine if d can be predicted time t in advance with probability p.

   (b) Advice – Given a partial alarm allocation, a specified set of discrepancies $D_1 \subseteq D$, a time interval of length $\hat{t}$ and a probability value p, determine the optimal allocation for new alarms such that every $d \in D_1$ is predictable time $\hat{t}$ in advance with probability p.

Statistical Analysis

The analyses described above can be enhanced by adding a statistical component to them. The motivation comes from the fact that the above analyses are performed by considering only the worst case times for failure propagation, and ignoring the fact that there usually is a finite interval of time for propagation of failures.

For example, consider a failure mode $\hat{f}$ and a discrepancy $\hat{d}$, such that $A_{i,j}^{min} = t_1$ and $A_{i,j}^{max} = t_2$, where $i = Node(\hat{f}),\ j = Node(\hat{d}),\ t_1 < t_2$. The detectability analysis described above will find that $\hat{f}$ is *not* detectable in time t, $t_1 < t < t_2$. This is because the detectability evaluation algorithm looks at the *maximum* time that $\hat{f}$ can take to propagate to $\hat{d}$, and the maximum time, t_2, is greater than t. However, the algorithm will find $\hat{f}$ always detectable in time t_2.

In the actual system, the propagations will take place somewhere within the propagation interval. As a consequence, $\hat{f}$ may occasionally propagate to $\hat{d}$ in time t. That is, $\hat{f}$ may be detectable in time t with a probability less than 1. The statistical analysis considers these probabilities when computing detectability etc.

For the statistical analysis, the propagation intervals on the edges in FPM are treated as probability distributions, representing the probability of propagation of failure over that time interval. This probability distribution can be uniform, gaussian or any other distribution. Then the probability of a propagation occurring at or before a certain time within that interval is computed. From this, the probabilities that a particular failure mode will propagate to the discrepancies within a given time period are computed.

2. **Distinguishability** : For a given time t, for all failure modes $f \in F$, determine the distinguishability of f, i.e., find if $DIS(f,t)$ is true or false.

3. **Predictability** : For all discrepancies $d \in D'$, find $PRED(d)$, i.e., compute t_{min} and t_{max} for d.

The algorithms for the above evaluation tasks are given in Appendix B.

Advice

The advisory analysis is used to generate suggestions for alarm allocation by specifying the diagnosability criteria that is required to be met. The diagnosability criteria consists of sets of failure modes and discrepancies and their desired detectability, distinguishability and predictability. Thus, advice generation task is an inverse of evaluation task. While in evaluation, the metrics for diagnosability are computed given an alarm allocation, in advice the desired metrics are specified and the algorithms try to find an optimal alarm allocation to ensure the metrics.

1. **Detectability** : Given a partial alarm allocation, a specified set of failure modes $F_1 \subseteq F$, and a time interval of length $\hat{t}$, determine the optimal allocation for new alarms such that $t_{lat}(f) \leq \hat{t} \ \ \forall f \in F_1$.

2. **Distinguishability** : Given a partial alarm allocation and a set of failure modes $F_1 \subseteq F$, and a time interval of length $\hat{t}$, determine the optimal allocation for new alarms such that all $f \in F_1$ are distinguishable in time $\hat{t}$.

3. **Predictability** : Given a partial alarm allocation, a specified set of discrepancies $D_1 \subseteq D$, and a time interval of length $\hat{t}$, determine the optimal allocation for new alarms such that $t_{min} \geq \hat{t} \ \forall d \in D_1$.

The algorithms are described in Appendix B.

Diagnosability Analyses

In the following discussion, a partial alarm allocation denotes a subset of alarms that are already part of the system design because of, for example, control requirements. The diagnosability analysis may suggest allocation of additional alarms or the removal of some of these alarms. A partial alarm allocation includes the scenario when there are alarms on all the discrepancies, or no alarms on any discrepancy.

There are three types of diagnosability analyses that can be performed :

1. Evaluation : This analysis involves computing the detectability and distinguishability of failure modes and predictability of discrepancies in the system, given a partial alarm allocation.

2. Advice : This analysis generates suggestions for alarm allocation for attaining a desired detectability and distinguishability of the failure modes and predictability of the discrepancies.

3. Test tree generation : This analysis generates a test tree for isolating faults in the system.

The analyses and algorithms used are described in the following sections.

Evaluation

Evaluation of a system's diagnosability means the computation of the values for the metrics defined above for each failure mode and discrepancy in the system, given a particular alarm allocation. The evaluation tasks are the following :

1. **Detectability** : For all failure modes $f \in F$, determine $DET(f)$, i.e., compute t_{erl} and t_{lat} for f.

the node number of $v(f)$ in G', which will be between 1 and n. Similarly, $Node(d)$ represents the node number of $v(d)$ in G', which will be between $n+1$ and $n+m+o$.

Data structures derived from G'

The following is the list of additional definitions derived from G' for the purpose of diagnosability analysis :

1. A is the $(n+m+o) \times (n+m+o)$ adjacency matrix for G' where $A_{i,j} = 1$ iff $< v_i, v_j > \in E$ else $A_{i,j} = 0$.

2. A^* is the $(n+m+o) \times (n+m+o)$ matrix representing the transitive closure of G'.

3. A^{min} is a $(n+m+o) \times (n+m+o)$ matrix whose elements represent the minimum time needed for one failure to propagate to another. $A^{min}_{i,j} = \hat{t}$ means that the failure at vertex v_i takes *at least* time $\hat{t}$ to propagate to the failure at vertex v_j. If $A^*_{i,j} = 0$ then $A^{min}_{i,j} = \infty$. A^{min} is obtained by finding the all pairs shortest paths between the vertices in G', using t_{min} as the cost on the edges.

4. A^{max} is a $(n+m+o) \times (n+m+o)$ matrix whose elements represent the maximum time needed for one failure to propagate to another. $A^{max}_{i,j} = \hat{t}$ means that the failure at vertex v_i takes *at most* time $\hat{t}$ to propagate to the failure at vertex v_j. If $A^*_{i,j} = 0$ then $A^{max}_{i,j} = \infty$. A^{max} is obtained by finding the all pairs shortest paths between the vertices in G', using t_{max} as the cost on the edges.

discrepancies in the FPM, where $D = \{d_1 \dots d_m\}$ represents the actual discrepancies and $D^{ss} = \{d_{m+1} \dots d_{m+o}\}$ represents the sensor states in FPM that were mapped to discrepancies. $\underline{\mathrm{m}}$ is a vector of length $m + o$, representing the monitored status of discrepancies. $\underline{\mathrm{m}}_i = 1$ if

- $1 \leq i \leq m$ and $\exists a \in A$ such that $< a, d_i > \in M$, OR
- $m < i \leq m + o$ and $\underline{\mathrm{e}}_{i-m} = 1$.

otherwise $\underline{\mathrm{m}}_i = 0$. In the following discussion, then, an alarm may mean an actual alarm for monitoring a discrepancy or the BIT and enabled status of a sensor state. Similarly, "alarm allocation" will be used to denote the association of alarms to actual discrepancies as well as the subset of sensor states that are BIT and enabled.

The diagnosability problem is defined by :

$$DYP =< F, D', \underline{\mathrm{m}}, G' >$$

where F, D' and $\underline{\mathrm{m}}$ are as defined previously. $G' = (V, E)$ is a directed graph derived from the modified FPG described in the previous section. The vertex set V has $n + m + o$ vertices, representing n failure modes, m discrepancies and o sensor states. Without loss of generality, we will assume that the vertices representing failure modes are numbered 1 to n, the vertices representing discrepancies are numbered $n + 1$ to $n+m+o$. An edge $e_{i,j} =< v_i, v_j > \in E$ iff the failure represented by v_i propagates and causes the failure represented by v_j. Each edge in E is weighted by two parameters :

1. t_{min}, which is the minimum time for propagation of failure along the edge, and
2. t_{max}, which is the maximum time for propagation of failure along the edge.

Notation : A vertex in G' which represents a failure mode f will be denoted by $v(f)$ and a vertex representing discrepancy d will denoted by $v(d)$. $Node(f)$ represents

c_1, c_2 and c_3 can be computed. The worst case minimum time, $\hat{t}_{min}$, for propagation to *any* sensor state is given by the largest of the minimum time for propagation to the the three groups. Similarly, the worst case maximum time $\hat{t}_{max}$ is given by the largest maximum time for propagation to the the three groups.

Effectively, the t_{min} and t_{max} on all the edges from d to $ss_1, \ldots ss_8$ can be replaced by $\hat{t}_{min}$ and $\hat{t}_{max}$ respectively, eliminating the LOP. The modified FPG is shown in Figure 22b.

Note that the above modification needs to be done only for discrepancies with LOPs whose expression contains a disjunction. There is no need to do the above for discrepancies with LOPs which have only one product term and no disjunction since there is no uncertainty about propagation of the failure to sensor states. The failure *will* propagate to all the sensor states in this case. The same holds for discrepancies that do not have explicit LOPs associated with them.

Diagnosability Problem

Let $SS = \{ss_1, ss_2, \ldots ss_o\}$ be the set of o sensor states in the system. Let $\underline{e}$ be a vector of length o representing the enabled/disabled status of sensor states. If sensor state ss_i is BIT and enabled, $\underline{e}_i = 1$ else $\underline{e}_i = 0$. $\underline{e}_i = 0$ could mean that ss is a manual sensor state or a disabled BIT.

As noted in Chapter II, sensor states are equivalent to primitive discrepancies. The diagnosability analysis algorithms treat sensor states as discrepancies. Thus, an enabled BIT sensor state is the same as a discrepancy monitored by an alarm. A manual or a disabled BIT is the same as a discrepancy that does not have an alarm monitoring it.

Using the above correspondence between discrepancies and sensor states, we extend the set of discrepancies. $D' = D \cup D^{ss} = \{d_1, d_2, \ldots d_{m+o}\}$ is a set of $m + o$

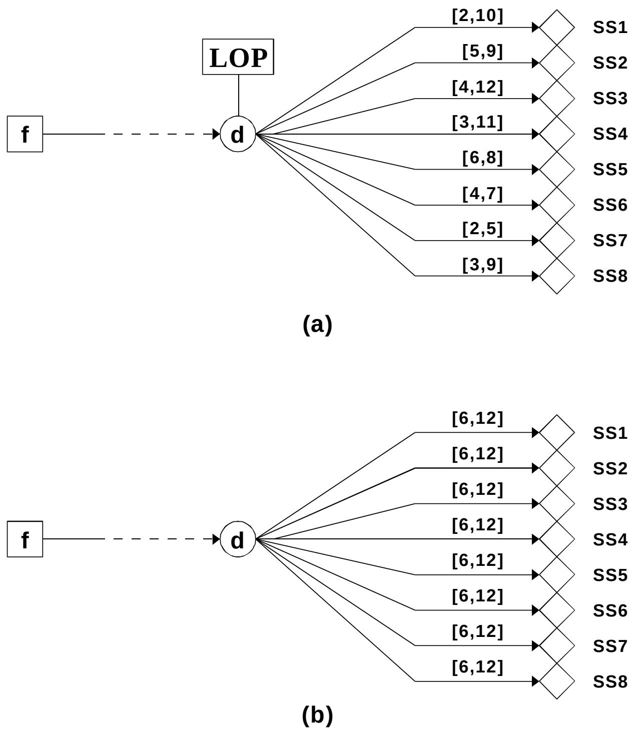


Figure 22. LOP Handling Example

known, they can not be used by the analyses since in the real system, ss_1 may never become active. However, since the analysis algorithms work with only the *worst case* minimum and maximum times for propagation, the problem can be solved by modifying the FPG in the following manner.

Let the product terms in the example above be represented by c_1, c_2 and c_3, where $c_1 = ss_1.ss_2.ss_3$, $c_2 = ss_4.ss_5$ and $c_3 = ss_6.ss_7.ss_8$. We will denote the minimum and maximum times for propagation form d to the sensor states by $< t_{min,1}, t_{max,1} >$ $\ldots < t_{min,8}, t_{max,8} >$. Consider the case when d causes the sensor states in the group c_1, i.e., ss_1, ss_2 and ss_3. Then, the *earliest* time when it can be concluded that d has occurred is when d has caused *all* the sensor states in c_1. This time is given by the largest of the t_{min} for the sensor states. Reasoning along the same lines, the latest time is given by the largest of the t_{max} for the sensor states.

In this manner, the "minimum" and "maximum" time for propagation to groups

- What are the values of $< t_{erl}, t_{lat} >$ for an $f \in F$ (How long will it take to detect a fault in a specific component?)
- Is $t_{min} > 0$ for $d \in D$? (Can a particular catastrophic failure be predicted?)
- Is $DIS(f, \infty) = TRUE \; \forall \; f \in F$? (Does adequate sensor coverage exist to isolate all the faults ?)
- What sensor assignment is needed such that $t_{lat} \neq \infty \; \forall \; f \in F$? (What sensor assignment is needed to detect every fault ?)

Handling LOPs

The diagnosability analysis algorithms described below compute the worst case minimum and maximum time that a given failure takes to propagate to a discrepancy or a sensor state. This is done by enumerating all the paths in FPG from the vertex representing the failure to the vertices representing discrepancies (or sensor states) and looking at the t_{min} and t_{max} along the edges in the path (using the all-pairs shortest path algorithm).

The propagations to sensor states, however, need to be handled in a different manner due to LOPs associated with the discrepancies. Consider the example shown in Figure 22a where a failure mode f propagates to a discrepancy d which can cause sensor states $ss_1, \ldots, ss_8$. The time intervals for d to cause the sensor states are also shown in the figure. Let the expression for the LOP associated with d be

$$ss_1.ss_2.ss_3 + ss_4.ss_5 + ss_6.ss_7.ss_8$$

The disjunction in the expression means that the failure *may or may not* cause some of the sensor states that make up the literals of the LOP. Thus, even though the minimum and maximum times for failure mode f to propagate to sensor state ss_1 is

Fault Prediction – It is always true that not the faults but their consequences that are of concern. Fault propagation is the mechanism which may trigger a discrepancy in the system behavior with critical consequences. A task related to diagnosis is the timely prediction of critical failures that can take place. This task becomes essential in order to predict potentially catastrophic failures and take preventive measures.

Based on the diagnostic concepts described above, the diagnosability metrics are :

Detectability of a failure mode $f \in F$, denoted by $DET(f)$, is the pair $< t_{erl}, t_{lat} >$, $t_{erl} \leq t_{lat}$, where t_{erl} is the minimum time that will pass before f can be included in the ambiguity set after its occurrence, and t_{lat} is the maximum time that will pass before its inclusion in the ambiguity set. If $t_{erl} = t_{lat} = \infty$, then f is not detectable.

Distinguishability of a failure mode $f \in F$ in time t, denoted by $DIS(f, t)$, is true if the ambiguity set contains only f when time t has passed after the occurrence of f, else it is false. Note that even if f is not distinguishable in time t, it might be distinguishable in time $t_1 > t$ because at time t_1 there might be more evidence (observed discrepancies) available for the diagnosis.

Predictability of a discrepancy $d \in D$, denoted by $PRED(d)$, is the pair $< t_{min}, t_{max} >$, $t_{min} \leq t_{max}$, where t_{min} is the shortest available time period between a forewarning and the actual occurrence of d, and t_{max} is the longest available time period between a forewarning and the actual occurrence of d. If $t_{min} = t_{max} = 0$, d is not predictable at all.

These three metrics can be used to characterize the diagnosability of the system. For example, the four questions listed in the previous section can be paraphrased in terms of these metrics as :

Diagnosability Metrics

Diagnosability analysis and sensor placement require the definition of metrics which can be used to measure the diagnosability characteristics. Here we define the metrics for diagnosability. The rationale for these definitions comes from the fundamental concepts of diagnosis and are independent of the actual reasoning method.

Fault Detection – Presence of a fault is detected by observation of one or more discrepancies. Because of the complex relationship between faults and discrepancies, more than on fault may be suspected when a discrepancy is observed. The set of all these possible faults has been called by many different names – the hypotheses set, the suspect set, the candidate set, the ambiguity set, etc. This set represents the ambiguity in the diagnostic results at any given time.

Because of the dynamics of the system, some time may pass before a fault causes the observed discrepancy(ies). The fault is said to be detected at the time it is first included in the ambiguity set. The fault may propagate to more discrepancies, but the evidence provided by these additional discrepancies will not go towards detecting the fault. The evidence will, instead, be used to diagnose the fault.

Fault Isolation – Once a fault has been detected, it needs to be isolated from other possible faults. A diagnoser does this by examining the elements of ambiguity set and by using some reasoning technique to prune the set down to the actual fault. During the process of diagnosis, more evidence may come in, causing the ambiguity set to shrink or grow. The goal is that at the end of diagnosis, the ambiguity set should contain only the fault that actually occurred, i.e., there should be no ambiguity in the diagnosis. In case of multiple faults, the set should contain all the faults that actually occurred and only the faults that actually occurred. The time needed for the resolution of ambiguity between faults may vary depending on the dynamics of the system and sensor coverage which determines the available evidence at a given time.

Hierarchical Fault Models

The diagnosability analysis algorithms operate on FPMs of a system. As noted in Chapter II, a system is modeled by breaking it down hierarchically into subsystems. The hierarchy thus generated consists of many functionalities and sub-functionalities. All the functionalities have their own FPMs, which also interact with each other. Thus, all the FPMs in the hierarchy are linked together. In fact, the whole hierarchy can be "flattened" into one FPM which contains all the failures and their propagations in the system.

The diagnosability analysis can be performed on the flat FPM or on FPMs of individual functionalities. When operating on the flat FPM, all the failure modes, discrepancies and sensor states can be considered at once. But the flat graph may become too big and unmanageable. On the other hand, analysis of individual FPMs is manageable. But in this case, combining the results of individual analyses requires some extra work. For example, there might be a sensor that can be used to generate alarms in different parts of the hierarchy. Thus, committing to an alarm in a functionality might effect the decision in other functionalities.

