INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17” x 23” black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6” x 9” black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
A knowledge-based system architecture for diagnosis and sensor validation in chemical process plants

Shum, Sik Kwan, Ph.D.

The Ohio State University, 1989

Copyright ©1989 by Shum, Sik Kwan. All rights reserved.
A KNOWLEDGE-BASED SYSTEM ARCHITECTURE
FOR DIAGNOSIS AND SENSOR VALIDATION
IN CHEMICAL PROCESS PLANTS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of the Ohio State University

By

Sik Kwan Shum, B.S., M.S.

The Ohio State University

1989

Dissertation Committee:  Approved by
James F. Davis  Adviser
Don W. Miller
Won-Kyoo Lee
Department of Chemical Engineering
Copyright by
Sik Kwan Shum
1989
To My Mother
My Wife and
Memory of My Father
ACKNOWLEDGMENTS

I would like to thank my adviser, Dr. James F. Davis, for his guidance and valuable suggestions during the course of my research and preparation of the dissertation. My thanks also to Dr. Don W. Miller and Dr. Won-Kyoo Lee for their comments on the dissertation. I would also like to thank Dr. B. Chandrasekaran and members of LAIR for their unique perspective on problem solving. Thanks to members of our AI in Engineering group for the many helpful suggestions on my research and the enjoyable working environment. Last, but certainly not least, thanks to my wife Grace for being caring and supportive, making the years of graduate school more pleasant.

This research is supported by the National Science Foundation under Grant No. CBT-8703746 and the Ohio State University Grant Program.
VITA

May 12, 1961 .............................. Born - Hong Kong

December, 1983 ........................ Honors B.S., Chemical Engineering
Pennsylvania State University
University Park, Pennsylvania

June, 1986 ............................... M.S., Chemical Engineering
Ohio State University
Columbus, Ohio

1986-Present ........................  Graduate Research Associate
Department of Chemical Engineering
Ohio State University
Columbus, Ohio

PUBLICATIONS AND PRESENTATIONS


**FIELDS OF STUDY**

Major Field: Chemical Engineering

Studies in development and application of knowledge-based problem-solving techniques to chemical process plants

Adviser: James F. Davis
# TABLE OF CONTENTS

DEDICATION ................................................................. ii
ACKNOWLEDGMENTS ...................................................... iii
VITA ................................................................................ iv
LIST OF FIGURES .................................................. ix

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION ......................................................... 1</td>
<td></td>
</tr>
<tr>
<td>1.1 CHARACTERIZATION OF DIAGNOSIS IN PROCESS PLANTS</td>
<td>1</td>
</tr>
<tr>
<td>1.2 OVERVIEW OF PROBLEMS ADDRESSED</td>
<td>4</td>
</tr>
<tr>
<td>1.3 RESEARCH FOCUS</td>
<td>6</td>
</tr>
<tr>
<td>1.4 DISSERTATION ORGANIZATION</td>
<td>9</td>
</tr>
<tr>
<td>II. DIAGNOSTIC FRAMEWORK ............................................ 10</td>
<td></td>
</tr>
<tr>
<td>2.1 GENERIC TASK VIEWPOINT</td>
<td>10</td>
</tr>
<tr>
<td>2.2 OVERVIEW OF DIAGNOSTIC FRAMEWORK</td>
<td>11</td>
</tr>
<tr>
<td>2.3 CLASSIFICATION</td>
<td>15</td>
</tr>
<tr>
<td>III. FUNCTIONAL DECOMPOSITION ...................................... 18</td>
<td></td>
</tr>
<tr>
<td>3.1 BASIS FOR THE CLASSIFICATION HIERARCHY</td>
<td>19</td>
</tr>
<tr>
<td>3.2 FUNCTIONAL DECOMPOSITION STRATEGY</td>
<td>25</td>
</tr>
<tr>
<td>3.3 MALFUNCTION HIERARCHY</td>
<td>32</td>
</tr>
<tr>
<td>3.4 EXAMPLE FUNCTIONAL DECOMPOSITION</td>
<td>36</td>
</tr>
<tr>
<td>3.5 CONCLUSIONS</td>
<td>42</td>
</tr>
<tr>
<td>IV. SENSOR VALIDATION .................................................. 43</td>
<td></td>
</tr>
<tr>
<td>4.1 SENSOR VALIDATION PROBLEM</td>
<td>43</td>
</tr>
<tr>
<td>4.1.1 Impact of Sensor Failure</td>
<td>44</td>
</tr>
<tr>
<td>4.1.2 Modes of Sensor Failure</td>
<td>46</td>
</tr>
</tbody>
</table>
4.2 REVIEW OF SENSOR VALIDATION METHODS .......................... 47  
 4.2.1 Independent Checks ................................................ 48  
 4.2.1.1 Limit Checking ..................................................... 48  
 4.2.1.2 Hardware Redundant Sensor Comparison ................... 49  
 4.2.2 Model Correlations .................................................. 49  
 4.2.2.1 Analytical Redundancy .......................................... 50  
 4.2.2.2 Parity Space Checking .......................................... 50  
 4.2.2.3 Measurement and Nodal Tests ............................... 51  
 4.2.2.4 Generalized Likelihood Ratio ................................. 52  
 4.2.3 New Techniques ..................................................... 53  
 4.2.4 Existing Validation Method in Classification .................. 54  
 4.2.5 Weaknesses of Conventional Methods ......................... 57  

V. SENSOR VALIDATION STRATEGY ....................................... 59  
 5.1 GENERAL CHARACTERISTICS ........................................ 59  
 5.1.1 Impact of Malfunctions on Validation ......................... 60  
 5.1.2 Diagnosis and Validation Integration ......................... 61  
 5.1.3 Flexibility .......................................................... 63  
 5.1.4 Sources of Alternate Data ...................................... 64  
 5.1.4.1 Hardware Redundant Sensors ................................. 65  
 5.1.4.2 Direct Qualitative Estimate by Independent Sensors .......... 67  
 5.1.4.3 Calculations With Empirical or Process Models .......... 68  
 5.2 SENSOR VALIDATION ALGORITHM ................................ 70  
 5.3 DISCUSSION ON THE ALGORITHM .................................. 74  
 5.3.1 Independent Data Checks ....................................... 75  
 5.3.2 Error Elimination .................................................. 75  
 5.3.3 Rigorous Validation Application Criteria .................... 76  
 5.3.4 Rigorous Validation and Conflict Resolution .................. 78  
 5.4 OTHER RELATED ISSUES .............................................. 82  
 5.4.1 Algorithm Failure Conditions ................................... 82  
 5.4.2 Reliability ranking ............................................... 82  
 5.4.2.1 Role of Reliability Ranking .................................. 83  
 5.4.2.2 Assignment of reliability .................................... 85  
 5.4.2.3 Measure of reliability ........................................ 85  
 5.4.3 Results of Validation ............................................. 86  

VI. IMPLEMENTATION ..................................................... 88  
 6.1 CHEMICAL PROCESS FOR IMPLEMENTATION ...................... 89
## Table of Contents

6.2 OVERALL SYSTEM ARCHITECTURE ................................... 93  
6.2.1 Malfunction Hierarchy ..............................................................95  
6.2.2 Symptoms Database ....................................................................99  
6.2.3 Validation Methods and Quantitative Models  
  Hierarchy ...................................................................................... 103  
  6.2.3.1 Sensor and Alarm Nodes .............................................103  
  6.2.3.2 Equipment nodes ........................................................ 106  
6.3 GENERAL FUNCTIONS ...............................................................108  
6.4 FUNCTION INTERFACE WINDOWS ....................................... 108  
  6.4.1 Symptoms Database Interface Window .............................. 109  
  6.4.2 Sensor Validation Interface Window ................................... 111  
6.5 COMPUTER HARDWARE .............................................................113  

VII. PERFORMANCE CHARACTERISTICS ............................................. 114  

7.1 INTEGRATED DIAGNOSIS AND SENSOR VALIDATION  
  7.1.1 Organization of Validation Information .............................. 116  
  7.1.2 Hypothesis Evaluation by Direct Pattern Matching ............117  
  7.1.3 Hypothesis Evaluation by Calculation ................................. 121  

7.2 CASE STUDIES .......................................................................  124  
  7.2.1 Case 1 - Category I Sensor Failure Alone ............................126  
  7.2.2 Extent of Rigorous Validation ............................................ 139  
  7.2.2.1 Straight Through Route Removed ...............................140  
  7.2.2.2 Validity Update Removed .......................................... 143  
  7.2.2.3 Both Approaches Removed ........................................143  
  7.2.3 Case 2 - Equipment Malfunction Plus Category I Sensor Failure ............................................ 146  
  7.2.4 Case 3 - Equipment Malfunction Plus Category II Sensor Failure ............................................ 155  
  7.2.5 Concluding Remarks ............................................................ 162  

CONCLUSIONS AND RECOMMENDATIONS .......................................... 164  

LIST OF REFERENCES ................................................................. 172  

APPENDICES  

A. DYNAMIC CSTR PROCESS SIMULATION ..................................... 176  

B. CSRL CODE FOR MALFUNCTION HIERARCHY ....................... 187
C. CODE FOR SYMPTOMS DATABASE ................................................. 202

D. CODE FOR VALIDATION HIERARCHY AND GENERAL
   FUNCTIONS ................................................................. 212
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURES</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Generic Task Framework for Diagnosis</td>
<td>12</td>
</tr>
<tr>
<td>2. Simplified Terephthalic Acid Oxidation Reactor Flowsheet</td>
<td>23</td>
</tr>
<tr>
<td>3. A General Functional Hierarchy</td>
<td>31</td>
</tr>
<tr>
<td>4. Cyclohexane Production Plant Flowsheet</td>
<td>37</td>
</tr>
<tr>
<td>5. Feed System Functional Hierarchy</td>
<td>39</td>
</tr>
<tr>
<td>6. Reaction System Functional Hierarchy</td>
<td>40</td>
</tr>
<tr>
<td>7. Product System Functional Hierarchy</td>
<td>41</td>
</tr>
<tr>
<td>8. Sensor Validation Algorithm</td>
<td>71</td>
</tr>
<tr>
<td>9. Recycle Reactor Process Flowsheet</td>
<td>91</td>
</tr>
<tr>
<td>10. Overall Computational Architecture</td>
<td>94</td>
</tr>
<tr>
<td>11. Malfunction Hierarchy for CSTR Process</td>
<td>96</td>
</tr>
<tr>
<td>12. CSRL Code for the HeatExchanger Specialist</td>
<td>98</td>
</tr>
<tr>
<td>13. Symptoms Database</td>
<td>100</td>
</tr>
<tr>
<td>15. Validation Methods and Quantitative Models Hierarchy</td>
<td>104</td>
</tr>
<tr>
<td>16. Code for the TT8-Ob Sensor Node</td>
<td>105</td>
</tr>
<tr>
<td>18. Symptoms Database Interface Window</td>
<td>110</td>
</tr>
</tbody>
</table>
19. Sensor Validation Interface Windows ................................................. 112
20. Hypothesis Evaluation by Direct Pattern Matching ............................. 119
21. Hypothesis Evaluation by Calculation ................................................. 123
22. Plot of Coolant Flowrate for Case 1 ................................................... 127
23. Plots of Actual and Measured Recycle Temperatures for Case 1 ... 128
24. Plot of Reactor Temperature for Case 1 ............................................ 129
25. Confidence Value Hierarchy for Case 1 .......................................... 131
26. Symptoms DataBase Interface Window With Tracing of Data Abstraction Results for Case 1 ............................................................ 132
27. Sensor Validation Interface Windows With Partial Tracing of Validation Results for Case 1 .............................................................. 134
28. Complete Validation Message Tracing for Case 1 .............................. 135
29. Sensor Validation User Input Window ............................................... 138
30. Message Tracing for Case 1 With Straight-Through Route Removed 142
31. Message Tracing for Case 1 With Update-Validity Removed ........ 144
32. Message Tracing for Case 1 With Both Approaches Removed ................ 145
33. Plots of Feed and Product Flowrates for Case 2 ........................ 147
34. Plots of Reactor and Recycle Temperatures for Case 2 .......... 148
35. Plots of Coolant Flowrate and Outlet Temperature for Case 2 ... 149
36. Plots of Actual and Measured Reactor Levels for Case 2 ........ 151
37. Confidence Value Hierarchy for Case 2 ............................................ 152
38. Complete Message Tracing for Case 2 .............................................. 153
39. Plots of Actual and Measured Coolant Flowrates for Case 3 .......... 156
40. Plot of Coolant Outlet Temperature for Case 3 ............................... 157
41. Plots of Reactor and Recycle Temperatures for Case 3 ............... 158
42. Confidence Value Hierarchy for Case 3 ................................. 160
43. Complete Message Tracing for Case 3 ................................. 161
CHAPTER I

INTRODUCTION

1.1 CHARACTERIZATION OF DIAGNOSIS IN PROCESS PLANTS

Processing plant operations rely heavily on control systems to establish operating conditions for maximum production of desired products and to maintain conditions in the face of process disturbances. It is recognized that conventional process control systems are effective for automatically making local process changes within some range to establish or maintain operating conditions. However, they are ineffective when an operation requires the identification of malfunctioning equipment or changes in operating parameters. As a result, process expertise for diagnosis of malfunctioning equipment and inappropriate process settings complements the role of hardwired process control systems, and is necessary for effective operation of these processes. Diagnosis in the process plant domain, therefore, can be generally characterized as the mapping from the symptoms on hand to a conclusion which is useful in resolving or alleviating the problem. Two distinct categories of diagnosis are evident. In response to a process problem, the appropriate diagnostic problem solving, and in turn the extent of detail of the conclusion obtained, depend on the response time involved and the criticality of the diagnostic situation. The two categories can be characterized as follows.
In one category, the conclusion is in terms of an abstraction of the state of the process such that corrective actions can be prescribed to alleviate the effects of a process problem without necessarily identifying the source. The purpose of diagnosis here is to identify appropriate corrective actions to avoid an imminent safety, operational or economic concern. During the time that an undesirable situation is imminent, the objective of the diagnosis is one of determining the state of the operation so that proper corrective action can be taken to avert a catastrophe rather than one of identifying and fixing a specific malfunctioning component. These kinds of "fast response" problems are typically recognizable in terms of 1) a time scale on the order of seconds or minutes in which the malfunction manifests itself, 2) a small amount of time available for doing the diagnosis, 3) rapidly varying data and 4) an alarm-driven diagnosis. Response time in this case is on the order of seconds and minutes and, therefore, becomes a stringent demand on a viable computational system. As far as diagnosis is concerned, the ability to reason through the details of the process is de-emphasized, while the ability to rapidly index symptoms into corrective actions is strongly emphasized.

In the other category, however, the conclusion is in terms of the primary cause(s) of the observed symptoms so that the source of the symptoms can be fixed or removed. This category includes problems which involve a degradation in performance due to drift or decay as well as malfunctions which do not critically or imminently affect the integrity of the process. However, it is still important to prevent the continued economic impact of the problem or further deterioration into
a potentially dangerous situation. The diagnostic objective is to carry through the diagnosis to a level of detail which identifies a root cause of the observed symptoms, and the emphasis is on thorough reasoning for accomplishing it. In turn, the corrective action is to fix the primary cause with the intention of preventing the problem from continuing or reoccurring. Typical characteristics of these so called "slow response" problems include 1) sufficient time to perform detailed tests, 2) current and past process data used in a "snapshot" style, and 3) an emphasis on sensor data, product quality data and maintenance records, in addition to alarm data. An expert system for slow response diagnosis is strained not by real time needs on the order of seconds, but on its ability to efficiently reason with diverse types of knowledge in identifying a malfunctioning aspect of the process at a level which is detailed enough to fix the problem. Note that the term snapshot is used here to refer to the set of process data from the various sensors that is taken in a specified window in time.

With this characterization of slow and fast response diagnosis, our emphasis is on slow response problems. In this context, there are two distinct problems in slow response diagnosis, namely, malfunction diagnosis and process diagnosis. The goal of malfunction diagnosis is the identification of a hardware malfunction, i.e. breakdown or deterioration of a physical process equipment. Process diagnosis involves the identification of deviating operating parameters. This problem is non-trivial because the notion of normality for many operating parameters is premised upon certain conditions, such as feed and catalyst properties, any of which may have changed during the course of operation. The identification of abnormal
operating parameters provides additional symptomatic evidence for identifying a hardware malfunction. If no hardware malfunction is identified, then the abnormal operating parameters provide a starting point for determining adjustments to the process.

1.2 OVERVIEW OF PROBLEMS ADDRESSED

This research concentrates on the malfunction diagnosis problem. Since complex plants generally have a large number of process components and each component can potentially malfunction, it is essential to have a problem-solving strategy which can effectively navigate through the solution space of all the potentially malfunctioning process components. This need is addressed in the hierarchical classification approach [4,5,6]. The structured problem solving in classification results in efficiency because of the organizing concepts captured in the levels of the hierarchy. Therefore, in order to apply classification to malfunction diagnosis in process plants, a decomposition of the process from general organizing concepts to specific equipment in the plant is necessary.

An appropriate organizing concept for constructing the classification hierarchy in the Chemical Processing Plant (CPP) domain is the abstraction of process details. With this conceptual basis, the hierarchical problem-solving approach starts with a highly simplified view of the process. At that level, the process is viewed as simple blocks with input and output streams so that the complexity is suppressed. Focus is quickly narrowed down so that further problem solving with progressively more detailed considerations is only necessary for a small portion of the process.
It is found that a significant portion of this process decomposition is addressed by a functional decomposition, which captures the levels of functional abstraction between the purposes (functions) of the systems at high levels and the physical process equipment components at low levels. Due to the diversified nature of the process plant domain, constructing a hierarchical decomposition of the plant based on the functional decomposition viewpoint is non-trivial. Therefore, it is important to define a systematic strategy which facilitates the construction of the hierarchical decomposition. That is the purpose of the systematic functional decomposition strategy developed in this work.

With the hierarchical decomposition, the individual organizing concepts and specific equipment malfunctions are evaluated based on the symptoms provided primarily by sensors, alarms and operator observations. Therefore, sensor failures have significant impact on malfunction diagnosis. Sensors can be classified into two distinct categories according to their intended functions. In the first category are process control system sensors. Since the control systems depend on these measurements for properly maintaining the process operating conditions, failures of control system sensors lead to undesirable process conditions, i.e. the cause of the malfunctioning conditions. The second category includes non-control system sensors. These sensors do not directly affect process operation, but they provide information for monitoring and diagnosis. Despite their different effects on the process operation, failures of either category of sensors can lead to misdiagnosis when the erroneous measurements are used for diagnosis. The sensor can fail in such a way that it exhibits the symptom of a malfunction when there is no actual
malfunction, or it can fail in the opposite manner, i.e. not showing the symptom when there is an actual malfunction. Therefore, sensor validation is an important problem in the diagnostic system.

Conventional approaches to sensor validation studied in the CPP domain generally involve some form of statistical analysis. These approaches are rarely considered in the context of malfunction diagnosis. Therefore, many of the methods developed assume normal operating conditions and only test for measurement errors. However, in a malfunction diagnosis situation, deviation from the normal operating conditions and malfunctions other than sensor failures have to be considered. This further complicates the validation problem and makes many statistical analysis methods ineffective. Moreover, in these approaches, only quantitative equation-based methods can be used and the statistical testing is applied to a large number of plant sensor data by manipulating a large set of constraint equations simultaneously. Therefore, these approaches are extremely computation expensive. Besides, due to limited instrumentation, many constraint equations cannot be applied. For these reasons, these approaches are not widely employed in chemical plants so that the general availability of statistically validated data cannot be assumed. Therefore, a novel sensor validation strategy which addresses these shortcomings is developed in this research.

1.3 RESEARCH FOCUS

The application of hierarchical classification to malfunction diagnosis in the CPP domain has been studied in my previous work [33]. In that study, the use of
a malfunction hierarchy, which captured malfunction hypotheses in different levels of process detail in the levels of the hierarchy, was shown to be effective for diagnosis in the domain. The hierarchy was constructed based on a functional decomposition of the process. The hypotheses were evaluated based on direct pattern matching between the observed symptoms and the discriminating symptoms of the malfunction hypothesis. This research builds on the previous results to enhance the application of the structured approach to malfunction diagnosis in the CPP domain. Specifically, the focus of the research is on three issues.

1. **Functional decomposition strategy** - The computational efficiency of hierarchical problem solving is derived from the levels of process detail captured in the diagnostic hierarchy. By eliminating general hypotheses at higher levels of the hierarchy, large number of specific hypotheses down the hierarchy can be pruned from further consideration. Therefore, the construction of an appropriate hierarchy for the process under consideration is an important concern in the application of this structured diagnostic approach. In order to facilitate the construction of an appropriate hierarchy, a systematic functional decomposition strategy is developed. Based on an analysis of the general chemical process plant characteristics, this strategy guides the identification of functional systems and subsystems of the plant from the process flowsheet.

2. **Sensor validation** - Since the diagnosis extensively uses data provided by sensors, it is important to validate the data in order to prevent erroneous diagnostic conclusions. To address this need, a novel sensor validation
strategy is developed in this research. This strategy takes into account the characteristics of chemical plants to take full advantage of the resources available for validation. Unlike conventional validation approaches, which are generally independent of diagnosis, the validation strategy here is developed specifically for diagnosis. As a result, sensor validation is integrated into the diagnostic process and the focus of validation is on eliminating sensor failures which can affect the results of diagnosis. The resulting computational architecture effectively combines the use of both symbolic and numerical model-based knowledge in solving the diagnosis and sensor validation problems.

3. Quantitative hypothesis evaluation - Besides directly observed symptom patterns, the discriminating characteristics of a malfunction can also be in terms of results of calculational tests, which are obtained by manipulating the sensor data on hand. In this way, given the necessary sensor measurements, the appropriate material and energy balance closures are calculated. The results can then be combined into higher abstractions pertaining to the operational status of the process equipment. With the numerical calculation capability developed for sensor validation, it is natural to develop this type of quantitative hypothesis evaluation methods such that they can be applied along with the direct pattern matching technique. This leads to more flexibility in the diagnostic determination.
To illustrate the computational methodology developed, a working prototype knowledge-based system for diagnosis and sensor validation has been built and tested. The prototype system is based on a dynamically simulated continuous stirred tank reactor (CSTR) with recycle. The simulation is used for two distinct purposes. During the development of the diagnostic system, the simulation can be run to study the effects of different malfunctions on the process operation. This establishes the discriminating symptoms of the various malfunctions. When the prototype system development is completed, the simulation provides the numerical sensor data for testing the performance of prototype system.

1.4 DISSERTATION ORGANIZATION

The organization of the dissertation is as follows. The overall diagnostic framework of our AI in Engineering research group will be discussed in Chapter II. In Chapter III, a systematic functional decomposition strategy will be presented. In Chapter IV, characterization of the sensor validation problem and a brief review of sensor validation techniques will be given. The sensor validation strategy developed in this research will be discussed in Chapter V. Then, the issues associated with the implementation of the prototype system will be discussed in Chapter VI. Finally, the performance characteristics of the integrated diagnosis and sensor validation system will be illustrated with representative case studies on the prototype system in Chapter VII, followed by the Conclusions and Recommendations. The complete program listings of the dynamic simulation and prototype system are included in the Appendices.
CHAPTER II
DIAGNOSTIC FRAMEWORK

2.1 GENERIC TASK VIEWPOINT

Our knowledge-based system approach to diagnosis is based on the generic task viewpoint, which was originally proposed by Chandrasekaran [4,5,6] and addresses the issue of explicitly structuring the domain knowledge and the problem-solving methodology by identifying the underlying tasks that comprise a problem-solving activity in a domain. Indeed, structure in knowledge-based systems has been a major research issue of our AI in Engineering group at The Ohio State University [8,9,25,28,33,35,36]. With this task viewpoint, diagnosis is treated as an activity which can be decomposed into a finite set of distinct tasks. Each task is responsible for performing a unique function which supports the overall diagnostic activity. Due to distinctively different inherent characteristics, each task is associated with its own form and organization of knowledge, and a problem-solving strategy which is specifically designed for performing the task effectively.

By recognizing the underlying organizational and problem-solving strategies for a broadly defined domain, it is possible to develop a set of generic problem-solving modules for the tasks. Each module provides a priori conceptual definitions of the types of knowledge used, and provides effective programming structures and
computational strategies to support the use of this knowledge. A computational framework for a specific domain application can then be constructed by integrating the appropriate modules which capture the required tasks. Conceptually, the framework facilitates the development of knowledge-based systems by providing a medium for analyzing an application at an appropriate level of detail.

2.2 OVERVIEW OF DIAGNOSTIC FRAMEWORK

Within the context of the task-oriented approach to diagnosis, our generalized framework for a knowledge-based diagnostic system in the process plant domain is shown in Figure 1 to consist of the following distinct problem-solving tasks:

1. The core task is called classification and is captured in a malfunction hierarchy in which the nodes represent primarily malfunction hypotheses. Each node in the hierarchy contains both the local control strategy and the knowledge structure which organizes the diagnostic knowledge required to evaluate the malfunction hypothesis, i.e. an integrated knowledge base and inference mechanism. This problem-solving structure allows one to proceed from general malfunction hypotheses at the top of the hierarchy to successively refined hypotheses down the hierarchy, thereby exploring the large solution space in an efficient manner. The classification task identifies plausible malfunction hypotheses based on the observed symptoms, which are in terms of process state parameters provided by sensors, alarms and operator observations.
Figure 1. Generic Task Framework for Diagnosis.
2. A symptoms database is necessary in maintaining data and handling data queries from the various problem solvers. The database performs data abstraction, which refers to the preprocessing of numerical data to generate diagnostically useful data required by the problem solvers. The task is to map numerical data into data which are directly useful in the decision making.

3. Diagnosis makes extensive use of information provided by sensors. However, like other process equipment, sensors are susceptible to failures. As a result, sensor validation is important in preventing erroneous diagnostic conclusions. Methods for validating sensor measurements and for numerically evaluating malfunction hypotheses of process equipment are organized in the validation methods and quantitative models hierarchy.

4. Besides sensor data, product quality data provide additional symptomatic information for diagnosis. Product quality information reflects the cumulative effects of all process abnormalities. Unlike sensor data, which provide a localized view of the operating state of the process, product quality deviations provide general evidence of malfunctions in the plant. In order to reason about product quality deviations, a task called hypothesis assembly is necessary. The task is to construct a plausible explanation for deviating product quality attributes in terms of deviations in operating parameters, i.e. process diagnosis. The identified operating
parameter deviations can then be used in the classification hierarchy as symptoms for evaluating a relevant malfunction hypothesis.

5. Due to the integrated nature of process plants, it is conceivable that a malfunction in one functional system of the plant may affect other functional systems. Also, in multiple malfunction situations, the malfunctions may have interacting effects on certain operating parameter so that a symptom pattern which is unexpected for any single malfunction may be obtained. As a result, additional processing by causal simulation may be necessary in the diagnosis. This task involves cause and effect reasoning about operating parameters across the process topology. Causal simulation is constrained by the goals and knowledge derived from the classificatory problem solving.

6. Finally, an explanation facility is used to provide clear explanation of the domain knowledge and control strategy used in the problem solving. It shows how knowledge is used and how the diagnostic conclusions are obtained. Four basic types of explanation are considered. Three basic types rely upon the control strategy and the deep source from which the knowledge is compiled. The fourth type justifies the information processing involved in the problem solving.

The problem-solving framework for the overall diagnostic activity in the CPP domain involves the integration of these distinct tasks. The modular nature of this framework allows the incorporation of any necessary auxiliary tasks into the main
activity of diagnosis. Thus, there is a high potential for further extending the applicability of the diagnostic system.

With this view of our diagnostic framework, the focus of this research is on the classification and sensor validation tasks. Since classification is the primary task in our diagnostic approach, it is discussed in more detail below to provide the context for further discussion in this work. The sensor validation task will be discussed in later chapters.

2.3 CLASSIFICATION

In the core task of classification, the hierarchical structure allows the decision process to proceed from a high level of generality to a high level of detail. By reasoning from the most general node to the more specific nodes, the number of considerations can be greatly reduced by rejecting upper level nodes and thereby removing from consideration entire branches in the hierarchy. This pruning strategy is called establish-refine. Specifically, with respect to malfunction diagnosis, this approach results in the organization of sets of malfunctions in terms of single, generally defined concepts at higher levels of the malfunction hierarchy. By eliminating individual organizing concepts instead of individual malfunctions, large groups of the total number of malfunctions can be removed from consideration. Thus, although the hierarchy represents the entire set of possible malfunctions, diagnosis at run-time involves the consideration of only a small part of the solution space of all potentially malfunctioning equipment components.
The malfunction hierarchy is both an abstraction hierarchy and the diagnostic problem solver. In addition to being associated with specific malfunction hypotheses, the nodes in the hierarchy conceptually represent diagnostic specialists. Instead of being a static collection of knowledge, the nodes participate in the problem solving. Each node has knowledge pertinent to establishing or rejecting the relevance of a particular malfunction hypothesis. Nodes higher in the hierarchy are associated with more general malfunction hypotheses and as a result have more general knowledge while lower nodes have more specific knowledge to make more detailed decisions. The diagnostic hierarchy can be visualized as a community of many small diagnostic experts which are coordinated to arrive at an overall decision about malfunctions in the plant.

The malfunction hypothesis under consideration determines what symptomatic information is relevant for establishing or rejecting that hypothesis. There is no restriction on the techniques that can be used for evaluating the individual malfunction hypotheses. In general terms, the malfunction hypothesis represented in a node of the hierarchy can be evaluated by comparing the symptoms on hand with the discriminating characteristics of the malfunction. The symptoms on hand are provided in the form of alarm states, sensor measurements, product qualities, and operator tests and observations. The discriminating characteristics of the malfunction can be represented in terms of directly observed symptom patterns. In that case, the observed symptoms can be directly matched with the discriminating symptom patterns to determine the relevancy of the malfunction hypothesis. Indeed,
direct pattern matching has been the only technique used in previous knowledge-based diagnostic systems in the CPP domain [25,34].

The hierarchical classification approach resulted in the design of a high-level programming language called Conceptual Structures Representation Language (CSRL) [3]. CSRL provides a basic structure for organizing knowledge hierarchically and a mechanism for top-down, parallel refinement, i.e. the establish-refine pruning strategy. Although CSRL enhances the programming of a classification hierarchy, it does not provide any conceptual help in the construction of an appropriate diagnostic hierarchy from the process flowsheet. Therefore, in order to facilitate the application of hierarchical classification to malfunction diagnosis in the CPP domain, a conceptual basis for the hierarchy needs to be defined.
CHAPTER III
FUNCTIONAL DECOMPOSITION

One basic requirement for the application of hierarchical classification to malfunction diagnosis is the construction of a hierarchical decomposition for the process under consideration. The problem-solving efficiency of hierarchical classification as the backbone of our structured diagnostic approach is derived from the levels of process abstraction captured by the hierarchy. Therefore, a hierarchical structure is not necessarily efficient for diagnosis and a careful evaluation of the conceptual basis of the hierarchy is needed. Various conceptual bases have been used in constructing the diagnostic hierarchy. Some of these bases have been discussed in my previous work [33]. In this research, a more in-depth analysis of three commonly used bases is performed. Due to the diversified nature and complexity of the CPP domain, constructing a hierarchical decomposition of a process plant is non-trivial. Therefore, a systematic functional decomposition strategy is developed in this work. This strategy facilitates the construction of a malfunction hierarchy, which contains malfunction hypotheses in progressive levels of detail in the nodes.
3.1 BASIS FOR THE CLASSIFICATION HIERARCHY

Different bases may be used in the construction of a classification hierarchy. The appropriate basis depends on the characteristics of the problem being solved. For diagnosis, the hierarchy can be based on events, structures or functions. The term event is used here to refer to any consequence of a malfunction in the plant, and it may or may not be directly observable. For example, a high temperature event can be directly observed by a temperature sensor, i.e. also a symptom. On the other hand, a blocked valve is an event which must be indirectly inferred from other observable symptoms such as flow and pressure sensor readings. Note that the categories of hierarchy discussed below refer to the overall viewpoint adopted in the construction of the hierarchy, and so the distinction may not be clearly visible throughout the hierarchy. For example, a functional hierarchy may contain nodes which represent structural components at lower levels in the hierarchy. The reason is that the primary purpose of malfunction diagnosis is to identify the specific equipment component that is malfunctioning or responsible for the occurrence of the event so that it can be repaired. As a result, all the diagnostic hierarchies somehow lead to considerations of specific structural components. The three commonly used hierarchical structures for diagnosis are first characterized below, followed by a discussion on their usefulness in CPP domain diagnosis.

1. **Event hierarchy** - it consists of undesirable events at the top of the hierarchy and possible causes of those events down the hierarchy. The levels in the hierarchy represent explicit causal relationships. For example, the top undesirable event may be a high reactor temperature,
and one of the possible causes is that the coolant flow valve is stuck closed.

2. **Structural hierarchy** - it is based on the physical relationships between the process equipment of the plant. Therefore, the nodes in the hierarchy are linked by physical system-subsystem kind of relationships, no functional abstraction is considered. For example, for a stirred tank reactor, the agitator and the motor drive unit would be considered as components of the reactor because the mixer is physically a part of the reactor. This hierarchy is easy to construct because the physical relationships between components are usually simple and can be obtained directly without much analysis of the process flowsheet.

3. **Functional hierarchy** - it is based on the functional relationships between the process equipment of the plant. This hierarchy can be constructed through knowledge of the functions of various systems in the plant and how these functions are achieved by means of lower level functions of more specific subsystems.