Thus, we provide the choice of analyzing individual FPMs or a flat FPM. Further, the design engineer is allowed to flatten any subtree of the functional hierarchy tree and analyze the flattened FPM for just the subtree. This way one can choose to analyze a whole sub-system. One may also flatten the whole hierarchy, thereby performing a complete fault analysis. The analysis algorithms remain unaffected whether the FPM belongs to an individual functionality or it has been obtained by flattening a part or whole of hierarchy.

CHAPTER IV

DIAGNOSABILITY ANALYSES

In this chapter, the different types of analyses that assist the design engineer in assessing and enhancing the diagnosability of a system are described. The design engineer is interested in two basic enquiries :

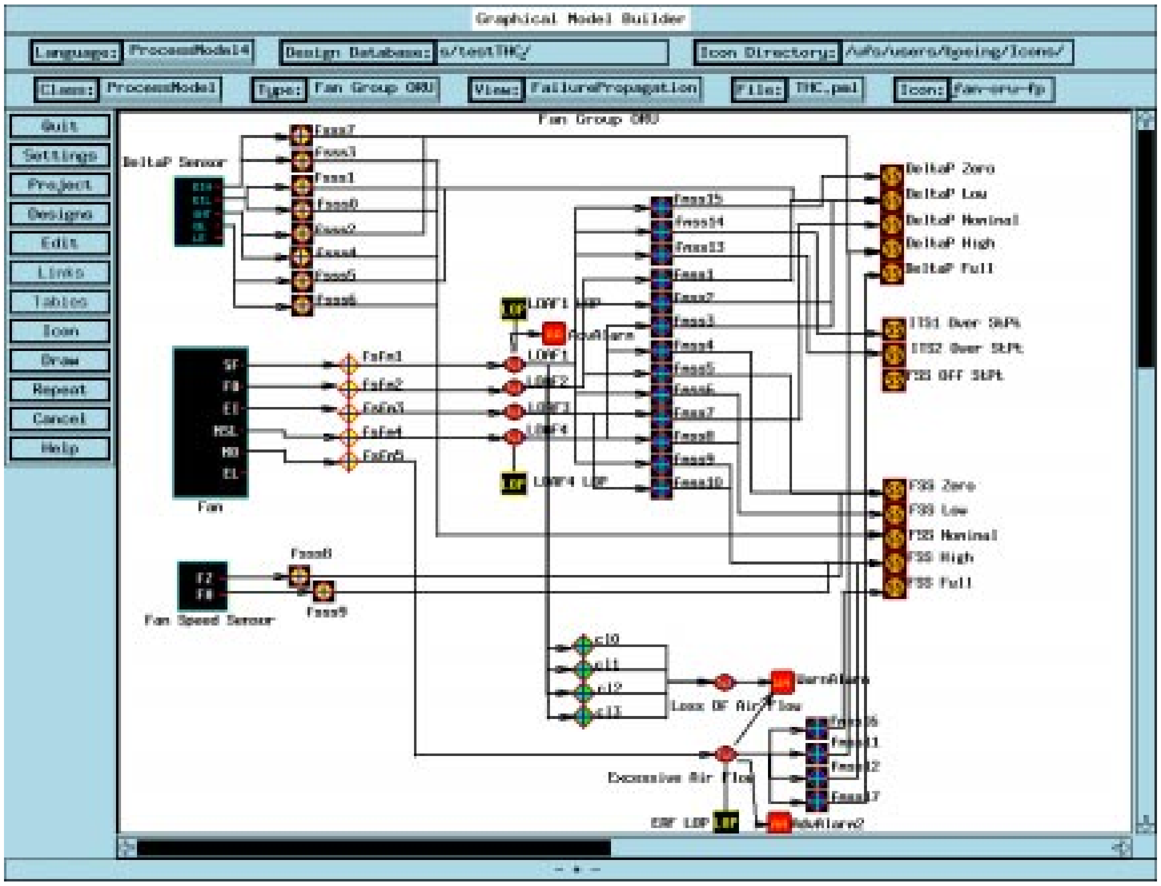
1. Is the system adequately covered by sensors such that the fault status of the system can be maintained correctly at all times ? If not, where does it fall short ?

2. What are the different possible sensor assignments that will preserve the diagnosability of the system and also save on the costs of sensors and tests ?

The same questions can be asked in a more specific manner, for example :

- How long will it take to detect a fault in a specific component?
- Can a particular catastrophic failure be predicted?
- Does adequate sensor coverage exist to isolate faults to the components ?
- What sensor assignment is needed to detect every fault ?

We have defined metrics to quantitatively characterize the diagnosability of a system. The diagnosability of a system can be evaluated in terms of these metrics and the metrics can be used to specify some diagnosability criteria to ask for suggestions about sensor assignment. The metrics and the analyses are described in the upcoming sections.


Graphical Model Builder

Language: ProcessModel4
Design Database: s/testTHC/
Icon Directory: /ufs/users/bgeing/Icons/
Class: ProcessModel
Type: Fan Group ORU
View: FailurePropagation
File: THC.pml
Icon: fan-oru-fp

Figure 21. Failure Propagations in Functionality Fan Group ORU

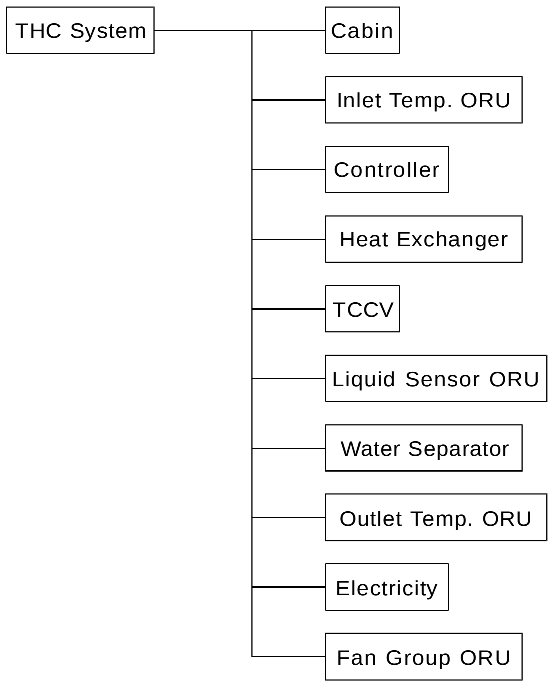


Figure 20. Functional Hierarchy of THC

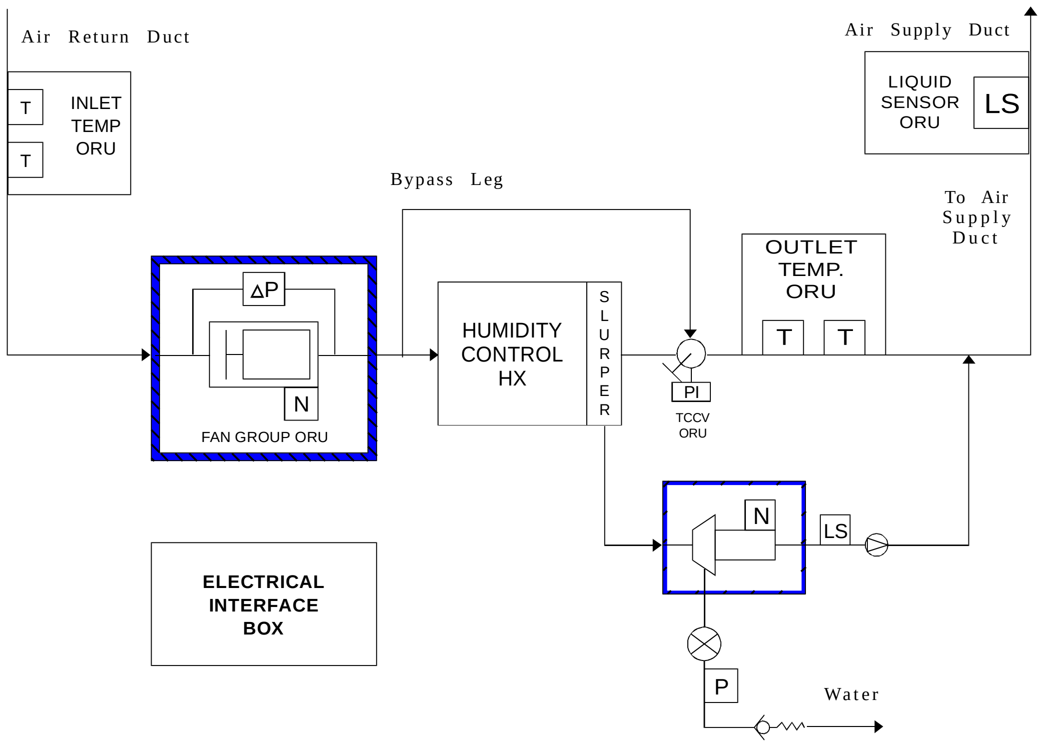


Figure 19. Schematic of THC

states in its fault models.

This system was the Temperature and Humidity Control system (THC) for cabins in Space Station Freedom. For testing the system, an interface through which signal values for the various sensors could be fed, was developed. The monitoring subsystem converted these signal values into sensor states and passed the SSEs to the diagnostic system. The schematic of THC is shown in Figure 19. Figure 20 shows the functional hierarchy of THC. Figure 21 shows the fault model of the Fan Group ORU complete with sensor states.

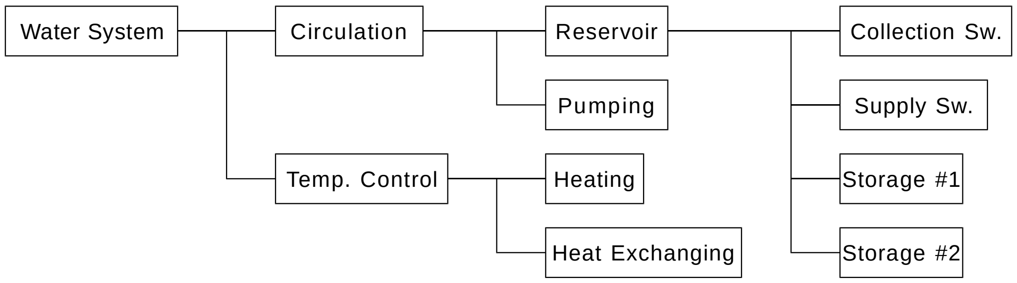
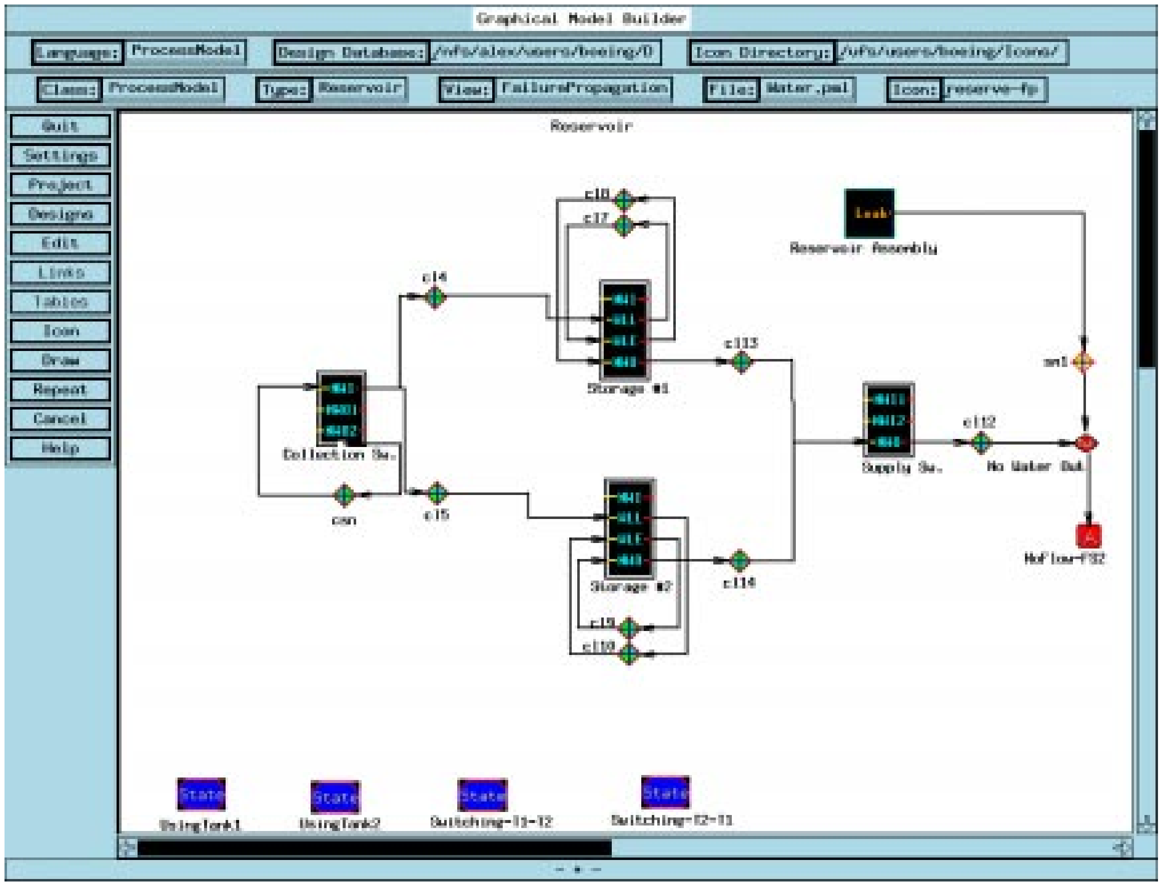


Figure 17. Functional Hierarchy of Thermal Control System


Graphical Model Builder

Language: ProcessModel
Design Database: /nfs/alex/users/boeing/D
Icon Directory: /nfs/users/boeing/Icons/
Class: ProcessModel
Type: Reservoir
View: FailurePropagation
File: Water.pml
Icon: reserve-fp

Figure 18. Failure Propagations in Functionality Reservoir

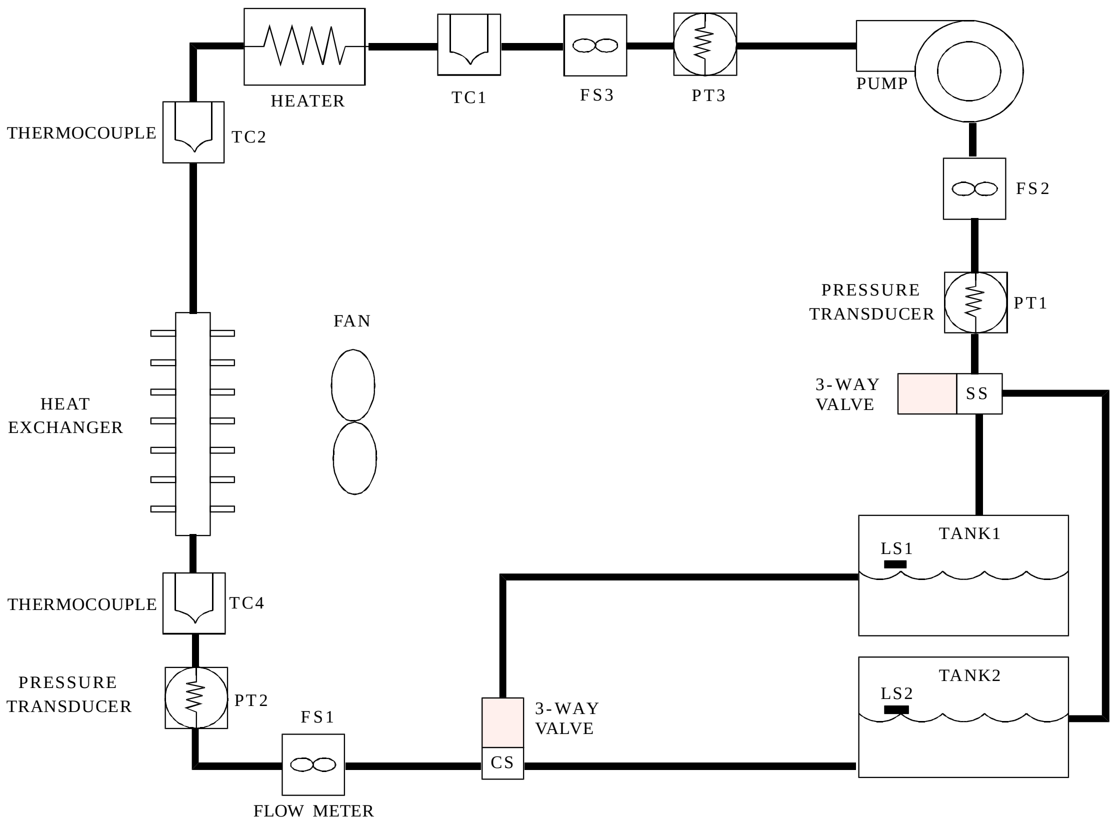


Figure 16. Schematic of Thermal Control System

TABLE 3 Silent Alarm Sensor Failures

Faults	Time in seconds
PS1	1.223147
PS1+PS2	0.967154
PS1+PS2+PS3	0.842267
PS1+PS2+PS3+PS4	0.767575

is shown in Figure 16, the functional hierarchy is shown in Figure 17 and the FPM of one of the functionalities is shown in Figure 18. This system has the simple task of maintaining the temperature of water within certain limits. There are two tanks in the system, only one of which is on-line at a time. The water is pumped through a heater and then a heat exchanger. By controlling the amount of heating in the heater and cooling in the heat exchanger, the temperature of the water is maintained. The three way valves before and after the tanks are used to put a tank on-line or off-line.

For TCS, the tests were conducted with the actual hardware. The run-time system consisted of the diagnostic system running on a workstation, talking to a data acquisition unit over a serial line. The data acquisition unit collected the signals provided by the sensors in the system and sent them over to the diagnostic system. The alarm generator part of the diagnostic system read these signals and rang and/or shut off alarm(s). The diagnoser then updated the fault status of the components in the system and displayed the results. We tested leaks in the tanks, fan failure, pump failure etc. and an assortment of sensor failures. The diagnoser returned correct results in case of component failures. Sensor failures were also diagnosed properly. In fact, even when all the three pressure transducers were failed, the diagnoser deduced the sensor failures correctly since none of the flow meters indicated any abnormal values.

The above two examples were of systems which were modeled without using the concept of sensor states and LOPs. The next example described here used sensor

TABLE 2 False Alarm Sensor Failures

Faults	Time in seconds
TS	0.390315
PS	0.687290
PS+TS	0.466055
TS+CS	0.489567
TS+CS+PS	0.484502

shows the results with single and combinations of sensor failures. The sensor failures in this experiment gave rise to false alarms. They were all diagnosed as sensor failures since the evidence against an actual fault and for a false alarm was quite strong. In the next experiment, the pump was failed and also one, two, three and four pressure sensors downstream from the pump were failed successively such that the sensors did not indicate a drop in pressure when they were supposed to (missing alarm). The results are shown in Table 3. With the first three sensor failures, the diagnoser returned correct diagnostic results – failure of pump and failures in the three sensors which did not detect a drop in pressure. When the fourth sensor was failed, the diagnoser returns the following result – the pressure sensor and flow meter in the pump assembly which are reading low values are faulty, the pump is all right and none of the four pressure sensors mentioned above are faulty. This is a case when the diagnoser gets misled by sensor failures but it requires four sensor failures to cause the diagnoser to go astray. As long as there is sufficient evidence for diagnoser to work with, it returns the correct results. Note that as the number of missing alarms increases, the diagnosis time goes down. This is because the diagnosis is event driven and is triggered every time that an alarm rings. Thus with a smaller number of ringing alarms, the diagnosis gets triggered less number of times and takes less cumulative time for diagnosis.

The second example is that of a Thermal Control System (TCS) whose schematic

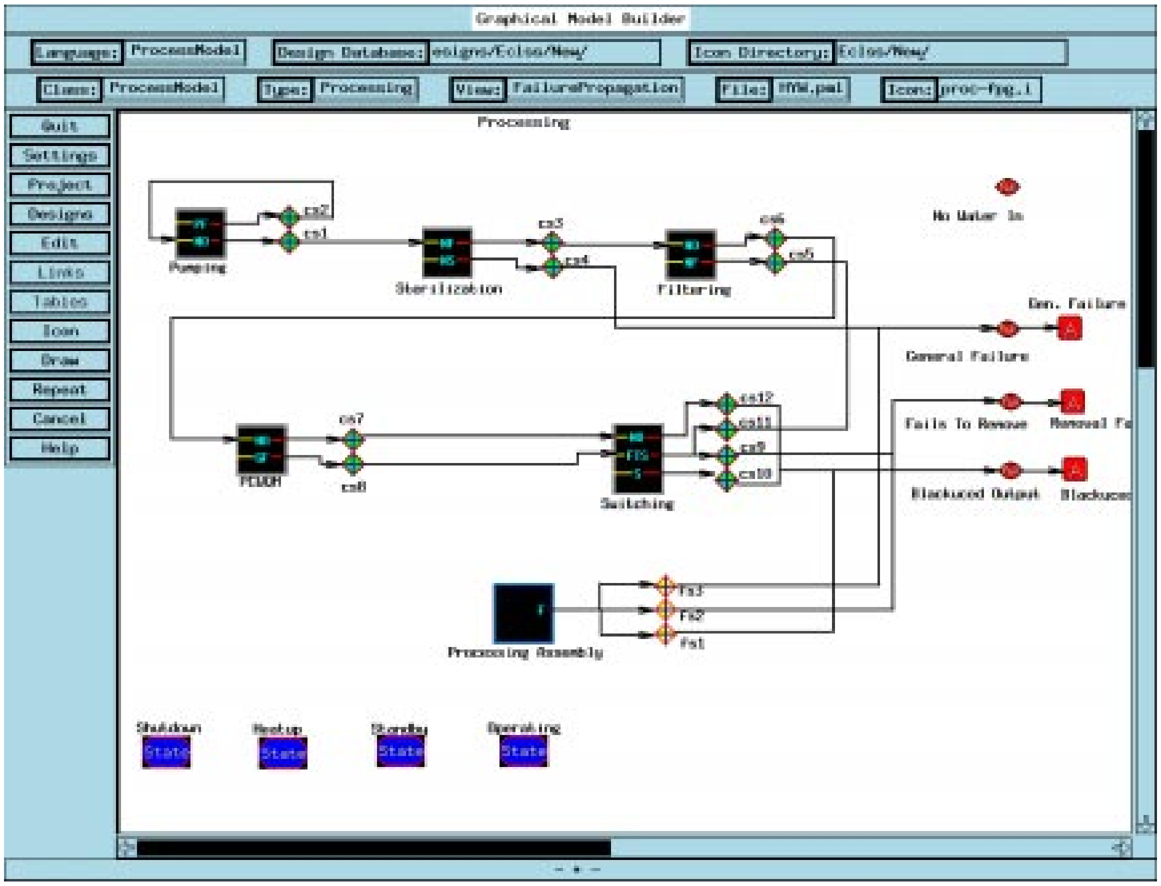
TABLE 1 Single and Multiple Failures

Faults	Time in seconds
TNK	0.450163
HTR	0.874088
PMP	1.004877
TNK+HTR	1.451494
TNK+PMP	1.365014
HTR+PMP	1.249900
TNK+HTR+PMP	1.572464
HTR+PMP+WQM	1.652971
TNK+HTR+PMP+DST	2.148210
TNK+HTR+PMP+WQM	2.106865

(PCWQM), (5) `DST` is the distribution assembly which supplies the clean water to user points, (6) `TS` is the temperature sensor in the heater, (7) `PS` is the pressure sensor in the pump assembly, (8) `CS` is the pressure sensor in the pressurization assembly, (9) `PS1` is the pressure sensor after the 3-way valve, (10) `PS2` is the pressure sensor after PCWQM, (11) `PS3` is the pressure sensor after the UF/RO filter assembly, and (12) `PS4` is the pressure sensor in the Sterilization assembly. The fault column shows the fault(s) that were simulated. Multiple faults are indicated by `+`, e.g., `TNK+HTR` means that a leak in the tank and a fault in the heater (fail high) were the two faults simulated.

The time column shows the time spent on diagnosing the faults. Note that the diagnosis system consists of an ILC and a number of instances of functionality diagnosers (FD), as discussed before. The diagnostic system is implemented using the communicating actors model of parallel, distributed computation [10]. The ILC and all the FDs are implemented as actors which exchange information, commands and results by using message passing. Thus, time shown in this column includes the time spent in ILC, in the various FDs and in the message passing.

Table 1 shows the results of single and multiple fault scenario simulations. Table 2


Graphical Model Builder

Language: ProcessModel
Design Database: esigns/Eclss/New/
Icon Directory: Eclss/New/
Class: ProcessModel
Type: Processing
View: FailurePropagation
File: HYW.pml
Icon: proc-fpg.i

Figure 15. Failure Propagations in Functionality Processing

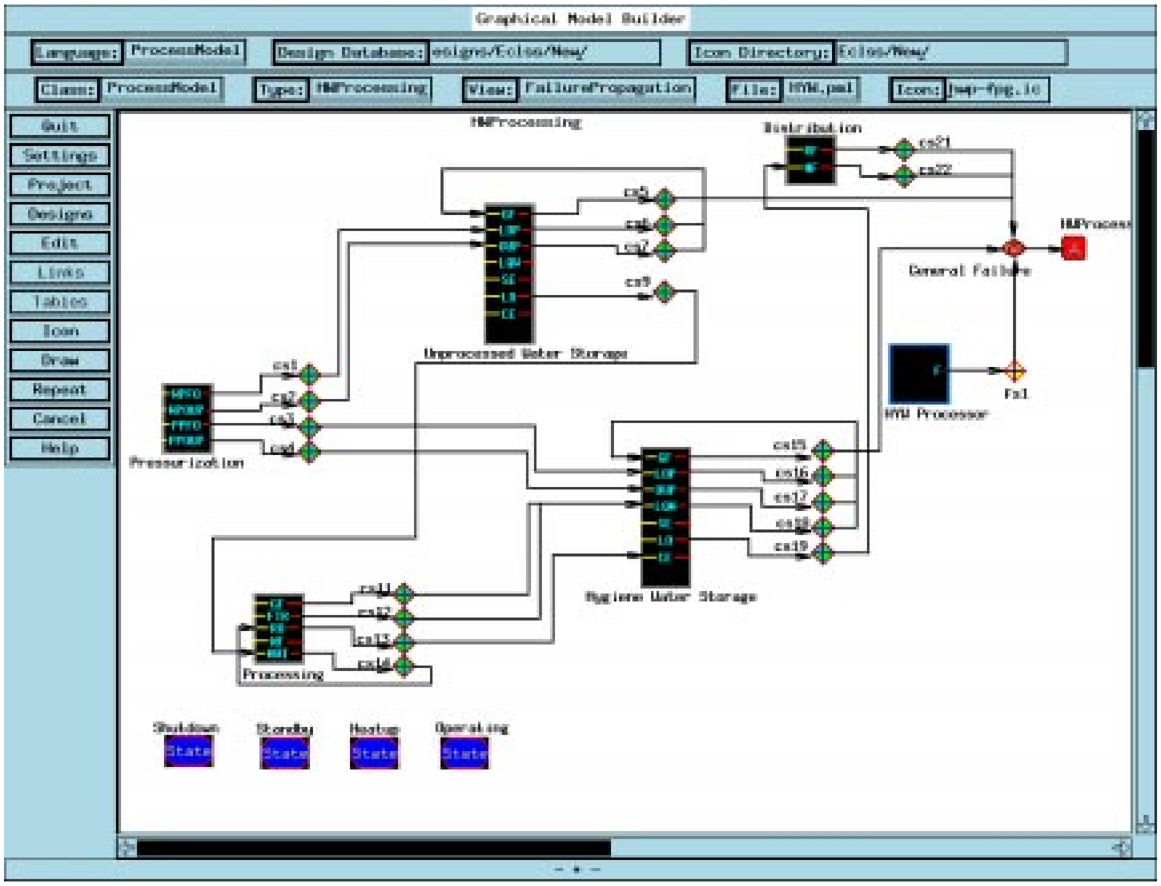

Language: ProcessModel
Design Database: esigns/Eclss/New/
Icon Directory: Eclss/New/
Class: ProcessModel
Type: HWProcessing
View: FailurePropagation
File: HYW.pml
Icon: hwp-fpg.ic

Figure 14. Failure Propagations in Functionality HWProcessing

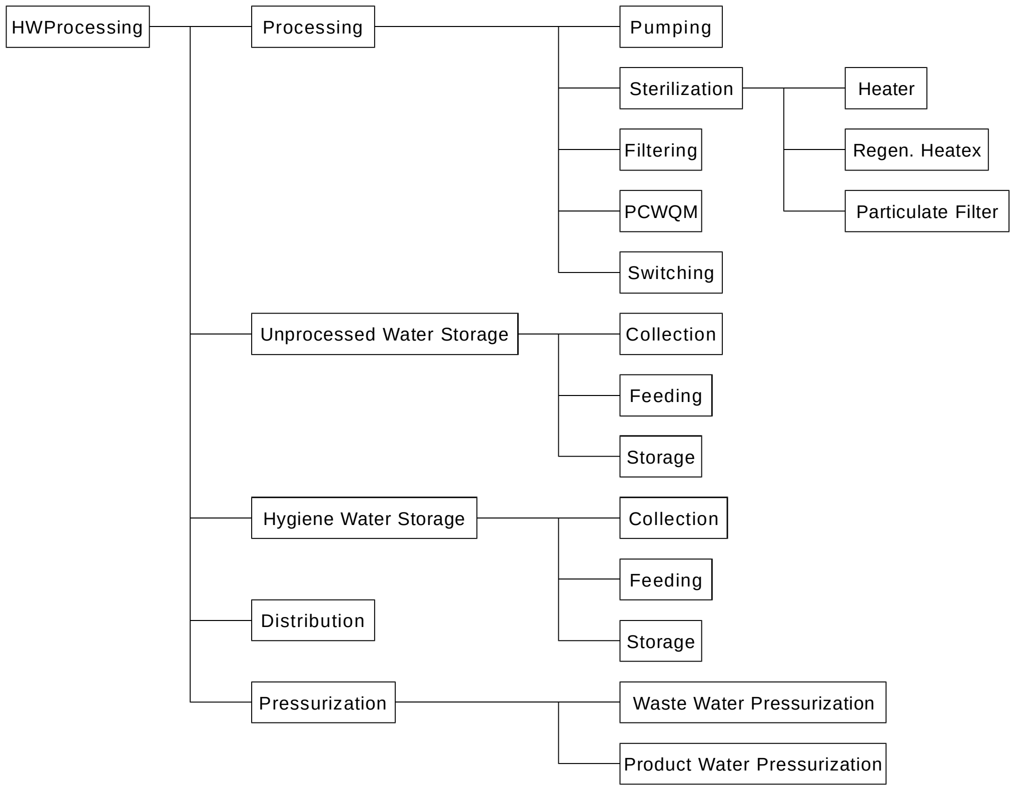


Figure 13. Functional Hierarchy of Hygiene Water Processor

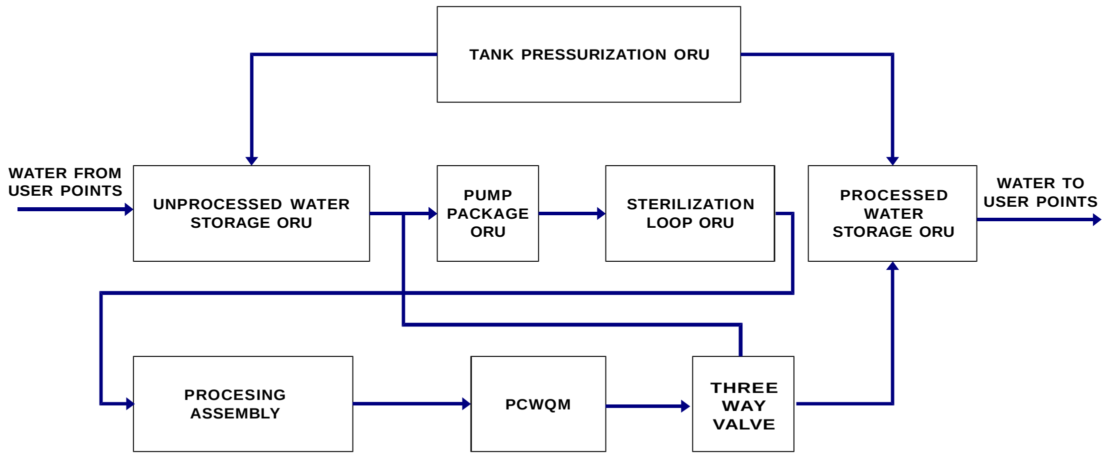


Figure 12. Schematic of Hygiene Water Processor

the diagnostic system using these models. The functional hierarchy for the HWP is shown in Figure 13. The FPM of `HWProcessing`, which is the top functionality is shown in Figure 14 and the FPM for functionality `Processing` is shown in Figure 15.

For testing purposes, a Fault Pattern Simulator (FPS) was used. The FPS provides a graphical interface to the user. The user can select the failure mode(s) of component(s) to fail and also the relative times of failures. The FPS then generates the alarms in real-time, using the failure propagation information in the models. These alarms are passed to the ILC, as and when they are supposed to ring, which diagnoses the fault(s) using the information provided by the unfolding alarm scenario. For testing sensor failure scenarios, the user can specify the sensors to fail. Then, FPS will generate a false alarm or will not ring an alarm when it is supposed to.

The results from simulations using HWP models are shown in Tables 1, 2 and 3. The abbreviations used are: (1) `TNK` is the tank in the Unprocessed Water Storage that supplies the water to the purification circuit, (2) `HTR` is the heater in the sterilization assembly, (3) `PMP` is the pump assembly, (4) `WQM` is the water quality monitor

The Hierarchy Navigator then looks at `FuncSet` and schedules the FD on one or more functionalities. The criteria for selecting a functionality is that (1) there should be a new event (ringing or shutoff alarm, timeout) since the last time FD was run on the functionality and (2) No ancestor of the functionality in the hierarchy should be in `FuncSet`. This way a top-down search in the hierarchy is performed, refining the diagnostic results with more details as the search goes down the hierarchy.

Tests and Discussion

In this section some examples of component faults, sensor faults and their combinations are given and the robustness of the diagnostic system is examined. First, the results obtained by testing using simulation of faults are discussed. Then the test with an actual system is described.

These tests show the capability of the algorithm to diagnose single and multiple faults in components and sensors accurately, failing only when the number of sensor faults becomes large.

The first example is in the context of the models for Hygiene Water Processor for the Space Station Freedom. Figure 12 shows a simplified schematic of the HWP system. The tanks in the Unprocessed Water Storage (1) collect the water from user points, (2) store it, and (3) supply it to the purifying circuit. The tanks in the Hygiene Water Storage (1) collect water after purification, (2) store it and (3) supply it to user points. The purification circuit itself consists of sterilization, bacteria removal and some other processing. The PCWQM senses the amount of contaminants in the water and controls the 3 Way Switch to either send it back to the purification circuit or to the Hygiene Water Storage tank. The models for the HWP were built and tested

3. Timeout occurs.

Upon being triggered, ILC goes through the following steps :

1. If the trigger is an alarm ringing/shutoff, then add to `FuncSet` all the functionalities that have the alarm as part of their FPM and add the alarm to the event list of the functionalities. This step is performed by the Monitoring Interface.

2. If the trigger was a timeout, add to `FuncSet` the functionality which was waiting on the timeout and add the timeout to the event list of the functionality. This step is performed by the Timeout Manager.

3. If the trigger was diagnostic result from FD,

   (a) If the result contains a next expected event, set a timeout for the functionality that the result is from. This step is performed by the Timeout Manager.