As discussed in the previous section, for efficient problem solving, a classification hierarchy needs to capture the levels of abstraction of the problem involved. Therefore, a classification hierarchy for malfunction diagnosis in the CPP domain needs to represent the decomposition of the processing plant into systems and subsystems of various levels of process details.

It is often difficult to use an event hierarchy for diagnosis in the CPP domain because the diagnostic knowledge is not commonly available in terms of events.
Furthermore, since current approaches to the systematic generation of an event hierarchy generally does not capture the levels of process detail in the plant, extra effort must be applied in order to impose levels of abstraction in the events. This makes the event hierarchy unattractive for the CPP domain.

It should be noted that an event hierarchy has been effectively used in a hierarchical diagnostic system in the Nuclear Power Plant (NPP) domain [18]. In that system, the major task is to examine the important events (faults) associated with the nuclear reactor cooling system and identify the root cause when an event occurs. In other words, the classification hierarchy is used to examine the possible causes of the various types of cooling system faults to determine the actual cause. Due to the overwhelming concern for safety in the NPP domain, it is natural to think in terms of the observable characteristics of undesirable events and the corrective actions to those events. When certain symptoms are observed, the state of the process, i.e. the fault condition, can be determined and the corrective actions performed without first identifying the root cause of the event. Moreover, for root cause analysis, the diagnostic knowledge is readily organized in terms of the various undesirable events and the causes of those events. Thus, it is convenient to capture the knowledge in an event hierarchy.

Both structural and functional hierarchies capture levels of process detail and can be directly constructed by decomposing the plant into systems and subsystems. In other words, both structural and functional decomposition viewpoints can be directly applied to a process flowsheet. However, since physical components in a plant are inherently configured to perform certain functions, it is natural to think
about the process with respect to a functional rather than a physical point of view. Moreover, any physical component may be part of several functional systems, i.e. serves more than one function. Since functional reasoning is necessary in diagnosis, a structural hierarchy would be inappropriate because components in different branches of the hierarchy may be functionally related, i.e. strong interaction among the branches. As a result, it is difficult to define the diagnostic knowledge necessary for each node in the resulting hierarchy to be evaluated independently in the diagnosis. Furthermore, a functional viewpoint results in more levels of process abstraction because high-level abstract functions can be extracted out of the organization of the physical components and their respective low-level functions. For example, referring to the plant segment shown in Figure 2, Pressure Control System, Cooling System, Mixing System and so on are identified as functional subsystems of the reactor because they control certain aspects of the reaction that takes place inside the reactor. If a structural viewpoint is used, this further organization would not be possible. For classificatory problem solving, more levels of abstraction generally result in higher efficiency because more low-level concepts can be ruled out by rejecting a high-level organizing concept.

The primary purpose of malfunction diagnosis is to explain the observed symptoms by identifying the reason (cause) why a certain process function is not performed correctly. Since reasoning with functions is needed, a functional hierarchy is most direct and appropriate, i.e. the natural decomposition. The diagnostic knowledge can be derived from a basic understanding of the functions of the various systems and subsystems of the process. For example, the function of
Figure 2. Simplified Terephthalic Acid Oxidation Reactor Flowsheet.
a temperature control system is to control the temperature. Therefore, when the
temperature control system malfunctions, the symptom for diagnosis includes an
abnormal temperature measurement. Unlike an event hierarchy, where every
important event must be represented in the hierarchy, only the functional systems
and subsystems need to be considered here. Depending on the malfunction, i.e. how
the functional system fails, different events would need to be considered. For
example, referring to Figure 2, a pressure control valve malfunction results in a
high pressure or low pressure event in the reactor when the valve fails closed and
open, respectively. By considering functional system and subsystem malfunctions
instead of events, fewer nodes are needed to represent the same information in the
hierarchy. Thus, functional decomposition results in a more compact hierarchy
compared to an event hierarchy. This idea has been discussed in some detail
previously [34]. Moreover, due to the functional basis, it is straightforward to
construct a complete hierarchy which includes all the functional systems and
subsystems (and the corresponding possible malfunctions). Finally, functions
performed by various systems in the plant can be generally defined and are
essentially independent of the exact plant configuration. Thus, a systematic
functional decomposition strategy can be defined for a generalized plant.

It is important to note that the functional process decomposition strategy
developed in this work should be viewed as a primary decomposition strategy
rather than the only strategy for the CPP domain. Although it does capture the
backbone structure for a diagnostic hierarchy such that a good initial hierarchy is
provided, it is recognized that, depending on the process under consideration, other
concepts in addition to functional systems may need to be incorporated. For example, specific events may need to be considered at a certain part of the hierarchy in addition to the functions.

It should also be noted that a functional system decomposition strategy for bringing in more focus in the diagnosis has been discussed by Finch and Kramer [15]. In their approach, there are only two levels of abstraction, namely, the unit function level and the system level. Each system is assigned the corresponding unit functions, no intermediate or higher (above system level) abstractions are considered. Potential fault propagation pathways and dependencies between the systems are represented by a process graph. Due to the inherent differences in the diagnostic approaches, the focus of our decomposition and the organization of functional systems are different.

3.2 FUNCTIONAL DECOMPOSITION STRATEGY

Functional decomposition is an operation in which systems and subsystems of a process plant are identified and organized according to the functions they perform. Therefore, given a process flowsheet, the outcome of functional decomposition is a functional hierarchy in which each node represents a functional subsystem of the plant and the level of process detail increases from top to bottom of the hierarchy. This can be effectively done by gradually zooming in to reveal more and more functional details of the plant. With this approach, a process plant is initially viewed as an input-output structure, i.e. a black box with inflow and outflow. The decomposition continues to the detailed equipment component level
so that when a malfunction is identified, the specific corrective action can be prescribed by the operator. Through studies of general characteristics of the CPP domain, a systematic decomposition strategy has been defined.

In abstract terms, the function of a typical chemical processing plant can be generally defined as the conversion of some raw material(s) to some final product(s) which satisfies given specifications. Therefore, three general functional systems can be identified at this high level of abstraction.

1. *Feed system* - this system maintains the flow and properties of the input stream of the raw material(s). If any recycle stream is added to the raw material feed before it enters the reaction system, the feed system needs to maintain the flow and properties of the combined fresh feed plus recycle stream. Therefore, in that case, the recycle stream is also considered here.

2. *Reaction system* - this system is responsible for the conversion of raw material(s) to product(s). It primarily contains the reactor and any support systems that maintain the desired operating conditions under which the reaction should take place.

3. *Product system* - this system performs the purification and output of final product(s). Separation process, transfer of intermediate and final product(s) and storage of the final product(s) are included in this system.

In order to ensure the performance of these functional systems, the key operating parameters are controlled either directly or indirectly by various regulatory systems. Therefore, each of these functional systems can be decomposed
into the corresponding regulatory systems that are present. Note that the term regulatory system is used here (instead of control system to prevent potential confusion) to refer to the entire functional subsystem through which a certain function is maintained, i.e. a parameter is controlled, not simply the closed control loop itself.

Although the actual regulatory systems present depend on the particular plant configuration under consideration, only a few important operating parameters are typically controlled. These include temperature, pressure, fluid flow rate, fluid concentration, liquid level and mixing rate (in a stirred tank). Note that important parameters such as reaction rate and catalyst activity often are not directly controlled because of difficulties in both on-line monitoring and adjustment. Such parameters are indirectly monitored and controlled.

These regulatory systems can be further decomposed. The exact details of how the control is achieved would naturally depend on the plant configuration. However, a regulatory system has certain common characteristics which allow a general decomposition to be defined. A regulatory system can be decomposed into the following functional subsystems:

1. **Supply function** - this function includes the supply of two different streams. The process stream is always needed while the additional stream may or may not be needed.

   a) Supply of the process stream fluid the properties of which are being controlled. For example, the supply of raw material feed to
allow the temperature and/or flow rate of the feed stream to be controlled.

b) Supply of any necessary additional medium to effect the control of the process stream property. For a closed loop control system, the flow rate of this medium is always controlled by the final control element of the control loop. For example, the supply of cooling water or steam to effect temperature control, the supply of electricity to the motor to effect mixing rate control.

2. **Equipment function** - functions associated with the auxiliary equipment without which proper control cannot be achieved. For example, a heat exchanger for temperature control, a compressor for pressure control.

3. **Control configuration** - it is the part of the regulatory system where actual control action is generated. The control configuration can be decomposed into:

   a) **Sensor** - monitors the controlled variable.

   b) **Controller** - given a setpoint, calculates signal to the final control element according to the control algorithm and the deviation of the controlled variable from setpoint.

   c) **Final control element** - achieves control action by adjusting the manipulated variable according to signal from controller. In case of cascade control, the secondary control system is considered here instead of the final control element.
For example, for closed loop temperature control, the control configuration may consist of a temperature sensor, PI control algorithm, temperature setpoint and a control valve to control the flow rate of cooling water or steam. On the other hand, for mixing rate, the control configuration may simply include the rpm setting and the electric motor of the agitator. The rpm may not be directly monitored and adjusted at all.

At this point, the decomposition has reached the specific component level for the control configuration. This level of process detail is sufficient for corrective actions to be taken once a specific malfunction is identified. On the other hand, the supply and equipment functions may be further elaborated into the various important functional aspects if necessary. However, it should be pointed out that the additional details about the functions are often unnecessary. For example, it is generally enough to find out whether or not there is a malfunction in the auxiliary equipment of the regulatory system. Whether the malfunction is in the containment or flow path of the equipment is not important.

1. The supply function can be broken down into the following:
   a) Property - the important state/property of the supply. For example, temperature of cooling water supply, concentration of raw material feed.
   b) Source - the reserve of the supply. For example, tanks of raw material feed for flow control.
c) Path - the flow path from the source to the control system. For example, pipelines for fluid flow, cables for electricity.

d) Driving force - the mechanism which provides the potential to move the supply from the source to the control system. For example, pumping to maintain pressure drop for flow from storage tank.

2. The equipment function depends on the type of equipment under consideration. For example, the primary functions of a heat exchanger in temperature control include the provision of a large heat transfer area and a large overall heat transfer coefficient to allow efficient heat exchange. In addition, like any equipment which handles fluids, a path for fluid flow (over the heat transfer area in this case) and fluid containment are necessary support functions which must also be fulfilled in order for the equipment to function properly.

A fragment of a general functional hierarchy is shown in Figure 3. Notice that from the functional decomposition strategy described above, at a detailed process level, structural components and process equipment are often considered directly instead of their functions. Nevertheless, the overall functional viewpoint is still maintained. These components are entered directly because, at that level, a function can be attributed directly to a specific and small component. Moreover, the ultimate goal of diagnosis is to identify which structural component is responsible for the abnormal state of the plant such that the operator can easily correct the problem. This also coincides with the observation that hierarchical
Figure 3. A General Functional Hierarchy.
problem solving using a functional viewpoint proceeds from functions at high levels to physical components at low levels [30].

3.3 MALFUNCTION HIERARCHY

The functional decomposition strategy described above results in a functional hierarchy which is a representation of the functional relationships among the various systems and subsystems in the plant. For diagnosis using a malfunction-driven approach, this functional hierarchy is most conveniently recast as a malfunction hierarchy in which each node represents a malfunction hypothesis concerning the functioning of the corresponding subsystem. This conversion is mostly straightforward because a malfunction is simply a violation of a certain function, i.e. there is a one-to-one correspondence. However, a few modifications and extensions may be necessary in some of the nodes.

1. A malfunction hypothesis derived directly from the functional hierarchy sometimes cannot be evaluated independently because of the lack of symptomatic information at the appropriate level. In that case, the inclusion of the corresponding node in the malfunction hierarchy results in extraneous problem solving because the hypothesis can only be evaluated through the use of more detailed knowledge derived from its subnodes. Therefore, the node may need to be removed from the hierarchy so that its subnodes are directly evaluated in the diagnosis. This involves a tradeoff because the additional node may make clear the organization of the diagnostic knowledge and the problem-solving focus
in the diagnosis. Indeed, this situation suggests the need for additional symptomatic information which in some cases may be provided by installing more sensors.

2. Every piece of process equipment which handles fluids must be able to contain the fluid and provide a flow path for the fluid. For the fluid containment and path functions in the functional hierarchy, the corresponding nodes in the malfunction hierarchy would represent malfunction hypotheses associated with leakages and blockages, respectively. These malfunctions can be considered through an exhaustive approach, i.e. tracing through every piece of equipment and connecting pipe in the plant. However, this approach is computation expensive and often difficult to carry out because the required symptomatic information (e.g. sensor data) may not be readily available. Alternatively, leaks can be detected through the use of material balance calculations. However, in order to associate a leak with a specific piece of equipment, an individual material balance calculation for the piece of equipment needs to be performed. Due to the generally limited instrumentation in the CPP domain, it is possible that the flowrates necessary for the calculation are not measured by sensors. Thus, it is much more reasonable to consider only the likely places for the leaks to occur. In this way, when a leak is detected, the likely sources of leak can be physically check to locate the leak. For example, a leak in a valve (the valve itself and the fitting to the pipe) is much more likely than a
leak in the middle of a pipe section. Therefore, it is not unreasonable to exclude pipes from leakage consideration in the malfunction hierarchy. In this way, expert knowledge about the equipment and process is used to better specify the scope of the diagnostic system.

3. Functional decomposition only identifies systems and subsystems with specific functions. Although the majority of the diagnostic knowledge represented in the corresponding nodes of the malfunction hierarchy can be derived from a functional understanding of the equipment, it may not provide all the diagnostic knowledge required to evaluate the malfunction hypotheses. For diagnosis, there is often additional knowledge derived entirely from experience or other sources. These other forms of knowledge need to be included where appropriate because they can explain unusual phenomena, i.e. symptoms, which cannot be explained by direct functional reasoning alone. Therefore, when the diagnostic knowledge for evaluating the malfunction hypotheses associated with the functional systems and subsystems is encoded, this additional knowledge will be entered as additional knowledge constructs (referred to as knowledge groups) in the CSRL specialists. These knowledge groups represent additional possible situations, or even unusual events, which can lead to a malfunction of a functional subsystem although there are no direct functional relationships. For example, it might happen that when any work is done on the cooling water line upstream, the flow control valve has a good possibility of
getting stuck because the mineral deposits on the pipe may come off and plug the valve. This is a piece of highly compiled process specific experiential knowledge which cannot be revealed by functional reasoning alone because of the lack of a model for such a situation. In that case, a knowledge group may be added to the valve node in the malfunction hierarchy so that this would also be considered, possibly after knowledge groups for other modes of malfunction fail to establish.

For any plant configuration, similar functional subsystems often appear in more than one system. For example, at a general process level, control systems for flow, temperature, etc., are often present in more than one major functional system in the plant. At a more detailed process level, every feedback control system can be similarly decomposed into sensor, controller and final control element. There are also similarities in the intermediate levels. For example, although there are many different types of heat exchangers, the functional characteristics are generally the same. Therefore, although the overall functional hierarchy may be different from one plant configuration to another, smaller segments of the hierarchy remain essentially the same. These segments may be used as building blocks for new hierarchies when different plant configurations are considered.

When the functional hierarchy is recast as a malfunction hierarchy, it forms the major backbone of the malfunction hierarchy. For similar functional systems, the diagnostic knowledge, i.e. the discriminating symptoms to look for, derived from functional reasoning is often generally applicable to each of the systems. For example, high temperature alarm and sensor reading are general symptoms that
suggest the probability of a malfunction in the temperature regulatory system. Therefore, much of the diagnostic knowledge developed for one application is transportable to another application without significant modification. On the other hand, interactions among the functional systems are process specific and need to be examined for each application. Moreover, the diagnostic knowledge in the additional knowledge groups of the specialists in the malfunction hierarchy, which is not derived from functional relationships, is generally more compiled and possibly applies only to a particular plant. Therefore, a malfunction hierarchy is made up of nodes containing mainly knowledge derived from functional reasoning for widely applicable malfunction hypotheses. This hierarchy can be extended, if necessary, to account for interactions and to accommodate more compiled and process specific knowledge and new discoveries in the operation of the plant.

3.4 EXAMPLE FUNCTIONAL DECOMPOSITION

In this section, a processing plant which produces cyclohexane by hydrogenation of benzene is considered to illustrate the functional decomposition strategy discussed above [14]. The process flowsheet is shown in Figure 4. Using the flowsheet, Feed System, Reaction System and Product System can be identified.

For the Feed System, there are both liquid feed (benzene and catalyst) and gas feed (hydrogen) streams. The flow rate of the liquid stream is controlled by a Flow Control System. The gas feed is composed of both fresh and recycled hydrogen. Note that since both the hydrogen and catalyst recycles are combined with the fresh feed streams before they are fed to the reactor, the recycles are
Figure 4. Cyclohexane Production Plant Flowsheet.
also considered under the feed system. The concentration of the recycle hydrogen stream is controlled by a Concentration Feedback Control System using an analytical indicator and controller. The Flow rate and Pressure of the combined hydrogen feed stream are also controlled. Refer to Figure 5 for the functional hierarchy of the Feed System. For the Liquid Feed Flow Control System, the supply function of the liquid includes the Benzene Tank and Catalyst Supply. The catalyst supply consists of makeup and recycle from the filter. The equipment function in this case consists of the Pump. The control mechanism is a Feedback Flow Control System which consists of a Flow Sensor, Controller and Control Valve. Similar decomposition can be done for the other control systems in the Feed System.

The functional hierarchy for the Reaction system is shown in Figure 6. It follows the functional decomposition in a straightforward manner. The part that deserve particular attention in this hierarchy is the Temperature Control System. It illustrates how a cascade control system is handled in the functional decomposition. Specifically, the Steam Pressure Control System (secondary control loop) is considered under the Temperature Feedback Control System (primary control loop) by replacing the final control element node of the Temperature Feedback Control System with the Steam Pressure Control System node.

The decomposition of the Product System is also straightforward and is shown in Figure 7. This example shows how the systematic functional decomposition strategy can be used to construct a functional hierarchy by inspection of the process flowsheet. As discussed above, the resulting functional hierarchy may need to be
Figure 5. Feed System Functional Hierarchy.
Figure 6. Reaction System Functional Hierarchy.
Figure 7. Product System Functional Hierarchy.
slightly modified according to the diagnostic knowledge available when the malfunction hierarchy is constructed for diagnostic application.

3.5 CONCLUSIONS

Efficiency in hierarchical problem solving is derived from the levels of process abstraction captured by the hierarchy. The proposed systematic functional decomposition strategy allows a process to be decomposed into subsystems in various levels of detail in a straightforward manner given a process flowsheet. This enhances the application of the classificatory diagnostic approach. Control system sensors are identified in the decomposition of the control configuration such that malfunctions of these sensors are explicitly represented in the diagnostic hierarchy. Moreover, the functional basis of the hierarchy facilitates the formulation of diagnostic knowledge. Both qualitative and quantitative model knowledge can be derived from the functional understanding of the process subsystems identified in functional decomposition. This knowledge forms the backbone of the diagnostic knowledge, which is supplemented by other knowledge derived from operational experience of the process.
CHAPTER IV
SENSOR VALIDATION

4.1 SENSOR VALIDATION PROBLEM

In hierarchical classification, the individual malfunction hypotheses represented in the nodes of the diagnostic hierarchy are evaluated based on the observed symptoms primarily provided by sensors, alarms and operator observations. As a result, the quality of the diagnostic conclusions depends on the availability of valid sensor data. However, sensors are known to fail. Due to the generally limited instrumentation in the CPP domain, sensor failure presents a severe problem in the operation and monitoring of the plant. Although a sensor is just another type of equipment in the plant, it is unique because of its monitoring function. Indeed, it is the primary source of information about the operation of the process. This information is useful in both process control and diagnosis. Closed loop process controllers depend on the operating parameter measurements provided by the sensors to determine the control action. With respect to malfunction diagnosis, sensors supply an essential portion of symptomatic information required to determine the status of the malfunction hypotheses. As a result, sensor validation is a very important problem in the process plant domain.
In the following sections, the impact of sensor failure and effectiveness of various existing validation methods are discussed. In particular, the focus of discussion is on the impact of sensor failure on malfunction diagnosis and the applicability of the validation methods to diagnostic problem solving.

4.1.1 Impact of Sensor Failure

In the process plant domain, sensors can be generally classified into two distinct categories based on their intended functions. Due to their functional differences, failure of each category of sensors has different ramifications in the process operation and malfunction diagnosis.

In the first category (Category I) are sensors which are components of process control loops. Since the sensor reading is used in determining the control action, a sensor failure leads to erroneous control actions, which in turn can lead to other undesirable operating conditions in the plant. As a result, a failure of a sensor in this category adversely affects the process operation and so it can be the source of the malfunction.

The second category (Category II) includes all sensors which are not components of closed control loops. Failures of these sensors do not directly lead to other operating problems in the plant. However, the validity of these sensor readings is important because they provide useful information for process monitoring and malfunction diagnosis.

The distinction of these two sensor categories is important for diagnostic purposes. When a Category I sensor failure is identified, it is possible to associate
other observed symptoms of a malfunction to the sensor failure because the sensor failure can actually cause other symptoms. A Category I sensor is represented in the malfunction hierarchy because the sensor is a piece of equipment in the control system which can be a source of the control system malfunction. This has indeed been identified in the functional decomposition strategy discussed in Chapter III. When the hypothesis of a Category I sensor malfunction is considered during diagnosis, the symptomatic information is provided by the result of sensor validation. In this case, sensor validation identifies the sensor failure, which is also the source of malfunction. Thus, sensor validation simultaneously serves the purpose of malfunction diagnosis.

On the other hand, a Category II sensor failure, although it cannot cause a malfunction, can lead to two possible problems in malfunction diagnosis. First, if the sensor fails in such a way that it exhibits the discriminating symptom of a malfunction when there is no actual malfunction, then the malfunction hypothesis may be established based on the wrong sensor data. Second, the sensor can also fail in the opposite manner, i.e. it does not show the symptom of a malfunction although there is an actual malfunction. This failure masks the symptom of a malfunction hypothesis such that the hypothesis may be rejected based on the wrong data. Obviously, both failure conditions result in misdiagnosis. As a result, although identifying a Category II sensor failure does not solve the diagnostic problem, validating the sensor data before a final diagnostic conclusion is reached facilitates the diagnosis.
The sensor validation task can be decomposed into two subproblems, 1) identification of sensor failure and 2) elimination of failed sensor reading by substituting with an alternate value.

4.1.2 Modes of Sensor Failure

Ease of identification of sensor failure depends on the particular mode of failure under consideration. Therefore, it is important to list the possible modes of failure such that appropriate failure identification methods can be applied.

1. *Sensor reading is outside the sensor and/or process limits.* For example, a flow transmitter with a signal range of 3 to 15 psig may be calibrated to read flow rate from 0 to 12 gallon per minute (gpm). With such a flow transmitter, it is not possible to have a flow reading greater than 12 gpm, i.e. the sensor limit. The process limit can be illustrated with a level sensor. Obviously, the level reading cannot be greater than the height of the tank, i.e. the process limit.

2. *Sensor reading is changing at a rate that is outside the physical limit of the process or inconsistent with the sensor characteristic.* For example, it may not be physically possible for the pressure in a tank to change by 100 psi in 5 seconds.

3. *Sensor is not measuring but gives out a constant reading.* This mode of failure is characterized by the total absence of variation in the data.

4. *Sensor biases.* This is generally a slow developing problem leading to a gradual deterioration of the sensor performance. The sensor reading is
biased but still within the various limits discussed above and exhibits random variations.

Besides the modes of failure above, the sensor signal may also exhibit large fluctuations due to normal disturbances in the process. Although that is undesirable and makes it difficult to abstract the trend and normality information from the readings, it is not a sensor failure mode per se. Indeed, as long as the fluctuation is random and the signal is within the absolute and rate of change limits discussed above, it is possible to establish the range of normality in the reading. This type of problem is not further discussed in this work.

4.2 REVIEW OF SENSOR VALIDATION METHODS

Sensor validation is by no means a new problem. Over the years, many different methods have been developed [12,13,23,41,43]. However, the actual implementation of these methods, although common in aerospace and NPP domains, is seldom seen in the CPP domain. The reason is that the effective application of a sensor validation approach is strongly affected by the availability of sensors. In the aerospace and NPP domains, sensors are available for measuring essentially all useful data and substantial physical redundancy is present. On the other hand, hardware redundancy is rare in the CPP domain and quite often even useful measurements are not monitored by sensors.

Sensor failure analysis methods can be broadly classified into two categories. In the first category, the sensor data are checked independently without using any correlation with other sensor data based on any kind of process models. The
second category of methods make use of empirical or physical process and measurement models to correlate one sensor reading to others. Sensor failure modes 1 to 3 described in Section 4.1.2 above can be identified by checks in the first category. However, sensor biases (mode 4) often require the use of model correlation methods.

In the following sections, the various common validation methods are presented. These methods are organized around the two categories discussed above, i.e., independent data checks and model correlation techniques. Then, new techniques which are being developed are reviewed. These sections will be followed by a discussion on the weaknesses of the conventional validation methods as they apply to the CPP domain. Finally, an existing validation approach in the context of classification is discussed.

4.2.1 Independent Checks

Common validation methods in this category include limit checking and hardware redundant sensor comparison.

4.2.1.1 Limit Checking

This method involves the comparison of sensor readings to fixed absolute or rate of change limits. As discussed in Section 4.1.2, specific limits on the range and slope of sensor readings are imposed by both process and sensor characteristics. This method is particularly suitable for the CPP domain because it
does not require sensor redundancy. However, only large gross sensor failure can be identified.

4.2.1.2 Hardware Redundant Sensor Comparison

This method involves the comparison of measurements provided by redundant sensors. Obviously, this method requires hardware redundancy and, in order to identify the failed sensor, at least three sensors are needed. As a result, this method is generally inapplicable in the CPP domain.

4.2.2 Model Correlations

Besides sensor hardware redundancy, models can be used to provide additional sources of redundancy. That is the purpose of analytical redundancy techniques [11,21,40]. With the hardware and analytical redundant measurements, a choice among them is necessary. If three or more readings are available, simple majority voting or the more advanced parity space checking [13] can be done.

The most commonly used gross error detecting methods are based on statistical hypothesis testing. The basic idea is to test the measured data against alternative hypotheses. The null hypothesis, $H_0$, is that no gross error is present, while the alternative hypothesis, $H_a$, is that one or more gross errors are present. Examples are measurement and nodal tests [19,20,32], and the generalized likelihood ratio method [26,39].
In the following sections, four common model correlation techniques are presented, namely, analytical redundancy, parity space checking, measurement and nodal tests, and generalized likelihood ratio.

4.2.2.1 Analytical Redundancy

Analytical redundancy (AR) refers to the use of physical or empirical relationships between variables to generate redundant readings from a set of nonredundant sensors. For example, given the valve position and pressure drop across a valve, the flowrate can be estimated. This provides an analytical redundant reading of the flowrate through the valve, which is measured with an orifice meter. Most work with AR uses physical models based on conservation equations. However, empirical models generated by methods such as recursive least square have also been applied [21].

Since hardware redundancy is practically nonexistent in chemical plants, analytical redundancy of some form is important for sensor validation. However, large scale application is difficult, if not impossible, because, 1) measurements required by the models may not be available and 2) model parameters cannot be precisely determined.

4.2.2.2 Parity Space Checking

In this method, redundant measurements (sensor and analytical) are modeled as the underlying true value plus a measurement error term. Sensor failures are identified by recognizing inconsistencies between the redundant data sources. The
visibility of these inconsistencies is enhanced by eliminating the underlying measured quantity from the data. This is done by creating linearly independent relationships between the measurements, which are called parity equations. As a result, for \( n \) redundant readings, \((n-1)\) parity equations can be defined and each equation contains only the error terms, including those that may result from a sensor failure.

The space spanned by the parity equations is called the parity space. The variables generated from the parity equations become the coordinates of the parity vector, which lies in the parity space. The magnitude of the parity vector represents the inconsistencies between the various redundant measurements, and its direction identifies the failure of a sensor or the plant unit from which the analytical measurement is derived.

This method requires the availability of at least three-fold redundancy so that two or more parity equations can be derived. This requirement hinders the application of this method in the CPP domain.

4.2.2.3 Measurement and Nodal Tests

Measurement test is based on a statistical test for outliers of the least-squares residuals. In order to apply this test, all the data in the process network under investigation must first be adjusted by least-squares analysis. Then the differences (residuals) between the least-squares estimates and the original measurements are calculated. If any residual is greater than the critical value, i.e. an outlier, a gross error in the corresponding measurement is identified.
Nodal or constraint test is based on analyzing the nodal imbalance. Given a steady state constraint equation ($\Sigma$ Inputs - $\Sigma$ Outputs = 0), the left hand side of the equation is called the nodal imbalance. If the value is greater than the critical value, there are one or more gross errors in the measurements associated with the constraint.

The critical value for these tests is selected based on the desired level of significance. For example, for a test at the 95% significance level, $\alpha = 0.05$ and therefore 1.96 times the standard deviation is the critical value.

Both tests are generally for linear constraint processes only. The major problem with the measurement test is that least-squares analysis (data reconciliation) of all the data in the process network is necessary before the method can be applied. On the other hand, the constraint test can only detect the presence of gross errors in the measurements, an additional procedure is required to specifically identify them.

4.2.2.4 Generalized Likelihood Ratio

The generalized likelihood ratio (GLR) approach provides a framework for identifying any type of gross errors that can be mathematically modeled. To use this method for gross error identification, a model that describes the effect of each type of gross errors is necessary. Specific sensor failure modes such as impulse and step, and process leaks can be modeled. The null hypothesis is tested against the alternative by using the generalized likelihood ratio. If the ratio is greater than the predetermined threshold value, a gross error is declared.
The major drawbacks with this approach include 1) each type of failure must be modeled and tested for specifically, thereby precluding the detection of all possible failures, 2) the required linear model is based on a specific operating point so that it is only valid for perturbations about that point, and 3) large computational burden in calculating the likelihood ratio.

4.2.3 New Techniques

In recent years, the use of pattern recognition techniques for sensor validation has been investigated [24, 42]. The unique feature of this technique is that it does not use an analytical model of the plant or associated systems to provide an estimate of sensor readings and the overall plant state. Instead, pattern recognition based on learned states from previous operations is used. The major difficulty in the application of this technique is that periods of operation which are typical for the process conditions of interest must be selected. Then past data collected for these operating periods are used to "teach" the system what to expect from each of the signals involved in the system. Thus, the effectiveness of the system depends on how representative of the process operating conditions the training data are.

An approach which utilizes a number of independent sensor validation modules on a set of sensors and combines the results of the various tests is being developed [43]. The signal validation is performed at primarily steady state or quasi steady state plant operational conditions. This approach requires the availability of properly conditioned plant signals through preprocessing. In addition to modules which make use of standard statistical testing techniques, one module employs
qualitative signal analysis similar to the reasoning used by an operator. Since this approach is still being developed, no details on the qualitative validation are provided.

4.2.4 Existing Validation Method in Classification

Within the context of hierarchical classification, an expectation-based sensor failure identification method has been applied in the NPP domain by Hashemi [16,17,18]. In that work, precompiled expected symptoms of malfunctions are used to identify suspicious data during pattern matching in a specialist node in the diagnostic hierarchy. In this way, pattern matching is used to both evaluate the malfunction hypothesis and identify sensor failure. If the majority of the expected symptoms of a malfunction hypothesis is observed while there are one or two pieces of unexpected data (not a perfect match of the expected symptom pattern), then the unexpected data are collected into a suspicious data list and diagnosis continues as usual. When the preliminary diagnostic session is finished, the suspicious data are validated by comparison with new values which are based on inferences with the initial diagnostic conclusions and qualitative analytical redundancy. Since preliminary diagnostic conclusions are reached before any identified sensor failure is eliminated, all malfunction hypotheses which use the failed sensor must be evaluated again to obtain the final diagnosis. There are several issues with this validation approach which deserve further discussion.

1. The approach relies on the identification of suspicious data through the use of precompiled combinations of mostly expected symptom patterns
along with one or two unexpected symptoms. The ramifications are as follows:

a) All failure identification knowledge must be somehow captured in the pattern matching format. That restriction makes the knowledge engineering difficult. Furthermore, if a malfunction hypothesis is evaluated based on the results of calculational tests, this approach is inapplicable. In that case, pattern matching with the sensor data is not done.

b) Patterns which have no relevancy in malfunction diagnosis must be included in the pattern matching table. That is undesirable because the resulting diagnostic specialist is more difficult to modify and maintain.

c) The encoding of many extra possible patterns is required. These patterns are not generally applicable to other plants. This does not present a major problem for nuclear plants because there are only a few different nuclear reactor designs in use. However, a more general approach is highly desirable in the CPP domain because of the diversity of chemical processes.

2. The approach only considers sensor failure as the cause of the unexpected symptoms. However, interacting multiple malfunctions can have additive or cancelling effects on a particular symptom used in the pattern matching. As a result, the parameter value may be different from what is expected based on the consideration of a single malfunction
hypothesis, i.e. an unexpected symptom exists. It is not possible to
distinguish the two problems based on the expectation alone.

3. It is assumed that the data have gone through certain conventional
validation routines such as limit checking and like sensor comparison.
Thus, the only purpose of the expectation-based method is to identify
common cause failure or instrumentation calibration drifts. This is a
valid assumption for nuclear plants because of the physical sensor
redundancy and the various data screening and management systems
generally implemented in the plants. However, sensor redundancy is
virtually nonexistent and data analysis is not widely applied in chemical
plants. As a result, these validation methods should also be taken into
consideration in the overall diagnostic approach.