   (b) Examine the fault hypotheses.

      i. Are there any sub-functionalities of the current functionality that need to be diagnosed ? This involves identification of the sub-functionalities that use one of the components that are being indicated faulty and the sub-functionalities that have alarms ringing in them. If such a sub-functionality exists, it is added to `FuncSet`.

      ii. If the number of failures indicated by the results is beyond a preset limit, conclude that the models in the subtree rooted at the functionality are no longer valid, mark all the functionalities in the tree as invalid and remove them from `FuncSet`, if any were in the set.

   This step is performed by the Hypotheses Manager. The User Interface displays the current fault status of the system.

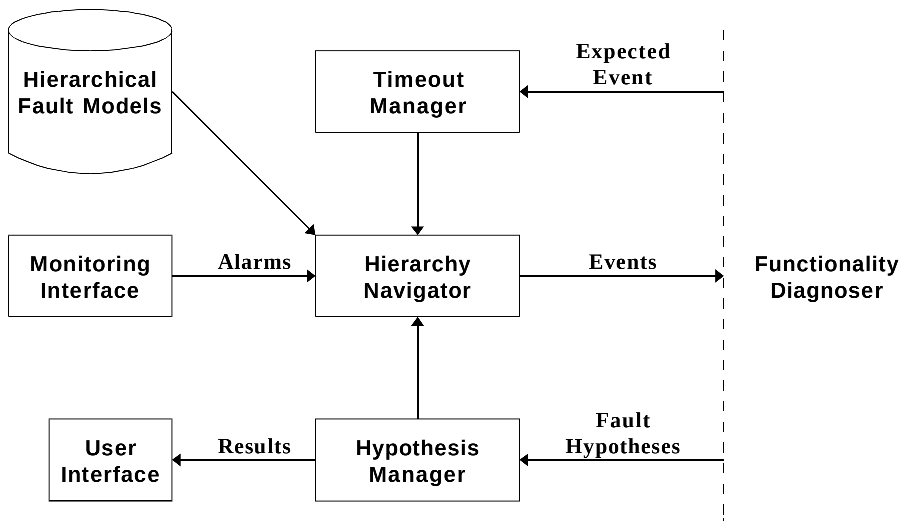


Figure 11. Block Diagram of the Inter-Level Coordinator

3. Once the SSE has been mapped to alarm event(s), execute the FD on the alarm event(s). The FD algorithm remains the same except for some minor changes which check for alarm events that were mapped from SSEs.

Inter-Level Coordinator

Figure 11 shows the block diagram of ILC. The Monitoring Interface receives alarm ringing and alarm shutoff events from the monitoring sub-system. The User interface collates the diagnostic results and displays them to the user.

The ILC maintains `FuncSet`, a set of functionalities that might need diagnosing. It adds and/or deletes functionalities from this set depending upon the fault scenario. The ILC gets triggered whenever :

1. Monitoring Interface receives an alarm ringing or shutoff event.

2. Diagnostic results from FD comes in.

Handling Sensor States

The foregoing discussion of diagnostic algorithms assumed that the FPM consisted of only failure modes and discrepancies. In this section we describe how sensor states effect the algorithms and what changes are needed to incorporate sensor states in the algorithm.

To handle sensor states, the algorithm is modified in the following manner :

1. Define a new event type, called the sensor state event (SSE), denoted ev^{ss}. The SSE is a triple $< ss, t, c >$ where ss is the sensor state with which the event is associated, t is the time of occurrence of the event and c is the description that characterizes the event. c can be one of the following :

   (a) *Active* – signifies that sensor state ss became active at time t.

   (b) *Inactive* – signifies that sensor state ss, which had been active for a while, became inactive at time t (for whatever reason, perhaps because the fault was repaired).

   (c) *Timeout* – signifies that sensor state ss which was expected to become active at or before time t did not do so.

2. When a SSE comes in, it gets mapped to an alarm event. The mapping of a SSE to an alarm event is done in the following manner :

   (a) If there is propagation edge from a failure mode to the sensor state, treat this sensor state as a monitored discrepancy, and add a corresponding alarm event to EV.

   (b) If there is a propagation from a discrepancy to this sensor state, check if the expression in the LOP associated with the discrepancy evaluates to true. If so, add a corresponding alarm event to EV.

of the sets. And, since these two steps are repeated for all the alarms and all the hypotheses, the algorithm *FindMissingAlarms* is $O(n_{hf}.m.(n+m)^2)$.

Analysis of *LocateFSC*

The second step takes $O(n_{hf}\ logn_{hf})$ time. The loop in step 4 is executed *at most alarmsToExplain* times, even though there might be many more hypotheses. This is because each hypothesis that is examined will explain at least one alarm. If, however, the next hypothesis does not explain any alarm, i.e., its rank is less than 1, the loop is exited. Thus, no more than *alarmsToExplain* hypotheses are examined. And since *alarmsToExplain* can not be more than the number of alarms, the loop in step 4 will be executed at most m times. The sub-steps of step 4 are $O(m)$. Thus, step 4 is $O(m^2)$.

Step 5 is $O(m)$ since we just go through the list of alarms once. Thus, the complexity of *LocateFSC* is $O(n_{hf}logn_{hf}\ +\ m^2)$.

Analysis of *FindNextAlarm*

The complexity of this algorithm is $O(m^2)$ since there are at most m alarms and for each ringing alarm we examine, in the worst case, all the other alarms exactly once.

Complexity of the Diagnostic Algorithm

From the complexity of the three steps described above, the complexity of the whole algorithm is $O(n_{hf}.m^2 + n.n_{hf} + n_{hf}.m.(n+m)^2 + n_{hf}logn_{hf} + m^2 + m^2)$. In the worst case, with $n_{hf} = n.m$, the complexity reduces to $O(n.m^4 + n^3m^2 + n^2m^3)$. In the more common case, n_{hf} will be $O(n)$ and then the complexity reduces to $O(n.m^3 + n^3m + n^2m^2)$.

Analysis of *UpdateHypotheses*

The first step in the algorithm checks if the alarm is a primary alarm of any failure mode. This takes $O(1)$ time since the information is maintained in matrices. If the new alarm is a primary alarm of a failure mode and the failure mode does not have any corresponding hypothesis in $\hat{r}$, a hypothesis is added to $\hat{r}$. This checking and adding takes $O(n_{h^f})$ time. Since the above is repeated for every failure mode, the first step is $O(n.n_{h^f})$.

The algorithm for *RecomputeTime()* is $O(m^2)$, as is explained later. Thus the second step takes $O(n_{h^f}.m^2)$ time.

Steps $3(a)$, $3(b), 3(c)$ and $3(d)$ are $O(m^2)$. Thus, step 3 takes $O(n_{h^f}.m^2)$ time. Thus, the complexity of *UpdateHypotheses* is $O(n_{h^f}.m^2 + n.n_{h^f})$.

Analysis of *RecomputeTime()*

This algorithm involves going through the list of all the hypotheses that a failure mode may have. Since a failure mode can be incident upon at most m discrepancies, there can be at most m such hypotheses. The loop in step 2 is executed $O(m)$ times because with every failure mode, we maintain a list of pointers to the entries $\hat{h}^f \in \hat{r}$ such that $\hat{h}^f$ and h^f stand for same f, and there can be at most m such entries. For each such hypothesis, the new alarm is added to either P or SP set. Thus *RecomputeTime()* takes $O(m^2)$ time.

Analysis of *FindMissingAlarms*

The procedure *ShouldHaveRung* takes $O(1)$ time since all it does is one mathematical calculation and one comparison. The procedure *IsOnlyPath*, however, takes $O((n+m)^2)$ time since it makes a DFS of the graph. Thus, step $3.a.i$ in the algorithm is $O((n+m)^2)$. Step $3.a.ii$ is only $O(m)$ since all it does is add the alarm to one these hypotheses.

The algorithm is given in Appendix A.

Procedure *FindNextAlarm*

This step finds the next alarm that should ring, given the hypotheses under consideration, and the latest time by which it should ring. The algorithm is given in Appendix A. The time computed by the algorithm is used to set the next timeout. It might happen that there are no more alarms that are expect to ring, or that the current time is such that timeouts for all the alarms have occurred. In such a case, no timeout is set.

Complexity Analysis

There are n failure modes and m discrepancies. The total number of nodes in G is $(n + m)$. Some of the above algorithms use Depth First Search (DFS) on G. The order of DFS is $O((n + m)^2)$ since the graph is represented using adjacency matrix A.

Many of the loops iterate on the number of hypotheses n_{hf}. The worst case number of hypotheses can be $O(nm)$. But this will be the case when every failure mode can cause every discrepancy and all the discrepancies have alarms associated with them *and* every alarm is ringing *and* none of the alarms agree with each other temporally. To begin with, no reasonable fault model will have all the failure modes causing all the discrepancies. Secondly, in case of a valid fault scenario, most of the alarms *will* agree with each other. Usually, the number of discrepancies that a failure mode can cause will be $O(1)$, not $O(m)$, and the number of hypotheses will be $O(n)$.

The alarm sets P, S etc. are maintained as arrays and the operation of adding or deleting the alarms is $O(m)$ since the maximum number of alarms is m.

- The alarms in the MP and MS sets of a failure mode hypothesis will require inclusion of sensor fault hypothesis(es).
- A sensor fault hypothesis can reduce the number of failure mode hypotheses (and also, perhaps, other sensor fault hypotheses) needed.
- On the other hand, a sensor fault hypothesis may reduce the confidence in some failure mode hypothesis(es), forcing the selection of some other failure mode hypothesis(es).

Another factor which effects the selection of hypotheses is the *diagnostic bias* towards component fault or sensor fault. If sensors are more likely to fail than components, the hypothesis selection heuristic can be modified to be biased towards sensor faults, i.e., it will try to explain the events with more sensor fault hypotheses than failure mode hypotheses. With a bias toward component faults, it will try to explain events with failure mode hypotheses.

This bias can also be included in the ranking of hypotheses. As explained earlier, the evidence from alarms can be weighed with the level of belief in sensors. If sensor failure probabilities are high, the level of support provided by alarms in the P and S sets gets reduced and the level of negation provided by alarms in MP, SP, MS and SS gets increased, thereby reducing the belief in the failure mode hypothesis.

Thus we see that depending on the requirements and the information available, various ranking and hypothesis selection methods can be used. For the simple ranking method described previously, a selection method with neutral bias is : for inclusion in r, consider only those failure mode hypotheses that have a rank higher than 0 (since only the hypotheses with rank 1 or higher have more evidence supporting it than negating it). The sensor fault hypotheses are selected according to the missing alarms of these hypotheses and any alarms that are not in the P or S sets of any of can be caused by a component fault or a sensor fault. An alarm shutoff or timeout could be caused either by sensor fault or by the fact that no fault exists. Thus, an event $ev =< a, t, c >$ is said to be explained by diagnosis $r \in 2^H$ if

1. $c = Ringing$ and

   - $\exists h^f \in r$ such that $a \in P(h^f) \vee a \in S(h^f)$ (ev is caused by component fault), OR
   - $\exists h^s \in r$ such that $a = a(h^s) \wedge type(h^s) = false$ (ev is caused by sensor fault).

   OR

2. $c = Shutoff$ or $c = Timeout$ and

   - $\neg\exists h^f \in r$ (there is no component fault), OR
   - $\exists h^f \in r$ such that $a \in MP(h^f) \vee a \in MS(h^f)$ and $\exists h^s \in r$ such that $a = a(h^s) \wedge type(h^s) = missing$ (there is at least one component fault and ev is a missing alarm caused by sensor fault).

How can the hypotheses be chosen for inclusion in r such that all the events are explained *and* r is minimal ? Trying all combinations of component and sensor fault is computationally prohibitive. The hypotheses have to be added to r incrementally, choosing at each step a hypothesis that will require the least number of additional hypotheses to explain the events. The selection of a hypothesis can influence other selections in following ways :

- A failure mode hypothesis can explain the ringing alarms which are in its P and S sets, thus reducing the number of other failure mode and sensor fault hypotheses needed to explain the alarms.

2. might have one or more ancestor ringing alarm(s).

3. might or might not be a descendant of one or more hypotheses already in $\hat{r}$.

4. might or might not be a descendant of one or more hypotheses that are not in $\hat{r}$.

5. might be an ancestor of some other ringing or silent alarms.

When the event comes in, its effect of $\hat{r}$ might be :

- recompute t_{erl} and t_{lat} for some (possibly all) the hypotheses $h^f \in \hat{r}$.
- add new hypotheses to $\hat{r}$.
- update the P, S, SP andSS sets of $h^f \in \hat{r}$.

The algorithm for updating $\hat{r}$ in Appendix A.

Procedure *FindMissingAlarms*

For each hypothesis h^f, there might be some alarms that should be ringing, given $t_{erl}(h^f)$, $t_{lat}(h^f)$ and the present time, but are not ringing. Such alarms are called the (primary or secondary) *missing* alarms of hypothesis h^f. To be able to compute the rank of hypothesis h^f, the missing alarms of h^f need to be identified. *Note that this and the following two steps are performed when* $type(ev) = Shutoff$ *or* $type(ev) = Timeout$ *in addition to when* $type(ev) = Ringing$. This is because a silent alarm ($Shutoff$ or $Timeout$) is expected to be a missing alarm of some hypothesis(es). The algorithm is given in Appendix A.

Procedure *LocateFSC*

The diagnoser next explains the current set of events by selecting component and/or sensor fault hypothesis(es) that comprise diagnosis $r \in 2^H$. A ringing alarm

1. Updating the hypotheses. This involves adding/removing hypotheses from $\hat{r}$ and adding/removing alarms from the P, SP, S and SS sets of $h^f \in \hat{r}$ (procedure *UpdateHypotheses*).

2. Finding the missing alarms of $h^f \in \hat{r}$ and add these to MP or MS (procedure *FindMissingAlarms*). The Update module shown in Figure 10 acts as the corrector by performing this and the first step.

3. Locate fault sources, i.e., identify the h^f and h^s to include in r (procedure *LocateFSC*). This step is performed by the Evaluator module shown in Figure 10.

4. Given the current set of hypotheses in $\hat{r}$, determine the next alarm event $ev =< a, t, ringing >$, where $a \in A$ is the the next alarm that should ring at or before time t (procedure *FindNextAlarm*). This expected event is passed on to the ILC along with the diagnostic results r. The ILC is then responsible to generate a corresponding timeout event $< a, t, timeout >$ and invoke the FD if the alarm does not start ringing by time t. The Predictor shown in Figure 10.

These steps are described in the following sections. Step 1, 2, and 4 use the principle of structural redundancy, while step 3 uses the principle of parsimony. These four steps are explained for an alarm ringing event, i.e., $ev =< a, t, ringing >$. For the other events, i.e., alarm becoming silent and timeout, the step *UpdateHypotheses* is not performed. Only the last three of the above steps are performed.

Procedure *UpdateHypotheses*

Let the event be $ev =< a, t, Ringing >$ where a is the alarm which starts ringing. The new alarm a

1. might not have any ancestor alarm that is ringing.

$s(h^s)$ is used to denote the sensor for which h^s stands, $a(h^s)$ for the alarm which is incorrect and $type(h^s)$ for the type of misbehavior. Note that, since SA is a many-to-many mapping, there might be more that one h^s for one sensor (if it causes more than one alarm to be incorrect) and there could be more than one h^s for one alarm. Since SA, the mapping from sensors to alarms, does not contain enough information, if an inconsistent alarm is generated by more than one sensor, all the sensors will be implicated. Having more than one h^s for one sensor does not cause any problems since they implicate only one actual sensor and provide stronger evidence in favor of the sensor fault.

The sensor fault hypothesis represents a hypothesis about *observation error*. As noted earlier, observation errors can be caused by an actual fault in a sensor or by modeling errors. Because of lack of information in SA, it is difficult to distinguish between sensor fault and modeling error with confidence; all that can be said is that the sensor, in some way, is responsible for the observation error. However, if only one of the many alarms generated by a sensor is inconsistent, it is possible to deduce modeling error.

FD Algorithms

The FD is invoked with a diagnostic problem DP and returns the diagnostic result r and the predicted next event ev. The FD maintains a possible hypotheses set $\hat{r} \subseteq H^f$ (possibly ϕ) for the FPM of a functionality, adding or removing hypothesis from it as needed. To generate the diagnosis r, it begin with $r = \phi$, selects some of the hypotheses in $\hat{r}$ and adds them to r. It might also select some hypotheses from H^s to include in r.

Whenever an event comes in, FD goes through the following four steps :

in set P.

- $MS \subseteq A$ is the set of *missing secondary alarms* for h^f. These are the secondary alarms of f that are silent but should have been ringing given $t_{lat}(h^f)$ and A^{max}.

- $SS \subseteq A$ is the set of *spurious secondary alarms*. These are the ringing secondary alarms of f that are that are *not* consistent with the alarms in set P.

$P(h^f), SP(h^f)$ etc. are used to denote the corresponding alarm sets of hypothesis h^f.

The alarm sets defined above contain the evidences that confirm or contradict a hypothesis and can thus be used for ranking the hypotheses. A simple way to do this is – When hypothesis h^f is first created, set its rank to 0. Whenever an alarm is added to the P or the S set of the hypothesis, increment its rank by one. Whenever an alarm is added to any of the other sets, the decrement the rank by one. This method of ranking gives equal weight to all the alarms. To incorporate the sensor reliability, the contributions from alarms can be weighted with the reliability data. To account for component failure rates, the final sum can be weighted with the corresponding failure rates. There can conceivably be more complex ranking methods.

Sensor Fault Hypotheses

A sensor fault hypothesis h^s is the 3-tuple $< s, a, type >$ where s is the sensor which is faulty, a is the alarm that is behaving incorrectly because of the fault in s, and $type$ characterizes the misbehavior. $type$ can be

1. $false$ – meaning that the fault in sensor s is causing alarm a to ring when it should be silent.

2. $missing$ – meaning that a is silent when it should be ringing.

$t_{erl}(h^f)$ denotes the estimated earliest time of occurrence of f, $t_{lat}(h^f)$ denotes the estimated latest time of occurrence of f and $rank(h^f)$ denotes the rank of h^f.

Let there be a hypothesis h^f for failure mode f such that $Node(f) = i$ and an alarm a such that $Node(d(a)) = j$. Let the time of ringing of the alarm be t. Then alarm a is said to <u>support</u> hypothesis h^f if

$(A^*_{i,j} = 1)$

and

$\left[\left(t_{erl}(h^f) \leq (t - A^{min}_{i,j}) \leq t_{lat}(h^f)\right) \vee \left(t_{erl}(h^f) \leq (t - A^{max}_{i,j}) \leq t_{lat}(h^f)\right)\right]$

The first condition, $A^*_{i,j} = 1$, checks whether a is reachable from f or not. If it is reachable, alarm a is said to be a <u>descendant</u> of hypothesis h^f and h^f is said to be <u>ancestor</u> of a. The second condition compares the time of ringing of a against the current estimated time interval for f which had been computed from earlier events. If h^f is ancestor of a, the estimated time of occurrence of f may be changed after this comparison according to the time of the new alarm.

P, MP, SP, S, MS and SS are all sets of alarms defined as :

- $P \subseteq A$ is the set of ringing *primary alarms* of f which *support* h^f. Note that there may be some primary alarms of f which do not support h^f. Such cases are discussed later.

- $MP \subseteq A$ is the set of *missing primary alarms* of f. Missing primary alarms are those primary alarms of f that are silent but should have been ringing, given $t_{lat}(h^f)$ and A^{max}.

- $SP \subseteq A$ is the set of *spurious primary alarms* for h^f. These are the ringing primary alarms of f that do not support h^f.

- $S \subseteq A$ is the set of *secondary alarms* of f that are *consistent* with the alarms

The alarm $a(d^m)$ is called a primary alarm of f.

If $A^*_{i,k} = 1$ but d^m is not a primary discrepancy of f, then it is called a secondary discrepancy of f. The alarm $a(d^m)$ is then a secondary alarm of f.

2. Let there be two ringing alarms a_1 and a_2 such that $Node(d(a_1)) = i$ and $Node(d(a_2)) = j$. Let the times when the alarms started ringing be t_1 and t_2 respectively. Then alarm a_2 is said to be temporally consistent with alarm a_1 if $A^*_{i,j} = 1 \wedge A^{min}_{i,j} \leq (t_2 - t_1) \leq A^{max}_{i,j}$. Also, if $A^*_{i,j} = 1$, a_1 is said to be ancestor of a_2 and a_2, descendant of a_1. If $A^*_{i,j} = 0$, temporal consistency is irrelevant.

3. An alarm a is called a ringing alarm if event $< a, t, Ringing > \in EV$. An alarm a is called a silent alarm if event $< a, t, Ringing > \notin EV$ or $< a, t, Shutoff >$ $\in EV$. If there are two events $ev_1 =< a, t_1, Ringing >$ and $ev_2 =< a, t_2, Shutoff >$, $ev_1, ev_2 \in EV$, then if $t_1 < t_2$, alarm a is silent else it is ringing. Obviously, if $< a, t, Timeout > \in EV$, a is silent.

The Hypotheses

Failure Mode Hypothesis

A failure mode hypothesis h^f is the n-tuple

$$h^f = \ < f, t_{erl}, t_{lat}, rank, P, MP, SP, S, MS, SS >$$

where f is the failure mode of a component for which this hypothesis stands and t_{erl} and t_{lat} give the estimated earliest and latest time of occurrence of failure mode f. The $rank$ of a hypothesis is a number which gives the measure of belief in the hypothesis. The higher the $rank$, the more likely we are to believe in the hypothesis.

3. A^{min} is a $(n+m) \times (n+m)$ matrix whose elements represent the minimum time needed for one failure to propagate to another. $A^{min}_{i,j} = \hat{t}$ means that the failure at vertex v_i takes *at least* time $\hat{t}$ to propagate to the failure at vertex v_j. If $A^{*}_{i,j} = 0$ then $A^{min}_{i,j} = \infty$. A^{min} is obtained by finding the all pairs shortest paths [25] between the vertices in G, using t_{min} as the cost on the edges.

4. A^{max} is a $(n+m) \times (n+m)$ matrix whose elements represent the maximum time needed for one failure to propagate to another. $A^{max}_{i,j} = \hat{t}$ means that the failure at vertex v_i takes *at most* time $\hat{t}$ to propagate to the failure at vertex v_j. If $A^{*}_{i,j} = 0$ then $A^{max}_{i,j} = \infty$. A^{max} is obtained by finding the all pairs shortest paths between the vertices in G, using t_{max} as the cost on the edges.

Some Definitions

1. Let there be a failure mode f such that $Node(f) = i$ and a monitored discrepancy d^m such that $Node(d^m) = k$. Then d^m is called a primary discrepancy of failure mode f if $A^{*}_{i,k} = 1$ and

   (a) $\neg\exists\ \hat{d},\ Node(\hat{d}) = j$ such that $\hat{d}$ is monitored and $A^{*}_{i,j} = 1 \wedge A^{*}_{j,k} = 1$. That is, no path in G from $v(f)$ to $v(d^m)$ goes through a vertex $v(\hat{d})$ such that $\hat{d}$ is monitored, *OR*

   (b) $\exists\ \hat{d}$ such that $\hat{d}$ is monitored and $A^{*}_{i,j} = 1 \wedge A^{*}_{j,k} = 1 \wedge A^{max}_{i,k} \leq A^{max}_{i,j}$. That is, if there does exist a path which goes through (possibly more than one) monitored discrepancy vertex $v(\hat{d})$, d^m is still a primary discrepancy if the maximum time for f to propagate to d^m (given by A^{max}) is less than the maximum time for propagation to $\hat{d}$. This can happen if there are more than one path from $v(f)$ to $v(d^m)$.

1. *Ringing* – signifies that alarm a started ringing at time t.

2. $Shutoff$ – signifies that alarm a, which had been ringing for a while, became quite at time t (for whatever reason, perhaps because the fault was repaired).

3. $Timeout$ – signifies that alarm a which was expected to ring at or before time t did not ring.

The diagnostic task, then, is to find a $r \in 2^H$ which *explains* the current events in the set EV and also to predict the next expected event $ev =< a, t, ringing >$, where $a \in A$ is the alarm expected to ring at or before time t. Note that, depending on the fault scenario, there might not be any such expected alarm.

Notation : A vertex in G which represents a failure mode f is denoted by $v(f)$ and a vertex representing discrepancy d is denoted by $v(d)$. If $< a, d > \in M$, then alarm a is also referred to as $a(d)$, and $d(a)$ stands for the discrepancy to which alarm a is assigned. Thus, $v(d(a))$ is the vertex in G representing the discrepancy which is monitored by alarm a. $Node(f)$ represents the node number of $v(f)$ in G, which is between 1 and n. Similarly, $Node(d)$ represents the node number of $v(d)$ in G, which is between $n+1$ and $n+m$.

Data structures derived from G

The following is the list of data structures derived from G :

1. A is the $(n+m)\times(n+m)$ adjacency matrix for G where $A_{i,j} = 1$ iff $< v_i, v_j > \in E$ else $A_{i,j} = 0$.

2. A^* is the $(n+m) \times (n+m)$ matrix representing the transitive closure of G, such that $A^*_{i,j} = 1$ iff there is a path from v_i to v_j else it is 0.

- $A = \{a_1, a_2, \ldots a_o\}$ is a set of $o \leq m$ alarms which are used to monitor some (perhaps all) of the discrepancies.

- $S = \{s_1, s_2, \ldots s_p\}$ is a set of p sensors.

- $M : A \rightarrow D$ is a mapping which describes the monitoring used in the system, where $< a_i, d_j > \in M$ represents the fact that alarm a_i is assigned to discrepancy d_j. Note that M is a one-to-one mapping. Discrepancy d_j is monitored means that there is an alarm $a_i \in A$ such that $< a_i, d_j > \in M$.

- $SA : S \rightarrow A$ is a mapping that describes the associations between sensors and alarms, $< s_i, a_j > \in SA$ represents the fact that sensor s_i contributes to the computation of alarm a_j. SA can be a many-to-many mapping.

- $G = (V, E)$ is a directed graph derived from the FPM described in Chapter II. The vertex set V has $n + m$ vertices, representing n failure modes and m discrepancies. Without loss of generality, we will assume that the vertices representing failure modes are numbered 1 to n and the vertices representing discrepancies are numbered $n + 1$ to $n + m$. An edge $e_{i,j} =< v_i, v_j > \in E$ iff the failure represented by v_i propagates and causes the failure represented by v_j. Each edge in E is weighted by two parameters :

  1. t_{min}, which is the minimum time for propagation of failure along the edge, and
  2. t_{max}, which is the maximum time for propagation of failure along the edge.

- EV is a set of events. An event, denoted ev, is a 3-tuple $< a, t, c >$ where $a \in A$ is the alarm with which the event is associated, t is the time of occurrence of the event and c is the description that characterizes the event. c can be one of the following :

of the fault state. The output is a ranked list of hypotheses which is generated by the Evaluator module. Ranking occurs according to the plausibility of the individual hypotheses. The Results module then selects the hypotheses that explain the current observations, identifies the fault source functionality and sends the results to ILC.

The Diagnostic Problem

The diagnoser characterizes the fault status of a system by hypothesizing about faults in components or sensors. h^f denotes a hypothesis about a failure in a component, i.e., a hypothesis about the occurrence of failure mode f. The hypothesis for a fault in a sensor is denoted by h^s.

Let H^f be the set of all possible hypotheses about the failure modes in the system and H^s be the set of all possible hypotheses about the sensor failures and let $H = H^f \cup H^s$ represent all possible failure hypotheses. A diagnostic result r is a set consisting of failure mode and sensor hypotheses which is a subset of the hypotheses set H, i.e., $r \subseteq H$. Then, the set of all possible diagnoses is 2^H, the superset of H. The goal of the diagnoser is to select one minimal diagnosis $r \in 2^H$, possibly ϕ, which gives the current fault status. A minimal diagnosis is the smallest set of hypothesis that explain all the events such that removing any hypothesis will leave one or more events unexplained.

The diagnostic problem DP is defined by the n-tuple

$$DP =< F, D, A, S, M, SA, G, EV >$$

where

- $F = \{f_1, f_2, \ldots f_n\}$ is a set of n failure modes in the FPM.

- $D = \{d_1, d_2, \ldots d_m\}$ is a set of m discrepancies in the FPM.

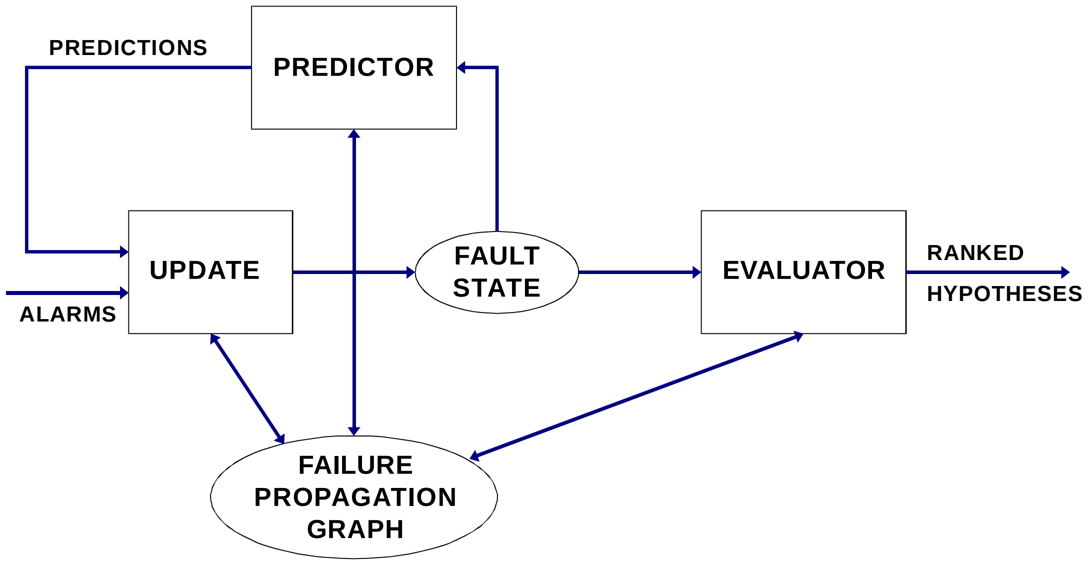


Figure 10. Conceptual structure of the robust reasoning method

Functionality Diagnoser

The Functionality Diagnoser is triggered by one of these events – (1) an alarm ringing, (2) an alarm becoming silent after ringing for a while, or (3) a timeout occurring, i.e., a predicted alarm did not ring.

The structure of the reasoning algorithm of FD uses the *predictor–corrector principle*. The conceptual block diagram of the reasoning method is shown in Figure 10.

The reasoning algorithm uses two basic data structures, the *system fault status* and the FPM. The system fault status consists of the current set of fault hypotheses and the corresponding evidence. Because the failure propagation graph expresses the dynamics of failure propagation, it can be used to predict future failures based on the current fault state. The function of the Predictor is to derive deadlines for the future incoming alarms. The event comparator compares the incoming alarms and the predicted deadlines, and drives the Update module. The Update module processes the received event (alarm or passed deadline) and creates a new update

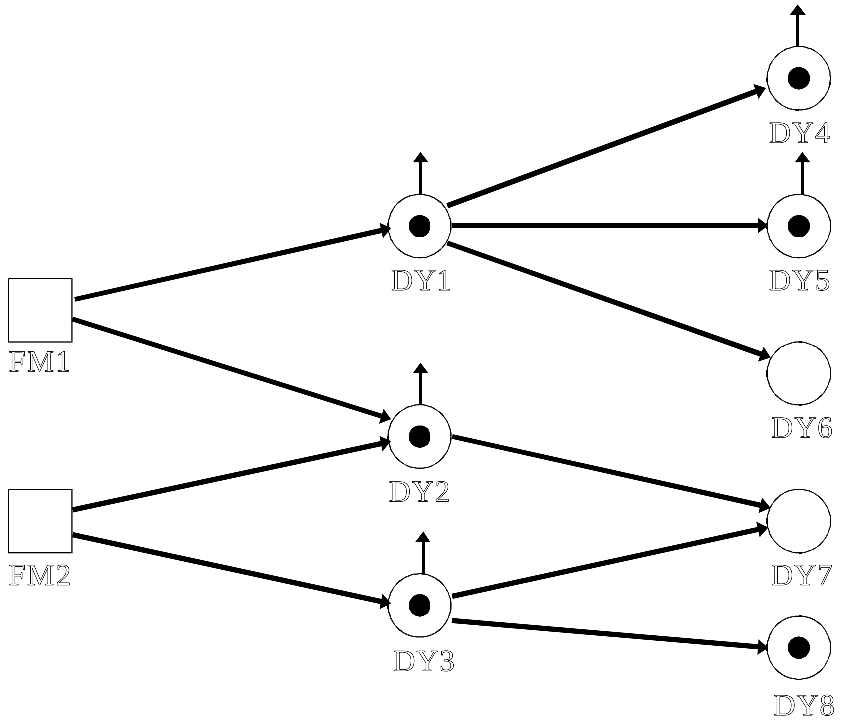


Figure 9. Use of structural redundancy for sensor fault detection

that $FM2$ is also a fault source. However, if $FM2$ is a fault source, the alarm at $DY8$ should also ring (structural redundancy), therefore this hypothesis implies that the sensor at $DY8$ must be faulty as well. An alternative explanation is that the sensor associated with $DY3$ is faulty and is giving rise to a spurious alarm. This hypothesis is more plausible than the previous one, since it explains the alarm scenario with 2 fault sources ($FM1$ and the sensor associated with $DY3$) instead of 3 ($FM1$, $FM2$ and the sensor associated with $DY8$) (parsimony). A number of other explanations can be found for the alarm pattern that are all less plausible than the previous ones.

Thus, structural redundancy means the use of the interdependence among the alarms in the reasoning algorithm. The actual reasoning method is considerably more complex than the illustrative example due to the temporal aspect of FPM.

the actual state of the system. Of course, in any realistic case, most of the observed events are directly related to underlying faults, therefore diagnosis is possible.

The principle of parsimony suggests that *the simplest explanation is the best.* If a hypothesis can explain consistently all of the observed events, it should be considered more plausible than another one, which additionally requires the assumption of a sensor fault as well. Application of the principle of parsimony means that the set of plausible hypotheses should be *minimal.*

Structural redundancy

As we have discussed before, the physical interactions in dynamic systems impose spatial and temporal constraints on the observed events. In those parts of the system where failure propagation occurs, a single fault results in multiple manifestations. Obviously, these manifestations are not independent of each other. They provide a *redundant* observation of the fault. Because the failure propagation models primarily represent structural relationships in the systems, we call this redundancy *structural redundancy.*

Due to the structural redundancy, events can confirm or contradict other events in a propagation model, therefore, a concept similar to that of the analytical redundancy approach can be developed. The idea is illustrated with the simple FPM shown in Figure 9. The graph shown here includes only failure propagations. For the sake of simplicity, all of the monitored discrepancies are associated with a unique sensor, whose output signal is used by a monitoring algorithm to generate the alarm.