The expectation-based sensor failure identification method is not appropriate
for diagnosis in the CPP domain. First, the use of quantitative methods is found to
be very useful for evaluating certain malfunction hypotheses. Since the sensor data
are used in the calculation instead of directly used in the pattern matching in this
case, the expectation-based approach cannot be used to identify any sensor failures.
Second, although it is not the focus of this work, our diagnostic framework is being
extended to resolve interacting multiple malfunctions [36]. As discussed in point 2
above, interacting malfunctions can render the expectation-based validation method
ineffective.
4.2.5 Weaknesses of Conventional Methods

Generally speaking, classical sensor validation methods are rarely discussed in the context of malfunction diagnosis in the CPP domain. Indeed, the primary focus of many of these methods is to identify and remove gross errors in the data when the plant is operating normally such that reconciled data can be provided to accounting and optimization programs in the plant. As a result, many of those methods developed assume normal steady state operating conditions [23] and are limited to linear or linearized constraint equations. Most methods only test for measurement errors so that only sensor failures can be identified although the actual sources of the gross errors might include a process leak. On the other hand, as discussed above, the GLR method can be used to identify any type of gross errors that can be mathematically modeled. However, each type of gross error must be modeled and tested for specifically. For multiple gross errors, the general approach requires serial identification and compensation [23], or elimination [31], of all gross errors. In the serial compensation strategy, if a gross error is identified, it is compensated by its estimated magnitude. This is repeated until no further gross errors are detected. On the other hand, with the serial elimination strategy, the identified gross error is deleted. The serial compensation strategy is applicable to all types of gross errors while the serial elimination strategy is only applicable to gross errors directly associated with measurements. As a result, the serial elimination strategy is not suitable for the GLR method.

Classical statistical gross error identification methods are generally applied in such a way that a large number of plant sensor data are manipulated together,
using a large set of often complex constraint equations. As a result, these methods are extremely computation expensive. Moreover, many constraint equations cannot be applied because 1) the instrumentation is limited, and 2) model parameters often cannot be determined accurately enough to support the kind of precise manipulation called for in the statistical tests. Where these methods consider the measurement process, some estimate of the statistics of the measurement noise is required. Some methods require the full statistical components, such as covariance matrices, of the noise. Such detailed statistics of process noise are not generally available. Only in especially well-constrained process network has the use been demonstrated in the CPP domain. One such example is the metering system of a steam network, where the physical stream properties (steam in this case) are well defined and only linear constraint equations (material balances) are necessary.

At the present time, this type of analysis is not widely employed in process plants so that it is not possible to assume the general availability of statistically validated data. Moreover, with respect to diagnosis, deviation from steady state conditions and malfunctions other than sensor failures have to be considered. In that case, detailed and robust process models, which are applicable over a wide range of process operation are required for these methods. This further increases the complexity of the validation problem and makes many statistical analysis methods ineffective. As a result, statistical gross error identification is not a very attractive general validation strategy for use in our diagnostic problem-solving framework.
CHAPTER V
SENSOR VALIDATION STRATEGY

5.1 GENERAL CHARACTERISTICS

The sole intention of the validation strategy developed in this research is to eliminate any non-random sensor errors including biases, such that the diagnostic conclusions can be reached based on valid data. The focus of the validation here is to support the diagnostic task. In comparison to classical sensor failure identification techniques discussed in Chapter IV, there are three important differences. First, although classical techniques are generally applied to situations where only sensor failures are present, validation under general malfunctioning conditions (malfunctions other than sensor failures may be present) are considered in this strategy. Second, instead of applying the validation procedure to all the sensors in the plant independent of diagnosis, validation is only applied selectively to those sensors which are involved in the diagnosis. Finally, instead of applying uniform testing statistics to all the sensors, flexibility in the methods is stressed in this strategy. These important characteristics, along with other related issues will be discussed in the following sections.
5.1.1 Impact of Malfunctions on Validation

The consideration of malfunctions in the plant has major ramifications on the sensor validation process. If only sensor errors are considered, i.e. no other process equipment failures, the process model equations including material and energy balances are valid. Therefore, when an equation is violated based on the observed sensor data, it can be concluded that there is an error in at least one of the sensor measurements used. However, when other equipment malfunctions are also considered, then the model equations themselves may not be valid. For example, material balances are violated when there is a process leak. As a result, when the equation is violated, it can be due to either a process malfunction or a sensor error, or even both.

One way to resolve this problem is to use relationships which are not affected by the malfunctions. Among the various available relationships which will be discussed in Section 5.1.4 below, there are two different categories of such relationships. These categories differ in the extent to which the relationships are not affected by malfunctions.

1. Some relationships are generally applicable in spite of equipment malfunctions. These include unique process specific relationships between sensor readings which are not based on conservation equations. For example, referring to Figure 1, the reactor level can be directly estimated by the temperature sensors along the wall of the reactor because the temperature in the slurry is different from that in the vapor phase.
2. Some relationships are not affected by certain malfunctions. Thus, they are valid under both normal and some malfunction conditions. For example, considering a tube and shell heat exchanger, energy balances can be written for the tube or shell side alone, or a combined tube and shell side balance can be used. Since the single side balances make use of the overall heat transfer coefficient, they are not valid when there is a heat exchanger fouling problem. However, the coefficient is not used in the combined balance so that it is valid even when the heat exchanger is fouled. Therefore, the combined balance is more widely applicable.

Since various forms of relationships need to be used in the validation, the validation strategy developed in this work involves a framework which allows any unique relationships to be incorporated. This will be further discussed in Section 5.1.3 below.

5.1.2 Diagnosis and Validation Integration

The validation strategy developed in this research integrates sensor validation with malfunction diagnosis. The validation strategy is designed to function as an auxiliary problem-solving module to the diagnostic problem solver. As a result, instead of trying to apply the strategy exhaustively to all sensors in the plant, many of which are not even needed for any given diagnostic scenario, the strategy is applied only to those sensors which are actually used during the diagnosis. By integrating the validation with malfunction diagnosis, the context of the current diagnosis specifies the sensors which need to be validated.
The diagnosis is based on hierarchical classification, which has been discussed in detail previously. Basically, it makes use of a malfunction hierarchy which is primarily based on a systematic functional decomposition of the process into subsystems of various levels of process detail. The diagnosis proceeds from the general malfunction hypotheses at the top of the hierarchy to more detailed sub-hypotheses at lower levels in the hierarchy. If any malfunction hypothesis is rejected, then all associated sub-hypotheses are also rejected from further consideration. In this manner, the diagnostic strategy effectively focuses the search such that only a portion of the solution space needs to be explored for a given diagnostic situation.

Each hypothesis node in the hierarchy represents a local problem solver with knowledge specific for establishing or rejecting the associated hypothesis. Sensor measurements provide an important source of symptomatic information for evaluating the malfunction hypotheses. As a result, when a malfunction hypothesis is evaluated in a node, the appropriate sensor readings are requested by the diagnostic problem solver. That is the point when the sensor validation strategy is initiated. In this way, the validation strategy takes advantage of the hierarchical classification by letting the distributed diagnostic problem solving direct the focus of the validation to those data that are actually needed for evaluating the hypotheses. Since hierarchical classification is efficient in searching the diagnostic solution space, i.e. only a small number of the malfunction hypotheses represented in the hierarchy need to be considered in detail, it is expected that the sensor validation strategy needs to be applied to a small portion of the sensors in the
Although not proven (a direct comparison is difficult because the validation techniques used are quite different), it is believed that this strategy will result in improved computational efficiency when compared with an exhaustive validation strategy which is applied to all sensors in the plant. It should be noted that since the systematic functional decomposition strategy facilitates the construction of an effective diagnostic hierarchy which captures the various levels of process detail, it indirectly leads to efficiency in the sensor validation.

If a sensor failure is identified in the validation, the reading is first replaced with an alternate value before a diagnostic conclusion is reached. Thus, the diagnostic conclusion is only based on validated data and it is not necessary to redo the diagnosis. However, it should be noted that under certain conditions, an alternate value cannot be determined to replace a failed sensor reading. In that event, UNKNOWN is used as the final value for diagnosis. This will be discussed in detail in Section 5.3.

5.1.3 Flexibility

In classical validation techniques, only quantitative equations can be used. Also, uniform testing statistics are applied to all the sensor data. However, instrumentation in chemical plants is limited to the extent that even useful measurements often are not monitored. As a result, exact model correlations can only be selectively applied to some sensors because the data needed in the model may not be available. For the other sensors, it is necessary to rely on approximate or short cut relationships between sensor readings. Thus, many of those statistical
validation methods are very difficult, if not impossible, to apply. Recognizing the severe limitation in the instrumentation, the validation approach is designed to allow the incorporation of any kind of applicable relationships. These relationships can be empirical or physical, and they can be exact or approximate. Thus, different relationships can be applied to different sensor types, process configurations and operating conditions. In other words, any specific relationship designed for a particular sensor can be incorporated into the validation strategy. In this way, relationships for validation which are particularly suitable for incorporation into the malfunction diagnosis problem-solving framework can be developed.

It should be emphasized that the focus of this research is on the organizational framework which allows these process specific relationships to be incorporated. The actual relationships themselves, which are highly process specific, are developed only for the purpose of illustrating the computational methodology involved in the validation approach.

5.1.4 Sources of Alternate Data

Based on the discussion above, the relationships which can be used to determine alternate sensor values warrant further discussion. The assumption here is that the plant and instrumentation are fixed. The development of the relationships may well help identify the need for additional sensors or different sensor placements so that the data can be validated.

The sources which can be derived for any process configuration primarily depend on the availability and location of sensors. The availability of sensors
determines whether hardware redundancy is available and which model correlations can be applied. The location of sensors often determines whether any unique relationships can be derived. Furthermore, the reliability history of the various sensors is also an important consideration when these sources of data are used in the validation.

5.1.4.1 Hardware Redundant Sensors

Hardware redundancy is very uncommon in chemical process plants. It can at most be found in a few important measurements. Since the same operating parameter is measured by the redundant sensors, the various readings are directly related (equivalent) and no process correlation is necessary. Thus, hardware redundancy is generally applicable whether or not the process is malfunctioning. However, hardware redundancy alone cannot identify common mode sensor failures, which have similar effect on all the redundant sensors. Moreover, it should be noted that occasionally even the direct relationship between the redundant sensors may implicitly assume certain conditions, which must be satisfied for the relationship to apply. For example, referring to the process in Figure 1, the temperature sensors along the wall of the reactor provide redundant measurements of the reactor temperature. However, when a temperature sensor reading is related to another sensor reading, it is implicitly assumed that both sensors are in the same phase, i.e. both in the slurry or vapor phase. Otherwise, the two readings are not comparable because each phase has a different temperature. As a result, in order to obtain an alternate reactor temperature reading using the redundant sensors, the
reactor level measurement needs to be checked to make sure that the sensors are in the same phase.

Physical sensor redundancy can be categorized into two classes: 1) one redundant sensor, and 2) two or more redundant sensors. Furthermore, the redundant sensor(s) may or may not be at the same location or using the same measuring technique. The ramifications are as follows.

If only one redundant sensor is available, then one of the two sensors must be known to be more reliable based on past operation. Otherwise, any conflict between the two sensor readings cannot be resolved without using additional sources of data. One sensor may be more reliable than another because the measuring technique involves more robust equipment, or the sensor is placed at a location which is less susceptible to failure. Generally speaking, multiple techniques have been developed for sensing various process variables. For example, flowrate can be measured by an orifice meter or a turbine meter, among others. Each type of sensor has a different accuracy and potential problems, thereby leading to a different operational reliability under the conditions of the process. The importance of the sensor location can be illustrated with the following example. Referring to Figure 1, the gas analysis system above the condenser is known to be more reliable than the one at the reactor. The reason is that for the analysis system at the reactor, hydrocarbon vapor may be collected in the gas sample along with oxygen and nitrogen. This introduces a bias into the oxygen concentration determined by the analysis system. On the other hand, above the condenser, the gas is free of
hydrocarbon vapor. Thus, the gas analysis system above the condenser is likely to be more reliable.

If there are two or more redundant sensors, i.e. total of three or more sensors, then majority voting or parity space can be used to resolve any conflict among the sensor readings. However, as discussed above, hardware redundancy alone cannot identify common cause failures. In order to identify common cause failures, some form of analytical redundancy is necessary. It should be noted that this problem is rare in chemical plants because it is uncommon to have even one redundant sensor.

5.1.4.2 Direct Qualitative Estimate by Independent Sensors

In order to apply this method effectively, the relative reliability of the sensors should be known such that when a conflict arises, the more reliable data can be chosen. The qualitative process relationships in this case are often generally applicable because they are based on unique characteristics of the process configuration which are rarely, if at all, invalidated by malfunctions. Although the relationship is only approximate, it does provide a reliable alternate value. This type of approximate relationship is very important in CPP because more exact relationships often cannot be used due to the lack of necessary sensor measurements. For example, referring to Figure 1, the level indicator is known to be unreliable because of the slurry in the reactor. Since there are temperature sensors along the wall of the reactor and there is a large temperature difference between the slurry and vapor phase, the temperature readings can be used to
provide a reliable estimate on the range of the level reading. The level measurement provided by the level sensor can then be checked against the range provided by the temperature sensors.

5.1.4.3 Calculations With Empirical or Process Models

In this case, material and energy balances, process model equations and empirical correlations are used to determine the alternate value quantitatively. Due to the general lack of hardware redundancy in chemical plants, this represents the primary source of alternate values for the sensors. Three important points should be noted.

First, only a small fraction of the balances which can be directly derived from a basic understanding of the process can be applied because quite often the measurements required for the calculations are not available. As a result, the calculational models must be formulated with the existent process instrumentation in mind. For example, a material balance can be written for every piece of process equipment in the plant. However, many of these balances cannot be applied because flowrates of one or more of the input and output streams are not measured. By considering the available sensor readings, a material balance can be written over several pieces of equipment such that the unmeasured streams are not included in the equation.

Second, mathematical models are generally developed based on certain assumptions on the process conditions, basically the normal operating conditions. Therefore, as discussed above, a malfunction may invalidate the underlying
assumptions so that the model is not applicable. For example, a material balance assumes that there is not a process leak. If there is an actual leak, then the balance equation is not valid because the magnitude of the leak is not accounted for in the equation. As a result, if the application of the model equation leads to a conflict situation in the validation which cannot be resolved, these assumptions need to be checked to make sure they are valid.

Third, due to inherent process disturbances and noise in the sensor readings, tolerance limits for normality and trend need to be specified for the data. Moreover, for any calculation using the data, the tolerance for the closure of the equation also needs to account for the uncertainties in the process model parameters. As a result, an appropriate tolerance limit must be determined based on past data and associated with each calculation. This process specific judgement on the tolerance has an impact on both malfunction diagnosis and sensor validation. With respect to malfunction diagnosis, it is assumed that when a malfunction occurs, "large" deviation from normal condition will be observed in some sensor readings. Therefore, the diagnostic problem solver is looking for these large deviations as symptoms, including trend in the data, deviation from some normality band, and nonclosures of model equations. Of course the abstraction of what is large here is relative and depends on the tolerance limit. If the tolerance band is too wide, then symptoms of an actual malfunction become unobservable. On the other hand, if the tolerance is too small, then symptoms are observed even when a malfunction is not present. The tolerance has similar effects on the validation. Basically, with the use of the tolerance, it is not generally possible to
identify sensor biases which are small enough to be well within the normal variation range of the data.

5.2 SENSOR VALIDATION ALGORITHM

When sensor data are requested by the diagnostic problem solver for evaluating a malfunction hypothesis, the following validation procedure is first carried out for each piece of data requested. If a sensor error is identified, the validation procedure attempts to eliminate the error by replacing the original sensor reading with an alternate value. Thus, the diagnosis is always performed with validated data. If any data are used in more than one hypothesis, then the validation procedure is done only the first time the data are requested. The flow diagram for this algorithm is shown in Figure 8. The important functional aspects of the algorithm will be discussed in Section 5.3.

1. The sensor reading is checked against the absolute sensor or process limit, whichever is more restrictive. These limits are specified based on the sensor range and process properties. If the value is outside the limit, then the reading is erroneous. Go to Step 9.

2. The rate of change, i.e. slope, of consecutive data over the specified past time interval is calculated and checked against the rate of change limit imposed by the sensor or process time response characteristics. If the rate of change is outside either of the limits, gross error is present in the data. Go to Step 9.
Figure 8. Sensor Validation Algorithm.
3. The standard deviation is calculated using the same set of data as in Step 2. If the standard deviation is zero, i.e. no variation in the data at all during the time interval, the sensor is probably not measuring. Go to Step 9.

4. Check to see whether or not the reading is from a Category I sensor (sensor measurements used in a process control system). If yes, an alternate value needs to be determined to confirm. Go to Step 11.

5. Perform data abstraction on the reading to determine its normality classification. If the reading is either HIGH or LOW, go to Step 11.

6. Using the slope calculated in Step 2, data abstraction is done to assign the trend of the past data. If the data show a significant UP or DOWN trend, go to Step 11.

7. Check the reliability history of the sensor. This requires a notion of the failure probabilities of the various sensors based on their past operation. If the reliability is low, then an alternate value needs to be obtained to confirm. Go to Step 11.

8. Data that get to this step are within the absolute and rate of change limits, not dead set at a constant reading, not coming from a control system sensor, normal and steady, and have a high reliability ranking. The probability of a sensor failure is very low here so that the data is assumed to be valid. Go to Step 13.

9. An alternate value obtained from a different source, such as a redundant sensor or a calculation based on empirical or physical models, is used
to replace the original data which failed the testing in Step 1, 2 or 3. Since a comparison with the original reading is not possible here, the data used in determining the alternate value are first validated.

10. During the validation of the sensor readings involved in determining the alternate value, one or more of the readings may have been set to UNKNOWN. As a result, it is possible that an alternate value cannot be obtained.

a) If a new value has been obtained, then it is used as the final data for diagnosis. Go to Step 13.

b) On the other hand, if no value has been obtained, then if another source of alternate value is available, go back to Step 9. Finally, if after exhausting the sources for alternate data, still no alternate value has been obtained, then the final data is taken as UNKNOWN. If UNKNOWN is selected, the user is presented with the situation and allowed to input a new value. Go to Step 13.

11. An alternate value is obtained from a different source (as in Step 9). However, instead of replacing the original reading, the new value is used to compare with the original reading to confirm its validity. Also, since a comparison is done, the data used in determining the alternate value are not validated first. Instead, they may be validated when a conflict arises in the comparison.
12. The comparison procedure is as follows.

a) If the alternate value agrees with the original reading, then the original reading is taken as valid and used as final data for diagnosis. The other sensors, the reading of which are used in determining the alternate data, are also taken as validated. Go to Step 13.

b) If the two sources of data do not agree, a choice is necessary. (This additional conflict resolution will be discussed in detail in the next section.) Basically, it makes use of additional sources of data, reliability rankings of the various sources of the data, etc., to resolve the conflict and determine the final data for diagnosis.

i) If the conflict is resolved, the final value would have been selected during the conflict resolution. Go to Step 13.

ii) If the conflict cannot be resolved, then if the sensor being validated has high reliability, the original reading is used. Otherwise, UNKNOWN is used as the final data. The user is presented with the conflict situation and allowed to input a new value. Go to Step 13.

13. The validation stops and the final value is returned to the problem solver which requested the value before the start of the validation.

5.3 DISCUSSION ON THE ALGORITHM

The algorithm can be broken down into four distinct functional aspects. Each of these aspects is discussed in detail in the following sections.
5.3.1 Independent Data Checks

The checks performed in Steps 1, 2 and 3 in the algorithm are designed for identifying the first three modes of sensor failures described in Section 4.1.2. As a result, if the data passed these tests, only sensor biases may be in the data, i.e. sensor failure mode 4. On the other hand, if the data failed one of these checks, then errors are obviously in the data. These checks are applied first because they are independent checks on the sensor characteristics, which can be applied without using readings from any other sensor. Since the data are checked against various prespecified limits, no complicated comparison logic is necessary. However, these tests are only capable of identifying large, obvious modes of sensor failures.

5.3.2 Error Elimination

Once a sensor failure has been identified in the independent checks above, an alternate value needs to be determined to replace the erroneous reading. This is the purpose of Steps 9 and 10. Since the original sensor reading has been rejected, it is not possible to compare it with the alternate value that is determined. Thus, in order to assure the validity of the alternate value, the validation procedure is first applied to those sensor measurements used in determining the alternate value. However, it is possible that during the validation of the sensor data, one or more of the readings have been replaced with UNKNOWN. The reason may be that an alternate value cannot be obtained to replace a rejected sensor reading or that a conflict situation cannot be resolved, and the user is not able to supply a value. In that case, the alternate value cannot be determined. If all the sources of
alternate value have been exhausted and still no value can be determined, then
UNKNOWN has to be used as the final value for the sensor. When UNKNOWN
has to be used in the validation, the user is alerted of the situation so that physical
checks may be performed and a value provided to the system. This step (Step 10)
allows the user to take control when the validation module fails to determine a
validated alternate value for the diagnostic system.

5.3.3 Rigorous Validation Application Criteria

Steps 4 to 7 include checks which are done simply to determine whether
rigorous validation can be avoided. Since only biases may be in the data that pass
the checks in Steps 1 to 3, the goal here is to determine whether there is a low
probability that significant biases are present in the data. If the probability is very
low, rigorous validation is not performed and the original sensor reading is taken
as valid. The premise is that it is possible to get a rough idea on the probability of
a gross error in the data by looking at the sensor characteristics and the current
data qualitatively. The implication is that if the data pass all these tests, then the
probability of a sensor error is very low.

This part of the algorithm introduces a heuristic element into the algorithm
which resembles what a human being would do to minimize the computational
demand. It also improves the performance of the validation module. These checks
themselves are much easier to do than the rigorous validation. More importantly,
by making it possible for data to go through the algorithm without invoking the
rigorous validation, data with very low failure probability can be collected for
validating other sensor readings where rigorous validation is required. This decreases the occurrence of complicated conflict situation in the validation algorithm, which may or may not be resolvable. Note that if the conflict cannot be resolved by the algorithm, then the final value used for diagnosis is determined based on the reliability of the sensor (Step 12). If the reliability is low, UNKNOWN has to be used, which is an undesirable result. Thus, both computational efficiency and results of the validation are improved by these steps.

Step 4 checks whether or not a control system sensor is involved. The behavior of a control system sensor is unique in that any sensor bias is not directly observable as a deviation in the reading from the normal condition as long as the controller is functioning properly. The reason is that the control system will adjust the manipulated variable accordingly so that the controlled variable, i.e. the sensor reading, is at the specified setpoint. Although the sensor reading is at the setpoint, the actual operating parameter is not at the setpoint, which is the desired operating condition. As a result, the process operation and eventually the product quality is adversely affected. Due to the unique behavior of control system sensors, their readings are always examined by the rigorous validation procedure. Thus, the other criteria are checked only for readings from sensors that are not in control systems.

Steps 5 and 6 are based on the premise that if any sensor data are not normal or exhibit a trend, then the rigorous validation should be performed to confirm the data. The process is assumed to be initially at some normal operating state. As a result, if the sensor reading does not show any deviation from the normal condition, then there has not been any significant change in that operating
parameter. Hypothetically, this can also happen if a malfunction affects the parameter in one direction, while at the same time, the sensor calibration drifts with the same magnitude in the opposite direction so that the effects are almost totally cancelled out. However, this scenario is taken to be extremely rare, if at all possible. The reason is that the magnitude and the rate of degradation due to the malfunction must match closely with the sensor drift in the opposite direction in order to cancel out the effects. As a result, if the reading is normal and steady, then it is probably valid and not affected by any malfunctions that may be present in that diagnostic session.

Finally, the reliability history of the sensor is checked in Step 7. Past operational performance history of sensors can help predict the probability that a sensor will fail at the present time. If a sensor has a high reliability ranking, the probability of a sensor failure is low, especially when the reading has satisfied the other conditions in the algorithm. On the other hand, some sensors are known to be unreliable. For example, as discussed previously, the level sensor in the reactor process shown in Figure 1. In that case, rigorous validation is always performed on the sensor reading. Other aspects of the reliability ranking will be discussed later in this chapter.

5.3.4 Rigorous Validation and Conflict Resolution

Steps 11 and 12 are designed for the identification and elimination of biases in the data. An alternate value is obtained and then compared with the original sensor reading. When the alternate value does not match the original sensor
reading, i.e. a conflict situation, additional problem solving is necessary. This conflict condition can only occur when the original reading has not been eliminated, i.e. a sensor failure has not been identified, so that the alternate value is simply obtained to try to confirm the original value. If the original sensor measurement has been rejected, a comparison is not possible and the various scenarios have been discussed in Step 9 in the algorithm.

If the alternate value is in conflict with the original sensor reading, it may be due to a failure in any of the sensors involved. For an alternate value provided by a hardware redundant sensor, either one of the sensors may have failed. If the alternate value is based on a direct estimate or a calculation, then one or several other sensor readings are involved, and a failure in any of the sensors can cause the conflict. It is also possible that the model relationship itself is not valid because of a malfunction. In the validation algorithm, if the two values are in conflict, then the following additional steps are carried out.

1. A second independent source of an alternate value using a different set of sensor readings, which does not include the original sensor reading under scrutiny, is used if possible.
   a) If the two independent alternate values agree, then the original sensor reading is ruled invalid. Thus, it is replaced with the alternate value.
   b) If the second alternate value agrees with the original sensor reading, then the original reading is taken as valid. This implies that there is a failure in the sensor(s) used in the source of the
first alternate value. The validation algorithm is thus applied to those sensor readings used in the first but not the second source to try to identify and remove the failure.

c) If none of the values agree, then there is probably one or more errors in the sensor readings involved in the alternate data sources. The validation algorithm is first applied to those sensor readings used in both sources of data. The other readings are validated only when no error is found in the readings common to both sources. After any identified error is eliminated, new alternate values for the original sensor reading under investigation are determined again and the comparisons are repeated.

2. If a second independent source of an alternate value is not available, then the assumption of the model relationship needs to be checked. If the relationship is valid, then the validation algorithm is applied to each of the sensor readings used in the first source of alternate value. Otherwise, the relationship cannot be applied and the original reading cannot be validated.

a) If all the sensor readings used in determining the alternate value are found to be valid, then the alternate value is valid. Since the original reading does not match the alternate value, it is ruled invalid. The original sensor reading is replaced by the alternate value.
b) If a failure has been identified and eliminated in validating the other sensors, then the alternate value is determined again using the validated values. The new alternate value, which is based on the validated data, is chosen as the final value for the original sensor being validated.

It is possible that the validation process for the sensor readings used in determining the alternate value cannot be completed without using the original sensor reading currently being validated. Also, one or more of those sensor readings may be replaced by UNKNOWN during the validation. That results in a conflict situation which cannot be resolved. In that event, since a failure has not been identified in the original sensor reading by the independent tests in the first three steps of the validation algorithm, the original reading is used as the final value as long as the sensor has a high reliability ranking. On the other hand, if the reliability is low, the final value is taken as UNKNOWN. The implication of this final decision step is that readings of low reliability sensors, which are likely to fail based on past experience, will not be used in diagnosis unless they can be confirmed in the validation. Whenever the conflict cannot be resolved, whether the original reading or UNKNOWN is used, the user is presented with the situation so that physical checks can be done and the final sensor value input to the system.
5.4 OTHER RELATED ISSUES

5.4.1 Algorithm Failure Conditions

From the above discussion on the validation algorithm, two failure conditions can be identified. First, the sensor data do not satisfy the independent checks in Steps 1 to 3 and an alternate value cannot be determined in Step 10. As a result, the algorithm sets the value to UNKNOWN. Second, there is a conflict condition that cannot be resolved in Step 12. In that case, the original reading is used if the sensor reliability is high. Otherwise, UNKNOWN is used.

The use of UNKNOWN is clearly undesirable because it decreases the strength of the diagnostic conclusion that can be obtained based on the data. However, it is much better to have a weaker diagnostic conclusion than to have a wrong conclusion based on incorrect data.

Whenever the algorithm fails to complete the validation successfully, the user is presented with the failure condition of the algorithm. The algorithm also prompts the user for an input. In this way, physical checks on the sensor may be performed by the user and a value provided to the system for diagnosis. This step allows the user to come in and take control when the algorithm is unable to validate the sensor data.

5.4.2 Reliability ranking

Although reliability ranking of sensors provides an additional piece of useful information for sensor validation, it is generally neglected in conventional validation techniques. Past operational performance history of sensors can help
predict the likeliness that a sensor will fail at the present time. This can be combined with other relevant information to streamline the validation algorithm. The use of reliability ranking has been discussed in various validation steps above, this section tries to present the whole idea of reliability ranking in sensor validation.

5.4.2.1 Role of Reliability Ranking

1. **Focus the validation process**

   As discussed in Section 5.3.3, reliability ranking is a test criterion in deciding whether rigorous validation needs to be applied to the sensor data. In this way, it helps focus the rigorous validation on the readings of low reliability sensors, which need the validation the most. Indeed, low reliability readings are never used unless they can be confirmed by the sensor validation module.

2. **Prioritize the validation**

   Conflict situations in the validation algorithm discussed in Section 5.3.4 may require the sensor data involved in the alternate data source to be validated. Also, as discussed in Section 5.3.2, when an alternate value is determined to replace a failed sensor reading, the sensor data used in the alternate data source are always validated first. In this case, instead of randomly validating the group of sensors involved, the sensors should be ordered according to their reliability ranking. It is beneficial to validate high reliability sensors first because they are more likely to be
valid for any given time. As a result, a set of validated data can be collected quickly for use in validating other sensor data.

This result is actually obtained in two different ways. First, a high reliability sensor reading can possibly pass straight through the algorithm, i.e. taken as valid without invoking the rigorous validation. Second, if the rigorous validation is necessary, it is more likely that the comparison between the original and alternate values is successful because at least the original reading itself is likely to be valid. It is assumed that when the comparison is successful, the data used in the alternate value source as well as the original reading are valid. The rationale is that it is very unlikely for the original reading and those sensor readings involved in the model equation to fail in such a way that the comparison is successful. Therefore, several sensor data are validated at once, and they can be used for validating other sensor data.

3. **Resolve conflicts**

When two sensors show conflicting readings, their relative reliability can be used to make a choice. Reliability ranking is also used when the validation ends in a conflict which cannot be resolved. In that case, if the sensor being validated has high reliability, the original reading is chosen as the final value. Otherwise, the final value is taken as **UNKNOWN**.
5.4.2.2 Assignment of reliability

The reliability ranking is not simply based on instrument manufacturer's rating, which is based on ideal conditions independent of process operation. Instead, it should be based on actual operational history of the sensor, which takes into account the effect of the process configuration and the materials being processed on the sensor. For a new process, initial reliability can be assigned in a conservative manner based on consideration of process characteristics so that likely sources of sensor problems can be identified. For example, a level sensor is likely to be unreliable for a slurry. The initial reliability can be updated when operational experience of the process is accumulated.

5.4.2.3 Measure of reliability

The reliability ranking is a symbolic measure of sensor reliability. This ranking can be used individually to check the reliability history of a sensor or relatively to determine which one of the sensors is more reliable. However, it is never intended to be manipulated mathematically. An integer scale ranging from 1 to 9 is selected, with 1 meaning highly unreliable and 9 highly reliable. This scale is large enough to allow the reliability ranking to spread out so that useful comparison among sensors is possible. For an individual sensor, a ranking of 6 or above is considered reliable for validation purposes. Although the scale of the value used is arbitrary, the value does qualitatively represent the reliability information of the sensor based on past operation.
5.4.3 Results of Validation

Using the validation algorithm discussed above, there are four distinct categories of data after the validation. They represent different extents of validation and the particular category that is obtained for a sensor depends on how, and whether or not, the validation is successfully completed. Although all these different categories of data are used similarly in diagnosis, they are marked differently so that the data can be questioned if necessary.

1. Rigorously validated data. This category includes the original sensor readings that are determined to be valid through the various comparison steps during the validation and those alternate values which are chosen to replace the original sensor readings as the final data for diagnosis. Thus, readings from additional sensors have been utilized during the validation process.

2. Data that pass straight through the validation algorithm, i.e. non-control system sensor readings that are within absolute and rate of change limits, exhibit random noise, normal, steady and has a high reliability ranking. Since all the checks involved are based on the sensor alone, no other sensors are involved during the validation. As discussed previously, these data are taken as valid values so that they are used for validating other sensor data and for the diagnosis.

3. Original data used although the rigorous validation cannot be successfully completed. The data cannot be rigorously validated because of a conflict with the alternate data that cannot be resolved. In this
case, the original sensor reading is used because it has high reliability ranking. Note that this implies that the data is also within the absolute and rate of change limits, and exhibit random noise. Otherwise, the original data would have been rejected and the conflict condition would not occur.

4. **UNKNOWN** is used as the final data for diagnosis because no good value is available. There are two possible scenarios.
   
i) The original data failed one of the checks in the first three steps of the algorithm, and no alternate value could be determined.
   
ii) There is a conflict in the validation that cannot be resolved and the reliability ranking of the sensor being validated is low. The original sensor reading is rejected although no other value is available because the low reliability reading cannot be confirmed.