In the particular fault scenario in Figure 9, the alarms associated with $DY1$, $DY2$, $DY3$, $DY4$ and $DY5$ are ringing (indicated by up arrow), while the alarm assigned to $DY8$ is silent. The simplest explanation for the alarms at $DY1$, $DY2$, $DY4$ and $DY5$ is that $FM1$ is a fault source (parsimony). Possible explanation for alarm $DY3$ is

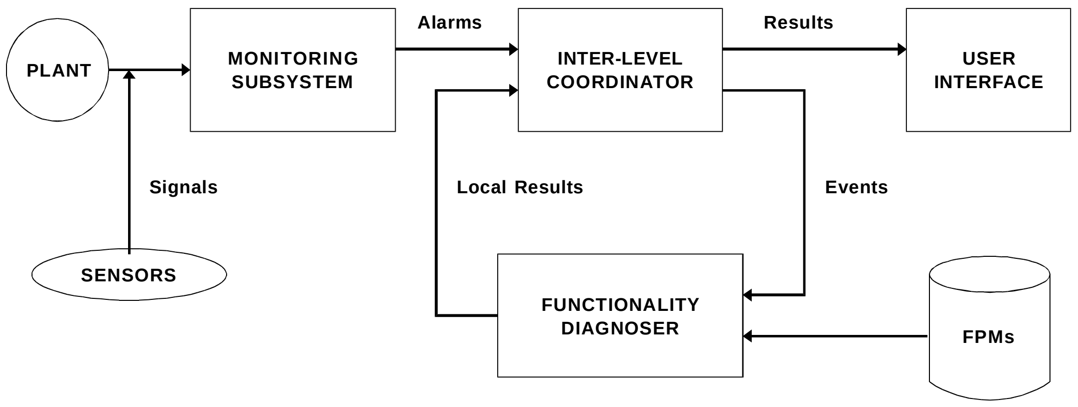


Figure 8. Block Diagram of the Diagnostic System

The ILC looks at the local results and decides which functionality to diagnose next, registers timeout requests and in general, coordinates the diagnosis.

Diagnostic Strategy of FD

The FD (1) receives events, (2) generates hypotheses, and (3) selects some hypothesis(es) according to how consistently they explain the observations. For this it uses the principles of *parsimony* and *structural redundancy*.

Parsimony

A particular hypothesis is said to be consistent with the received events (observations) if the spatial and temporal constraints imposed by the propagation models are satisfied.

If observation errors are possible, not all of the received events have to comply with the spatial and temporal constraints. Consequently, the number of hypotheses that are plausible under the given set of observed events will become larger. In the extreme case when the sensors and/or fault detection algorithms are completely unreliable, any fault hypotheses is plausible, since the observations do not carry information about

The Diagnostic System

The diagnostic system gets triggered whenever an event arrives. An event can be – (1) an alarm ringing, (2) an alarm becoming silent and (3) a timeout on an alarm. These events are defined more precisely in later sections. The diagnoser then has to interpret the event in the context of FPM and identify the fault(s) that caused the event. Since the system is modeled as a hierarchy of separate functionalities, each with its own FPM, the arrival of an event gives rise to the following questions :

- Which functionality's FPM should the diagnoser operate on to identify faults ? That is, which are the functionalities which have an alarm as part of the fault model with which the event is associated ?
- Once the results from that functionality are returned, which other functionalities need to be examined ?
- How are the results from different functionalities combined ?
- How is the search in the hierarchy guided ?

This gives rise to two separate tasks – guiding search and performing diagnosis on FPMs. Thus, the diagnostic reasoning is divided into two modules – Functionality Diagnoser (FD) and Inter-Level Coordinator (ILC). Figure 8 shows the block diagram of the diagnostic system. The ILC receives the external events, provides an interface to the user and controls the diagnostic search. When an event comes in, it invokes the FD to perform diagnosis on the FPM of some functionality(ies) and waits for the results from FD. The FD operates on the FPMs of individual functionalities and generates *local* results. These local results consist of a list of component and/or sensor fault(s) and the next expected alarm event. The results are passed on to the ILC.

The following discussion only describes the core diagnostic routines for the sake of concise presentation. The algorithm that was developed and tested has some other features which will be discussed only briefly. These are :

- Probability : The failure rates of components and the uncertainty in sensor signals does not form part of the discussion. However, in the sections on hypothesis ranking and selection, we briefly discuss how they are incorporated.

- System State : Any real system has many operational states, e.g., `Shutdown`, `Standby` etc. The relevant failures and their interactions differ from state to state. The FPM described in last chapter includes the state dependencies of failures and failure propagations. For diagnosis, effectively, the FPG changes with state. This change is handled by the overall diagnostic system. Here we only describe diagnosis when the system is in any one state.

- Generalized Graphs : The algorithm described here operates on a simple FPG which does not have AND nodes. To include the AND nodes in diagnosis, the reachability analysis is modified.

- Sensor States : We assume that there are no sensor states in the system, i.e., the fault models consist of only failure modes and discrepancies (monitored with an alarm or non-monitored) and interactions between them. Thus, the nodes in the FPG only consist of failure modes or discrepancies. At the end of this chapter, we briefly describe how the sensor states are incorporated in the reasoning.

CHAPTER III

ROBUST DIAGNOSTICS

In this chapter we describe a robust diagnostic system which provides reliable diagnosis in the presence of observation errors. The diagnostic system operates on hierarchical fault models of a system, described in the previous chapter. The input to the diagnostic system are events that arrive asynchronously. The diagnoser interprets the incoming events in the context of FPM and generates hypotheses about the fault(s) in component(s) and/or sensor(s).

The goal of this work was to develop a diagnostic system which :

- Diagnoses multiple faults, and thus, there should not be any underlying assumption of single or multiple points of failures.
- Identifies observation errors.
- Is robust against a large number of sensor failures and degrades gracefully as the number of sensor failures increase.
- Is event-driven and uses incremental non-monotonic reasoning.
- Predicts future events and uses the predictor-corrector principle to revise its hypotheses.
- Restricts the diagnostic search to the relevant parts of the functional hierarchy.
- Identifies loss of model validity in case of large faults and restricts its search to those parts of the hierarchy where the model of the system seems to be valid.
- Uses algorithms of polynomial complexity.

graph $G = (V, E)$. The vertex set V has $n + m + o$ vertices, representing n failure modes, m discrepancies and o sensor states. An edge $e_{i,j} =< v_i, v_j > \in E$ iff the failure represented by v_i propagates and causes the failure represented by v_j. Each edge in E is weighted by two parameters :

1. t_{min}, which is the minimum time for propagation of failure along the edge, and

2. t_{max}, which is the maximum time for propagation of failure along the edge.

Thus, the edges in E represent the causal interactions and the dynamics of the interactions between failures.

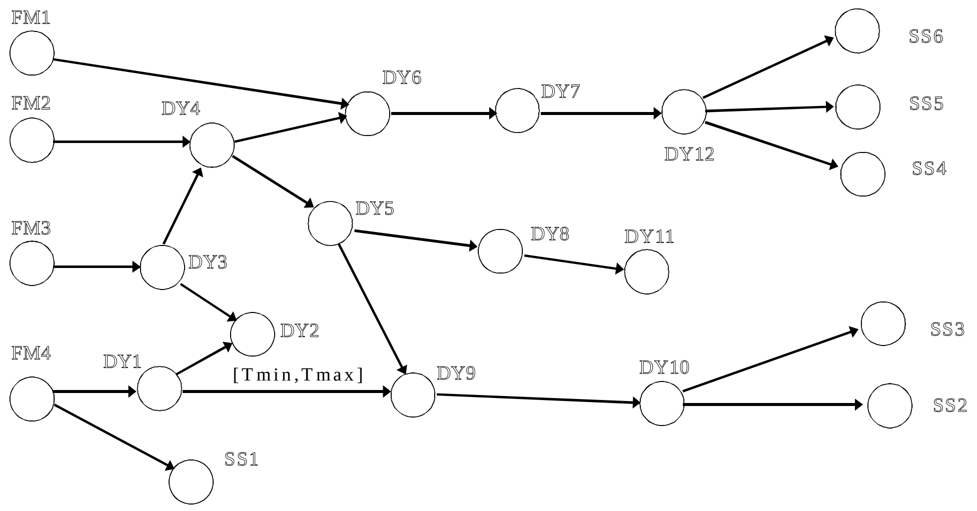


Figure 7. Failure Propagation Graph

means that the LOP is associated with that discrepancy. The ellipses represent the sensors and the dotted lines between the sensors and monitored discrepancies represent the sensor allocation. The association between sensors and sensor states is not shown because such information is implicit in the sensor states.

Failure Propagation Graph

From the entities in FPM and their interactions, we can derive a directed graph which represents the propagation of failures in the system. The vertices of such a graph consist of failure modes, discrepancies and sensor states. The edges in the graph represent the propagation of the failures and correspond directly to the arrows as shown in Figure 6. The graph is called a Failure Propagation Graph (FPG). An example of the FPG obtained from the model shown in Figure 6 is shown in Figure 7.

Let us assume that there are n failure modes, m discrepancies and o sensor states in the system. The diagnostic and diagnosability algorithms operate on a directed

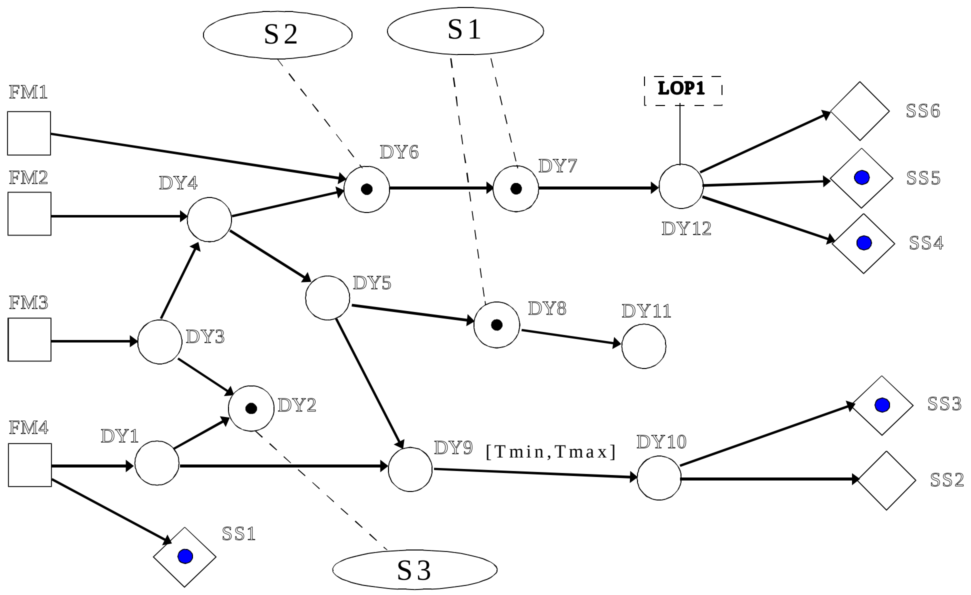


Figure 6. Failure Propagation Model

can be modeled as failure propagations even though, strictly speaking, sensor states are not failures. But it allows us to integrate the causality into fault models more coherently. Also, the sensor delays and polling rates for sensors can be incorporated in the propagation interval. The combined fault model with failure modes, discrepancies, alarms, sensor states etc. is called the Failure Propagation Model (FPM).

The pictorial representation used for failure propagations can now be extended to include alarms and sensor states as shown in Figure 6. As before, the square boxes represent the failure modes of components and the circles represent the discrepancies. The dotted circles are discrepancies that are monitored with an alarm. The empty circles are discrepancies that don't have any alarm explicitly associated them, but they may still be monitored through sensor states, e.g., `DY12`. The diamonds represent the sensor states. Sensor states that are BIT and enabled are represented by a dotted diamond, while the manual and disabled BIT sensor states are shown with the empty diamond. The dashed boxes represent the LOPs. A line from a LOP to a discrepancy disjunction or an arbitrary sum-of-products expression with the sensor states as literals. If there is no modeled LOP associated with a discrepancy, a default conjunction LOP is associated with it, which means that the discrepancy will cause *all* the sensor states. If the sensors are in states such that the LOP associated with a particular discrepancy evaluates to a true value, it can be inferred that the discrepancy exists.

Thus, the monitoring mechanism for a discrepancy can be modeled by specifying the alarm on that discrepancy or by listing the sensor states that the discrepancy impacts along with the associated LOP. The alarm representation does not give any information about how an alarm is generated from the sensors, while sensor state representation does.

A sensor state can have the following attributes :

- Built In Test (BIT)/Manual : A BIT sensor is part of the system and provides data without operator intervention as opposed to manual sensors. Correspondingly, the sensor states can be BIT or manual.

- Enabled/Disabled : Even if a sensor is a BIT, it may not always be turned on (to save power, for example). If the BIT is turned on and is providing signal readings, we say that the sensor is enabled, otherwise it is disabled. We associate the same attribute with corresponding sensor states.

Finally, sensor state allocation represents the set of sensor states that are BIT and enabled. This is similar to alarm allocation and gives the sensor states that are "on-line".

Failure Propagation Model

There is another way of looking at sensor states. Sensor states can be considered to be *primitive* discrepancies and the causality between discrepancies and sensor states discrepancies. These evidences are called alarms, which have been defined in Chapter I. Sensor allocation describes which sensors are used for which alarms.

However, this representation is not comprehensive, in that, it does not allow one to describe exactly *how* the sensor values relate to the alarms and discrepancies. This relationship is modeled by using sensor states. A sensor in the system typically provides continuous valued readings over a wide range. The continuous range of sensor values can be divided into a set of ranges which are called sensor states. When a sensor value is within one of these sub-ranges, the sensor is said to be in the state corresponding to that sub-range, or, that the particular sensor state has "occurred".

For example, a temperature sensor might have a continuous range from $20^o C$ to $90^o C$. Typically, the sensor states for this sensor might be `Temp-Zero`, `Temp-Low`, `Temp-Nominal`, `Temp-High` and `Temp-Full`, representing ranges $20^o C - 25^o C$, $26^o C - 40^o C$, $41^o C - 70^o C$, $71^o C - 85^o C$ and $86^o C - 90^o C$.

There is no prescription as to the number and kinds of states that a sensor can be in. There can be as many sensor states as are required to model the monitoring scheme. A sensor can be in only one of these states at any given time. Whenever the sensor is reading values that correspond to a particular state, the sensor state is said to be active. Since the sensor can be in only one state at a given time, all the other sensor states corresponding to the sensor are said to be inactive.

A discrepancy *causes* one or more sensors to be in particular state(s). Thus there is a causal relationship which goes from discrepancies to sensor states. During diagnosis, by examining the combinations of current states of the sensors, it can be ascertained if a discrepancy exists. The combination is expressed using a Logic Operator (LOP). A LOP is associated with a discrepancy and represents the logical relationship between the sensor states caused by the discrepancy. The LOP could be a conjunction, a

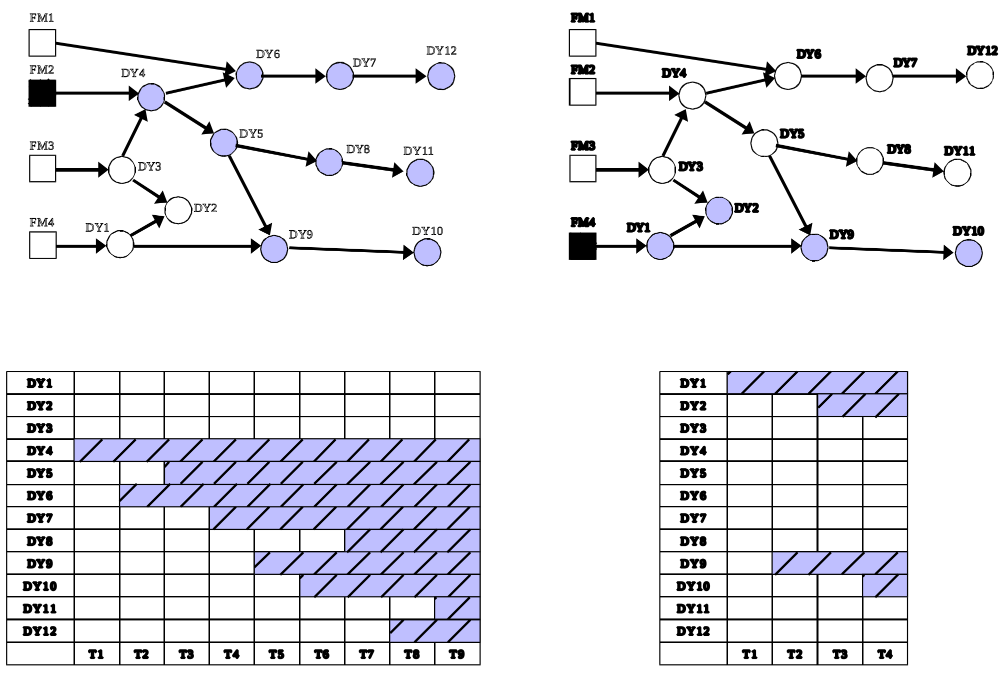


Figure 5. Propagation of Faults

the faults. An inconsistency in the pattern can be used to detect sensor failures. For diagnosability studies, it should be possible to determine the relative importance of a discrepancy for detecting and/or diagnosing a failure mode and the time periods involved.

Failure Monitoring

The observation of anomalies in system behavior is provided by sensors. This section discusses their role in diagnostics and how they are modeled.

Sensors measure the values of physical variables and provide signals. These signals can be used for control or they can be used for monitoring the health of the system. Following the dichotomy of physical and functional structure, we model sensors as physical components that monitor the functional failures.

From the diagnostic point of view, sensors provide evidence about the existence of

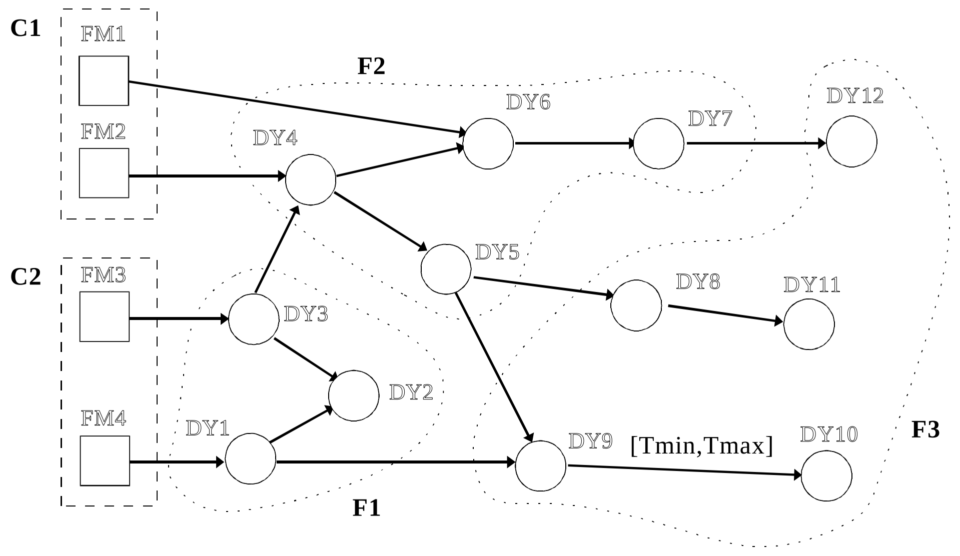


Figure 4. Pictorial Representation of Failure Propagation

consequent failure.

The interactions between failure modes and discrepancies can be represented pictorially, as shown in Figure 4 for a system with two components `C1` and `C2` and three functionalities, `F1`, `F2` and `F3`. The failure modes of `C1` are `FM1` and `FM2` and those of `C2`, `FM3` and `FM4`. The discrepancies of `F1` are `DY1`, `DY2` and `DY3`; of `F2`, `DY4`, `DY5`, `DY6` and `DY7`; of `F3`, `DY8`, `DY9`, `DY10`, `DY11` and `DY12`. The square boxes represent the failure modes while the circles represent the discrepancies. The arrows between the nodes represent failure propagation. All the propagations shown are parameterized with $[t_{min}, t_{max}]$, though it is shown only for the propagation between `DY9` and `DY10`.

Figure 5 shows the propagation of two faults in the system. The figures at the top show the discrepancies (dark circles) that will occur at some time as the faults propagate. The tables show the time line for these discrepancies. The time intervals shown in the tables are not necessarily equal. They just represent the fact that the discrepancies will occur at different points in time as the fault propagates. For fault diagnosis, the spatial and temporal pattern of discrepancies can be used to isolate

Failure Propagations

The occurrence of a failure mode causes one or more discrepancies in the system. These discrepancies usually appear as out of range physical variables. For instance, an output valve of the pump assembly, when stuck closed, causes the output flow rate to drop and the internal pressure to build up. Because the physical variables are related to each other (through laws of thermodynamics, for example), an out of range physical variable may cause some more physical variables to go out of limits. For example, a rise in temperature inside a gas container will cause the internal pressure to build up. Further, an out of range physical variable can cause a fault in a physical component, e.g., high pressure may lead to a leak in a pipe.

This phenomenon of causation between failure modes and discrepancies is called failure propagation. We say that the antecedent failure (failure mode or discrepancy) propagates to the consequent failure. Following the chain of antecedent and consequent failures, we can enumerate the failure propagation paths starting from any given failure. The failure propagation paths are the mechanisms by which a fault in one part of the system can cause failures to occur in a "remote" part.

Due to the dynamics of the system, the failure propagations do not take place instantaneously; instead they take a finite amount of time. For example, a heater broken high will take some finite amount of time to cause the temperature to rise beyond acceptable limits. Further, in any real system, the time taken can not be specified exactly. However, the *minimum* and *maximum* time that a propagation takes can be known fairly accurately, allowing us to use a time interval to express the uncertainty.

To incorporate the dynamics into fault models, each failure propagation is parameterized with a time interval $[t_{min}, t_{max}]$, called propagation interval, which gives the minimum and the maximum time that the antecedent failure can take to cause the

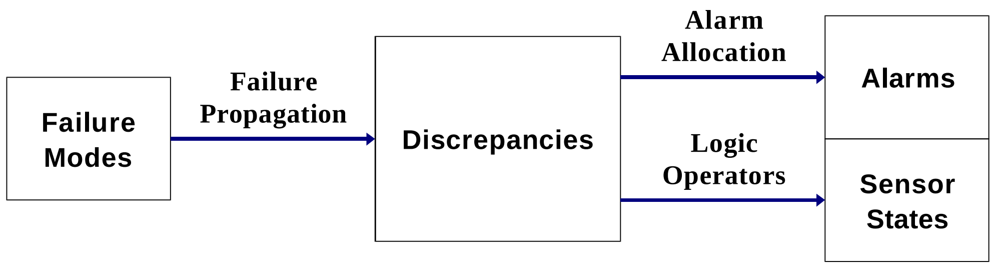


Figure 3. Fault Model Aspects

breakdown of the system in this manner can continue down to any level depending upon the available knowledge and diagnostic requirements.

The hierarchies are linked together at each level, i.e, the relationships between the functionalities and physical components (including the fault models) are described at each level. A one-to-one correspondence between the levels in the hierarchies is not necessary. The linkages between functional and physical models may go from any level to any level.

Fault Models

There are three aspects to the fault models :

1. Failure modes of physical components.

2. Discrepancies in functionalities.

3. Alarm and sensor states.

Figure 3 shows the relationships between the three aspects. The interactions between failure modes and discrepancies are captured by failure propagations. The alarm allocation and Logic Operators (LOPs) describe how the discrepancies are monitored. These are discussed in the following sections.

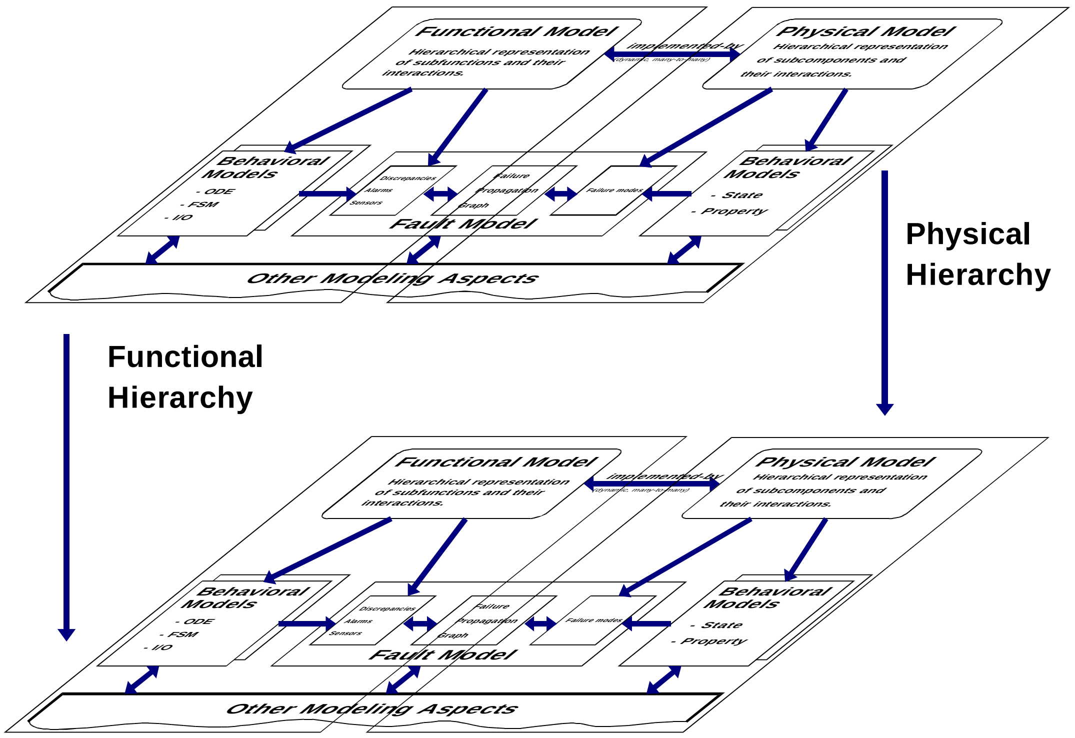


Figure 2. Hierarchical Modeling

is discussed in the later sections.

Hierarchical Models

Hierarchical decomposition, commonly used to deal with complex systems, is also employed by the modeling paradigm used in this thesis. Not only is a system modeled by breaking it down into multiple aspects, it is simultaneously decomposed into a hierarchy, as shown in figure 2. The physical models consist of hierarchy of components. At each level in the hierarchy, the fault characteristics for the component(s) are also described. Similarly, the functional models consist of a hierarchy of functionalities and there are fault models for each functionality in the hierarchy. The hierarchical

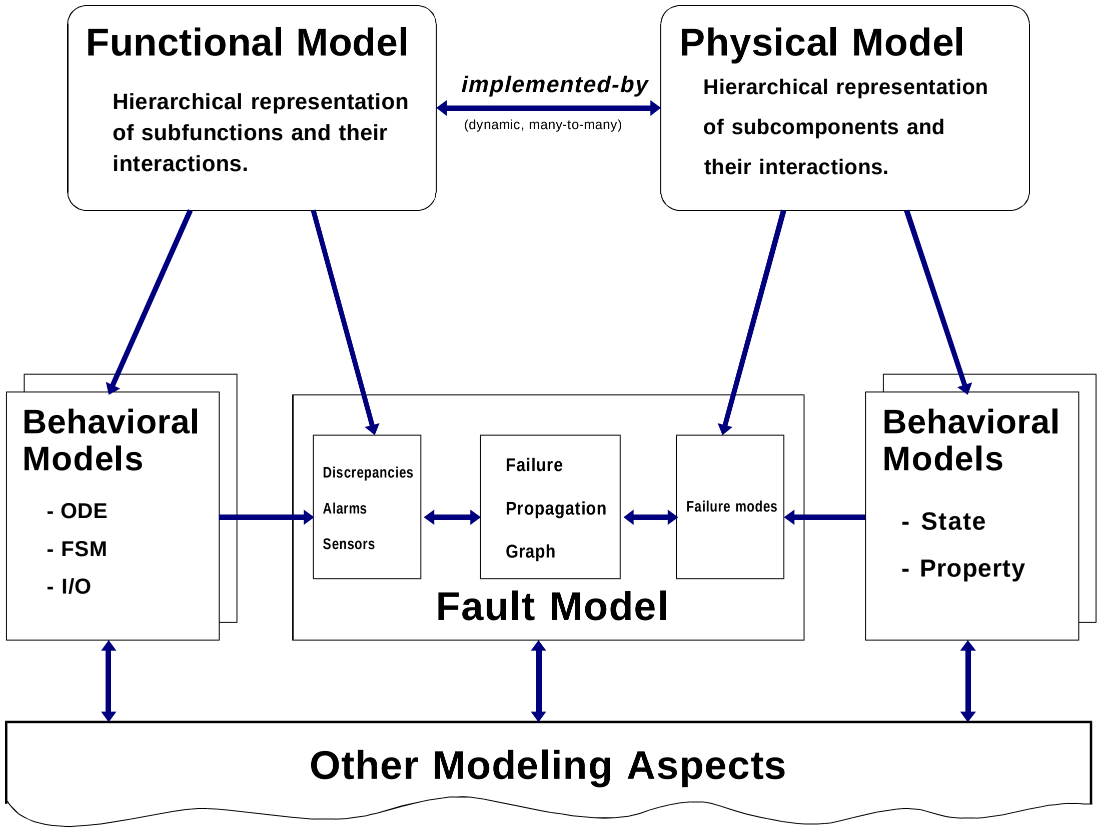


Figure 1. Multiple Aspect Modeling

structure. Each of these has “sub-aspects”, which will be discussed below.

Physical structure of a system consists of physical components and their assemblies. The interesting features (relevant sub-aspect) of components are their failure modes and properties such as failure rate, whether the component is a sensor, what kind of tasks the component can be used for, etc. The occurrence of a failure mode is manifested as some abnormal behavior of the system. Since the abnormal behaviors relate to the function that a system performs, they are modeled in the context of functional structure.

Functional structure consists of a set of sub-systems. Each of these sub-systems is called a functionality and is characterized by the task it performs. The behvioral description of a functionality can include such aspects as ordinary differential equations, finite state machines etc. However, from the point of view of diagnosis, the ways in which a functionality can fail to perform its task is important. A failure of a functionality is represented by a discrepancy (abnormal behavior), which is part of the failure sub-aspect of the functionality. The observation of discrepancies is provided by the monitoring scheme used, which is the other sub-aspect related to diagnosis.

The breakdown of a system in terms of its aspects is shown in figure 1. The mapping between the two primary aspects describes the way that physical components are used to implement the tasks in a functionality. This mapping between the functions and components is dynamic and many-to-many – one function may use different components at different times; one component may be used for different functionalities. Behavioral models are associated with the physical components and functionalities. For a component, the behavioral models consist of state and properties like failure rate etc. For functionalities the behavioral models can consist of Ordinary Differential Equations (ODE), Finite State Machines (FSM) etc. The fault models describe the relationships and the interactions between the failure-related sub-aspects. This

The concept of structure of a system is fairly intuitive. One tries to identify a suitable breakdown of the system into interacting blocks such that the failures in the blocks and their interactions can be abstracted. The granularity of the decomposition can be guided by the amount of knowledge available and by the requirements. For example, it may suffice to know that a heat exchanger is faulty. There may be no need to find out exactly which component in the heat exchanger assembly has failed, perhaps because the repair action involves replacement of the whole heat exchanger. In this case, the heat exchanger assembly can be considered as a single block.

There are two types of structural descriptions commonly used by researchers in the field :

1. Physical structure, which describes the interconnections of physical components.

2. Functional structure, which describes a system in terms of the tasks that its parts perform.

Gallanti et al. [14] have described a method to identify deviations in control and design parameters of a continuous, dynamic system using the system equations. Yu and Biswas [13] relate the system equations to structure by extendeding the above methodology to relate the deviant parameters to component faults. The work described in [15] models the structure of a system in terms of its physical components and attaches functional descriptions to each component. Qualitative functional models of the kind described in [20, 21] etc. also use physical structure. Hamscher [12] describes a system in terms of its functional as well as physical structure.

In the modeling paradigm used in this research, we model both the physical and functional structure of a system with a clear distinction between them. This method of modeling a system from more than one distinct viewpoint is called multiple aspect modeling. The two primary aspects are the physical structure and the functional of the system. The connectionist models of the kind described in [4] also suffer from the same drawbacks. Further, the probability values used to describe the likelihood of one fault causing another do not always reflect the real-world situation. The reason is that a good statistical evaluation of this causality is usually not available. The uncertainty in the *time* taken for faults to propagate is usually better understood. However, the above models do not capture this information.

Fault models using diagnostic dictionaries (the kind used in [8, 22] etc.), provide a simple mapping form faults to effects. The effects of a fault, once all the propagations have taken place and the system has settled down, are listed. Thus the temporal relationships between faults and the dynamics are lost.

The temporal relationships and the dynamics are captured in the fault models using directed graphs (as in [9, 11]). In this thesis, we use a similar representation which is described in the upcoming sections.

Multiple Aspect Hierarchical Modeling

For modeling large, complex systems, two techniques have been commonly used :

1. Focus on selected features.

2. Use hierarchical breakdown.

Multiple Aspects

For sensor-based diagnosis, the features that are of interest are :

1. Structure.

2. Behavior under fault conditions.

3. Specify the causality between the failure modes and discrepancies.

Functional models (qualitative or analytical) are quite useful in detecting faults. Given the current inputs to the system, the models can be used to generate the expected behavior and from there, to detect any discrepancies. Fault *diagnosis*, however, proves to be quite difficult using these models (as also with component connection models, discussed above). This is primarily because the diagnostic search space can become combinatorially large when (1) hypothesizing about possible faults and (2) selecting plausible hypothesis(es) that are responsible for the observed discrepancies. The problem is further compounded when observation errors need to be diagnosed as well.

The large diagnostic search spaces when using functional models arise out of the attempt to reason about *abnormal* behavior of a system using models that describe the behavior under *normal* conditions. Except in some cases, such attempts have not been successful. To reason about faults, it makes sense to use fault models, even though they only give an approximation to the actual behavior.

Fault models help in diagnosis by reducing the diagnostic search space. Hypothesis generation is straight-forward – just consider all the failure modes that could have caused the discrepancies. Diagnosing with a single fault assumption is simple. Diagnosing with multiple faults and/or sensor failure assumption can possibly result in a large number of combinations of faults to be examined. In this case, some reasonable heuristics can be used which are derived from the fault characteristics.

In this thesis, we use fault models. Many different representations have been used by researchers to describe the interactions between the failure modes and discrepancies.

Rule-based fault models lack depth and clarity and fail to capture the dynamics like in [7], are not useful for modeling large systems. Fault diagnosis using component connection models is difficult. The reasons for this are the same as in the case of the qualitative functional models, described below.

To address the complexity in most engineering systems, researchers have used *qualitative* models. Qualitative models divide the (possibly infinite) space of all possible behaviors of a system into a finite set of ranges. Qualitative *functional* models do this by dividing the process variable space into "ranges of interest" and using qualitative physics to generate the behavior of a system. They have met with varying degrees of success in analyzing and predicting the behavior.

Fault models (fault trees, cause-consequence diagrams, diagnostic dictionaries etc.) describe system behavior when faults are present. These models use qualitative representation of faults, discrepancies and their interactions. This is done in the following manner :

1. Discretize the failure space of components and specify the failure modes of components thus discretized. For example, a failure of a heater when it starts overheating might be represented with just two failure modes, "Broken-high" and "Broken-high-high", by declaring two cutoff levels for these two failure modes. Defining failure modes for components in this manner is a standard engineering practice and thus, the information about failure modes is readily available.

2. Discretize the behavioral space of the system into sets of normal and abnormal behaviors. The abnormal behaviors are represented by discrepancies. For example, corresponding to the failure modes above, we may specify two discrepancies – `Temp-High` and `Temp-Full`. The information about discrepancies is also usually part of the readily available engineering design data.

Functional Models vs. Fault models

For diagnosis, two kinds of modeling paradigms have been commonly used to describe the behavior of engineering systems – (1) functional models and (2) fault models. Functional models are used to describe the "correct" behavior of a system, i.e., the behavior when no faults are present. Fault models describe the behavior when faults are present. In this section, the two paradigms are examined for their usefulness in diagnosis.