Note that from the discussion in Section 5.3.3, when the sensor data pass straight through the algorithm, i.e. data category 2 above, it is believed that the probability of an error in the data is very low. Therefore, those data are taken as valid. However, it is recognized that such a probability, no matter how low, still exists. In the rare event that such an error actually occurs, the diagnosis will be using some erroneous data and the results of diagnosis will be adversely affected (as discussed in Section 4.1.1). That is indeed the motivation behind marking this category of data differently from rigorously validated data. In this way, the validation strategy can be extended in the future by incorporating a tactic for going back to this category of data and perform the rigorous validation.
CHAPTER VI
IMPLEMENTATION

The implementation of the sensor validation strategy discussed in the previous chapter requires a computational architecture which integrates diagnostic problem solving with sensor validation. Since calculation with mathematical models is necessary in the validation procedure, the effective application of numerical model-based knowledge, such as material and energy balances, in the computational architecture is an important issue. As a logical outgrowth, malfunction diagnosis based on hypothesis evaluation by means of mathematical calculations is also developed. This further demonstrates the flexibility and capability of the architecture for capturing both numeric and symbolic problem solving.

This architecture represents an extension of the previously developed diagnostic framework in the process engineering domain. As before, diagnostic problem solving is by means of hierarchical classification using a malfunction hierarchy. However, in addition to qualitative hypothesis evaluation by directly matching the observed symptoms with the precompiled symptom patterns of a malfunction, the hypothesis is also evaluated quantitatively based on the results of calculations with mathematical models. To facilitate the manipulation of recorded process data, a database structure is necessary. An intelligent database has been
used in diagnostic systems in both CPP and NPP domains [2]. However, methods for abstracting normality and trend information from numerical data, which are implemented in this work, have not been rigorously applied in the database in previous systems. The most unique feature in this implementation is the additional structure designed for organizing all the domain knowledge for sensor validation and malfunction hypothesis evaluation by calculation. This structure facilitates the integration of qualitative knowledge and mathematical models for use in sensor validation and malfunction diagnosis. With the multiple structures in the prototype system, the effective communication among the different problem-solving modules to arrive at a consistent diagnostic conclusion is also an important issue in the implementation.

6.1 CHEMICAL PROCESS FOR IMPLEMENTATION

A dynamically simulated chemical process is used for the implementation of the prototype knowledge-based system. With the simulation, various malfunctioning conditions can be hypothesized and run. This serves two distinct purposes. During the implementation of the diagnostic module, simulation runs provide the discriminating symptoms of the various malfunctions, i.e. a source of diagnostic knowledge which need to be captured. When the implementation is completed, the numerical sensor data from the simulation are collected for testing the prototype system. The process was selected based on the following characteristics:

1. The process needs to have reasonable complexity. Specifically, it should be complex enough to require nontrivial reasoning, i.e. requires a
malfunction hierarchy with several branches and levels. At the same time, it should be small enough such that the simulation is manageable.

2. Continuous and well controlled such that some steady state operating conditions can be established before the onset of malfunction. This is the reference, normal operating point based on which other process operating conditions are defined. In other words, the symptoms of abnormality are defined as deviations compared with this point of reference.

3. The process configuration allows the various sources of alternate data for sensor validation discussed in Section 5.1.4 to be demonstrated, i.e. hardware sensor redundancy, direct qualitative estimates, and calculations with mathematical models.

The flowsheet for the chosen process is shown in Figure 9. It is obtained by modifying the process described by Oyeleye and Kramer [27]. The various model parameters, including physical properties such as density and heat capacity, and reaction characteristics such as the rate constant and heat of reaction, are based on a generic second order reactor process discussed in Principles and Practice of Automatic Process Control [37]. This process basically involves a continuous stirred tank reactor (CSTR), where the reactant A is converted to product B. The flowrate of the reactant A feed stream is controlled by a feedback proportional-integral (PI) controller, while the concentration and temperature are simply monitored. The reactor liquid level is controlled by a feedback PI controller, which manipulates the flowrate of the product stream. In the reactor, an exothermic second order reaction
Figure 9. Recycle Reactor Process Flowsheet.
takes place, i.e. $2A \rightarrow B$. In order to control the temperature inside the reactor, the excess energy from the heat of reaction is removed by cooling the recycle stream. The cooling is achieved by contacting the recycle stream with a coolant stream in a tube-and-shell heat exchanger. The flowrate of the recycle stream is controlled by a feedback PI controller. The recycle temperature is also controlled by a feedback PI controller, which manipulates the flowrate of the cooling water. However, the setpoint of the recycle temperature controller is specified by the reactor temperature controller, which is itself a feedback PI controller. In other words, a cascade control system is used to control the reactor temperature. Cascade control is used here to improve the dynamic response of the reactor temperature control system.

The dynamic simulation of the process is programmed in the Advanced Continuous Simulation Language (ACSL) [1], which runs on a VAX 8550. The process model and ACSL program listing for the simulation are included in Appendix A. In the simulation, random noise is added to all the sensor signals. By running the simulation for normal steady state operating conditions, tolerance limits for normality and trend in the data can be specified for the various sensor measurements. Also, the absolute and rate of change sensor limits are specified based on the simulated characteristics of the sensors and the process. In order to simulate malfunction conditions, the simulation is modified accordingly before it is run to obtain the data for diagnosis. For example, in order to simulate fouling of the tube and shell heat exchanger, the initial overall heat transfer coefficient is multiplied by a term which decreases with time.
6.2 OVERALL SYSTEM ARCHITECTURE

Based on our task viewpoint discussed previously, each distinct problem-solving task is associated with its own form and organization of knowledge, and problem-solving strategy which is specifically designed for performing the task effectively. Therefore, each problem-solving module for a task provides a priori conceptual definitions of the types of knowledge used, and provides effective programming structures and computational strategies to support the use of this knowledge. The computational framework for a particular implementation is then constructed by integrating the appropriate modules which capture the required tasks.

With the simulated process, the prototype diagnosis and sensor validation system can be developed to demonstrate the computational methodology. For this implementation, the framework consists of three problem-solving modules, i.e. a malfunction hierarchy, a symptoms database (hierarchy), and a validation methods and quantitative models hierarchy. These modules perform three distinct tasks in concert, namely, hierarchical classification, data abstraction and sensor validation, to reach consistent final diagnostic conclusions. This computational architecture is shown in Figure 10. These modules and their respective tasks are summarized below.

1. Hierarchical classification is the core task and is captured in a malfunction hierarchy in which the nodes represent malfunction hypotheses. The task here is to identify plausible malfunction hypotheses
Figure 10. Overall Computational Architecture.
based on the observed symptoms. The details of hierarchical classification has been discussed in Chapter II.

2. Data abstraction refers to the processing of numerical data to generate diagnostically useful data required by the problem solvers. This task is captured in a symptoms database, which maintains the plant data and handles data queries from the various problem solvers.

3. In the validation method and quantitative model hierarchy, both qualitative and quantitative forms of knowledge are organized for sensor validation and hypothesis evaluation. There are methods for determining alternate sensor values and calculational models which can provide symptomatic information for the evaluation of malfunction hypotheses.

6.2.1 Malfunction Hierarchy

The malfunction hierarchy for the CSTR process is shown in Figure 11. It is constructed based on a functional decomposition of the process. The details of the decomposition strategy have been discussed in Chapter III. This diagnostic hierarchy is implemented in the version of CSRL that has been developed by Battelle Memorial Institute [7]. The malfunction hierarchy maintains the overall control of hierarchical classification, which is the core task in malfunction diagnosis. Precompiled knowledge for determining the status of the malfunction hypotheses is encoded in the diagnostic specialists (nodes) in the hierarchy. The complete CSRL specialist code for the malfunction hierarchy is included in Appendix B.
Figure 11. Malfunction Hierarchy for CSTR Process.
To illustrate the CSRL programming construct, the code for the HeatExchanger specialist is shown in Figure 12. In this particular specialist, the malfunction hypothesis is evaluated both by means of numerical calculations and direct pattern matching with the observed symptom patterns. There are two separate knowledge groups (kgs). In the "heatex-test" knowledge group, a numerical method is used to examine the hypothesis. The function "HYPOTHESIS-TEST" executes the necessary calculation specified in the "HeatExchanger-Ob" node in the validation methods and quantitative models hierarchy and returns the result of whether the hypothesis should be "Established" or "Rejected". On the other hand, in the "heatex" knowledge group, the abstracted normality values of the various data are directly matched with the discriminating symptom patterns to evaluate the hypothesis. The function "DB-FETCH", which is used to obtain data, performs two functions. First, it initiates the sensor validation procedure. By letting the data fetching function initiate the validation, the validation procedure is directly applied only to those data which are actually used during the diagnosis. Second, after the validation is completed, it returns the final value that is in the database. If abstracted data are requested, as in this case, the function which performs the necessary data abstraction is called. It should be noted that when a hypothesis is evaluated quantitatively, the data required for the calculation are also obtained by means of "DB-FETCH". Thus, the sensor measurements that are needed for diagnosis are always validated before they are used in making a diagnostic conclusion.
[Specialist HeatExchanger (declare (superspecialist RTempCtrlSys)
  (subspecialists))
  [kgs (heatex-test Rules (match (HYPOTHESIS-TEST 'HeatExchanger-Ob)
    with
    (if Established
     then 3
     else Rejected
     then -3
     else 0)))
  (heatex Table /* ; "if the heat exchanger is fouled, TT7 should decrease while VP3
  and FT3 should increase (MTC8 decreases) when other conditions
  are constant"
  (match (DB-FETCH 'FT2 'value)
    (DB-FETCH 'FT3 'value)
    (DB-FETCH 'MTC8 'value)
    (DB-FETCH 'TT4 'value)
    (DB-FETCH 'TT7 'value)
    (DB-FETCH 'TT8 'value)
    with
    (if N (OR H HH)
     (OR L LL)
     (OR L LL)
     N
     then 3
     else ? (OR H HH)
     (OR L LL)
     ? (OR L LL)
     ?
     then 2
     else ? (OR N L LL)
     (OR N H HH)
     ? (OR N H HH)
     ?
     then -3
     else 0)
  (messages (Establish (if (NOT (EQ heatex-test 0))
    then
    (SetConfidence self heatex-test)
    else
    (SetConfidence self heatex)]

Figure 12. CSRL code for the HeatExchanger specialist.
6.2.2 Symptoms Database

The symptoms database for the recycle CSTR process is shown in Figure 13. In this hierarchy, the data that are useful for diagnosis and sensor validation are stored. Basically, three general categories of data are considered. They include sensor data, alarm states and controller information. The particular types of controller information considered include the setpoints, controller signals (control actions) and control valve positions. This hierarchy is implemented in the Medley release of Xerox LOOPS [44], which is an object-oriented programming environment. The complete code for the database is included in Appendix C.

Each sensor node in this hierarchy contains all the necessary information for the data abstraction (for normality and trend) and data checking (absolute and rate of change limit checks, and noise check described in the first three steps of the validation algorithm discussed in Section 5.2) methods. The code for the temperature sensor "TT8" is shown in Figure 14 to illustrate the information stored in a sensor node in the hierarchy. Referring to Figure 14, various variables are defined to store the information. These variables can be classified into three different conceptual categories.

1. Information for data abstraction:

   - "normal" stores the temperature that is observed under normal operating conditions.
   - "delhl" and "delhhll" are tolerance limits for the assignment of "high/low" and "very high/very low" abstracted values, respectively. For
Figure 13. Symptoms Database.
(DEFCCLASS TT8
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:39 by "))
  (Supers TempSensors)
  (InstanceVariables (ivs (numvalue pastdata))
    (normal 68.1)
    (delhi 1)
    (delhhi 2)
    (deltrend 0.08)
    (hlimit 90)
    (llimit 50)
    (ratelimit 10)
    (devlimit 0.001)
    (numvalue NIL question
      "What is the reading of recycle temp. sensor TT8 ?"
      doc (* measurement from sensor))
    (pastdata NIL question
      "What are the past ten readings of recycle temp. sensor TT8 ?"
      doc (* list of measurements from sensor))
    (trend NIL)
    (value NIL)))

Figure 14. Code for the TT8 sensor node in database hierarchy.
instance, if the temperature is greater than (normal + delhl), then the abstracted temperature value is high.
- "deltrend" is the tolerance for the slope in the data above which a trend, i.e. up or down, is assigned.

2. Information for independent data checks:
- "hlimit" and "llimit" are the absolute high and low limits for the sensor data which are used for limit checking.
- "ratelimit" is the rate of change limit for the slope in the data that is used in the rate of change check.
- "devlimit" is the low limit on the standard deviation of the data that is used in the presence of noise check.

3. Actual process data and queries for data:
- "ivs" specifies the variables which are used for storing actual numerical process data. This is useful for functions which input or delete these data from the database.
- "numvalue" stores the numerical sensor reading from the last data point collected for the diagnosis. The question can be used to query the user for the numerical reading. Note that "NIL" is shown where the actual value will be stored.
- "pastdata" stores numerical values from the past ten data points. The question here is for the past ten readings of the sensor.
- "trend" stores the abstracted trend information based on the numerical values in the "pastdata".
"value" stores the abstracted value based on the numerical value in "numvalue".

6.2.3 Validation Methods and Quantitative Models Hierarchy

This hierarchy is shown in Figure 15. The nodes in this hierarchy are used for organizing information which can be used for sensor validation or quantitative hypothesis evaluation. Therefore, there are two general categories of nodes. In the first category are sensors and alarms, which represent the process data that need to be validated. In the second category, there are those process equipment where quantitative methods can be used to provide symptomatic information for evaluating the malfunction hypotheses. Like the symptoms database, this hierarchy is implemented in Xerox LOOPS. The complete code for this hierarchy can be found in Appendix D.

6.2.3.1 Sensor and Alarm Nodes

Unlike the corresponding sensor or alarm node in the symptoms database, a sensor or alarm node in this hierarchy maintains only sensor validation information. The sensor node "TT8-Ob", which corresponds to the "TT8" node in the symptoms database shown previously in Figure 14, is shown in Figure 16 to illustrate the kind of information organized in the node. The variables defined in the node can be classified into two general categories.
Figure 15. Validation Methods and Quantitative Models Hierarchy.
(DEFCLASS TT8-Ob
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:39 by "))
(Supers Sensor-Ob)
(InstanceVariables (reliability 6)
  (reliability-test NIL)
  (limtest NIL)
  (ratetest NIL)
  (devtest NIL)
  (validity NIL)
  (status NIL)
  (function TT8-VALUE)
  (tolerance 1.5)
  (sensors-used (FT2 FT3 TT5 TT6 TT7))))

Figure 16. Code for the TT8-Ob sensor node.

1. Information for completing validation:
   - "reliability" stores the reliability ranking of the sensor based on its operational history. As discussed in Chapter V, a symbolic scale ranging from 1 to 9 is used.
   - "function" stores the name of the method which can be used for determining an alternate value for the sensor reading. There is no restriction on the types of method that can be used. The various sources of alternate value discussed in Chapter V can be incorporated.
   - "tolerance" specifies the tolerance limit within which the comparison between the original reading and the alternate value is considered successful.
   - "sensors-used" contains a list of all the sensors which are used by the method in determining the alternate value. This list is used when the
sensor data used in the calculation need to be set to valid (Step 12a in algorithm) or need to be validated themselves.

2. Results of validation:
   - "reliability-test" stores the result of the check on the reliability performed in the validation algorithm. Reliability ranking of 6 of above will pass the reliability test, i.e. considered reliable.
   - "limtest" is the result of absolute limit checking
   - "ratetest" is the result of rate of change limit checking
   - "devtest" is the result of presence of noise checking
   - "validity" is the final conclusion concerning the validity of the sensor data assigned by the validation algorithm.
   - "status" stores the information pertaining to the progress of the validation procedure.

6.2.3.2 Equipment nodes

These nodes contain pointers to equation-based methods which can be used to determine the operational status of the equipment represented. Generally speaking, malfunction hypotheses can be established or rejected by means of direct pattern matching with qualitative abstractions of individual sensor readings. However, in some situations, it is more appropriate to consider a hypothesis in terms of information obtained from manipulating the sensor readings with mathematical models. For example, a heat exchanger fouling problem can be checked by performing energy balances on the exchanger. Referring back to the
"HeatExchanger" diagnostic specialist in Figure 12, the "HeatExchanger-Ob" node in this hierarchy is called to apply the method which evaluates the malfunction hypothesis by numerical calculation. The code for the "HeatExchanger-Ob" node is shown in Figure 17. Only three variables are defined in this type of nodes. The "function" variable represents the name of the calculation that needs to be performed, "HTEX-TEST" in this case. The "HTEX-TEST" function checks the closures of both the single (tube or shell) side and the combined tube and shell side energy balances. If the combined balance closes while the single side balance fails to close, then it is concluded that the heat exchanger is probably fouled. The variable "tolerance" specifies the tolerance limit on the closure of the balance equation. Finally, "hypothesis" contains the result of the hypothesis evaluation.

(DEFCLASS HeatExchanger-Ob
  (MetaClass Class Edited%: (* ; "Edited 2-Aug-89 12:48 by "))
  (Supers Equipment-Ob)
  (InstanceVariables (function HTEX-TEST)
    (tolerance 1.6)
    (hypothesis NIL)))

Figure 17. Code for the HeatExchanger-Ob equipment node.
6.3 GENERAL FUNCTIONS

With the three individual hierarchies each containing some portion of the important information for diagnosis and sensor validation, an effective mechanism for directing the information processing is essential. In this particular implementation, the integration of these structures is by means of Lisp function calls. As discussed above, the sensor validation algorithm is initiated by the function "DB-FETCH" by calling the "VALIDATE" function. The "VALIDATE" function then calls on other functions to carry out the various tests in the algorithm. For example, in order to find the trend in the data, the "LSTSQ" function is called to find the slope by performing linear regression on the list of past data. Similarly, as discussed in the section above, to evaluate a malfunction hypothesis by calculation, the appropriate equipment node in the validation method and quantitative model hierarchy is called. The operational status of the equipment is then determined based on the closures of the various constraint equations involved. The overall information flow in the integrated diagnosis and sensor validation system will be discussed in the next chapter when the performance of the prototype system is discussed. The Lisp code for all the functions used in the system can be found in Appendix D.

6.4 FUNCTION INTERFACE WINDOWS

In order to provide convenient access to selected data abstraction and sensor validation functions in the system, and to keep track of the results of function execution, two interface windows have been implemented.
6.4.1 Symptoms Database Interface Window

As shown in Figure 18, this window has a large blank area for printing out the results of database operations, including data abstraction. The attached function menu includes useful database functions. These functions are described below.

- "CLEAR-ALL-DATA" removes all process data from the database. That includes all the numerical data as well as abstracted values.

- "CLEAR-ABSTRACTED-DATA" removes only the abstracted process data from the database.

- "INPUT-ALL-DATA" is a function for entering numerical process data into the database interactively. The questions for data will be asked so that the data for each node can be typed in.

- "MAKE-BATCH-DATAFILE" is a function for opening a file into which all the numerical process data can be typed in. After the data have been typed in, they can be read into the database.

- "READ-DATA-FROM-FILE" is a function for reading the data from a data file into the various nodes in the database.

- "CLEAR-WINDOW" is a convenient function for clearing the blank area in the database window.

- "INFER-NORMALITY" is a data abstraction function for the normality classification of the numerical reading.

- "INFER-TREND" is a data abstraction function for the trend of the data based on the slope determined by linear least squares using a list of numerical data.
### Database Functions

<table>
<thead>
<tr>
<th>CLEAR-ALL-DATA</th>
<th>READ-DATA-FROM-FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR-ABSTRACTED-DATA</td>
<td>CLEAR-WINDOW</td>
</tr>
<tr>
<td>INPUT-ALL-DATA</td>
<td>INFER-NORMALLITY</td>
</tr>
<tr>
<td>MAKE-BATCH-DATAFILE</td>
<td>INFER-TREND</td>
</tr>
</tbody>
</table>

**Database User Window**

---

**Figure 18. Symptoms Database Interface Window.**
6.4.2 Sensor Validation Interface Window

This window is shown in Figure 19. It consists of three major areas. The bottom blank area, the Sensor Validation Important Results Window, is for printing out the important conclusions of the validation algorithm. The middle blank area, called Sensor Validation Information Window, is for printing out the results of individual validation tests as they are performed. Finally, the menu contains useful sensor validation functions which will be described below.

- "INITIALIZE-VALIDATION" is a function for removing all the sensor validation test results. This should be performed before a new diagnostic session begins.
- "INITIALIZE-HYPOTHESIS" is a function for removing all the quantitative hypothesis evaluation results obtained from the diagnosis. Again, this should be done before a new diagnostic session starts.
- "CLEAR-WINDOWS" is a convenient function for clearing both the Sensor Validation Important Results Window and the Sensor Validation Information Window.
- "SENSOR-VALIDATION" applies the validation procedure to the sensor.
- "LIMIT-CHECKING" performs absolute limit checking on the sensor.
- "RATE-OF-CHANGE-CHECKING" checks the slope determined from a set of sensor data against the rate of change limit of the sensor.
- "NOISE-CHECKING" checks for the presence of noise in the data by evaluating the standard deviation.
## Validation Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Checking Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALIZE-VALIDATION</td>
<td>LIMIT-CHECKING</td>
</tr>
<tr>
<td>INITIALIZE-HYPOTHESIS</td>
<td>RATE-OF-CHANGE-CHECKING</td>
</tr>
<tr>
<td>CLEAR-WINDOWS</td>
<td>NOISE-CHECKING</td>
</tr>
<tr>
<td>SENSOR-VALIDATION</td>
<td>RELIABILITY-CHECKING</td>
</tr>
</tbody>
</table>

Sensor Validation Information Window

Sensor Validation Important Results Window

Figure 19. Sensor Validation Interface Window.
- "RELIABILITY-CHECKING" checks the reliability ranking of the sensor.

6.5 COMPUTER HARDWARE

The implementation of the prototype diagnosis and sensor validation system has been carried out on a Xerox 1186 Lisp machine, which is running the Medley programming environment [22]. This environment supports both Common Lisp and Interlisp programming languages, and the Xerox LOOPS object-oriented language. The Battelle CSRL that runs on this machine is built on top of the Xerox LOOPS.
CHAPTER VII

PERFORMANCE CHARACTERISTICS

In this chapter, the performance of the prototype diagnosis and sensor validation system is discussed. As discussed in Chapter VI, the computational architecture consists of three distinct hierarchies in which three distinct problem-solving tasks are captured. To facilitate an understanding of how these problem-solving modules are used in the overall information processing to arrive at a final diagnostic conclusion, the sequence of steps involved in the problem solving are first discussed in terms of the particular task that are being performed and where the necessary information for performing that task is organized. This is followed by case studies which illustrate the overall problem-solving methodology and the various important aspects of the sensor validation strategy.

7.1 INTEGRATED DIAGNOSIS AND SENSOR VALIDATION

In the computational architecture developed, the main problem-solving task is malfunction diagnosis. Sensor validation is integrated into diagnosis to ensure that incorrect data are not used to make a decision. Therefore, the malfunction hierarchy, which performs the diagnostic task, maintains the overall control of the information flow in the problem solving. In other words, the progress of problem
solving is determined by which hypothesis nodes in the malfunction hierarchy need to be examined during the diagnosis. A malfunction hypothesis is evaluated based on symptomatic information provided by the sensor data. Therefore, any sensor failures will adversely affect the result of diagnosis. The impact of sensor failure on malfunction diagnosis has been discussed in detail in Section 4.1.1. The sensor validation task is thus performed on the data required for the diagnosis before a decision on the status of the malfunction hypothesis is made.

In the evaluation of a hypothesis, the sensor data may be used qualitatively in direct pattern matching with the characteristic symptoms of the malfunction. If the observed symptoms on hand match the discriminating symptom patterns of a malfunction, then the malfunction hypothesis is established, i.e. there is probably a malfunction. The sensor data can also be used quantitatively in mathematical model calculations such as material and energy balances. In that event, the status of the malfunction hypothesis is determined based on the results of the calculations. These two different uses of data require different problem-solving processes. Both qualitative and quantitative methods of hypothesis evaluation will be described below. Note that the descriptions are for the evaluation of a single hypothesis in a node of the malfunction hierarchy. The processing is repeated for the other nodes in the hierarchy based on the hierarchical classification computational methodology, which is the overall search (pruning) strategy in the diagnosis.
7.1.1 Organization of Validation Information

The specific steps of the sensor validation algorithm which are performed in the various stages of the problem solving are also included in the discussion below. Since the functions of the various steps in the validation algorithm have been discussed in detail in Chapter V, they will not be repeated below. The main reason for indicating the validation steps involved is to show the particular structures involved in the various steps of the validation. It should be noted that although the majority of sensor validation information is stored in the validation method and quantitative model hierarchy, information for the independent data checks is stored in the symptoms database. There are two related reasons behind this organization.

First, the information used in these data checks actually represents fixed quantitative data pertaining to the characteristics of the sensor reading. These data characteristics are directly applied to the sensor reading to check its integrity. These quantitative data are conceptually similar to the sensor data characteristics used for data abstraction. For example, the rate of change limit for limit checking is very similar to the tolerance limit used for data abstraction. Indeed, they represent limits on the same parameter, i.e. slope in the data. The only difference is in the magnitude. Data abstraction information certainly needs to be represented in the database. Thus, although the sensor nodes in the validation method and quantitative model hierarchy have been developed specifically for organizing validation information, the data for these independent checks are represented in the symptoms database because they fit in well.
Second, the symptoms database is not used to organize the additional information for sensor validation (i.e. along with the information for independent checks) because the nature of the knowledge is conceptually different. Rather than direct quantitative characteristics of the sensor data, symbolic information needs to be organized for these additional steps in the validation. Reliability ranking is a symbolic measure of the past operational reliability history of the sensor. Also, the function that can be performed to validate the sensor reading is represented symbolically by the name of the function. By representing the pointer (name) to a function that can be performed, the actual function can be implemented in a separate environment. Although the functions are developed in the LISP environment in this particular implementation, this representation simulates how a call to a function in a foreign environment (such as FORTRAN) can be incorporated. Furthermore, other symbolic information, including the results of the various tests performed, the status (i.e. progress) of the validation and the final conclusion also needs to be organized. Due to the different conceptual content, this additional information for sensor validation is not stored in the database. Instead, an independent structure is developed, i.e. the validation method and quantitative model hierarchy.

7.1.2 Hypothesis Evaluation by Direct Pattern Matching

In this case, the symptomatic data from sensors and alarms are directly matched with the discriminating symptoms of a malfunction hypothesis to determine whether or not the hypothesis should be established. Since the
discriminating symptom patterns are generally qualitative, data abstraction needs to be performed on the observed data to determine the normality and trend information. The numerical sensor data are not used in a calculation to help determine the status of the hypothesis. This diagnostic process is shown diagrammatically in Figure 20. In order to make clear the interaction among the hierarchies, the individual steps described below are marked in the figure.

1. Diagnosis starts with the malfunction hierarchy. At a node in the hierarchy, the malfunction hypothesis is evaluated by direct pattern matching. Therefore, the data which are required for the pattern matching are requested from the symptoms database. Before the data are returned to the hypothesis node, the sensor validation algorithm is applied to the numerical sensor data involved. In this way, only validated data are used in the pattern matching.

2. The focus of the sensor validation is now on the symptoms database. As discussed in Chapter VI (see Figure 14), each sensor node in the symptoms database contains the numerical process data and the necessary information for performing data abstraction ("normal", "delhi", "delhhl" and deltrend") and the data checks ("hlimit", "llimit", "ratelimit" and "devlimit") in Steps 1 to 3 in the validation algorithm (VA).

(a) If the data pass these checks, then only sensor biases may be in the data. Following the validation algorithm, the sensor category is then checked (VA Step 4). If a control system sensor is not involved, then data abstraction is done to determine the normality and trend
Figure 20. Hypothesis Evaluation by Direct Pattern Matching.
information (VA Steps 5 and 6). Otherwise, rigorous validation is first invoked to confirm the sensor data.

(b) On the other hand, if the data fail any of the checks, an alternate value needs to be determined to replace the original sensor reading.

3. In any event, the data validation continues with the corresponding sensor node in the validation method and quantitative model hierarchy. As discussed in Chapter VI and illustrated in Figure 16, a sensor node in this hierarchy contains information pertaining to the reliability history of the sensors ("reliability") and methods which can be used to determine alternate values for validating the sensors ("function", "tolerance" and "sensors-used").

4. Depending on the result of the processing with the symptoms database, different additional problem-solving steps with the validation method and quantitative model hierarchy are necessary. These different steps are specified according to the validation algorithm.

(a) If the data are determined to be normal and steady in the data abstraction, then the reliability history of the sensor is checked using the information in the validation method hierarchy (VA Step 7). If the sensor reliability is high, the data are taken as valid (VA Step 8). On the other hand, if the reliability is low, then rigorous validation is performed (VA Steps 11 and 12).

(b) If the data are not normal or steady, or if the data are from a control system sensor, then rigorous validation using the method(s) in
the validation method hierarchy is performed to confirm the original sensor data (VA Steps 11 and 12).

(c) If the data fail any of the data checks performed with the symptoms database, then an alternate value is determined, using the method(s) specified in the sensor node in the validation method hierarchy, to replace the original sensor reading (VA Steps 9 and 10).

5. Final numerical data are stored back into the symptoms database when the validation algorithm is completed. If any alternate values determined during the validation are chosen as final values, the original sensor data in the database are replaced.

6. The data originally requested by the malfunction hypothesis are returned to the appropriate node in the malfunction hierarchy. Since the hypothesis requires trend or normality data for pattern matching, the abstracted values based on the numerical data are returned. Any needed data abstraction, if not already done during the validation process, is performed using the final numerical data before the values are returned.

7. The final abstracted data are matched against the precompiled discriminating symptom patterns of the malfunction hypothesis to evaluate its relevancy.

7.1.3 Hypothesis Evaluation by Calculation

Calculation with mathematical equations provides an additional source of symptomatic information for evaluating a malfunction hypothesis. In this case, the
result of the calculation is used simply as another piece of symptomatic information. Thus, the result of the calculation, such as the closure of a constraint equation, may be used along with other symptoms (direct symptoms and other calculational results) in the pattern matching. This is most useful when other symptomatic information, in addition to the calculational result, is necessary in evaluating the hypothesis. The calculational hypothesis evaluation process described below is shown diagrammatically in Figure 21.

1. The diagnosis again starts with the malfunction hierarchy. However, in this case, quantitative evaluation is involved. At a node in the malfunction hierarchy, the hypothesis requests the result of a calculational test. As a result, the corresponding node in the validation method and quantitative model hierarchy is accessed to initiate the calculation.

2. The data which are needed to do the calculation are requested from the symptoms database. The sensor validation algorithm is first applied to the data involved before they are used in the calculation.

3. The focus of the information processing is now on the symptoms database. This step involves the same problem solving as that in Step 2 of the hypothesis evaluation process described in the previous section.

4. Again, similar to Step 3 in the previous section, the data validation continues with the validation method and quantitative model hierarchy.

5. This validation step is the same as Step 4 in the section above. If any sensor failure is identified, the new alternate value is stored back into
VALIDATION METHOD AND QUANTITATIVE MODEL HIERARCHY

PLANT

OBTAIN ALTERNATE DATA USE RELIABILITY HISTORY PERFORM CALCULATION

SYMPTOMS DATABASE HIERARCHY

DATA

DATA CHECKING DATA ABSTRACTION

MALFUNCTION HIERARCHY

HYPOTHESIS EVALUATED BASED ON THE RESULT OF CALCULATION

Figure 21. Hypothesis Evaluation by Calculation.
the database, i.e. Step 5 above. The final validated numerical data are
used to perform the calculation specified in the node in the validation
method and quantitative model hierarchy. The calculation may involve
a mathematical constraint equation, such as a material or energy balance
using the input and output data of the functional system being
considered in the malfunction hypothesis.

6. The result of the calculation is returned to the original hypothesis node
in the malfunction hierarchy which requested the information.

7. The hypothesis is evaluated based on the result of the calculation. If the
constraint equation is satisfied, then there is probably no malfunction in
the functional system and the malfunction hypothesis for the system can
be rejected. On the other hand, a failed constraint equation indicates
there is a probable malfunction in the functional system so that the
malfunction hypothesis is established.

7.2 CASE STUDIES

In this section, three diagnostic cases are discussed. These cases have been
carefully selected to illustrate the overall computational methodology and the
sensor validation algorithm developed in this research. Note that the following
discussion is based on the particular set of validation methods for the simulated
process developed in this implementation. As discussed in Chapter V, the methods
available for validation can strongly affect the performance of the validation
strategy. Based on that discussion, the ideal situation is to have all direct
relationships because the probability of a conflict situation in the validation that cannot be resolved is expected to be lower if more direct relationships are available.

The case studies were carried out in the following manner. With the malfunction and sensor failure specified, the dynamic simulation was run to obtain the numerical data for the various sensors in the process. The symptoms on hand, including numerical sensor data and alarm states, were then read into the symptoms database from a data file. Then the diagnosis was carried out to try to identify the originally assumed malfunction and sensor failure. The results of the various problem-solving activities performed during the diagnostic session were collected for analysis.

As discussed in Section 4.1.1, failure of a Category I (control system) sensor alone can lead to undesirable process conditions. On the other hand, failure of a Category II (non-control) sensor does not have any direct effect on the process operation. Therefore, the same mode of failure in a different sensor category can lead to very different scenarios. The cases discussed are organized around these two categories to illustrate the effects. They include 1) Category I sensor with a mode 3 failure (i.e. not measuring) alone, 2) equipment malfunction plus Category I sensor with a mode 4 failure (i.e. bias), and 3) equipment malfunction plus Category II sensor with a mode 4 failure. Since the failure of a Category II sensor alone does not affect the process operation, it does not constitute an interesting diagnostic case. As a result, this particular scenario is not discussed below. However, it should be noted that this scenario is basically a simplified version of
case 3 because the additional equipment malfunction definitely would not make the case any easier. Therefore, the performance of the system in case 3 would be representative of cases with a Category II sensor failure.

7.2.1 Case 1 - Category I Sensor Failure Alone

In this case study, the recycle temperature sensor TT8 was not measuring but simply giving a constant reading of 78°F, i.e. sensor failure mode 3 discussed in Section 4.1.2. This case is representative of the information processing involved in the validation strategy when any failures of mode 1 to 3 are involved. The reason is that these three modes of failure are all directly identified by independent data checks (VA Steps 1 to 3) and then the same procedure is used (VA Steps 9 and 10) in obtaining a replacement value.

Referring to the process flowsheet in Figure 9, the TT8 sensor is used in the recycle temperature control system, which is the secondary controller in the reactor temperature control system. Since the recycle temperature controller was using the TT8 sensor reading to determine the control action, erroneous control action was calculated. Specifically, since the constant TT8 reading was higher than the setpoint value, the coolant flowrate was increased (see Figure 22), by opening the control valve, to try to bring the temperature back down to the setpoint. As a result, referring to Figure 23, although the TT8 reading remained constant, the actual recycle temperature decreased. As shown in Figure 24, this in turn led to a decrease in the reactor temperature. This discussion shows how the failure of a Category I sensor alone can lead to undesirable process operating conditions.
Figure 22. Plot of Coolant Flowrate for Case 1.
Figure 23. Plots of Actual and Measured Recycle Temperatures for Case 1.
Figure 24. Plot of Reactor Temperature for Case 1.
This case was run using the prototype system. The diagnostic conclusions are represented in the confidence value hierarchy shown in Figure 25. A confidence value is a symbolic measure of the degree of certainty that a malfunction hypothesis is potentially valid for a given set of process symptoms. On a discrete scale of -3 to 3, a confidence value of -3 suggests that the hypothesis is highly unlikely while a 3 indicates the opposite. Intermediate values suggest lower degrees of confidence. A diagnostic specialist is considered to be established when the confidence value is 2 or 3 and rejected when it is -2 or -3. It is suspended when the value is -1, 0 or 1, i.e. there is not enough evidence to make a final decision. In the malfunction hierarchy, the specialist nodes at the lowest level of the hierarchy (referred to as tip nodes) represent the most specific malfunction hypotheses. In other words, an established tip node indicates that a specific malfunction (root cause) has been identified. From the confidence value hierarchy in Figure 25, the "RRTSensorTT8" (Reactor Recycle Temperature Sensor TT8) specialist was established with a confidence value of 3, which indicated that the TT8 sensor malfunction was correctly identified in the diagnosis.

Data abstraction for trend and normality was necessary during the diagnosis for providing the symptomatic information for direct pattern matching. Also, it was required in Steps 5 and 6 of the validation algorithm. These results were written to the DataBase User Window, which is shown in Figure 26. From Figure 26, the data abstraction results showed "abnormal" conditions [i.e. high (H), low (L), increasing (UP) or decreasing (DOWN)] for the following data: 1) TT4 (reactor temperature sensor) was decreasing, 2) FT3 (coolant flow sensor) was high, 3) TT7...
Figure 25. Confidence Value Hierarchy for Case 1.
### DATABASE FUNCTIONS

<table>
<thead>
<tr>
<th>CLEAR-ALL-DATA</th>
<th>READ-DATA-FROM-FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR-ABSTRACTED-DATA</td>
<td>CLEAR-WINDOW</td>
</tr>
<tr>
<td>INPUT-ALL-DATA</td>
<td>INFER-NORMALITY</td>
</tr>
<tr>
<td>MAKE-BATCH-DATAFILE</td>
<td>INFER-TREND</td>
</tr>
</tbody>
</table>

#### Database User Window

- Abstracted normality value of FT1 is: N
- Abstracted normality value of TT4 is: N
- Abstracted trend value of TT4 is: DOWN
- Abstracted normality value of FT3 is: H
- Abstracted normality value of TT6 is: N
- Abstracted trend value of TT6 is: STEADY
- Abstracted normality value of TT7 is: L
- Abstracted normality value of TT8 is: L
- Abstracted normality value of MTC4 is: H
- Abstracted normality value of STC4 is: N
- Abstracted normality value of VP3 is: H
- Abstracted normality value of MTC8 is: L
- Abstracted normality value of FT2 is: N
- Abstracted normality value of LT1 is: N
- Abstracted trend value of LT1 is: STEADY

---

Figure 26. Symptoms DataBase Interface Window with Tracing of Data Abstraction Results for Case 1.
(coolant outlet temperature sensor) was low, 4) TT8 was low, 5) MTC4 (reactor temperature controller signal) was high, 5) VP3 (coolant flow control valve positioner) was high, and 6) MTC8 (recycle temperature controller signal) was low. These abstracted values were expected based on the discussion of the process conditions above. However, three of the values warrant more discussion. For sensor TT8, despite the sensor failure, the validated value represented the actual recycle temperature, which was shown to be low in Figure 23. The MTC4 reading is the setpoint value for the recycle temperature controller. Therefore, when the reactor temperature was decreasing, the MTC4 value was set to be high to try to bring the reactor temperature back up. For the MTC8 value, the coolant flow valve is fully open when MTC8 is zero, and the opening decreases as MTC8 increases. Thus, since a high coolant flowrate was specified by the recycle temperature controller, the MTC8 value was low.

During the diagnostic session, the results of the various data checks specified in the validation algorithm (VA Steps 1 to 3 and 5 to 7) were written to the Sensor Validation Information Window, while the key results of the validation were written to the Sensor Validation Important Results Window. These windows (showing a partial tracing of the messages written) are shown in Figure 27 to illustrate how the windows would look when a case is run. The complete tracing of all the messages written to these validation windows can be found in Figure 28.

Referring to the tracing in Figure 28, the following important points should be observed.
## VALIDATION FUNCTIONS

<table>
<thead>
<tr>
<th>SENSOR-VALIDATION</th>
<th>LIMIT-CHECKING</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALIZE-VALIDATION</td>
<td>CLEAR-WINDOWS</td>
</tr>
<tr>
<td>INITIALIZE-HYPOTHESIS</td>
<td>NOISE-CHECKING</td>
</tr>
<tr>
<td>CLEAR-WINDOWS</td>
<td>RELIABILITY-CHECKING</td>
</tr>
<tr>
<td>SENSOR-VALIDATION</td>
<td></td>
</tr>
</tbody>
</table>

### Sensor Validation Information Window
- Sensor TT6 rate of change limit checking result: Pass
- Sensor FT3 presence of noise checking result: Pass
- Sensor FT3 normality test result: Fail
- Sensor TT6 absolute limit checking result: Pass
- Sensor TT6 rate of change limit checking result: Pass
- Sensor TT6 presence of noise checking result: Pass
- Sensor TT6 normality test result: Pass
- Sensor TT6 trend test result: Pass
- Sensor TT6 reliability ranking test result: Pass
- Sensor TT7 absolute limit checking result: Pass
- Sensor TT7 rate of change limit checking result: Pass
- Sensor TT7 presence of noise checking result: Pass
- Sensor TT7 normality test result: Pass
- Calculated value for sensor TT7 is not within tolerance band of measurement. Since only one calculation is available, the data used in calculation are being validated.
- Sensor LT1 absolute limit checking result: Pass
- Sensor LT1 rate of change limit checking result: Pass
- Sensor LT1 presence of noise checking result: Pass

### Sensor Validation Important Results Window
- Sensor (TT6, TT8, TT2) are set to be Fail
- Alternate TT8 value being calculated to replace
- Sensor FT2 measurement is validated
- Alternate FT2 value being calculated to confirm
- Sensor FT3 normality test result: Fail
- Alternate FT3 value being calculated to confirm
- Sensor FT3 measurement is validated
- Sensor TT6 reading passed all tests in validation module
- Sensor TT7 normality test result: Fail
- Alternate TT7 value being calculated to confirm
- The conflict between calculated and measured values of sensor TT7 cannot be resolved
- Since sensor reliability is high, the original measured value is used
- Alternate value for sensor TT8 is obtained and used
- Sensor LT1 measurement is validated

Figure 27. Sensor Validation Interface Windows With Partial Tracing of Validation Results for Case 1.
Tracking of messages written to the windows during diagnosis

Case: Recycle temperature sensor TT8 stuck high

Sensor Validation Information Window:
- Sensor FT1 absolute limit checking result: Pass
- Sensor FT1 rate of change limit checking result: Pass
- Sensor FT1 presence of noise checking result: Pass
- Sensor TT4 absolute limit checking result: Pass
- Sensor TT4 rate of change limit checking result: Pass
- Sensor TT4 presence of noise checking result: Pass
- Sensor TT8 absolute limit checking result: Pass
- Sensor TT8 rate of change limit checking result: Pass
- Sensor TT8 presence of noise checking result: Pass
- Sensor TT2 absolute limit checking result: Pass
- Sensor TT2 rate of change limit checking result: Pass
- Sensor TT2 presence of noise checking result: Pass
- Sensor FT3 absolute limit checking result: Pass
- Sensor FT3 rate of change limit checking result: Pass
- Sensor FT3 presence of noise checking result: Pass
- Sensor TT6 absolute limit checking result: Pass
- Sensor TT6 rate of change limit checking result: Pass
- Sensor TT6 presence of noise checking result: Pass
- Sensor TT6 normality test result: Pass
- Sensor TT6 trend test result: Pass
- Sensor TT7 absolute limit checking result: Pass
- Sensor TT7 rate of change limit checking result: Pass
- Sensor TT7 presence of noise checking result: Pass
- Sensor TT7 normality test result: Pass
- Sensor TT7 reliability ranking test result: Pass
- Calculated value for sensor TT7 is not within tolerance band of measurement. Since only one calculation is available, the data used in calculation are being validated
- Sensor LT1 absolute limit checking result: Pass
- Sensor LT1 rate of change limit checking result: Pass
- Sensor LT1 presence of noise checking result: Pass

Sensor Validation Important Results Window:
- Sensors (FT4) are set to be valid
- Sensors (TT6 TT3 TT2) are set to be valid
- Sensor TT8 presence of noise checking result: Fail
- Alternate TT8 value being calculated to replace
- Sensor FT2 measurement is validated
- Sensor FT3 normality test result: Fail
- Alternate FT3 value being calculated to confirm
- Sensor FT3 measurement is validated
- Sensor TT6 reading passed all tests in validation module
- Sensor TT7 normality test result: Fail
- Alternate TT7 value being calculated to confirm
- The conflict between calculated and measured values of sensor TT7 cannot be resolved
- Alternate value for sensor TT8 is obtained and used
- Sensor LT1 measurement is validated

Sensor Validation User Input Window:
- Do you want to input new value for sensor TT7?
  - N
  - Value of TT7 not changed

Figure 28. Complete Validation Message Tracing for Case 1.
1. Only those sensor data which were actually needed during the diagnostic session were validated. In this particular case, based on the names of the sensors included in the tracing, FT1, TT4, TT8, FT2, FT3, TT6 TT7 and LT1 were validated.

2. Since the TT8 data were constant at 78°F, they failed the presence of noise check (VA Step 3). As a result, an alternate value was determined to replace the original reading (VA Steps 9 and 10). In this case, an energy balance around the heat exchanger was used to determine the alternate TT8 value. The sensor data involved in the calculation were first validated before the calculation was done. As shown in the tracing of the Validation Important Results Window in Figure 28, the data from sensors FT2, FT3, TT6 and TT7, which were used in the calculation were validated before the final TT8 value was obtained.

3. When the data from sensor TT7 were validated, an alternate value needed to be determined to confirm because the reading was low (VA Step 5). The alternate value was not within the tolerance band of the measurement, thus the data used in the calculation were validated. The conflict between the alternate and original values could not be resolved (VA Step 12b) because the calculation of the alternate TT7 value (using an energy balance around the heat exchanger) required the TT8 reading, which was itself being validated at that time. This point will be further elaborated in the next section. Since the reliability of the TT7 sensor was high, the original sensor reading was selected. As shown in the message
tracing in Figure 28, the user was then prompted for an input in the Sensor Validation User Input Window, which is also shown in Figure 29. This additional window only appears when there is an unresolvable conflict in the validation. That alerts the user of the situation and allows the user an opportunity to check the sensor and input a value. A new value was not input in this case.

4. The entire set of tests in the validation algorithm were performed only on sensor TT6. The reason is that when the sensor data fail a check, then the subsequent checks are not performed. For example, sensor FT3 (coolant flow sensor) failed the normality check, i.e. the reading was either high or low. Thus, the trend check and reliability check (VA Steps 6 and 7) were not done. In this case, the data from sensor TT6 passed all the tests. The data were taken as valid without going through the rigorous validation (VA Step 8).

5. For the control system sensors (FT1, TT4, TT8, FT2 and LT1), rigorous validation is always performed (VA Step 4). Therefore, the checks in VA Steps 5 to 7, which are used for determining whether rigorous validation can be avoided, are not necessary. Thus, for these sensors, the tracing of the messages written in the Validation Information Window did not include results from these checks.

6. Some sensor readings were taken as valid without going through the various steps in the validation algorithm. The reason was that the alternate value determined for a certain sensor using these sensor data
Figure 29. Sensor Validation User Input Window.
matched the original reading of that sensor (VA Step 12a). The message "[Sensors (list of sensor names) are set to be valid]" was used to indicate this condition. In this case, from the message tracing in Figure 28, FT4, TT2, TT3 and TT5 were set to be valid in this way.

7.2.2 Extent of Rigorous Validation

Point 3 in the previous section indicates that, depending on the methods available, it is probably impossible to validate all the sensor data rigorously. Due to the limited instrumentation in chemical plants, it is highly likely that only a small number of quantitative methods can be used for obtaining alternate values for different sensors. The reason is that even when additional equations are available, they probably cannot be used because some of the parameters required in the equations may not be measured by sensors. As a result, the same equation probably needs to be used for determining the alternate values for several sensors. Therefore, if all the sensor data need to be rigorously validated by quantitative methods, there are likely to be conflicts which cannot be resolved. In the particular case above, the same energy balance around the heat exchanger was used in determining alternate values for sensors TT6, TT7 and TT8. When the alternate value of TT8 had to be calculated to replace the original sensor reading which failed the presence of noise check, the rigorous validation of TT7 was required. That resulted in an unresolvable conflict because the calculation of the alternate TT7 value required the TT8 reading, which was itself being validated at that time. Therefore, the rigorous validation of TT7 could not be completed.
There are two general strategies which can help minimize the occurrence of these conflict situations. First, instead of relying heavily on quantitative balance equations, different types of methods as described in Section 5.1.4 should be applied in the rigorous validation. Indeed, that is the reason why the validation strategy developed in this work emphasizes a knowledge representation where flexibility in the methods can be incorporated. It is expected that if direct relationships can be generally applied to the sensors, the probability of conflict will be very low. The effects of different combinations of methods on the performance of the validation strategy are not studied in this work because the focus is on the computational framework rather than the specific methods available, which are process dependent. Second, tactics can be used to minimize the application of rigorous validation. As discussed in Chapter V, two approaches for minimizing the amount of rigorous validation have been used in the algorithm (VA Steps 8 and 12a). These were also indicated in points 4 and 5 in the previous section. In order to study the effects of these approaches in the performance of the sensor validation strategy, Case 1 was run again with each and then both approaches removed from the algorithm. Although the same diagnostic conclusion was obtained in all the cases discussed below, the amount of computation and the conflict conditions encountered were different among the runs.

7.2.2.1 Straight Through Route Removed

In this case, instead of checking the criteria in Steps 4 to 7 of the validation algorithm to determine whether rigorous validation could be avoided, all the data
needed during the diagnostic session were rigorously validated. The complete tracing of the messages written to the validation windows is shown in Figure 30. When this is compared with the tracing of the original case shown in Figure 28, the most important difference occurred in sensor TT6. Instead of passing straight through the algorithm, the TT6 data had to be rigorously validated. This rigorous validation led to another unresolvable conflict because, as discussed above, the same method (energy balance around the heat exchanger) was used for determining the alternate values for TT6, TT7 and TT8. As a result, instead of concluding that TT6 was valid, which was actually true here, the algorithm concluded that the conflict could not be resolved. The value was used because it passed the data checks in Steps 1 to 3 of the algorithm and the reliability was high. This was a much weaker conclusion and so the performance of the validation module was adversely affected.

It should also be noted that since rigorous validation was generally applied in this case, the criteria for invoking the rigorous validation were no longer needed. As a result, those tests in Steps 4 to 7 of the algorithm were not performed. However, these are simple independent checks which can be done easily. Moreover, the data abstraction for normality and trend (the key element of the normality and trend check in Steps 5 and 6) is often necessary to provide the appropriate data requested by the diagnostic problem solver anyway. Therefore, the savings in the computation by eliminating these tests is generally more than offset by any additional rigorous validation required, especially if quantitative calculational methods are involved.
Tracing of messages written to the window during diagnosis
Case: Recycle temperature sensor TT8 stuck high (straight through route turned off)

Sensor Validation Information Window:
- Sensor FT1 absolute limit checking result: Pass
- Sensor FT1 rate of change limit checking result: Pass
- Sensor FT1 presence of noise checking result: Pass
- Sensor TT4 absolute limit checking result: Pass
- Sensor TT4 rate of change limit checking result: Pass
- Sensor TT4 presence of noise checking result: Pass
- Sensor TT8 absolute limit checking result: Pass
- Sensor TT8 rate of change limit checking result: Pass
- Sensor TT8 presence of noise checking result: Fail
- Sensor FT2 absolute limit checking result: Pass
- Sensor FT2 rate of change limit checking result: Pass
- Sensor FT2 presence of noise checking result: Pass
- Sensor FT3 absolute limit checking result: Pass
- Sensor FT3 rate of change limit checking result: Pass
- Sensor FT3 presence of noise checking result: Pass
- Sensor FT3 normality test result: Fail
- Sensor TT6 absolute limit checking result: Pass
- Sensor TT6 rate of change limit checking result: Pass
- Sensor TT6 presence of noise checking result: Pass
- Sensor TT6 normality test result: Pass
- Sensor TT6 trend test result: Pass
- Sensor TT6 reliability ranking test result: Fail
- Calculated value for sensor TT6 is not within tolerance band of measurement. Since only one calculation is available, the data used in calculation are being validated
- Sensor TT7 absolute limit checking result: Pass
- Sensor TT7 rate of change limit checking result: Pass
- Sensor TT7 presence of noise checking result: Pass
- Sensor TT7 normality test result: Fail
- Calculated value for sensor TT7 is not within tolerance band of measurement. Since only one calculation is available, the data used in calculation are being validated
- Sensor LT1 absolute limit checking result: Pass
- Sensor LT1 rate of change limit checking result: Pass
- Sensor LT1 presence of noise checking result: Pass

Sensor Validation Important Results Window:
- Sensor FT1 measurement is validated
- Sensors (FT4) are set to be valid
- Sensors (TT5 TT3 TT2) are set to be valid
- Sensor TT8 presence of noise checking result: Fail
- Alternate TT8 value being calculated to replace
- Sensor FT2 measurement is validated
- Sensor FT3 normality test result: Fail
- Alternate FT3 value being calculated to confirm
- Sensor FT3 measurement is validated
- Sensor TT6 reliability ranking test result: Fail
- Alternate TT6 value being calculated to confirm
- Sensor TT7 normality test result: Fail
- Alternate TT7 value being calculated to confirm
- The conflict between calculated and measured values of sensor TT7 cannot be resolved
- Since sensor reliability is high, the original measured value is used
- The conflict between calculated and measured values of sensor TT6 cannot be resolved
- Since sensor reliability is high, the original measured value is used
- Alternate value for sensor TT6 is obtained and used
- Sensor LT1 measurement is validated

Sensor Validation User Input Window:
- Do you want to input new value for sensor TT7? [ ] > N
- Value of TT7 not changed
- Do you want to input new value for sensor TT6? [ ] > N
- Value of TT6 not changed

Figure 30. Message Tracing for Case 1 with Straight-Through Route Removed.
7.2.2.2 Validity Update Removed

In this case, the algorithm (VA Step 12) was modified such that even when the alternate value determined for a sensor matched with the original reading, the sensor data used in the calculation were not automatically set to valid. The complete tracing of the messages written to the validation windows is shown in Figure 31. When this is compared with the original case, the main difference is that the sensor TT5 data had to be directly validated. Instead of being set to valid automatically, the validation algorithm was applied to the data. Since the data were decreasing (one of the criteria for invoking rigorous validation), rigorous validation was performed. Although the additional validation in this case was completed without complicated conflict situations, two points should be noted. First, more computation is always necessary to carry out the additional validation. Second, whenever the rigorous validation is applied, a conflict condition is possible whether or not the sensor data being validated are valid. The reason is that the other data needed to determine the alternate value for comparison may not be valid. The validation of those data may result in conflict, which may eventually lead to a conflict in the original validation. As a result, if the probability of a sensor failure is very low, the sensor data should be directly set to valid.

7.2.2.3 Both Approaches Removed

In this case, both approaches for minimizing the amount of rigorous validation were removed, i.e. combination of the two sections above. The complete tracing of the messages written to the validation windows is shown in Figure 32. As
Tracing of messages written to the windows during diagnosis
Case: Recycle temperature sensor TT8 stuck high (update validity turned off)

Sensor Validation Information Window:
- Sensor FT1 absolute limit checking result: Pass
- Sensor FT1 rate of change limit checking result: Pass
- Sensor FT1 presence of noise checking result: Pass
- Sensor TT4 absolute limit checking result: Pass
- Sensor TT4 rate of change limit checking result: Pass
- Sensor TT4 presence of noise checking result: Pass
- Sensor TT8 absolute limit checking result: Pass
- Sensor TT8 rate of change limit checking result: Pass
- Sensor TT8 presence of noise checking result: Fail
- Sensor FT2 absolute limit checking result: Pass
- Sensor FT2 rate of change limit checking result: Pass
- Sensor FT2 presence of noise checking result: Pass
- Sensor FT3 absolute limit checking result: Pass
- Sensor FT3 rate of change limit checking result: Pass
- Sensor FT3 presence of noise checking result: Pass
- Sensor FT3 normality test result: Fail
- Sensor TT5 absolute limit checking result: Pass
- Sensor TT5 rate of change limit checking result: Pass
- Sensor TT5 presence of noise checking result: Pass
- Sensor TT5 trend test result: Fail
- Sensor TT6 absolute limit checking result: Pass
- Sensor TT6 rate of change limit checking result: Pass
- Sensor TT6 presence of noise checking result: Pass
- Sensor TT6 normality test result: Pass
- Sensor TT6 trend test result: Pass
- Sensor TT6 reliability ranking test result: Pass
- Sensor TT7 absolute limit checking result: Pass
- Sensor TT7 rate of change limit checking result: Pass
- Sensor TT7 presence of noise checking result: Pass
- Sensor TT7 normality test result: Fail
- Sensor TT7 normality test result: Fail
- Calculated value for sensor TT7 is not within tolerance band of measurement. Since only one calculation is available, the data used in calculation are being validated

Sensor Validation Important Results Window:
- Sensor FT1 measurement is validated
- Sensor TT4 measurement is validated
- Sensor TT8 presence of noise checking result: Fail
- Alternate TT8 value being calculated to replace
- Sensor FT2 measurement is validated
- Sensor FT3 normality test result: Fail
- Alternate FT3 value being calculated to confirm
- Sensor FT3 measurement is validated
- Sensor TT5 trend test result: Fail
- Alternate TT5 value being calculated to confirm
- Sensor TT5 measurement is validated
- Sensor TT6 reading passed all tests in validation module
- Sensor TT7 normality test result: Fail
- Alternate TT7 value being calculated to confirm
- The conflict between calculated and measured values of sensor TT7 cannot be resolved
- Since sensor reliability is high, the original measured value is used
- Alternate value for sensor TT8 is obtained and used
- Sensor LT1 measurement is validated

Sensor Validation User Input Window:
- Do you want to input new value for sensor TT7?
  > N
- Value of TT7 not changed

Figure 31. Message Tracing for Case 1 with Update-Validity Removed.
Case: Recycle temperature sensor TT8 stuck high (straight through route and update validity turned off)

Sensor Validation Information Window:
- Sensor FT1 absolute limit checking result: Pass
- Sensor FT1 rate of change limit checking result: Pass
- Sensor FT1 presence of noise checking result: Pass
- Sensor TT4 absolute limit checking result: Pass
- Sensor TT4 rate of change limit checking result: Pass
- Sensor TT4 presence of noise checking result: Pass
- Sensor TT8 absolute limit checking result: Pass
- Sensor TT8 rate of change limit checking result: Pass
- Sensor TT8 presence of noise checking result: Fail
- Sensor FT2 absolute limit checking result: Pass
- Sensor FT2 rate of change limit checking result: Pass
- Sensor FT2 presence of noise checking result: Pass
- Sensor FT3 absolute limit checking result: Pass
- Sensor FT3 rate of change limit checking result: Pass
- Sensor FT3 presence of noise checking result: Pass
- Sensor TT5 absolute limit checking result: Pass
- Sensor TT5 rate of change limit checking result: Pass
- Sensor TT5 presence of noise checking result: Pass
- Sensor TT5 normality test result: Pass
- Sensor TT5 trend test result: Fail
- Sensor TT6 absolute limit checking result: Pass
- Sensor TT6 rate of change limit checking result: Pass
- Sensor TT6 presence of noise checking result: Pass
- Sensor TT6 normality test result: Pass
- Sensor TT6 trend test result: Pass
- Sensor TT6 reliability ranking test result: Fall
- Calculated value for sensor TT6 is not within tolerance band of measurement. Since only one calculation is available, the data used in calculation are being validated.
- Sensor TT7 absolute limit checking result: Pass
- Sensor TT7 rate of change limit checking result: Pass
- Sensor TT7 presence of noise checking result: Pass
- Sensor TT7 normality test result: Fail
- Alternate TT7 value being calculated to replace
- Sensor FT3 measurement is validated
- Sensor TT6 measurement is validated
- Sensor TT8 measurement is validated
- Alternate TT8 value being calculated to replace
- Sensor FT2 measurement is validated
- Sensor FT3 normality test result: Fail
- Alternate FT3 value being calculated to confirm
- Sensor FT3 measurement is validated
- Sensor TT5 trend test result: Fail
- Alternate TT5 value being calculated to confirm
- Sensor TT5 measurement is validated
- Sensor TT6 reliability ranking test result: Fail
- Alternate TT6 value being calculated to confirm
- Sensor TT7 normality test result: Fail
- Alternate TT7 value being calculated to confirm
- The conflict between calculated and measured values of sensor TT7 cannot be resolved
- Since sensor reliability is high, the original measured value is used
- The conflict between calculated and measured values of sensor TT6 cannot be resolved
- Since sensor reliability is high, the original measured value is used
- Alternate value for sensor TT8 is obtained and used
- Sensor LT1 measurement is validated

Sensor Validation User Input Window:
- Do you want to input new value for sensor TT7? > N
  Value of TT7 not changed
- Do you want to input new value for sensor TT6? > N
  Value of TT6 not changed

Figure 32. Message Tracing for Case 1 with Both Approaches Removed.
expected, the effects of removing each approach as discussed above are all exhibited in this case. Specifically, additional conflict condition in validating sensor TT6 and sensor TT5 data had to be directly validated. Since these effects have already been discussed, the details are not repeated here.

7.2.3 Case 2 - Equipment Malfunction Plus Category I Sensor Failure

Referring to the process flowsheet in Figure 9, the scenario here was that the feed flow control valve was stuck at 25% open and the reactor level sensor LT1 had a positive bias of 0.3 m. With the constant 25% opening, the feed flowrate was 0.30 m³/min, which was much lower than the normal value of 0.45 m³/min (see Figure 33). This low feed flowrate led to a decreased concentration of reactant in the reactor. As a result, the reaction rate decreased and the exothermic heat of reaction also decreased. In order to maintain the reactor temperature, the reactor temperature control system dictated a higher setpoint for the recycle temperature controller. This led to an increase in the recycle temperature by means of a decrease in the coolant flowrate. As shown in Figure 34, the reactor temperature was decreasing very slowly and then slowly returning to the setpoint, when the recycle temperature increased. Referring to Figure 35, the decrease in coolant flowrate also led to an increase in the coolant outlet temperature. At the same time, the bias in LT1 affected the operation of the reactor level control system. Since the level was being controlled, the decrease in feed flowrate led to a decrease in the product flowrate so that the level could remain constant. However, since a bias in LT1 was also introduced, the product flowrate initially increased to
Figure 2.3. Plots of Feed and Product Flowrates for Case 2.
Figure 34. Plots of Reactor and Recycle Temperatures for Case 2.
Figure 35. Plots of Coolant Flowrate and Outlet Temperature for Case 2.
try to bring the level sensor reading back down to the setpoint (see Figure 33). From Figure 36, although the LT1 reading was returned to the setpoint value, the bias in LT1 resulted in an actual reactor level which was lower than the normal setpoint level.

The confidence value hierarchy for this diagnostic case study is shown in Figure 37. In that hierarchy, both the "FFValve" (Feed Flow Valve) and "RLSensorLT1" (Reactor Level Sensor LT1) specialist nodes were established with confidence values of 3. Thus, the originally assumed malfunctions, namely feed flow control valve failure and reactor level sensor bias, were correctly identified in the diagnosis. It should be noted that the "RTempCtrlSys" (Reactor Temperature Control System) specialist was established with a confidence value of 2. This branch was established because of the effect of the low feed flowrate on the reactor temperature discussed above. However, as expected, no tip node was established in that branch of the malfunction hierarchy, i.e. no actual malfunction was found in that branch.

The complete tracing of the messages written to the Database User Window and the Sensor Validation Information and Important Results Windows are included in Figure 38. From the tracing of messages written to the DataBase User Window, the data abstraction results showed "abnormal" conditions for the following data: 1) FT4 (product flow sensor) was low, 2) FT1 (feed flow sensor) was low, 3) VP1 (feed flow valve positioner) was low, 4) MFC1 (feed flow controller action) was low, 5) TT4 (reactor temperature sensor) was decreasing, 6) TT8 (recycle temperature sensor) was high and increasing, 7) LT1 was low, and
Figure 36. Plots of Actual and Measured Reactor Levels for Case 2.
Figure 37. Confidence Value Hierarchy for Case 2.
Tracing of messages written to the windows during diagnosis
Case: Feed flow valve stuck and reactor level sensor LT1 bias

**DataBase User Window:**
- Abstracted normality value of FT4 is: L
- Abstracted normality value of FT1 is: L
- Abstracted normality value of VP1 is: L
- Abstracted normality value of MFC1 is: L
- Abstracted normality value of SFC1 is: N
- Abstracted normality value of TT4 is: N
- Abstracted normality value of TT8 is: H
- Abstracted normality value of TT8 is: UP
- Abstracted normality value of FT2 is: N
- Abstracted normality value of LT1 is: L
- Abstracted normality value of SLC1 is: N

**Sensor Validation Information Window:**
- Sensor FT1 absolute limit checking result: Pass
- Sensor FT1 rate of change limit checking result: Pass
- Sensor FT1 presence of noise checking result: Pass
- Sensor FT4 absolute limit checking result: Pass
- Sensor FT4 rate of change limit checking result: Pass
- Sensor FT4 presence of noise checking result: Pass
- Sensor TT4 absolute limit checking result: Pass
- Sensor TT4 rate of change limit checking result: Pass
- Sensor TT4 presence of noise checking result: Pass
- Sensor TT8 absolute limit checking result: Pass
- Sensor TT8 rate of change limit checking result: Pass
- Sensor TT8 presence of noise checking result: Pass
- Sensor LT1 absolute limit checking result: Pass
- Sensor LT1 rate of change limit checking result: Pass
- Sensor LT1 presence of noise checking result: Pass
- Calculated value for sensor LT1 is not within tolerance band of measurement. Since only one calculation is available, the data used in calculation are being validated

**Sensor Validation Important Results Window:**
- Sensor FT1 measurement is validated
- Sensor FT4 measurement is validated
- Sensor TT4 measurement is validated
- Alternate FT4 value being calculated to confirm
- Sensor ALF1 is valid
- Sensors (TT7 TT6 FT3 FT2) are set to be valid
- Sensors (TT5 TT3 TT2) are set to be valid
- Sensor TT8 measurement is validated
- The calculated value of sensor LT1 is chosen as final value
- Alarm ALL1 failed but validated

Figure 38. Complete Message Tracing for Case 2.
8) MLCl (level controller action) was high. These abstracted values were expected based on the discussion above of the effects of the malfunctions on the various parameters. It should be noted that for LT1, the data abstraction was done after the data were validated so that the "real" low level was correctly identified. For the validation results shown in the tracing, the only interesting result was for LT1. In this case, the alternate value did not match the original reading. Since the data used in estimating the alternate value were all validated, the alternate value was chosen as the final value.

The important points which should be noted based on the results of this case study include:

1. Multiple malfunction situations can be identified with the hierarchical diagnostic approach. As discussed above, the two initially assumed malfunctions were correctly identified.

2. This case shows how a sensor bias is identified and eliminated through the rigorous validation (Steps 11 and 12 in the validation algorithm). The alternate value for sensor LT1 was directly estimated by the temperature readings along the wall of the reactor. Since the alternate value did not match with the original measurement, the data used in determining the alternate value were validated. Since all the data used in the method were found to be valid, the alternate value was chosen to replace the original reading.

3. When there is a bias in a control system sensor, a working controller would maintain the sensor reading at the setpoint. As a result, the bias
is not observable as an abnormality in the sensor data although it indeed affects the process operating condition. In this case study, the bias in LT1 was not directly observed since the sensor data were normal and steady. However, the real reactor level was controlled at a lower than normal value. This changed the residence time for the reaction and so the product quality was adversely affected.

7.2.4 Case 3 - Equipment Malfunction Plus Category II Sensor Failure

Referring to the flowsheet in Figure 9, this case study involved the fouling of the heat exchanger and a negative bias of 0.2 m³/hr in the coolant flow sensor FT3. The heat exchanger fouling decreased the overall heat transfer coefficient of the heat exchanger. As a result, in order to maintain the same recycle temperature, a higher coolant flowrate was needed. As shown in Figure 39, the coolant flowrate increased, but the sensor measurements were lower than the actual flowrate due to the negative bias. The outlet temperature of the coolant decreased (see Figure 40) because the flowrate was higher and the heat transfer coefficient was lower. Referring to Figure 41, both the reactor and recycle temperatures remained essentially constant due to the control action. Since the coolant flowrate is expected to increase when the heat exchanger is fouled, the flowrate provided by the flow sensor FT3 is an important piece of symptomatic information in the diagnosis. As a result, although the failure of FT3 (being a Category II sensor) could not directly lead to a malfunction, it could certainly affect the result of diagnosis if left undetected.
Figure 39. Plots of Actual and Measured Coolant Flowrates for Case 3.
Figure 40. Plot of Coolant Outlet Temperature for Case 3.
Figure 41. Plots of Reactor and Recycle Temperatures for Case 3.
The results of the diagnosis is shown in the confidence value hierarchy in Figure 42. From the hierarchy, the "HeatExchanger" specialist node was established with a confidence value of 3. Thus, the heat exchanger malfunction originally assumed was correctly identified in the diagnosis.

The complete tracing of the messages written to the DataBase User Window and the Sensor Validation Information and Important Results Windows during the diagnosis is shown in Figure 43. In this case, the "abnormal" abstracted values included 1) MTC8 was low, 2) FT3 was high and 3) TT7 was low. These data were expected based on the discussion above. For the messages written to the validation windows, the results of validating FT3 should be noted. Based on the messages written to the Important Results Window, the error in FT3 was identified and the alternate value determined was used for diagnosis. The important points in this case study are:

1. Referring to the CSRL specialist code shown previously in Figure 12, the malfunction hypothesis in the "HeatExchanger" node can be evaluated based on both direct pattern matching and the result of calculation. In this particular case, the calculational result was able to provide conclusive evidence of the malfunction so that direct pattern matching was not performed. In other words, the "heatex" knowledge group was not evaluated. Instead, the "HYPOTHESIS-TEST" function in the "heatex-test" knowledge group executed the "HTEX-TEST" function which was indexed in the "HeatExchanger-Ob" node in the validation methods and quantitative models hierarchy. The "HTEX-TEST" function
Figure 42. Confidence Value Hierarchy for Case 3.
Tracing of messages written to the windows during diagnosis
Case: Heat exchanger fouling and coolant flow sensor FT3 bias

DataBase User Window:
[Abstracted normality value of FT1 is: N ]
[Abstracted normality value of TT4 is: N ]
[Abstracted trend value of TT4 is: STEADY ]
[Abstracted normality value of MTC8 is: L ]
[Abstracted normality value of FT3 is: H ]
[Abstracted normality value of TT6 is: N ]
[Abstracted trend value of TT6 is: STEADY ]
[Abstracted normality value of TT7 is: L ]
[Abstracted normality value of LT1 is: N ]
[Abstracted trend value of LT1 is: STEADY ]

Sensor Validation Information Window:
[Sensor FT1 absolute limit checking result: Pass ]
[Sensor FT1 rate of change limit checking result: Pass ]
[Sensor FT1 presence of noise checking result: Pass ]
[Sensor TT4 absolute limit checking result: Pass ]
[Sensor TT4 rate of change limit checking result: Pass ]
[Sensor TT4 presence of noise checking result: Pass ]
[Sensor FT2 absolute limit checking result: Pass ]
[Sensor FT2 rate of change limit checking result: Pass ]
[Sensor FT2 presence of noise checking result: Pass ]
[Sensor FT3 absolute limit checking result: Pass ]
[Sensor FT3 rate of change limit checking result: Pass ]
[Sensor FT3 presence of noise checking result: Pass ]
[Sensor FT3 normality test result: Fail ]
[Calculated value for sensor FT3 is not within tolerance band of measurement. Since only one calculation is available, the data used in calculation are being validated ]
[Class VP3 is not a sensor or alarm. Validation not performed ]
[Sensor TT6 absolute limit checking result: Pass ]
[Sensor TT6 rate of change limit checking result: Pass ]
[Sensor TT6 presence of noise checking result: Pass ]
[Sensor TT6 normality test result: Pass ]
[Sensor TT6 trend test result: Pass ]
[Sensor TT6 reliability ranking test result: Pass ]
[Sensor TT7 absolute limit checking result: Pass ]
[Sensor TT7 rate of change limit checking result: Pass ]
[Sensor TT7 presence of noise checking result: Pass ]
[Sensor TT7 normality test result: Fail ]
[Sensor TT7 reliability test result: Pass ]
[Sensor LT1 absolute limit checking result: Pass ]
[Sensor LT1 rate of change limit checking result: Pass ]
[Sensor LT1 presence of noise checking result: Pass ]

Sensor Validation Important Results Window:
[Sensor FT1 measurement is validated ]
[Sensors (FT4) are set to be valid ]
[Sensor TT4 measurement is validated ]
[Sensors (TT5 TT3 TT2) are set to be valid ]
[Sensor FT2 measurement is validated ]
[Sensor FT3 normality test result: Fail ]
[Alternate FT3 value being calculated to confirm ]
[The calculated value of sensor FT3 is chosen as final value ]
[Sensor TT6 reading passed all tests in validation module ]
[Sensor TT7 normality test result: Fail ]
[Alternate TT7 value being calculated to confirm ]
[Sensor TT7 measurement is validated ]
[Sensors (TT6) are set to be valid ]
[Sensor LT1 measurement is validated ]

Figure 43. Complete Message Tracing for Case 3.
determined the status of the heat exchanger based on the closure of two constraint equations. If the combined tube and shell side energy balance closed while either the tube or shell side energy balance alone did not close, it was concluded that the heat exchanger was fouled. The reason is that the single side balance includes the overall heat transfer coefficient term but the combined balance does not contain that coefficient.

2. Since FT3 is not a control system sensor, it does not appear in the malfunction hierarchy. The reason is that a failure of such a non-control system sensor cannot actually lead to a malfunction in the process.

7.2.5 Concluding Remarks

The main points illustrated in the case studies are summarized in the following paragraphs.

1. The two different sensor categories have different effects on the diagnosis. It can be seen that a Category I sensor failure is represented in a node in the malfunction hierarchy such that it is explicitly considered. On the other hand, a Category II sensor failure is only considered when the sensor reading is needed for the diagnosis of another equipment malfunction.

2. The validation algorithm can identify the different modes of sensor failure, including biases.
3. The sensors are validated only when they are needed. Therefore, the sensors which are actually validated are different from case to case. This can be seen by comparing the different lists of sensors validated for the three cases studied.

4. Rigorous validation relying heavily on quantitative equation-based methods will probably lead to conflict situations which may not be resolvable. Thus, different types of methods should be used. Also, the approaches for minimizing rigorous validation are effective in improving the performance of the validation strategy.

5. The prototype system can successfully identify multiple malfunction situations.

6. A malfunction hypothesis can be successfully evaluated based on the results of quantitative calculations.
CONCLUSIONS AND RECOMMENDATIONS

The major contributions of this research include 1) a new, effective characterization of the sensor validation problem in terms of distinctions in sensor functions and modes of failure, 2) a novel sensor validation strategy which is designed for supporting chemical process plant diagnosis, 3) a systematic functional decomposition strategy which enhances the construction of a diagnostic hierarchy from the process flowsheet, and 4) a computational architecture which effectively integrates the use of both qualitative and quantitative knowledge. The significance of these contributions is summarized in the following paragraphs.

1. Problem Characterization

Through an analysis of the characteristics of sensor failures, it is realized that both the intended function of the sensor and the particular mode of failure involved have significant impact on the sensor validation problem. In this research, it is proposed that sensors can be classified into two distinct categories, where one category includes control system sensors (Category I) and the other category consists of non-control system sensors (Category II). This categorization facilitates the sensor validation in the context of malfunction diagnosis. It is shown that a failure of only a Category I sensor can directly lead to undesirable operating
conditions so that the failure is explicitly considered in malfunction diagnosis. This is accomplished by representing the Category I sensor malfunction in a hypothesis node in the malfunction hierarchy. Both sensor categories can exhibit the four different modes of sensor failure considered, which include mechanical failures as well as biases. Mechanical sensor failures are directly observable in the sensor measurements for both sensor categories. However, sensor biases can only be directly observed in Category II sensors because, for a Category I sensor, the control action maintains the measurements at the setpoint unless the controller is saturated or failed. Different approaches are used for identifying these different failure modes and it is shown that the identification of sensor biases is most difficult.

2. Novel Sensor Validation Strategy

The sensor validation strategy developed in this research takes into account the characteristics of chemical plants, the sensor characteristics and modes of failure, and the impact of malfunctions on the validation problem to take full advantage of the resources available for sensor validation. The important aspects of the validation strategy are listed below.

a. Recognizing the problem characteristics, various tactics are included in the validation algorithm for enhancing the computational efficiency. These tactics reflect the pragmatic use of knowledge about the problem, which resembles what a human being would do, in streamlining the problem solving involved.
- First, different steps are included in the algorithm to identify the different modes of sensor failures. The independent data checks are simple to apply and are effective for identifying those mechanical sensor failures. By applying these checks first in the algorithm, the more complicated failure identification steps do not need to be used if these modes of failure are present. On the other hand, sensor biases are identified by the more elaborate rigorous validation steps using some form of correlations including direct process relationships and quantitative models.

- Second, since the rigorous validation steps are often computation expensive, two approaches are employed to minimize the application of these steps. In one approach, if the alternate value determined for confirming a sensor reading matches the original reading, then the sensor data used in the determination are taken to be valid. The rationale is that the probability for all the sensor data involved to fail in a certain way at the same time such that the alternate value determined matches with the original value is very low. In the second approach, the sensor characteristics (reliability history, sensor category) and the current data (trend, normality, etc.) are looked at qualitatively to determine the probability of a sensor failure in the data. If the probability of a failure is very low, then the rigorous validation steps are not performed and the data are taken to be valid. Here, the algorithm is effectively leveraging information pertaining to the operational history.
and characteristics of the sensors to help reduce the amount of rigorous validation.

b. Due to the fact that instrumentation is very limited in the chemical process plant domain, the sensor validation strategy developed emphasizes flexibility in the knowledge representation. Flexibility in the representation allows the incorporation of any applicable methods for validation, so that methods specifically developed for an individual sensor in a particular process can be used. The ability to use a variety of methods is important because both qualitative knowledge and quantitative process model equations represent valuable sources of information for sensor validation.

c. The sensor validation algorithm is integrated into malfunction diagnosis such that the diagnosis has the overall problem-solving control and specifies which sensors need to be validated.

3. Systematic Functional Decomposition Strategy

For hierarchical classification, efficiency is derived from the levels of process detail captured in the levels of the hierarchy. Due to the complexity and diversified nature of the CPP domain, constructing a hierarchical decomposition of the process plant so that the classificatory diagnostic task can be applied is non-trivial. The systematic functional decomposition strategy developed in this research provides general guidelines for the identification of functional subsystems in various levels of process detail. It facilitates the construction of the diagnostic hierarchy, where
malfunction hypotheses for the various functional subsystems represent the primary conceptual basis of the hierarchy. Therefore, the functional decomposition strategy enhances the application of the classificatory diagnostic approach. Moreover, the functional decomposition strategy effectively identifies the Category I sensors which need to be represented in the malfunction hierarchy. On the other hand, Category II sensors are not included in the malfunction hierarchy because they cannot directly cause any actual malfunctioning process conditions.

4. Computational Architecture

The computational architecture for the integrated diagnosis and sensor validation system consists of three distinct hierarchical structures, each is responsible for a crucial part of the problem solving. This architecture provides organizing constructs for both symbolic and quantitative model-based knowledge, and is shown to be effective for diagnosis and sensor validation in the CPP domain. The important features of the tasks and the associated hierarchical structures are summarized below.

a. The classificatory diagnostic task is the core task in the architecture and is captured in a malfunction hierarchy, which is constructed primarily based on a functional decomposition of the process. The malfunction hypotheses in the hierarchy are effectively evaluated by both direct pattern matching and results of quantitative calculational methods.

b. The sensor validation task is an auxiliary task to diagnosis. The information for performing the validation is organized in the validation
methods and quantitative models hierarchy. The validation information includes various correlations, ranging from approximation to process models. The validation module successfully identifies the various modes of sensor failures, including biases, under malfunctioning process operating conditions. From the case studies, the tactics employed for minimizing the amount of rigorous validation are shown to be effective in improving the computational efficiency. This hierarchy also provides the organizing structure for the calculational methods used in the quantitative hypothesis evaluation, which is illustrated with a case study.

c. Data abstraction is another necessary auxiliary task to diagnosis. The symptomatic data for diagnosis and sensor validation are conveniently stored in the symptoms database. This hierarchy provides a central location where the numerical process data and the various parameters necessary for the data abstraction task and the independent data checks for the validation task are organized.

RECOMMENDATIONS

During the course of this research, several issues are found to be particularly valuable to further pursue.

1. It is believed that when the sensor data pass straight through the algorithm without invoking the rigorous validation, or when the data used in a calculation are taken to be valid because the alternate value determined matches with the original reading, the probability for a sensor failure in the
data is very low. However, it is recognized that this probability, no matter how low, still exists. In the rare event that such an error actually occurs, the diagnosis will be using some erroneous data and the results of diagnosis will be adversely affected. That is indeed the reason why this category of data are marked differently from the rigorously validated data when the validation strategy is done. In this way, the validation strategy can be extended by incorporating a tactic for going back to this category of data and perform the rigorous validation.

2. The computational architecture is demonstrated with data from a simulated process. The use of actual plant data from a real process should be the next step in further testing the methodology.

3. The performance of the validation strategy depends on the methods which are available for validating the sensor data. Therefore, it is important to study the methods which can be used. There are several related issues.

- The methods which can be applied ultimately depend on the availability of process data. Therefore, the sensors present in the plant and their locations have significant effects on the validation. These effects should be investigated so that general guidelines for placements of sensors may be specified.

- The effect of different combinations of direct relationships and quantitative model calculation in the validation methods should be studied. It is expected that if more direct relationships can be used, the performance of the validation strategy should be improved.
- Identification of additional sources of knowledge for determining the "true" values for the sensors. Although product quality is not used in this research, it is recognized that product quality is adversely affected by sensor failures. Therefore, product quality can possibly be used to suggest the probable failure of a sensor. The use of product quality information has been developed by Ramesh [29].

- Other independent data checks using advanced data analysis techniques can be developed and incorporated into the validation algorithm. The independent data checks currently employed in the validation module include absolute and rate of change limit checks and the check for presence of noise. Other checks such as analysis of the noise characteristics can also be incorporated if they are found to be necessary.

- Various data abstraction and analysis methods using neural network are currently being developed. The application of these advanced methods in the computational framework should be studied.

4. Although the quantitative methods used in this research are implemented in the LISP environment, the organizational constructs developed indeed facilitate the incorporation of quantitative functions implemented in a foreign programming environment. The use of these foreign functions in the computational architecture should be further studied.
LIST OF REFERENCES


APPENDIX A

DYNAMIC CSTR PROCESS SIMULATION
PROCESS MODEL EQUATIONS

Stirred Tank Reactor

Overall Mass Balance:
\[ \frac{dV}{dt} = F_i + F_r - F \] (constant density) (1)

Component A Mass Balance:
\[ \frac{d(CAV)}{dt} = F_i*C_{A,i} + (F_r-F)*C_A - k*C_{A}^2*V \] (2)

Combine (1) and (2):
\[ V\frac{d(C_A)}{dt} = F_i*(C_{A,i}-C_A) - k*C_{A}^2*V \] (3)

Energy Balance:
\[ \frac{dT}{dt} = F_i*\frac{(T_i-T)}{V} + F_r*\frac{(T_r-T)}{V} - \delta H_R*k*C_{A}^2/(\rho*C_p) \] (4)

Reaction Rate Coefficient:
\[ k = k_o * \exp\{-E/[R*(T+273.16)]\} \] (5)

Heat Exchanger Energy Balances

Recycle stream (shell side)
\[ \delta T/\delta t = -4F_i/[\pi*(D_o^2-n*D_i^2)]\delta T/\delta z - 4n*D_i*U*(T-T_c)/[\rho*C_p*(D_o^2-n*D_i^2)] \] (6)

Coolant stream (tube side)
\[ \delta T_c/\delta t = -4F_o/[(\pi*n*D_i^2)]\delta T_c/\delta z + 4*U*(T-T_c)/(\rho_c*C_{pc}*D_i) \] (7)
Nomenclature

\[ \begin{align*}
C_A &= \text{Concentration of A in reactor, kgmole/m}^3 \\
C_{fu} &= \text{Feed concentration of A, kgmole/m}^3 \\
C_p &= \text{Heat capacity of reactor liquid content, J/kgmole-°C} \\
C_{pc} &= \text{Specific heat capacity of coolant stream, J/kg-°C} \\
D_t &= \text{Heat exchanger tube diameter, m} \\
D_s &= \text{Heat exchanger shell diameter, m} \\
E &= \text{Activation energy of the reaction, J/kgmole} \\
F &= \text{Flowrate of reactor outlet stream, m}^3/\text{min} \\
F_c &= \text{Coolant flowrate, m}^3/\text{min} \\
F_f &= \text{Feed flowrate, m}^3/\text{min} \\
F_p &= \text{Product flowrate, m}^3/\text{min} \\
F_r &= \text{Recycle flowrate, m}^3/\text{min} \\
k &= \text{Reaction rate coefficient, m}^3/\text{min-kgmole} \\
k_p &= \text{Arrhenius coefficient, m}^3/\text{min-kgmole} \\
n &= \text{Number of tubes in heat exchanger} \\
R &= \text{Ideal gas law constant, 8314.39 J/kgmole-K} \\
T &= \text{Reactor temperature, °C} \\
T_c &= \text{Coolant temperature, °C} \\
T_f &= \text{Feed temperature, °C} \\
T_r &= \text{Recycle temperature, °C} \\
t &= \text{Time, min} \\
U &= \text{Overall heat-transfer coefficient, J/min-m}^2-\text{°C} \\
V &= \text{Reactor liquid volume, m}^3 \\
z &= \text{Length of heat exchanger, m} \\
\delta H_R &= \text{Heat of reaction, J/kgmole} \\
p &= \text{Density of reactor liquid content, kgmole/m}^3 \\
p_c &= \text{Density of coolant stream, kg/m}^3
\end{align*} \]
ACSL CODES

PROGRAM - Dynamic CSTR Simulation

INITIAL

ARRAY TH(50), TC(50), DTHDT(50), DCDT(50), THIC(50), TCIC(50)
INTEGER I, IEND, N
ALGORITHM IALG=1
XERROR V=0.07, TH=0.1, TC=0.1
CINTERVAL CINT=0.1
NSTEPS NSTP=1
MAXINTERVAL MAXT=0.01
CONSTANT ETIME=10.
CONSTANT IEND=50
CONSTANT PI=3.1416
CONSTANT R=8314.39 $ "J/KG MOLE-K"
II Reaction related constants 
CONSTANT K0=4.464 $ "CM/MIN-KGMOLE"
CONSTANT E=1.182E07 $ "J/KGMOLE"
CONSTANT DHR=-9.86E07 $ "J/KGMOLE"
CONSTANT VS=7.08 $ "CM"

II Heat exchanger related constants
CONSTANT 00=0.5, 01=0.03, LH =3. $ "M"
CONSTANT N=20, U=2.13E05 $ "J/MIN-SO.M-C"
CONSTANT RHOC=1000., CPC=4184. $ "KG/CM; J/KG-C"
CONSTANT RH0=19.2, CP=1.815E05 $ "KG MOLE/CM; J/KG MOLE-C"
CONSTANT TRS=68.0553 $ "DEG C"
CONSTANT TCS=40.0168 $ "DEG C"
CONSTANT FCS=0.9 $ "CM/MIN"

II Set TH IC and TC IC initial s.s. vectors for heat exchanger
CONSTANT THIC=83.2516,82.8253,82.4049,81.9905,81.5818,81.1789,...
78.7685,78.3468,75.1296,74.8169,74.5085,74.2044,73.9046,...
73.6090,73.3175,73.0301,72.7467,72.4673,72.1917,71.9200,...
71.6522,71.3880,71.1276,70.8708,70.6176,70.3679,70.1217,...
69.8790,69.6396,69.4036,69.1709,68.9415,68.7152,68.4922,...
68.2722,68.0553
CONSTANT TCIC=27.3601,27.7151,27.0652,27.4104,27.7508,28.0864,...
29.4173,29.7436,30.0653,30.3826,30.6954,31.0038,31.3079,...
31.6078,31.9034,32.2010,32.4824,32.7659,33.0424,33.3209,...
33.5927,33.8606,34.1268,34.3852,34.6421,34.8993,35.1540,...
35.3913,35.6340,35.8734,36.1095,36.3422,36.5717,36.7980,...
37.0211,37.2411,37.4580,37.6719,37.8828,38.0907,38.2958,...
38.4979,38.6973,38.8938,39.0877,39.2788,39.4672,39.6530,...
39.8362,40.0168

II-Initialize possible disturbance and setpoint changes
CONSTANT CAIS=2.88, DCAI=0.0, TCAI=0.0 $ "KG MOLES/CM" 
CONSTANT TIS=66., DTI=0.0, TDI=0.0 $ "DEG C"
CONSTANT TCDS=27., DTCO=0.0, TDCO=0.0 $ "DEG C"
CONSTANT FIS=0.45, DFI=0.0, TDFI=0.0 $ "CM/MIN"
CONSTANT FRS=0.9, DF=0.0, TDFR=0.0 $ "CM/MIN"
CONSTANT TRXS=83.6839, DTRX=0.0, TDRX=0.0 $ "DEG C"
CONSTANT DL=0.0, TDL=0.0 $ "M"
-----Calculate constants and initial steady state conditions-----

\[ \text{EOR} = \frac{E}{R} \]

\[ \text{DHRD} = \text{DHR}/(\text{RHOC} \times \text{CP}) \]

\[ \text{DEL} = \text{LH}/\text{IE HD} \]

\[ \text{A} = \left( \text{DO} \times 2 \times \text{NPDI} \times 2 \right) \]

\[ \text{C1} = \frac{\text{LH}}{\text{IEN}} \]

\[ \text{C2} = \frac{4 \times \text{N} \times \text{DP} / \text{U}}{\sqrt{\text{R} \times \text{CP} \times \text{A}}} \]

\[ \text{C3} = \frac{4 \times \text{D1}}{\text{PI} \times \text{A}} \]

\[ \text{C4} = \frac{4 \times \text{U} \times \text{PI} \times \text{CP} \times \text{DI}}{\text{RHOC} \times \text{A}} \]

\[ \text{TRX} = 0.95 \times \text{TRXS} \]

\[ \text{ARX} = \frac{\text{PI} \times 2.25 \times 2}{4} \]

\[ \text{LS} = \sqrt{\text{VU} \times \text{ARX}} \]

\[ \text{FPS} = \text{FIS} \]

\[ \text{FS} = \text{FIS} + \text{FRS} \]

\[ \text{KS} = \text{KOW} \times \exp(-\text{EOR}/(\text{TRXS} + 273.16)) \]

\[ \text{CAS} = \frac{(-\text{FIS} \times \sqrt{\text{FIS}^2 + 4 \times \text{KS} \times \text{VS} \times \text{CAS}})}{2 \times \text{KS} \times \text{VS}} \]

"---------- Sensor characteristics ----------"

"-----Feed conc. sensor AT1-----"

\[ \text{CONSTANT A1MIN} = 1.5 \]

\[ \text{CONSTANT D1A1} = 3 \]

\[ \text{TUA1} = 2./60. \]

"-----Product conc. sensor AT2-----"

\[ \text{CONSTANT A2MIN} = 2 \]

\[ \text{CONSTANT D2A2} = 3 \]

\[ \text{TUA2} = 2./60. \]

"-----Feed flow sensor FT1-----"

\[ \text{CONSTANT F1MIN} = 0.0 \]

\[ \text{CONSTANT D1FT1} = 1.2 \]

\[ \text{TUA1} = 2./60. \]

"-----Recycle flow sensor FT2-----"

\[ \text{CONSTANT F2MIN} = 0.0 \]

\[ \text{CONSTANT D2FT2} = 2.4 \]

\[ \text{TUA2} = 2./60. \]

"-----Coolant flow sensor FT3-----"

\[ \text{CONSTANT F3MIN} = 0.0 \]

\[ \text{CONSTANT D3FT3} = 2.4 \]

\[ \text{TUA3} = 2./60. \]

"-----Product flow sensor FT4-----"

\[ \text{CONSTANT F4MIN} = 0.0 \]

\[ \text{CONSTANT D4FT4} = 1.2 \]

\[ \text{TUA4} = 2./60. \]

"-----Level sensor LT1-----"

\[ \text{CONSTANT L1MIN} = 0.0 \]

\[ \text{CONSTANT D1LT1} = 2.5 \]

\[ \text{TUA1} = 2./60. \]

"-----Feed temp. sensor TT1-----"

\[ \text{CONSTANT T1MIN} = 45 \]

\[ \text{CONSTANT D1TT1} = 40 \]

\[ \text{TUA1} = 2./60. \]

"-----Reactor temp. sensor TT2-----"

\[ \text{CONSTANT T2MIN} = 65 \]

\[ \text{CONSTANT D1TT2} = 40 \]
TAUT2=2./60. $ "Temp. sensor time constant, MIN"

"--- Reactor temp. sensor TT3---"
CONSTANT T3MIN=65.
CONSTANT DT3T=40.
TAUT3=2./60.

"--- Reactor temp. sensor TT4---"
CONSTANT T4MIN=65.
CONSTANT DT4T=40.
TAUT4=2./60.

"--- Reactor temp. sensor TT5---"
CONSTANT T5MIN=65.
CONSTANT DT5T=40.
TAUT5=2./60.

"--- Coolant inlet temp. sensor TT6---"
CONSTANT T6MIN=10.
CONSTANT DT6T=40.
TAUT6=2./60.

"--- Coolant outlet temp. sensor TT7---"
CONSTANT T7MIN=20.
CONSTANT DT7T=40.
TAUT7=2./60.

"--- Recycle temp. sensor TT8---"
CONSTANT T8MIN=50.
CONSTANT DT8T=40.
TAUT8=2./60.

"-------- Control valve characteristics --------"

"-------- Coolant flow valve --------"
CONSTANT FC_MAX=2.4 $ "CU.M/MIN"
CONSTANT ALPFC=50. $ "Valve rangeability parameter"

"-------- Feed flow valve --------"
CONSTANT FIMAX=1.2 $ "CU.M/MIN"
CONSTANT ALPFI=50. $ "Valve rangeability parameter"

"-------- Recycle flow valve --------"
CONSTANT FRMAX=2.4 $ "CU.M/MIN"
CONSTANT ALPFR=50. $ "Valve rangeability parameter"

"-------- Product flow valve --------"
CONSTANT FPMAX=1.2 $ "CU.M/MIN"
CONSTANT ALPPP=50. $ "Valve rangeability parameter"

"-------- Calculate initial sensor and controller signals --------"

BAS=(CAIS-A1MIN)/DA1T
BA2S=(CAS-A2MIN)/DA2T
BF1S=(FIS-F1MIN)/DF1T
BF2S=(FIS-F2MIN)/DF2T
BF3S=(FCS-F3MIN)/DF3T
BF4S=(FPS-F4MIN)/DF4T
BL1S=(LS-L1MIN)/DL1T
BT1S=(TIS-TMIN)/DT1T
$BT2S=(TRXS-T2MIN)/DT2T$
$BT3S=(TRXS-T3MIN)/DT3T$
$BT4S=(TRXS-T4MIN)/DT4T$
$BT5S=(TRXS-T5MIN)/DT5T$
$BT6S=(TRXS-T6MIN)/DT6T$
$BT7S=(TRXS-T7MIN)/DT7T$
$BT8S=(TRXS-T8MIN)/DT8T$

$NFIS=ALOG(FIS/FIMAX)/ALOG(ALPF1)$ $"NF C1-Feed flow control"
$NF R S=ALOG(FRS/FRMAX)/ALOG(ALPF 2)$ $"NF C2-Recycle flow control"
$NFPS=ALOG(FPS/FPMAX)/ALOG(ALPF 3)$ $"MLC1-Reactor level control"
$MFCS=ALOG(FCS/FCMAX)/ALOG(ALPF C)$ $"MTCB-Recycle temp. control"

"---Controller tuning parameters---"

"Flow controller FRC1"
CONSTANT KCF1=-6., TAUIF1=.05

"Flow controller FRC2"
CONSTANT KCF2=-6., TAUIF2=.05

"Level controller LRC1"
CONSTANT KCL1=3., TAUI L1=3.

"Temperature primary controller TRC4"
CONSTANT KCT4=1.5, TAUIT4=2.5

"Temperature secondary controller TRC8"
CONSTANT KCT8=1.1, TAUIT8=.23

END $ "OF INITIAL"

DYNAMIC

DERIVATIVE
TIME=T

"---Conditions that may be changed during the simulation---"

"Disturbances---"

"Reactant conc. in KGMOLES/CU.M"
$CAI=CAI+DCAI*STEP(TDCAI)$

"Reactant temp. in DEG C"
$TI=TI S+DTI*STEP(TDTI)$

"Coolant inlet temp. in DEG C"
$TCO=TCOS+DTCO*STEP(TDCO)$

"Setpoints---"

"Steady state conditions are chosen as setpoints"

"Reactant flowrate setpoint SFC1 in CU.M/S"
$F I S E T=F I S+DFI*STEP(TDFI)$

"Recycle flowrate setpoint SFC2 in CU.M/S"
$FRSE T=FRS+DFR*STEP(TDFR)$

"Reactor level setpoint SL C1 in M"
$LSET=LSRL*STEP(TD L )$
"Reactor temp. setpoint ST C4 in DEG C"
TRXSET=TRXS+DTRX*STEP(DTSTRX)

"----Calculate normalized set point----"
NFISET=(FISET-F1MIN)/DF1T
NFSET=(FSET-F2MIN)/DF2T
NLSET=(LSET-L1MIN)/DL1T
NTXSET=(TRXSET-T4MIN)/DT4T

"----------------------------------------------"
"Feed conc. sensor AT1----"  
BA1SD=0.015/DA1T  
DBA1DT=((CA1-A1MIN)/DA1T-BA1)/TAU1  
BA1T=INTEG(DBA1DT, BA1S)+GAUSS(0.0, BA1SD)  
BA1=BOUND(0.0, 1.0, BA1T)  

"Product conc. sensor AT2----"  
BA2SD=0.015/DA2T  
DBA2DT=((CA2-A2MIN)/DA2T-BA2)/TAU2  
BA2T=INTEG(DBA2DT, BA2S)+GAUSS(0.0, BA2SD)  
BA2=BOUND(0.0, 1.0, BA2T)  

"Feed flow sensor FT1-----"  
BF1SD=0.005/DF1T  
DBF1DT=((FI-F1MIN)/DF1T-BF1)/TAUF1  
BF1T=INTEG(DBF1DT, BF1S)+GAUSS(0.0, BF1SD)  
BF1=BOUND(0.0, 1.0, BF1T)  

"Recycle flow sensor FT2-----"  
BF2SD=0.01/DF2T  
DBF2DT=((FR-F2MIN)/DF2T-BF2)/TAUF2  
BF2T=INTEG(DBF2DT, BF2S)+GAUSS(0.0, BF2SD)  
BF2=BOUND(0.0, 1.0, BF2T)  

"Coolant flow sensor FT3-----"  
BF3SD=0.01/DF3T  
DBF3DT=((FC-F3MIN)/DF3T-BF3)/TAUF3  
BF3T=INTEG(DBF3DT, BF3S)+GAUSS(0.0, BF3SD)  
BF3=BOUND(0.0, 1.0, BF3T)  

"Product flow sensor FT4-----"  
BF4SD=0.005/DF4T  
DBF4DT=((FP-F4MIN)/DF4T-BF4)/TAUF4  
BF4T=INTEG(DBF4DT, BF4S)+GAUSS(0.0, BF4SD)  
BF4=BOUND(0.0, 1.0, BF4T)  

"Reactor Level sensor LT1----"  
BL1SD=0.01/DL1T  
DBL1DT=((L-L1MIN)/DL1T-BL1)/TAUL1  
BL1T=INTEG(DBL1DT, BL1S)+GAUSS(0.0, BL1SD)  
BL1=BOUND(0.0, 1.0, BL1T)  

"Feed temp. sensor TT1-----"  
BT1SD=0.2/DT1T  
DBT1DT=((TI-T1MIN)/DT1T-BT1)/TAUT1  
BT1T=INTEG(DBT1DT, BT1S)+GAUSS(0.0, BT1SD)  
BT1=BOUND(0.0, 1.0, BT1T)  

"Reactor temp. sensor TT2-----"  
BT2SD=0.2/DT2T  
DBT2DT=((TRX-T2MIN)/DT2T-BT2)/TAUT2  
BT2T=INTEG(DBT2DT, BT2S)+GAUSS(0.0, BT2SD)  
BT2=BOUND(0.0, 1.0, BT2T)  

"Reactor temp. sensor TT3-----"  
BT3SD=0.2/DT3T  
DBT3DT=((TRX-T3MIN)/DT3T-BT3)/TAUT3  
BT3T=INTEG(DBT3DT, BT3S)+GAUSS(0.0, BT3SD)  
BT3=BOUND(0.0, 1.0, BT3T)  

"Reactor temp. sensor TT4-----"  
BT4SD=0.2/DT4T  
DBT4DT=((TRX-T4MIN)/DT4T-BT4)/TAUT4
\[ B_{T4} = \text{INTEG}(DB_{T4DT}, BT_{4S}) + \text{GAUSS}(0.0, BT_{4SD}) \]
\[ BT_{4} = \text{BOUND}(0.0, 1.0, BT_{4}) \]

"---Reactor temp. sensor TT5---"
\[ BT_{5SD} = 0.2/DT_{5T} \]
\[ DB_{T5DT} = ((TRX-TMIN)/DT_{5T}-BT_{5})/TAU_{5T} \]
\[ BT_{5T} = \text{INTEG}(DB_{T5DT}, BT_{5S}) + \text{GAUSS}(0.0, BT_{5SD}) \]
\[ BT_{5} = \text{BOUND}(0.0, 1.0, BT_{5}) \]

"---Coolant inlet temp. sensor TT6---"
\[ BT_{6SD} = 0.2/DT_{6T} \]
\[ DB_{T6DT} = ((TC0-T6MIN)/DT_{6T}-BT_{6})/TAU_{6T} \]
\[ BT_{6T} = \text{INTEG}(DB_{T6DT}, BT_{6S}) + \text{GAUSS}(0.0, BT_{6SD}) \]
\[ BT_{6} = \text{BOUND}(0.0, 1.0, BT_{6}) \]

"---Coolant outlet temp. sensor TT7---"
\[ BT_{7SD} = 0.2/DT_{7T} \]
\[ DB_{T7DT} = ((TT7-T7MIN)/DT_{7T}-BT_{7})/TAU_{7T} \]
\[ BT_{7T} = \text{INTEG}(DB_{T7DT}, BT_{7S}) + \text{GAUSS}(0.0, BT_{7SD}) \]
\[ BT_{7} = \text{BOUND}(0.0, 1.0, BT_{7}) \]

"---Recycle temp. sensor TT8---"
\[ BT_{8SD} = 0.2/DT_{8T} \]
\[ DB_{T8DT} = ((TT8-T8MIN)/DT_{8T}-BT_{8})/TAU_{8T} \]
\[ BT_{8T} = \text{INTEG}(DB_{T8DT}, BT_{8S}) + \text{GAUSS}(0.0, BT_{8SD}) \]
\[ BT_{8} = \text{BOUND}(0.0, 1.0, BT_{8}) \]

"--------------------------- Controller action ---------------------------"

**PROCEDURAL**

"---Feed flow controller FRC1---"
\[ ER_{RF1} = N_{FISET} - BF_{1} \]
\[ IN_{TF1} = \text{INTEG}(ER_{RF1}, 0.0)/TAU_{1F1} \]
\[ MFI = MFI + KCF1*(ER_{RF1} + IN_{TF1}) \]
\[ MFC1 = \text{BOUND}(0.0, 1.0, MFI) \]
\[ VP1 = ALPF1**(-MFC1) \]
\[ FI = FIAX*VP1 \]

"---Recycle flow controller FRC2---"
\[ ER_{RF2} = N_{FRSET} - BF_{2} \]
\[ IN_{TF2} = \text{INTEG}(ER_{RF2}, 0.0)/TAU_{1F2} \]
\[ MFR = MFR + KCF2*(ER_{RF2} + IN_{TF2}) \]
\[ MFC2 = \text{BOUND}(0.0, 1.0, MFR) \]
\[ VP2 = ALPF2**(-MFC2) \]
\[ FR = FRAX*VP2 \]

"---Level controller LRC1---"
\[ ER_{RL1} = NLSET - BL_{1} \]
\[ IN_{TL1} = \text{INTEG}(ER_{RL1}, 0.0)/TAU_{1L1} \]
\[ MFL = MFL + KCL1*(ER_{RL1} + IN_{TL1}) \]
\[ MLC1 = \text{BOUND}(0.0, 1.0, MFL) \]
\[ VP4 = ALPF4**(-MLC1) \]
\[ FP = FPAX*VP4 \]

"---Temperature controller TRC4---"
\[ ER_{RT4} = NTXSET - BT_{4} \]
\[ IN_{TT4} = \text{INTEG}(ER_{RT4}, 0.0)/TAU_{1T4} \]
\[ MTT8 = MTT8 + KCT4*(ER_{RT4} + IN_{TT4}) \]
\[ MTC4 = \text{BOUND}(0.0, 1.0, MTT8) \]

"---Temperature controller TAC8---"
ERRT8=MTC4*BT8
INTT8=INTEG(ERRT8, 0.0)/TAUITB
MFC=MFC5+KCT8*(ERRT8+INTT8)
MTC8=BOUND(0.0, 1.0, MFC)
VP3=ALPFC**(-MTC8)
FC=FCMAX**VP3

END $ "OF PROCEDURAL"

"------------------------------------------------------------------"
"------------ Calculate measured sensor data -------------""------------
"------------------------------------------------------------------"

A1H=BA1*DA1T+A1MIN
A2H=BA2*DA2T+A2MIN
F1H=BF1*DF1T+F1MIN
F2H=BF2*DF2T+F2MIN
F3H=BF3*DF3T+F3MIN
F4H=BF4*DF4T+F4MIN
L1H=BL1*DL1T+L1MIN
T1H=BT1*DT1T+T1MIN
T2H=BT2*DT2T+T2MIN
T3H=BT3*DT3T+T3MIN
T4H=BT4*DT4T+T4MIN
T5H=BT5*DT5T+T5MIN
T6H=BT6*DT6T+T6MIN
T7H=BT7*DT7T+T7MIN
T8H=BT8*DT8T+T8MIN

"----Termination condition-----"
TERM(TIME .GE. ETIME)

END $ "OF DERIVATIVE"

END $ "OF DYNAMIC"

TERMINAL

WRITE(9,901) FISET
WRITE(9,902) FRESET
WRITE(9,903) LSET
WRITE(9,904) TRXSET

901..FORMAT(1X,'Feed Flow Setpoint SFC1 =',3X,F9.7)
902..FORMAT(1X,'Recycle Flow Setpoint SFC2 =',3X,F9.7)
903..FORMAT(1X,'Reactor Level Setpoint SLC1 =',3X,F9.7)
904..FORMAT(1X,'Reactor Temp. Setpoint STC4 =',3X,F9.6)

END $ "OF TERMINAL"

END $ "OF PROGRAM"
(messages (Establish (SetConfidence self 3))

(Specialist FFControllerFC1 (declare (superspecialist FFFlowFBCtrl)
 (subspecialists))
 (kgs ( FFControllerFC1 Table (* ; ' Increase In FC1 controller signal MFC1 decreases valve position VP1, thus decreases
 feed flowrate")
 (match (DB-FETCH 'FT1 'value)
)

(Specialist FFSensorFTI (declare (superspecialist FFControllerFC1)
 (subspecialists))
 (kgs (FFSensorFTI Rules
 (match (DATA-FETCH 'FTI-Ob Validity)
 with
 (if Valid
 then -3
 else Invalid
 then 3
 else Through
 then -2
 else Probable
 then -1
 else 0]
 (messages (Establish (SetConfidence self sensorFTI)

(Specialist FFFlowFBCtrl (declare (superspecialist FFFlowCtrlSys)
 (subspecialists FFControllerFC1 FFValve)
 (kgs (FFFlowFBCtrl Table (match (DB-FETCH 'FT1 'value)
 (DB-FETCH 'VP1 'value)
 with
 (if (OR L LL)
 (OR L LL)
 then 3
 else (OR H HH)
 (OR H HH)
 then 3
 else (OR H HH)
 (OR L LL)
 then -3
 else (OR L LL)
 (OR H HH)
 then -3
 else 0]
 (messages (Establish (SetConfidence self fflowfb)

(Specialist FFFlowCtrlSys (declare (superspecialist Process)
 (subspecialists FFFlowCtrlSys RTempCtrlSys RRecycleFlowCtrlSys RLevelCtrlSys))
 (kgs)
 (messages (Establish (SetConfidence self 3)

(Specialist Process (declare (superspecialist)
 (subspecialists FFFlowCtrlSys RTempCtrlSys RRecycleFlowCtrlSys RLevelCtrlSys))
 (kgs)
(DB-FETCH 'MFC1 value)
with
(if (OR L LL)
  (OR N H HH)
  then 3
elseif (OR H HH)
  (OR N L LL)
  then 3
elseif (OR H HH)
  (OR H HH)
  then -3
elseif (OR L LL)
  (OR L LL)
  then -3
else 0)
)
(FC1-set Table (match (DB-FETCH 'FT1 value)
  (DB-FETCH 'SFC1 value)
  with
  (if (OR L LL)
    (OR L LL)
    then 3
  elseif (OR H HH)
    (OR H HH)
    then 3
  elseif ? N
    then -3
  elseif (OR H HH)
    (OR L LL)
    then -3
  elseif (OR L LL)
    (OR H HH)
    then -3
  else 0))
)
(messages (Establish (SetConfidence self (MAX ffFC1 FC1-set))

[Specialist FFValve (declare (superspecialist FFlowFBCtrl)
(subspecialists))]
[kgs (ffval-test Rules (match (HYPOTHESIS-TEST 'FFValve-Ob)
with
  (if Established
    then 3
  elseif Rejected
    then -3
  else 0)))]
(ffval Table
(match (AskYNU? "Does feed control valve pass physical check")
  (BCSR+ ? FFSensorFT1)
  (BCSR+ ? FFControllerFC1)
  with
  (if T ??
    then -3
  elseif F ??
    then 3
  elseif ? NIL NIL
    then 2
  elseif ? (NOT NIL) ?
    then -2
  elseif ?? (NOT NIL)
    then -2
  else 1)
  (messages (Establish (if (NOT (EQ ffval-test 0))
    then
    (SetConfidence self ffval-test)
    else
    (SetConfidence self ffval)
[Specialist FeedSupply (declare (superspecialist FFLOWCtrlSys)  
(subspecialists))
[kgs (feedsup Table  
(match (DB-FETCH 'FT1 Value)  
(AskYN? "Is feed supply depleted")  
with  
(if (OR H HH N)  
? then -3  
else ? F  
then -3  
else ? T  
then 3  
else 0)  
(messages (Establish (SetConfidence self feedsup]

(Specialist RTempCtrlSys (declare (superspecialist Process)  
(subspecialists RTempFBCtrl HeatExchanger SecRRecycleFlowCtrlSys))
[kgs (rttemp Table (match (DB-FETCH 'TT4 Value)  
(DB-FETCH 'ALT4 'activated)  
(DB-FETCH 'MTC8 Value)  
with  
(if (OR H HH L LL)  
? T ? then 3  
else (OR H HH L LL)  
?? ? then 2  
else (OR UP DOWN)  
?? then 2  
else (OR H HH L LL)  
then 2  
else (M STEADY ? N  
then -3  
else 0)  
(messages (Establish (SetConfidence self rttemp]

[SPECIALIST RTempFBCtrl (declare (superspecialist RTempCtrlSys)  
(subspecialists RTsensor'TT4 RTController'TC4 RRecycleTempCtrl))
[kgs (rttempfb Table (*: "recycle temp. TT8 should be low when reactor temp. is high")  
(match (DB-FETCH 'TT4 Value)  
(DB-FETCH 'TT8 Value)  
with  
(if (OR H HH)  
(OR N H HH)  
then 3  
else (OR L LL)  
(OR N L LL)  
then 3  
else N ?  
then -3  
else (OR H HH)  
(OR L LL)  
then -3  
else (OR L LL)  
(OR H HH)  
then -3  
else 0)))  
(messages (Establish (SetConfidence self rttempfb]

[rtempfb-trend Table (*: "recycle temp. TT8 should be low or decreasing when reactor temp. is increasing")  
(match (DB-FETCH 'TT4 'trend)  
(DB-FETCH 'TT8 'value)  
(DB-FETCH 'TT9 'trend)  
with  
(if UP (OR N H HH) ?
then 3  
elself UP ? UP  
then 3  
elself DOWN (OR N LL) ?  
then 3  
elself DOWN ? DOWN  
then 3  
elself STEADY ? ?  
then -3  
elself UP (OR L LL) ?  
then -3  
elself UP ? DOWN  
then -3  
elself DOWN (OR H HH) ?  
then -3  
elself DOWN ? UP  
then -3  
else 0

(messages [Establish [SetConfidence self (MAX rtempfb rtempfb-trend)]

(Specialist RTSensorTT4 (declare (superspecialist RTempFBCtrl)  
(subspecialists))
[kgs (sensorTT4 Rules  
(match (DATA-FETCH 'TT4-Ob 'validity)  
 with  
(if Valid  
then -3  
else Invalid  
then 3  
else Through  
then -2  
else Probable  
then -1  
else 0]

(messages [Establish [SetConfidence self sensorTT4]

(Specialist RTControllerTC4 (declare (superspecialist RTempFBCtrl)  
(subspecialists))
[kgs (rTC4 Table (* ; "Increase in MTC4 increases recycle temp. TT8, thus increases reactor temp. TT4")  
(match (DB-FETCH 'TT4 'value)  
(DB-FETCH 'TT4 'trend)  
(DB-FETCH 'MTC4 'value)  
with  
(if (OR H HH)  
? (OR N H HH)  
then 3  
elself (OR L LL)  
? (OR N L LL)  
then 3  
elself ? UP (OR N H HH)  
then 3  
elself ? DOWN (OR N L LL)  
then 3  
elself (OR H HH)  
? (OR L LL)  
then -3  
elself (OR L LL)  
? (OR H HH)  
then -3  
elself ? UP (OR L LL)  
then -3  
elself ? DOWN (OR H HH)  

)
(TC4set Table (match (DB-FETCH TT4 value)
    (DB-FETCH 'TT4 'trend)
    (DB-FETCH 'STC4 value)
    with
      (if (OR H HH)
        ?
        (OR H HH)
        then 3
        elseif (OR L LL)
        ?
        (OR L LL)
        then 3
        elself ? UP (OR H HH)
        then 3
        elseif ? DOWN (OR L LL)
        then 3
        elself ? ? N
        then -3
        elseif (OR H HH)
        ?
        (OR L LL)
        then -3
        elseif (OR L LL)
        ?
        (OR H HH)
        then -3
        elseif ? UP (OR L LL)
        then -3
        elseif ? DOWN (OR H HH)
        then -3
        else 0))
    (messages (Establish (SetConfidence self (MAX rTTC4 TC4set)))

[Specialist RRRecycleTempCtrl (declare (superspecialist RTempFBCtrl))
    (subspecialists CoolantSupply CoolantTemp RRTempFBCtrl))
[kgs (rrtemp Table (match (DB-FETCH MTC4 value)
    (DB-FETCH 'TT8 value)
    with
      (if (OR H HH)
        (OR N L LL)
        then 3
        else 0)))
    (rrtemp-aux Table (match (BCSRL+? RTControllerTC4)
    (BCSRL+? RTSensorTT4)
    with
      (if NIL NIL
        then 2
        else (NOT NIL)
        ?
        then -2
        else ? (NOT NIL)
        then -2
        else 0))
    (messages (Establish (if (NOT (EQ rrtemp 0)))
    then
      (SetConfidence self rrtemp))
else

(SetConfidence self rtemp-aux)

(Specialist CoolantSupply (declare (superspecialist RRecycleTempCtrl)
  (subspecialists)))
[kgs (cooltantsup Table
  (match (DB-FETCH 'FT3 Value)
    (AskYNU? "Does utilities department confirm loss of coolant supply")
      with
      (if (OR H HH N)
        ?
        then -3
        elseif ? F
        then -3
        elseif ? T
        then 3
        else 0)
    (messages (Establish (SetConfidence self coolantsup)

(Specialist CoolantTemp (declare (superspecialist RRecycleTempCtrl)
  (subspecialists))
[kgs (cooltemp Table (match (DB-FETCH 'TT8 Value)
    (DB-FETCH 'TT6 Value)
      with
      (if (OR H HH)
        (OR H HH)
        then 3
        elseif (OR L LL)
        (OR L LL)
        then 3
        elseif ? N
        then -3
        else 0)
    (messages (Establish (SetConfidence self cooltemp)

(Specialist RRTempFBCtrl (declare (superspecialist RRecycleTempCtrl)
  (subspecialists RRTSensorTTB RRTControllerTC8 RRTValve))
[kgs (rtempfb Table (* ; "coolant flow valve should be wide open when recycle temp. setpoint MTC4 is low")
  (match (DB-FETCH 'MTC4 Value)
    (DB-FETCH 'VP3 Value)
      with
      (if (OR H HH)
        (OR N H HH)
        then 3
        elseif (OR L LL)
        (OR N L LL)
        then 3
        elseif N N
        then -3
        elseif (OR H HH)
        (OR L LL)
        then -3
        elseif (OR L LL)
        (OR H HH)
        then -3
        else 0)
    (messages (Establish (SetConfidence self rtempfb)

(Specialist RRTSensorTTB (declare (superspecialist RRTempFBCtrl)
  (subspecialists))
[kgs (sensorTTB Rules
  (match (DATA-FETCH 'TT8-Ob validity)
with
(if Valid
then -3
elseif Invalid
then 3
elseif Through
then -2
elseif Probable
then -1
else 0)
(messages (Establish (SetConfidence self sensorTTB)

(Specialist RRTControllerTCB (declare (superspecialist RRTempFBCtrl)
(subspecialists))
(kgs (mTC8 Table
("; "when TT8 setpoint MTC4 is high, MTC8 should be high such that coolant flowrate
is low")
(match (DB-FETCH 'MTC4 Value)
(DB-FETCH 'MTC8 Value)
(BCSRL+? 'RRTSensorTTB)
with
(if (OR H HH)
(OR N L LL)
NIL
then 3
elseif (OR L LL)
(OR N H HH)
NIL
then 3
elseif N N
then -3
elseif (OR H HH)
(OR H HH)
NIL
then -3
elseif (OR L LL)
(OR L LL)
NIL then -3
elseif ? ?? (NOT NIL)
then -2
else 0)
(messages (Establish (SetConfidence self rrtTCB)

(Specialist RRTValve (declare (superspecialist RRTempFBCtrl)
(subspecialists))
(kgs (rrtval-test Rules (match (HYPOTHESIS-TEST 'RRTValve-Ob)
with
(if Established
then 3
elseif Rejected
then -3
else 0))
(rrtval Table
(match (AskYNU? "Does coolant flow control valve pass physical check")
(BCSRL+? 'RRTSensorTTB)
(BCSRL+? 'RRTControllerTCB)
with
(if T ??
then -3
elseif F ??
then 3
elseif ? NIL NIL
then 2
elseif (NOT NIL) ?
then -2
elseif ? ?? (NOT NIL)
then -2
else 1

(messages (Establish (if (NOT (EQ rrtval-test 0))
  then
  (SetConfldence self rrtval-test)
  else
  (SetConfldence self rrtval)

(Specialist HeatExchanger (declare (superspecialist RTempCtrlSys)
  (subspecialists))
  (kgs (heat-ex-test Rules (match (HYPOTHESIS-TEST 'HeatExchanger-Ob)
    with
    (if Established
      then 3
      else Rejected
      then -3
      else 0)))
  (heatex Table (* ; "if the heat exchanger is fouled, TT7 should decrease while VP3 and FT3 should increase (MTC8 decreases) when other conditions are constant")
    (match (DB-FETCH 'FT2 Value)
      (DB-FETCH 'FT3 Value)
      (DB-FETCH 'MT6 Value)
      (DB-FETCH 'TT4 Value)
      (DB-FETCH 'TT7 Value)
      (DB-FETCH 'TT8 Value)
      with
      (if N (OR H HH)
        (OR L LL)
        N (OR L LL)
        N
        then 3
        else Rejected
        then -3
        else 0))
    (messages (Establish (if (NOT (EQ heatex-test 0))
      then
      (SetConfldence self heatex-test)
      else
      (SetConfldence self heatex)

(Specialist SecRRecycleFlowCtrlSys (declare (superspecialist RTempCtrlSys)
  (subspecialists))
  (kgs (secRRF Table (match (DB-FETCH 'TT4 Value)
    (DB-FETCH 'FT2 Value)
    with
    (if N
      then -3
      else (OR L LL)
      (OR L LL)
      then -3
      else (OR H HH)
      (OR H HH)
      then -3
      else (OR L LL)
      (OR H HH)
      then 3
      else (OR H HH)
      (OR L LL)
      then 3
    else 0))
    (messages (Establish (if (NOT (EQ heatex-test 0))
      then
      (SetConfldence self heatex-test)
      else
      (SetConfldence self heatex)
else 0]
(messages (Establish (SetConfidence self secRRF)

(Specialist RRRecycleFlowCtrlSys (declare (superspecialist Process)
(subspecialists RRFlowFBCtrl RRFPump))
[kgs (rrflow Table (match (DB-FETCH 'FT2 value)
(DB-FETCH 'ALF2 activated)
with
(if (OR H HH L LL)
 then 3
 else (OR H HH L LL)
 ?
 then 2
 else N ?
 then -3
 else 0)
(messages (Establish (SetConfidence self rrflow)

(Specialist RRFlowFBCtrl (declare (superspecialist RRRecycleFlowCtrlSys)
(subspecialists RRFSensorFT2 RRControllerFC2 RRValve))
[kgs (rrflowfb Table (match (DB-FETCH 'FT2 value)
(DB-FETCH 'VP2 value)
with
(if (OR L LL)
 (OR L LL)
 then 3
 else (OR H HH)
 (OR H HH)
 then 3
 else (OR H HH)
 (OR L LL)
 then -3
 else (OR L LL)
 (OR H HH)
 then -3
 else 0)
(messages (Establish (SetConfidence self rrflowfb)

(Specialist RRFSensorFT2 (declare (superspecialist RRFlowFBCtrl)
(subspecialists))
[kgs (sensorFT2 Rules
(match (DATA-FETCH 'FT2-Ob validity)
with
(if Valid
 then -3
 else Invalid
 then 3
 else Through
 then -2
 else Probable
 then -1
 else 0)
(messages (Establish (SetConfidence self sensorFT2)

(Specialist RRControllerFC2 (declare (superspecialist RRFlowFBCtrl)
(subspecialists))
[kgs (rrFC2 Table "increase in FC2 controller signal MFC2 decreases valve position VP2, thus decreases recycle flowrate")
(match (DB-FETCH 'FT2 value)
(DB-FETCH 'MFC2 value)
with

(if (OR L LL)
   (OR N H HH)
   then 3
elseif (OR H HH)
   (OR L LL)
   then 3
elseif (OR H HH)
   (OR N HH)
   then -3
elseif (OR L LL)
   (OR L LL)
   then -3
else 0))

(FC2set Table (match (DB-FETCH 'FT2 value)
   (DB-FETCH 'SFC2 value)
   with
   (if (OR L LL)
      (OR L LL)
      then 3
   elseif (OR H HH)
      (OR H HH)
      then 3
   elseif ? N
      then -3
   elseif (OR H HH)
      (OR L LL)
      then -3
   elseif (OR L LL)
      (OR H HH)
      then -3
else 0)
  (messages (Establish (SetConfldence self (MAX rrfFC2 FC2set)

[Specialist RRFValve (declare (superspecialist RRFFlowFBCtrl)
   (subspecialists))]

[kgs (rrfval-test Rules (match (HYPOTHESIS-TEST 'RRFValve-Ob)
   with
   (if Established
      then 3
   elseif Rejected
      then -3
   else 0))]

(rrfval Table
   (match (AskYNU? "does recycle flow control valve pass physical check")
   (BCSRL+/? 'RRFSensorFT2)
   (BCSRL+/? 'RRFControllerFC2)
   with
   (if T ? ?
      then -3
   elseif F ? ?
      then 3
   elseif ? NIL NIL
      then 2
   elseif ? (NOT NIL) ?
      then -2
   elseif ? ? (NOT NIL)
      then -2
else 1)
  (messages (Establish (if (NOT (EQ rrfval-test 0))
   then
   (SetConfldence self rrfval-test)
   else
   (SetConfldence self rrfval)

[Specialist RRPump (declare (superspecialist RRRecycleFlowCtrlSys)
   (subspecialists))]
[kgs (rrfpump-test Rules (match (HYPOTHESIS-TEST 'Pump-Ob) with (if Established then 3 else Rejected then -3 else 0)) (rrfpump Table (match (DB-FETCH 'FT2 value) (DB-FETCH 'FT4 value) with (if (OR L LL) (OR L LL) then 3 else (OR N H HH) ? then -3 else if (OR N H HH) then -3 else 0) (messages (Establish (if (NOT (EQ rrfpump-test 0)) then (SetConfidence self rrfpump-test) else (SetConfidence self rrfpump) (Specialist RLevelCtriSys (declare (superspecialist Process) (subspecialists RLevelFBCtrl RLPump)) (kgs (rlevel Table (match (DB-FETCH 'LT1 value) (DB-FETCH 'LT1 trend) (DB-FETCH 'ALL1 'activated) with (if (OR H HH L LL) ? T then 3 else (OR H HH L LL) ? then 2 else if (OR UP DOWN) ? then 2 else if (OR H HH) ? then 3 else if (OR N H HH) ? then 3 else if (OR N L LL) ? then 3 else if (OR H HH) then -3 else if (OR L LL) then -3 else if (OR H HH) (OR H HH) (OR H HH) (OR L LL) (OR L LL)) (messages (Establish (SetConfidence self rlevel) (Specialist RLevelFBCtrl (declare (superspecialist RLevelCtriSys) (subspecialists RLSensorLT1 RLControllerLC1 RLValve)) (kgs (rlevelf Table (match (DATA-FETCH 'LT1-Ob 'validity) (DB-FETCH 'LT1 value) (DB-FETCH 'VP4 value) with (if Invalid ? ? then 3 else if (OR L LL) (OR N H HH) then 3 else if (OR H HH) (OR N L LL) then 3 else if (OR H HH) then -3 else if (OR L LL) then -3 else if (OR H HH) (OR H HH) (OR H HH))
(if DOWN (OR N H HH)
  then 3
  elseif UP (OR N L LL)
  then 3
  elseif STEADY ?
  then -3
  elseif DOWN (OR L LL)
  then -3
  elseif UP (OR H HH)
  then -3
  else 0)
}

(messages [Establish (SetConfidence self (MAX rievelfb rievelfb-trend)]

[Specialist RLSensorLT1 (declare (superspecialist RLLevelFBCtrl)
  (subspecialists))]
[kgs (sensorLT1 Rules
  (match (DATA-FETCH 'LT1-Ob validity)
  with
  (if Valid
   then -3
   elseif Invalid
   then 3
   elseif Through
   then -2
   elseif Probable
   then -1
   else 0]
  (messages [Establish (SetConfidence self sensorLT1)]

[Specialist RControllerLC1 (declare (superspecialist RLLevelFBCtrl)
  (subspecialists))]
[kgs (rILC1 Table (*; "decrease in MLC1 increases product flowrate, thus decreases LT1")
  (match (DB-FETCH 'LT1 'trend)
  (DB-FETCH 'LT1 'value)
  (DB-FETCH 'MLC1 'value)
  with
  (if (OR H HH)
   ?
   (OR N H HH)
   then 3
   elseif (OR L LL)
   ?
   (OR N L LL)
   then 3
   elseif (OR N H HH)
   then 3
   elseif (OR H LL)
   ?
   (OR L LL)
   then -3
   elseif (OR L LL)
   ?
   (OR H HH)
   then -3
   elseif (OR L LL)
   then -3
   elseif (OR H HH)
   then -3
   else 0)])}
(LC1set Table (match (DB-FETCH 'LT1 Value)
   (DB-FETCH 'LT1 trend)
   (DB-FETCH 'SLC1 Value)
   with
   (if (OR H HH)
      ?
      (OR H HH)
      then 3
      elseif (OR L LL)
      ?
      (OR L LL)
      then 3
      elseif ? UP (OR H HH)
      then 3
      elseif ? N
      then -3
      elseif (OR H HH)
      ?
      (OR L LL)
      then -3
      elseif (OR L LL)
      ?
      (OR H HH)
      ?
      (OR L LL)
      then -3
      elseif ? UP (OR L LL)
      then -3
      elseif ? DOWN (OR H HH)
      then -3
      else 0))

(messages (Establish (SetConfidence self (MAX rILC1 LC1set))

(Specialist RLValve (declare (superspecialist RLevelFBCtrl)
   (subspecialists))
   (kgs (rival-test Rules (match (HYPOTHESIS-TEST 'RLValve-Ob)
      with
      (if Established
      then 3
      elseif Rejected
      then -3
      else 0)))

(rival Table
   (match (AskYNUP "Does product flow control valve pass physical check")
   (BCSRL+? 'RLSensorLT1)
   (BCSRL+? 'RLControllerLC1)
   with
   (if T ??
      then -3
   elseif F ??
      then 3
   elseif NIL NIL
      then 2
   elseif (NOT NIL) ?
      then -2
   elseif ? (NOT NIL)
      then -2
   else 1)

(messages (Establish (if (NOT (EQ rival-test 0))
   then
   (SetConfidence self rival-test)
   else
   (SetConfidence self rival)

(Specialist RLPump (declare (superspecialist RLevelCtrlSys)
   (subspecialists))


(kgs (ripump-test Rules (match (HYPOTHESIS-TEST 'Pump-Ob) with
      (if Established
       then 3
       elseif Rejected
       then -3
       else 0))
      (ripump Table (match (DB-FETCH 'FT2 ' value)
               (DB-FETCH 'FT4 ' value) with
               (if (OR L LL)
                (OR L LL)
                then 3
               elseif (OR N H HH)
               ?
               then -3
               elseif (OR N H HH)
               ?
               then -3
               elseif 0)
      (messages (Establish fif (NOT (EQ ripump-test 0))
                      then
                      (SetConfidence self ripump-test)
                      else
                      (SetConfidence self ripump)
APPENDIX C

CODE FOR SYMPTOMS DATABASE
(DEFINE-FILE-INFO READTABLE "INTERLISP" PACKAGE "INTERLISP")
(FILECREATED "4-Sep-89 14:01:06" [{CASPION:LAIR:OHIO-STATE}<SHUM>CSTR>CSTRDATABASE.MPLOOPS;4]
34395
changes to:%: (CLASSES CSTRDataBase)
previous date%: "3-Aug-89 16:53:47"
{CASPION:LAIR:OHIO-STATE}<SHUM>CSTR>CSTRDATABASE.MPLOOPS;3})

(PRETTYCOMPRINT CSTRDATABASECOMS)

(RPAQQ CSTRDATABASECOMS

 (CLASSES ALF1 ALF2 AL1 ALT4 AT1 AT2 Alarms CSTRDataBase ConcSensors Controllers FT1 FT2 FT3 FT4 FlowAlarms FlowSensors LT1 LevelAlarms LevelSensors MFC1 MFC2 MLC1 MTC1 MTC4 MTC8 SFC1 SFC2 SLC1 STC4 SensorData SetPoints Signals TT1 TT2 TT3 TT4 TT5 TT6 TT7 TT8 TempAlarms TempSensors VP1 VP2 VP3 VP4 ValvePositions)

(METHODS)
(FNS)
(VARS)
(INSTANCES)
(PROP MAKEFILE-ENVIRONMENT CSTRDATABASE.MPLOOPS))

("* ; * File created by SHUM-S")

(DEFCCLASSES ALF1 ALF2 AL1 ALT4 AT1 AT2 Alarms CSTRDataBase ConcSensors Controllers FT1 FT2 FT3 FT4 FlowAlarms FlowSensors LT1 LevelAlarms LevelSensors MFC1 MFC2 MLC1 MTC1 MTC4 MTC8 SFC1 SFC2 SLC1 STC4 SensorData SetPoints Signals TT1 TT2 TT3 TT4 TT5 TT6 TT7 TT8 TempAlarms TempSensors VP1 VP2 VP3 VP4 ValvePositions)

(DEFCCLASS ALF1
(MetaClass Eclss Edited%: (* ; "Edited 7-Jun-89 16:20 by ")
(Supers RowAlarms)
(InstanceVariables (ivs (activated))
(activated NIL question "Is the feed flow alarm (ALF1) activated?" doc
(* takes T,F or U values)))

(DEFCCLASS ALF2
(MetaClass Eclss Edited%: (* ; "Edited 7-Jun-89 16:20 by ")
(Supers FlowAlarms)
(InstanceVariables (ivs (activated))
(activated NIL question "Is the recycle flow alarm (ALF2) activated?" doc
(* takes T,F or U values)))

(DEFCCLASS AL1
(MetaClass Eclss Edited%: (* ; "Edited 7-Jun-89 16:22 by ")
(Supers LevelAlarms)
(InstanceVariables (ivs (activated))
(activated NIL question "Is the reactor level alarm (AL1) activated?" doc
(* takes T,F or U values)))

(DEFCCLASS ALT4
(MetaClass Eclss Edited%: (* ; "Edited 7-Jun-89 16:15 by ")
(Supers TempAlarms)
(InstanceVariables (ivs (activated))
(activated NIL question "Is the reactor temperature alarm (ALT4) activated?" doc
(* takes T,F or U values)))

(DEFCCLASS AT1
(MetaClass Eclss Edited%: (* ; "Edited 1-Aug-89 13:24 by ")
(Supers ConcSensors)
(InstanceVariables (ivs (numvalue pastdata))
(normal 2.88)
(delhil 0.08)
(delhil 0.16)
(deltrend 0.01)
(hlimit 4.5)
(defclass AT2
  (meta-class class edited%; (* ; "edited 1-Aug-89 13:25 by ")
  (supers conc-sensors)
  (instance-variables (ivs (num-value past-data))
    (normal 1.15)
    (delh 0.06)
    (delhhi 0.16)
    (deltrh 0.01)
    (hilimit 2.5)
    (illimit 0.2)
    (ratelimit 1.0)
    (devlimit 0.001)
    (num-value nil question "what is the reading of product conc. sensor AT2 ?" doc
     (* measurement from sensor))
    (past-data nil question "what are the past ten readings of product conc. sensor AT2 ?" doc
     (* list of measurements from sensor))
    (trend nil)
    (value nil)))

(defclass alarms
  (meta-class class edited%: (* ; "edited 7-Jun-89 16:24 by ")
  (supers cstr-data-base))

(defclass cstr-data-base
  (meta-class class edited%: (* ; "edited 4-Sep-89 12:01 by ")
  (supers object)
  (instance-variables (ivs (with-data-cases cstr-data-base)
    doc (* these are the instances of subclasses with data)
    (with-data-instances (icstr-data-base)
      doc (* classes not listed under alarm or sensor classes))
    (alarm-instances (alf1 alf2 all1 allt4))
    (sensor-instances (at1 at2 ft1 ft2 ft3 ft4 lt1 tt1 tt2 tt3 tt4 tt5 tt6 tt7 tt8))
    (value-instances (mfc1 mfc2 mlc1 mtc4 mtc8 sfc1 sfc2 slc1 stc4 vp1 vp2 vp3 vp4))
    (controlsensors (ft1 ft2 lt1 tt4 tt8)))

(defclass conc-sensors
  (meta-class class edited%; (* ; "edited 7-Jun-89 16:25 by ")
  (supers sensor-data))

(defclass controllers
  (meta-class class edited%: (* ; "edited 27-Jun-89 13:47 by ")
  (supers cstr-data-base))

(defclass ft1
  (meta-class class edited%: (* ; "edited 3-Aug-89 16:44 by ")
  (supers flow-sensors)
  (instance-variables (ivs (num-value past-data))
    (normal 0.45)
    (delh 0.05)
    (delhhi 0.1)
    (deltrh 0.01)
    (hilimit 1.2)
    (illimit 0.0)
    (ratelimit 0.5)
    (devlimit 0.001)
    (num-value nil question "what is the reading of feed flow sensor FT1 ?" doc
     (* measurement from sensor))
(DEFCLASS FT2
(MetaClass Class Edited%: (* ; "Edited 3-Aug-89 16:44 by "))
(Supers FlowSensors)
(InstanceVariables (ivs (numvalue pastdata))
 (normal 0.9)
 (delhi 0.05)
 (delhhi 0.1)
 (deltrend 0.01)
 (hlimit 2.4)
 (llimit 0.0)
 (ratelimit 0.5)
 (devlimit 0.001)
 (numvalue NIL question "What is the reading of recycle flow sensor FT2 ?" doc
 (* measurement from sensor))
 (pastdata NIL question "What are the past ten readings of recycle flow sensor FT2 ?" doc
 (* list of measurements from sensor))
 (trend NIL)
 (value NIL)))

(DEFCLASS FT3
(MetaClass Class Edited%: (* ; "Edited 3-Aug-89 16:45 by "))
(Supers FlowSensors)
(InstanceVariables (ivs (numvalue pastdata))
 (normal 0.9)
 (delhi 0.08)
 (delhhi 0.16)
 (deltrend 0.01)
 (hlimit 2.4)
 (llimit 0.0)
 (ratelimit 0.5)
 (devlimit 0.001)
 (numvalue NIL question "What is the reading of coolant flow sensor FT3 ?" doc
 (* measurement from sensor))
 (pastdata NIL question "What are the past ten readings of coolant flow sensor FT3 ?" doc
 (* list of measurements from sensor))
 (trend NIL)
 (value NIL)))

(DEFCLASS FT4
(MetaClass Class Edited%: (* ; "Edited 3-Aug-89 16:45 by "))
(Supers FlowSensors)
(InstanceVariables (ivs (numvalue pastdata))
 (normal 0.45)
 (delhi 0.05)
 (delhhi 0.1)
 (deltrend 0.01)
 (hlimit 1.2)
 (llimit 0.0)
 (ratelimit 0.5)
 (devlimit 0.001)
 (numvalue NIL question "What is the reading of product flow sensor FT4 ?" doc
 (* measurement from sensor))
 (pastdata NIL question "What are the past ten readings of product flow sensor FT4 ?" doc
 (* list of measurements from sensor))
 (trend NIL)
 (value NIL)))

(DEFCLASS RowAlarms
(MetaClass Class Edited%: (* ; "Edited 6-Jun-89 17:56 by "))
(Supers Alarms))

(DEFCLASS RowSensors
(MetaClass Class Edited%: (* ; "Edited 6-Jun-89 13:46 by "))
(Supers SensorData))

(DEFCLASS LT1
(MetaClass Class Edited%: (* ; "Edited 3-Aug-89 16:45 by ")
(Supers LevelSensors)
(InstanceVariables (Ivs (numvalue pastdata))
(normal 1.78)
(delh 0.06)
(delhl 0.16)
(delhlh 0.01)
(limit 2.5)
(limitl 0.0)
(ratelimit 0.5)
(devlimit 0.001)
(numvalue NIL question "What is the reading of reactor level sensor LT1 ?" doc
(* measurement from sensor))
(pastdata NIL question "What are the past ten readings of reactor level sensor LT1 ?" doc
(* list of measurements from sensor))
(trend NIL)
(value NIL))

(DEFCLASS LevelAlarms
(MetaClass Class Edited%: (* ; "Edited 3-Aug-89 16:45 by ")
(Supers Alarms))

(DEFCLASS LevelSensors
(MetaClass Class Edited%: (* ; "Edited 6-Jun-89 14:34 by ")
(Supers SensorData))

(DEFCLASS MFC1
(MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:41 by shum-s")
(Supers Signals)
(InstanceVariables (Ivs (numvalue))
(normal 0.25)
(delh 0.05)
(delhl 0.1)
(numvalue NIL question "What is the reading of feed flow controller signal (FC1) ?" doc
(* measurement))
(value NIL))

(DEFCLASS MFC2
(MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:38 by shum-s")
(Supers Signals)
(InstanceVariables (Ivs (numvalue))
(normal 0.25)
(delh 0.05)
(delhl 0.1)
(numvalue NIL question "What is the reading of recycle flow controller signal (FC2) ?" doc
(* measurement))
(value NIL))

(DEFCLASS MLC1
(MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:38 by shum-s")
(Supers Signals)
(InstanceVariables (Ivs (numvalue))
(normal 0.25)
(delh 0.05)
(delhl 0.1)
(numvalue NIL question "What is the reading of reactor level controller signal (LC1) ?" doc
(* measurement))
(value NIL))

(DEFCLASS MTC4
(MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:41 by shum-s")
(Supers Signals)
(InstanceVariables (Ivs (numvalue))
(normal 0.45)
(delh 0.03)
(delhl 0.05)
"What is the reading of reactor temp. primary controller signal (TC4)?" doc (* measurement)
(value NIL))

(DECLASS MTc8
(MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:41 by shum-s")
(Supers Signals)
(InstanceVariables (ivs (numvalue)))
(normal 0.25)
(delhl 0.03)
(delhhll 0.06)
(numvalue NIL question "What is the reading of reactor temp. secondary controller signal (recycle temp. controller TC8)?" doc (* measurement)
(value NIL))

(DECLASS SFC1
(MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:46 by shum-s")
(Supers SetPoints)
(InstanceVariables (ivs (numvalue)))
(normal 0.45)
(delhl 0.05)
(delhhll 0.1)
(numvalue NIL question "What is the setpoint reading of feed flow controller FC1?" doc (* numerical reading)
(value NIL))

(DECLASS SFC2
(MetaClass Class Edited%: (* ; "Edited 12-Jul-89 16:14 by shum-s")
(Supers SetPoints)
(InstanceVariables (ivs (numvalue)))
(normal 0.9)
(delhl 0.05)
(delhhll 0.1)
(numvalue NIL question "What is the setpoint reading of recycle flow controller FC2?" doc (* numerical reading)
(value NIL))

(DECLASS SLC1
(MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:45 by shum-s")
(Supers SetPoints)
(InstanceVariables (ivs (numvalue)))
(normal 1.75)
(delhl 0.06)
(delhhll 0.16)
(numvalue NIL question "What is the setpoint reading of reactor level controller LC1?" doc (* numerical reading)
(value NIL))

(DECLASS STC4
(MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:44 by shum-s")
(Supers SetPoints)
(InstanceVariables (ivs (numvalue)))
(normal 83.7)
(delhl 3)
(delhhll 6)
(numvalue NIL question "What is the setpoint reading of reactor temp. controller TC4?" doc (* numerical reading)
(value NIL))

(DECLASS SensorData
(MetaClass Class Edited%: (* ; "Edited 6-Jun-89 13:38 by ")
(Supers CSTRDataBase))

(DECLASS SetPoints
(MetaClass Class Edited%: (* ; "Edited 27-Jun-89 13:54 by ")
(Supers Controllers))
(DEFCLASS Signals
  (MetaClass Class Edited%: (* ; "Edited 27-Jun-89 13:49 by "))
  (Supers Controllers))

(DEFCLASS TT1
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:41 by "))
  (Supers TempSensors)
  (InstanceVariables (ivs (numvalue pastdata))
    (normal 66.0)
    (delhi 3)
    (delhll 6)
    (deltrend 0.08)
    (lllimit 85)
    (lllimit 45)
    (devllimit 0.001)
    (numvalue NIL question * What is the reading of feed temp. sensor TT1 ? * doc
     (* measurement from sensor))
    (pastdata NIL question *What are the past ten readings of feed temp. sensor TT1 ?* doc
     (* list of measurements from sensor))
    (trend NIL)
    (value NIL)))

(DEFCLASS TT2
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:40 by "))
  (Supers TempSensors)
  (InstanceVariables (ivs (numvalue pastdata))
    (normal 79.5)
    (delhi 3)
    (delhll 6)
    (deltrend 0.08)
    (lllimit 105)
    (lllimit 65)
    (ratelimit 10)
    (devllimit 0.001)
    (numvalue NIL question * What is the reading of reactor temp. sensor TT2 ? * doc
     (* measurement from sensor))
    (pastdata NIL question *What are the past ten readings of reactor temp. sensor TT2 ?* doc
     (* list of measurements from sensor))
    (trend NIL)
    (value NIL)))

(DEFCLASS TT3
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:40 by "))
  (Supers TempSensors)
  (InstanceVariables (ivs (numvalue pastdata))
    (normal 83.7)
    (delhi 3)
    (delhll 6)
    (deltrend 0.08)
    (lllimit 105)
    (lllimit 65)
    (ratelimit 10)
    (devllimit 0.001)
    (numvalue NIL question * What is the reading of reactor temp. sensor TT3 ? * doc
     (* measurement from sensor))
    (pastdata NIL question *What are the past ten readings of reactor temp. sensor TT3 ?* doc
     (* list of measurements from sensor))
    (trend NIL)
    (value NIL)))

(DEFCLASS TT4
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:37 by "))
  (Supers TempSensors)
  (InstanceVariables (ivs (numvalue pastdata))
    (normal 83.7)
    (delhi 3)
    (delhll 6)
    (deltrend 0.08)
    (lllimit 105)
(InstanceVariables (ivs (numvalue pastdata))
  (normal 68.1)
  (delhl 1)
  (delhhll 2)
  (dehrend 0.08)
  (hllmh 90)
  (fitelim 10)
  (devlimh 0.001)
  (numvalue NIL question "What is the reading of recycle temp. sensor TT8 ?" doc (* measurement from sensor))
  (pastdata NIL question "What are the past ten readings of recycle temp. sensor TT8 ?" doc (* list of measurements from sensor))
  (trend NIL)
  (value NIL))

(DEFCCLASS TempAlarms
 (MetaClass Class Edited%: (* ; "Edited 1-Jan-68 09:33"))
 (Supers Alarms))

(DEFCCLASS TempSensors
 (MetaClass Class Edited%: (* ; "Edited 6-Jun-89 11:45 by ")
 (Supers SensorData))

(DEFCCLASS VP1
 (MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:49 by shum-s")
 (Super ValvePositions)
 (InstanceVariables (ivs (numvalue))
  (normal 0.38)
  (delhl 0.07)
  (delhhll 0.14)
  (numvalue NIL question "What is the reading of feed flow valve positioner (VP1) ?" doc (* measurement))
  (value NIL))

(DEFCCLASS VP2
 (MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:49 by shum-s")
 (Super ValvePositions)
 (InstanceVariables (ivs (numvalue))
  (normal 0.38)
  (delhl 0.1)
  (delhhll 0.2)
  (numvalue NIL question "What is the reading of recycle flow valve positioner (VP2) ?" doc (* measurement))
  (value NIL))

(DEFCCLASS VP3
 (MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:50 by shum-s")
 (Super ValvePositions)
 (InstanceVariables (ivs (numvalue))
  (normal 0.38)
  (delhl 0.05)
  (delhhll 0.1)
  (numvalue NIL question "What is the reading of coolant flow valve positioner (VP3) ?" doc (* measurement))
  (value NIL))

(DEFCCLASS VP4
 (MetaClass Class Edited%: (* ; "Edited 21-Jul-89 11:50 by shum-s")
 (Super ValvePositions)
 (InstanceVariables (ivs (numvalue))
  (normal 0.38)
  (delhl 0.1)
  (delhhll 0.2)
  (numvalue NIL question "What is the reading of product flow valve positioner (VP4) ?" doc (* measurement))
  (value NIL))
(DEFCLASS ValvePositions
  (MetaClass Class Edited%;
  (Supers Controllers))

(*) ; "Edited 27-Jun-89 13:53 by ")

(BatchMethodDefs)

(UnbatchMethodDefs)

(PUTPROPS:DATABASE:MPLOOPS:MAKEFILE-ENVIRONMENT(:READTABLE:"INTERLISP":PACKAGE:"INTERLISP"))

(DECLARE%: DONTCOPY
  (FILEMAP (NIL)))

STOP
APPENDIX D

CODE FOR VALIDATION HIERARCHY AND GENERAL FUNCTIONS
(DEFINE-FILE-INFO READTABLE "INTERUSP" PACKAGE "INTERUSP"
(FILECREATED * 9-Sep-89 14:31:37* (DSK) <LSFFILES> <SHUM> CSTROBJECT.MPLoops;26 120424

changes to:% (FNS VALIDATION VALUE-ASKUSER CSRLDB INIABST INIDATA INIHYPOTHESIS INVALIDITY 
TRADE-VALIDATION TRAD-VALIDATION-STOP VALUE-COMPARE-LIST VALUE-COMPARE-ONE 
VALUE-REPLACE VALIDATE VALUE-COMPARE-NEW UPDATE-VALIDITY TT2-COM)
(VARS CSTROBJECTCOMS)(CLASSES TT2-Ob)

previous date:% "1-Sep-89:15:16:15" {CASPIAN-LAIR:OHIO-STATE} <SHUM> CSTR> CSTROBJECT.MPLoops:10})

(PRETTYCOMPRINT CSTROBJECTCOMS)

(RPAQO CSTROBJECTCOMS
 (* ; "File created by SHUM-S")
 (CLASSES Alarm-Ob ALF1-Ob ALF2-Ob ALL1-Ob ALT4-Ob AT1-Ob AT2-Ob ControlValve-Ob Equipment-Ob 
FFValve-Ob FT1-Ob FT2-Ob FT3-Ob FT4-Ob HeatExchanger-Ob LT1-Ob Process-Ob Pump-Ob 
RLValve-Ob RRFValve-Ob RRTValve-Ob Sensor-Ob TT1-Ob TT2-Ob TT3-Ob TT4-Ob TT5-Ob TT6-Ob 
TT7-Ob TT8-Ob)

(METHODS)
(FNS ASKDATA ASKDATALOOPS AT1-VALUE AT2-VALUE BOLD CLEAR-VALIDATION-WINDOWS CSRLDB 
DATA-ETCH DATA-INFERENTIAL DB-FETCH DESTROYINSTANCES FT1-VALUE FT2-VALUE FT3-VALUE FT4-VALUE 
FT4-VALUE FT4-VALUE HTET-TEST HYPOTHESIS-TEST INFERNORMALITY INFERTREND INIABST INIDATA 
INIHYPOTHESIS INVALIDITY INPUTDATAMISSCHECK LISTSQLT1-VALUE MADEDATAMFILE MAKEINSTANCES 
MEAN NOISECHECK NORMAL NORMALITYCHECK PRINTCSRLSYSTEM PUMP-TEST RATECHECK 
READDATA RELIABILITYCHECK SIGMA STDDEV SUM TRACE-VALIDATION TRADEC-VALIDATION-STOP 
TRENDCHECK TT1-VALUE TT2-COM TT2-VALUE TT3-COM TT3-VALUE TT4-VALUE TT5-VALUE 
TT5-VALUE TT6-VALUE TT6-VALUE TT7-VALUE TT8-VALUE UPDATE-VALIDITY VALIDATE VALIDATE-ALARM VALIDATE-REST 
VALIDATE-SENSORLIST VALIDATE-SIMPLE VALIDATION VALUE-ASKUSER VALUE-COMPARE 
VALUE-COMPARE-LIST VALUE-COMPARE-ONE VALUE-REPLACE VALVE-TEST)

(VARS)
(INSTANCES)
(FAIL MAKEFILE-ENVIRONMENT CSTROBJECT.MPLoops))

(* ; "File created by SHUM-S")

(DEFCLASSES Alarm-Ob ALF1-Ob ALF2-Ob ALL1-Ob ALT4-Ob AT1-Ob AT2-Ob ControlValve-Ob Equipment-Ob 
FFValve-Ob FT1-Ob FT2-Ob FT3-Ob FT4-Ob HeatExchanger-Ob LT1-Ob Process-Ob Pump-Ob RLValve-Ob 
RRFValve-Ob RRTValve-Ob Sensor-Ob TT1-Ob TT2-Ob TT3-Ob TT4-Ob TT5-Ob TT6-Ob TT7-Ob TT8-Ob)

(DEFCLASS Alarm-Ob
 (MetaClass Class Edited%: (* ; "Edited 14-Aug-89 17:02 by ")
 (Supers Process-Ob))

(DEFCLASS ALF1-Ob
 (MetaClass Class Edited%: (* ; "Edited 14-Aug-89 17:40 by ")
 (Supers Alarm-Ob)
 (InstanceVariables (sensor FT1)
 (validity NIL)))

(DEFCLASS ALF2-Ob
 (MetaClass Class Edited%: (* ; "Edited 14-Aug-89 17:41 by ")
 (Supers Alarm-Ob)
 (InstanceVariables (sensor FT2)
 (validity NIL)))

(DEFCLASS ALL1-Ob
 (MetaClass Class Edited%: (* ; "Edited 14-Aug-89 17:41 by ")
 (Supers Alarm-Ob)
 (InstanceVariables (sensor LT1)
 (validity NIL)))

(DEFCLASS ALT4-Ob
 (MetaClass Class Edited%: (* ; "Edited 14-Aug-89 17:41 by ")
 (Supers Alarm-Ob)
 (InstanceVariables (sensor TT4)
 (validity NIL)))
(DEFCLASS ATI-Ob
  (MetaClass Class Edited%: (* ; 'Edited 1-Aug-89 13:24 by '))
  (Supers Sensor-Ob)
  (InstanceVariables (reliability 6)
   (reliability-test NIL)
   (ratetest NIL)
   (devtest NIL)
   (validity NIL)
   (status NIL)
   (function AT1-VALUE)
   (tolerance 0.1)
   (sensors-used (AT2 FT1 FT4 LT1 TT4))))

(DEFCLASS AT2-Ob
  (MetaClass Class Edited%: (* ; 'Edited 1-Aug-89 13:25 by '))
  (Supers Sensor-Ob)
  (InstanceVariables (reliability 6)
   (reliability-test NIL)
   (ratetest NIL)
   (devtest NIL)
   (validity NIL)
   (status NIL)
   (function AT2-VALUE)
   (tolerance 0.1)
   (sensors-used (AT1 FT1 FT4 LT1 TT4))))

(DEFCLASS ControlValve-Ob
  (MetaClass Class Edited%: (* ; "Edited 5-Jul-89 13:01 by shum-s")
  (Supers Process-Ob))

(DEFCLASS Equipment-Ob
  (MetaClass Class Edited%: (* ; "Edited 5-Jul-89 12:34 by shum-s")
  (Supers Prooess-Ob))

(DEFCLASS FFValve-Ob
  (MetaClass Class Edited%: (* ; "Edited 31-Jul-89 17:43 by '")
  (Supers ControlValve-Ob)
  (InstanceVariables (function VALVE-TEST)
   (tolerance 0.02)
   (positioner VP1)
   (action MFC1)
   (hypothesis NIL))))

(DEFCLASS FT1-Ob
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:31 by '")
  (Supers Sensor-Ob)
  (InstanceVariables (reliability 6)
   (reliability-test NIL)
   (ratetest NIL)
   (devtest NIL)
   (validity NIL)
   (status NIL)
   (function (FT1-VALUE FT1-VP1))
   (tolerance (0.03 0.04))
   (sensors-used (FT4) (VP1)))

(DEFCLASS FT2-Ob
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:32 by '")
  (Supers Sensor-Ob)
  (InstanceVariables (reliability 6)
   (reliability-test NIL)
   (ratetest NIL)
   (devtest NIL)
   (validity NIL)
   (status NIL)
(DEFCLASS FT3-Ob
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:33 by  
  (Supers Sensor-Ob)
  (InstanceVariables (reliability 6)
    (reliability-test NIL)
    (limit-test NIL)
    (dev-test NIL)
    (validity NIL)
    (status NIL)
    (function (FT3-VALUE FT3-VP3))
    (tolerance 0.03 0.04)
    (sensors-used (FT2-VP2))
  ))

(DEFCLASS FT4-Ob
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:33 by 
  (Supers Sensor-Ob)
  (InstanceVariables (reliability 6)
    (reliability-test NIL)
    (limit-test NIL)
    (dev-test NIL)
    (validity NIL)
    (status NIL)
    (function (FT4-VALUE FT4-VP4))
    (tolerance (0.03 0.04))
    (sensors-used (FT1-VP4))
  ))

(DEFCLASS HeatExchanger-Ob
  (MetaClass Class Edited%: (* ; "Edited 2-Aug-39 12:48 by 
  (Supers Equipment-Ob)
  (InstanceVariables (function HTEX-TEST)
    (tolerance 1.6)
    (hypothesis NIL))

(DEFCLASS LT1-Ob
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:34 by 
  (Supers Sensor-Ob)
  (InstanceVariables (reliability 4)
    (reliability-test NIL)
    (limit-test NIL)
    (dev-test NIL)
    (validity NIL)
    (status NIL)
    (function LT1-VALUE)
    (tolerance 0.25)
    (sensors-used (TT2 TT3 TT4 TT5))
  ))

(DEFCLASS Process-Ob
  (MetaClass Class Edited%: (* ; "Edited 14-Aug-89 17:49 by 
  (Supers Object)
  (InstanceVariables (alarmobjects (ALF1-Ob ALF2-Ob ALL1-Ob ALT4-Ob))
    (sensorobjects (AT1-Ob AT2-Ob FT1-Ob FT2-Ob FT3-Ob FT4-Ob LT1-Ob TT1-Ob TT2-Ob TT3-Ob
      TT4-Ob TT5-Ob TT6-Ob TT7-Ob TT8-Ob))
    (hypothesisobjects (Pump-Ob HeatExchanger-Ob RLValve-Ob RRFValve-Ob RRTValve-Ob FFValve-Ob)))

(DEFCLASS Pump-Ob
  (MetaClass Class Edited%: (* ; "Edited 26-Jul-89 13:59 by shum-s")
  (Supers Equipment-Ob)
  (InstanceVariables (function PUMP-TEST)
    (tolerance 0.05)
    (hypothesis NIL))
(DEFCLASS RLValve-Ob
  (MetaClass Class Edited%: (* ; "Edited 31-Jul-89 17:44 by "))
  (Supers ControlValve-Ob)
  (InstanceVariables (function VALVE-TEST)
    (tolerance 0.02)
    (positioner VP4)
    (action MLC1)
    (hypothesis NIL)))

(DEFCLASS RRFValve-Ob
  (MetaClass Class Edited%: (* ; "Edited 31-Jul-89 17:44 by "))
  (Supers ControlValve-Ob)
  (InstanceVariables (function VALVE-TEST)
    (tolerance 0.02)
    (positioner VP2)
    (action MFC2)
    (hypothesis NIL)))

(DEFCLASS RRTValve-Ob
  (MetaClass Class Edited%: (* ; "Edited 31-Jul-89 17:44 by "))
  (Supers ControlValve-Ob)
  (InstanceVariables (function VALVE-TEST)
    (tolerance 0.02)
    (positioner VP3)
    (action MTC8)
    (hypothesis NIL)))

(DEFCLASS Sensor-Ob
  (MetaClass Class Edited%: (* ; "Edited 5-Jul-89 12:27 by shum-a")
    (Supers Process-Ob))

(DEFCLASS TTI-Ob
  (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:35 by "))
  (Supers Sensor-Ob)
  (InstanceVariables (reliability 7)
    (reliability-test NIL)
    (limtest NIL)
    (ratetest NIL)
    (devtest NIL)
    (validity NIL)
    (status NIL)
    (function TT1-VALUE)
    (tolerance 1.5)
    (sensors-used (AT2 FT1 FT2 LT1 TT4 TT8)))

(DEFCLASS TT2-Ob
  (MetaClass Class Edited%: (* ; "Edited 4-Sep-89 15:55 by "))
  (Supers Sensor-Ob)
  (InstanceVariables (reliability 7)
    (reliability-test NIL)
    (limtest NIL)
    (ratetest NIL)
    (devtest NIL)
    (validity NIL)
    (status NIL)
    (function (TT2-VALUE TT2-COM))
    (tolerance (1.0 1.0))
    (sensors-used ([LT1 TT3 TT4 TT5] (TT3 TT4 TT5))))

(DEFCLASS TT3-Ob
  (MetaClass Class Edited%: (* ; "Edited 3-Aug-89 16:37 by "))
  (Supers Sensor-Ob)
  (InstanceVariables (reliability 7)
    (reliability-test NIL)
    (limtest NIL)
    (ratetest NIL)
    (devtest NIL)
    (validity NIL)
    (status NIL)
(function (TT3-VALUE TT3-COM))
(tolerance (1.5 1.5))
(sensors-used (LT1 TT2 TT4 TT5)
(TT2 TT4 TT5))

(DEFCLASS TT4-OB
 (MetaClass Class Edited%: (* ; "Edited 3-Aug-89 16:38 by "))
(Supers Sensor-Ob)
(InstanceVariables (reliability 7)
 (reliability-test NIL)
 (limitest NIL)
 (ratetest NIL)
 (devtest NIL)
 (validity NIL)
 (status NIL)
 (function TT4-VALUE)
 (tolerance 1.5)
 (sensors-used (LT1 TT2 TT3 TT5)))))

(DEFCLASS TT5-OB
 (MetaClass Class Edited%: (* ; "Edited 3-Aug-89 16:38 by "))
(Supers Sensor-Ob)
(InstanceVariables (reliability 7)
 (reliability-test NIL)
 (limitest NIL)
 (ratetest NIL)
 (devtest NIL)
 (validity NIL)
 (status NIL)
 (function TT5-VALUE)
 (tolerance 1.5)
 (sensors-used (LT1 TT2 TT3 TT4)))))

(DEFCLASS TT6-OB
 (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:38 by "))
(Supers Sensor-Ob)
(InstanceVariables (reliability 7)
 (reliability-test NIL)
 (limitest NIL)
 (ratetest NIL)
 (devtest NIL)
 (validity NIL)
 (status NIL)
 (function TT6-VALUE)
 (tolerance 1.5)
 (sensors-used (FT2 FT3 TT5 TT6 TT8)))))

(DEFCLASS TT7-OB
 (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:39 by "))
(Supers Sensor-Ob)
(InstanceVariables (reliability 7)
 (reliability-test NIL)
 (limitest NIL)
 (ratetest NIL)
 (devtest NIL)
 (validity NIL)
 (status NIL)
 (function TT7-VALUE)
 (tolerance 1.5)
 (sensors-used (FT2 FT3 TT5 TT6 TT8)))))

(DEFCLASS TT8-OB
 (MetaClass Class Edited%: (* ; "Edited 1-Aug-89 13:39 by "))
(Supers Sensor-Ob)
(InstanceVariables (reliability 6)
 (reliability-test NIL)
 (limitest NIL)
 (ratetest NIL)
 (devtest NIL)
 (validity NIL)
(status NIL)
(function TT8-VALUE)
(tolerance 1.5)
(sensors-used (FT2 FT3 TT5 TT6 TT7)))

(/BatchMethodDefs)
(/UnbatchMethodDefs)
(DEFINE)

(ASKDATA
 [LAMBDA (classname lvlist)
 (* ; *Edited 16-Jun-89 23:51 by *)
 (* ;function called by ASKDATA to update data interactively*)
 (for lv in lvlist do (PROG (string1 string2 answer)
 (SETQ string1 (CONCAT "* The current entry for "classname
 iv
 'question)
 " > is "
 (GETValue (classname)
 iv)
 "")
 (SETQ string2 (CONCAT string1
 " Please give the new value (q for no changes) : "
 (GETValue (classname)
 iv
 'doc)
 (SETQ answer (PromptRead2 CSRLDBWIN NIL))
 (if (EQUAL answer 'q)
 then (RETURN)
 else (PutValue (classname)
 iv answer))
 (RETURN))]
)

(ASKDATA
 [LAMBDA (topclass)
 (* ; *Edited 24-Jul-89 14:49 by shum-s*)
 (* ;function called by INPUTDATA to update database *)
 (LET [(classlist (APPEND (GETValue (classname) topclass)
 'alarmclasses)
 (APPEND (GETValue (classname) topclass)
 'sensorclasses)
 (GETValue (classname) topclass)
 'valueclasses)]
 (for classname in classlist do (LET [(lvlist (GETValue (classname)
 'lv)]
 (ASKDATA classname lvlist))]

(AT1-VALUE
 [LAMBDA NIL
 (* ; *Edited 24-Jul-89 13:52 by shum-s*)
 (* ;function to estimate the value of feed conc. measurement for validating AT1*)
 (LET [(ca2 (DATA-FETCH 'AT2 'numvalue))
 (f1 (DATA-FETCH 'FT1 'numvalue))
 (f4 (DATA-FETCH 'FT4 'numvalue))
 (f1 (DATA-FETCH 'LT1 'numvalue))
 (f4 (DATA-FETCH 'TT4 'numvalue))
 (ca1)
 (if (AND (NUMBERP ca2)
 (NUMBERP f1)
 (NUMBERP f4)
 (NUMBERP f11)
 (NUMBERP f4))
 then (SETQ ca1 (QUOTIENT (PLUS (TIMES ca2 f1)
 (TIMES 17.749 f1) (TIMES ca2 f4)
 (ANTILOG (QUOTIENT (MINUS 1421.632) (PLUS f1 273.16))
 (PLUS f11 273.16))
 (f1))


else (SETQ ca1 'UNKNOWN))

(AT2-VALUE
 [LAMBDA NIL
 ("* Edited 24-Jul-89 13:52 by shum-s")
 (*
 "function to estimate the value of product conc. measurement for validating AT2")
 (LET ((ca1 (DATA-FETCH 'AT1 'numvalue))
   (f1 (DATA-FETCH 'FT1 'numvalue))
   (f4 (DATA-FETCH 'FT4 'numvalue))
   (lt1 (DATA-FETCH 'LT1 'numvalue))
   (tt4 (DATA-FETCH 'TT4 'numvalue))
   (ca2 z)
   (if (AND (NUMBERP ca1)
            (NUMBERP f1)
            (NUMBERP f4)
            (NUMBERP lt1)
            (NUMBERP tt4))
     then (SETQ z (TIMES 17.749 lt1 (ANTILOG (QUOTIENT (MINUS 1421.632)
               (PLUS 14 273.16))))
            (SETQ ca2 (QUOTIENT (DIFFERENCE (SORT (PLUS (TIMES 1414)
               (TIMES 4.0 z f4 cal)))
               14)
              (TIMES 2.0 z)))))
     else (SETQ ca2 'UNKNOWN))
)

(BOLD
 [LAMBDA NIL
 ("* Edited 14-Jun-89 18:39 by ")
 (*
 "function to change font to bold font")
 (PRINTOUT (TTYDISPLAYSTREAM)
 .FONT BOLDFONT T "BOLD"))

(CLEAR-VALIDATION-WINDOWS
 [LAMBDA NIL
 ("* Edited 27-Jul-89 11:57 by shum-e")
 (*
 "function to