Heterogeneity

Developing a common paradigm for functional models of the different kinds of sub-systems in a heterogeneous system proves to be quite difficult. Previous work on model-based systems has shown that suitable paradigms for different kinds of systems tend to be quite different. For example, a digital circuit is described by the connections between modules and the logical relationships between the input(s) and output(s) of the modules. Such a represention can not be used for, say, chemical processes, which are usually described by analytical equations.

Fault models offer a way out, since it is not necessary to model the "correct" behavior. Instead, only the ways in which these systems can fail and the manifestations of faults are of interest. This allows the development a general framework for modeling most engineering systems.

Complexity

The next issue to be addressed is that of complexity of systems. Analytical models (state-space representation, differential equations etc.) prove to be useful for diagnosing faults in small, stable systems but not in large and complex systems. Component connection models with analytical relations between input and outputs of components

- Complexity : The system consists of a large number of components and sub-systems with complex interactions between them.

- Non-local effects of a fault : Because of the interactions between the sub-systems, a fault in one part of the system can effect the functioning of another "remote" sub-system (propagation of faults).

- Dynamic : Because of the dynamic nature of the processes in the system, the interactions between sub-systems take a finite amount of time. This means that faults will also take a finite amount of time to effect the functioning of the system. This introduces a temporal relationship between a fault and its manifestations.

The goal of the work in this thesis was to develop a diagnostic system that localizes faults spatially as well as temporally. In other words, we would like to know which component failed and *when* it failed. Knowing the time of occurrence of a fault can be very helpful in fault recovery and prevention of catastrophic failures. Also, as is explained later, temporal reasoning helps in localizing faults spatially. Hence, the modeling paradigm used should include a description of temporal characteristics of the system.

Given the above goal and the characteristics of systems to be modeled, there are two basic questions regarding the choice of models :

1. Functional models or fault models ?

2. What type of (functional or fault) models ?

CHAPTER II

MODELING CONCEPTS

In order to diagnose faults in a system and to perform diagnosability studies, we need a modeling paradigm which allows us to express the system structure, components and behavior in the form of comprehensive, well defined models. This chapter describes the modeling paradigm that is used in this research.

Domain Characteristics

The paradigm used for modeling a system has to be guided by the characteristics of systems to be modeled and by the application. Using the same kind of models for nuclear plant diagnosis and medical diagnosis is not feasible. This is simply because the interactions between faults (diseases) and their manifestations (symptoms) have quite different characteristics in the above two cases. Instead, a modeling paradigm should be developed which is focused on a class of systems with similar characteristics.

The application addressed in this thesis is that of diagnosis. Thus, system characteristics are examined from diagnostic point of view. The application domain consists of engineering systems like power plants, chemical plants, airplanes, etc. These systems have some common characteristics that strongly influence the selection of modeling paradigm :

- Heterogeneity : A large engineering system employs many different kind of sub-systems. For example, there are mechanical systems, chemical processes, controllers, digital systems, etc.

2. Develop algorithms which will operate on the fault models, analyze them and evaluate the diagnosability of the system using the metrics.

3. Develop algorithms which will generate alarm allocation advice, given the desired values for the diagnosability metrics.

4. Develop a tool which will :

   - allow the user to perform diagnosability experiments by changing alarm allocation and sensor allocation.

   - allow the user to specify the desired diagnosability and ask for alarm and sensor allocation suggestion.

   - generate comprehensive reports about the diagnosability of the system.

Thesis Outline

Chapter II discusses the modeling paradigm used in this research. Chapter III describes the robust diagnostic system, its strategy, structure, algorithms and tests. Chapter IV discusses the metrics and algorithms developed for diagnosability. Chapter V concludes the dissertation.

i.e., when a discrepancy is present). The diagnostic algorithm should be able to identify such observation errors and incorporate the information into generation and ranking of fault hypotheses.

2. Because the system under diagnosis is a dynamic system, faults will propagate through the system over time, giving rise to “event propagations”. Hence, the diagnostic algorithm should handle temporal relationships among events.

3. The event-driven nature of the diagnostic algorithm gives rise to few more requirements :

    - The algorithm should use non-monotonic reasoning.
    - The algorithm should be capable of incremental diagnosis.
    - The algorithm should be able to predict future events.
    - The algorithm should revise its hypotheses when a predicted event occurs (or does not occur).

4. Another requirement of the diagnostic algorithm is that it should be able to identify loss of model validity and should take it into account when diagnosing.

5. Lastly, the algorithms should be of polynomial complexity in order to ensure scalability.

Diagnosability Studies

We identify the following objectives for diagnosability studies :

1. Define metrics to quantify the ability to detect, diagnose and predict faults in the system.

faulty operation – (1) Discriminability, (2) Accuracy and (3) Time Lag. The metrics, however, capture the notion of *detectability* more than that of diagnosability. The discriminability score together with the time lag score tells us whether a particular sensor will provide any evidence in case of a fault, and if so, how soon. But it does not tell us if the evidence can be used effectively to *diagnose* the fault.

Objectives

In the previous section, the problems and issues associated with sensor-based diagnosis were identified and discussed. In particular, two areas for research have been identified – robust diagnostics and diagnosability studies. The research described here deals with these two issues. In this section we will describe the specific objectives of the research.

Modeling

In order to diagnose faults in a system and to perform diagnosability studies, we need a modeling paradigm which allows us to express the system structure, components and behavior in the form of comprehensive, well defined models. The models should describe the faults in the system, their manifestations and the role sensors play in monitoring and diagnosing the faults.

Robust Diagnostics

A diagnostic algorithm should be developed which is robust against observation errors. For this, the following requirements are identified :

1. The algorithm should be able to identify erroneous observations. The observation errors can manifest themselves as false alarms (alarm ringing when there is no discrepancy present) and/or silent alarm (alarm *not* ringing when it should,

involves developing test vectors that will exercise the circuit in a way to ensure that the circuit will work properly with all possible inputs.

Tanaka discusses diagnosability and optimal sensor allocation for linear discrete dynamic systems in [17]. He uses the state-space representation of a system and has defined indices for *detectability* of a fault and *separability* of faults. These indices can be computed from a given sensor allocation and can also be used to determine the optimal sensor allocation to achieve a certain detectability and separability. However, this method is good only for small, stable systems whose state-space description is available.

Scarl proposes a method to minimize sensor costs for device-centered, model-based systems [18]. A diagnosability criterion, which expresses the permissible ambiguity in the diagnosis, is defined for every component in the system. Then a Minimal Sensor Set (MSS) for each component set is derived by considering sets of all possible combinations of sensors, in the increasing order of cardinality of the sets. Once the MSS for each component are found, a global minimization is performed to come up with one MSS for the whole system that will keep the cost of sensors minimum. The drawbacks of this method are that (1) it is dependent on the diagnostic methodology and (2) it is computationally expensive, making it useless for large systems.

The work done by Chang et al. [8] uses diagnostic dictionaries to model the failures and computes the t-fault diagnosability (up to t faults can be diagnosed correctly or not) of the faults in a system. However, they ignore the dynamics of the system and do not address the issues related to the detection and prediction of faults.

Chien et al. have presented [19] an approach to evaluating sensor placement. They simulate the system behavior using analytical models to predict the signal values read by sensors under normal and faulty operation. They have defined three metrics to evaluate a sensor on the basis of its ability to distinguish normal operation and

Diagnosis in this system proves to be quite difficult, particularly in the presence of sensor failures. This is because the diagnostic algorithm has to search a large space of possible faults in order to trace the observed malfunctions to their original cause. The presence of sensor failures makes the search space larger.

Diagnosability

As mentioned in the previous section, diagnosability of a system is a measure of how effectively the faults can be detected and diagnosed. Two factors have a considerable effect on this – (1) the number of sensors and (2) their locations, i.e., the physical variables they measure and the signals they provide to the diagnostic system. The sensor allocation in the system describes the number and location of sensors. This, in turn determines the alarm allocation, i.e., the discrepancies that can be observed.

Also, as mentioned previously, the number, weight, cost etc. are important design concerns. Reducing the number of sensors may result in substantial savings in weight and cost. Having a large number of sensors, on the other hand, may result in faster, more accurate fault detection and diagnostics. Thus, we need to arrive at an optimal sensor allocation which does not sacrifice diagnosability of the system. Sensor allocation requires the understanding of the effects of sensor placement on basic characteristics of diagnosability. The ultimate goal is to minimize the sensor cost while maintaining the required level of diagnosability.

There has been considerable work done on design of digital circuits for testability [16]. The domain, however, is quite different from that of dynamic system. Also the tasks – testing and diagnosing a system – though similar, have important differences. Testing of digital circuits, whether a go/no-go testing or diagnostic testing, is done prior to putting the system in operation. Designing a circuit for testability is known, then under certain conditions, the state of the system can be calculated from the observed variables, which in turn can be used to derive the expected value of the other, related physical variables. Then, by comparing the computed values to the actual values read by the sensors, the health of sensors can be determined. The collection of papers in [1] addresses issues related to analytical redundancy method for sensor failure detection. The main drawback of this method is that the analytical model of the plant has to be available. Additionally, most of the results have been developed only for linear systems. These limitations make the method useful only for smaller, stable subsystems.

Qualitative methods have been used for diagnosis of complex systems which can not be modeled analytically. Association-based systems, a class of qualitative systems, emulate a human expert diagnosing the fault(s) [2]. Fox et al. [6] address sensor failures in a rule-based system called PDS by performing a “meta-diagnosis” of sensors before the actual diagnosis and computing the level of confidence in the evidence provided by the sensors. Chandrasekharan et al. describe a system which diagnoses faulty sensors by pattern matching on diagnostic expectations. Sensor failure detection has also been tried with neural networks [5].

The computational complexity of association-based systems, however, makes their usefulness limited. Such systems typically work with large fault-symptom knowledge bases. When observation errors need to be diagnosed along with component faults, there can be a combinatorial explosion in the number of fault-symptoms associations to be stored in the knowledge base, and this makes the diagnosis computationally prohibitive.

Scarl describes a methodology to diagnose sensor failures using functional models of a continuous time systems [7]. A system is represented as a network of component with functional relationships between the inputs and outputs of each component.

Problem Description

Robust Diagnostics

The sensory input signals provide information to diagnostic programs. When the information provided by the signals indicates incorrect system behavior, the diagnostic program should explain the incorrect behavior by identifying the component(s) in the system that have failed. In other words, the diagnostic program explains the *observed* system behavior in terms of one or more hypotheses about the state of the components in the system.

Since the diagnoser has to locate the fault(s) by interpreting the observations (information provided by sensor signals), the validity of diagnostic results depends upon the reliability of observations. The observations can be erroneous because (1) sensors can fail and thus provide the diagnostic system with wrong data values, (2) even if the sensors are working correctly, the signals read by them could be interpreted wrongly (in model-based systems, this would be a modeling error), and, (3) in case of a major fault, the assumptions about the system may become invalid (in model-based systems this means a loss of model validity).

One of the approachs to tackle the problem of unreliable sensors has been to use hardware redundancy. Signals from three or more sensors measuring the same physical variable are compared for consistency. However, in many systems such as space vehicles, due to the very high cost, weight and space penalties, the extensive use of hardware redundancy is prohibitive.

Another approach has been to use *analytical redundancy* for detecting sensor failures. Dynamic systems represent a complex set of constraints among the observable physical variables in the system. This means that there is an inherent redundancy in the system, which is often called analytical redundancy [1]. If the model of the plant and the diagnostic results can be incorrect and/or incomplete. A robust diagnostic system should be able to handle observation errors. By a diagnostic system that can handle observation errors we mean a system that will, ideally, be able to interpret the (possibly erroneous) observations properly and come up with the correct and complete diagnostic result. At worst, the diagnostic system should degrade gracefully as the number of observation errors increases. An important issue of the diagnosis of large scale systems is that of efficiency of the diagnostic algorithm. An algorithm of exponential complexity is not scalable. Thus the diagnostic algorithm should be able to handle observation errors and also be of polynomial complexity.

Alarm allocation means the assignment of alarms to discrepancies. A given alarm allocation describes the set of discrepancies which are being observed for occurrence. A discrepancy that is not being observed for occurrence will not have an alarm assigned to it. One alarm is allocated to only one discrepancy. *Sensor allocation*, although a similar concept, is more complex than alarm allocation. Sensor allocation is the assignment of sensors to alarms. What makes it complex is the fact that, because of inter-relationships between the physical variables in the system, the value read by one sensor may be used to generate more than one alarm and, also, an alarm may be generated using values from more than one sensor.

Diagnosability of a system, in its most general sense, means that property of the system which allows the faults in the system to be detected and diagnosed in a timely manner. In order to characterize the diagnosability of a system, one needs to develop some criteria or metrics, which express the property of diagnosability in a reasonable and coherent manner.

term “failure” is used, it should be clear from the context what is meant, otherwise it will be explicitly stated.

Fault detection means determining that there is something wrong with the system. Usually, faults are detected by observing the values of physical variables in the system and then deducing that one or more discrepancies exist, which implies that one or more faults in some component(s) have occurred. Once a discrepancy is observed, i.e., a fault has been detected, it needs to be diagnosed. Fault diagnosis means identifying the faults, i.e., locating the physical components that are not functioning properly. The diagnostic result consists of a set of one or more components in the system that are believed to be faulty.

Correctness of the diagnostic results means that only those components that are actually faulty are identified as faulty and no healthy component is part of the diagnostic result. Completeness of results means that *all* the components that are faulty are indicated.

Sensors are those components in the system that are used to measure values of physical variables like temperature, pressure, etc. The signals generated by these sensors can be used for control and monitoring. The fault detection algorithms can use these signals to determine whether a discrepancy exists or not. An alarm is an indication that a discrepancy has occurred. The alarm is said to ring when a discrepancy is observed. A discrepancy which has an alarm assigned to it is called a monitored discrepancy, while the ones without alarms are called non-monitored discrepancies. An alarm may become silent after ringing for a while (the discrepancy may not exist after a while, possibly because of a repair action). The ringing and silencing of alarms introduce events that trigger the diagnostics. All the events have a time stamp associated with them, signifying the time that the status of the discrepancy changed.

As mentioned in the previous section, sensor failures can lead a diagnoser astray example, in a space bound system, the number, weight, size, reliability and cost of the sensors are important design concerns, and keeping the number of sensors at the lowest possible level is very important.

What's needed is a tool to analyze the system in terms of its diagnostic characteristics. Such a tool could be used at design time to determine the relative importance of sensors from the point of view of diagnostics. It would also help in choosing the optimal allocation of sensors to insure diagnosability of the system while keeping costs at a minimum.

Terminology

In this section some of the terms and concepts used in diagnosis and diagnosability studies are defined and discussed briefly. For a detailed description of modeling concepts and definitions, please refer to Chapter II.

A component is part of the physical hardware assembly of the system. It may refer to a single component like a pipe or an assembly of components, e.g., a pump assembly. A failure mode is a failure of a component. A component may have more than one failure mode, i.e., a component may fail in more than one way. When a component malfunctions, we say that a failure mode of the component has occurred. The occurrence of a failure mode is called a *fault*. A component which exhibits one or more failure modes is referred to as a faulty component. A component which is not malfunctioning (none of the failure modes have occurred) is called a healthy component.

A fault in a component will produce anomalies in system behavior. These anomalies are called discrepancies. A discrepancy may be immediately observable or it may go unobserved depending upon sensor allocation and fault detection algorithms used.

The generic term for a failure mode, a fault and a discrepancy is *failure*. When the

CHAPTER I

INTRODUCTION

The degree of automation in large, complex systems such as chemical plants, power generation and aerospace systems has been steadily increasing in the recent past. Automated diagnosis and control forms a necessary part of these systems. Accurate and speedy diagnosis of faults is an important factor in maintaining their health and continued operation and in reducing of repair and recovery time. These systems employ a large number of sensors to read the values of physical variables. The signals provided by these sensors constitute the data that the control, monitoring and diagnostic algorithms use. The important role that these sensors play in the systems gives rise to some issues regarding their reliability and cost.

The first issue is that of reliability of sensors. Since the signals from sensors are used for control, monitoring and diagnosis, it follows that if a sensor fails and provides incorrect readings, the system performance will be adversely affected. It might result in an incorrect control action leading to a degradation in system performance. It might result in incorrect diagnosis, resulting in unnecessary repair and loss of productivity while the system is down for repairs. In the worst case, a wrong control action or wrong diagnosis may result in a complete loss of system. Thus, being able to identify failures in sensors will result in considerable improvement in safety and performance.

The second issue is that of sensor allocation for diagnosability. The diagnostic functions of a system are best performed when they have an ample amount of sensor readings, which allows them to compute the system state. However, it is not always possible to put a sensor on every important physical variable in the system. For

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

Let me begin with an expression of thanks and deep sense of appreciation for all the help, advice, guidance and support given by my advisor, Dr. Janos Sztipanovits. His insight, criticism, support and the "can be done" attitude has been of immense value to me.

Thanks to my sponsor, the Boeing Co., for continued financial support. Thanks are also due to James Ray Carnes, Dr. Byron Purves, Dr. David Throop and Al Underbrink, Jr., all at Boeing Co., for their critical evaluation of my work, suggestions for improving it and help in testing the ideas.

Thanks are due to my other committee members, Dr. Gautam Biswas, Dr. Benoit Dawant and Dr. George E. Cook for their help and support.

Two people that have aided me considerably towards implementing and testing the ideas in this thesis are Dr. Gabor Karsai and Dr. Csaba Biegl. Without their work in providing the framework for implementing the ideas and constant help, life would have been considerably more trying.

Thanks to Dr. Samir Padalkar for his advice and help. He laid the foundation on which most of the work described in the thesis is built.

I would like to thank Ben Abbott for his friendship and all the help he has given me over the years – work related and otherwise.

Thanks to my fellow graduate students and other friends, particularly Maggie, for their help, support and the good times.

Last, but most decidedly not the least, my deep gratitude to my family for their love and understanding, even though they would prefer that I wasn't halfway across the world from them.

SENSOR-BASED DIAGNOSIS OF DYNAMICAL SYSTEMS

By

Amit Misra

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

May, 1994

Nashville, Tennessee

Approved:	Date: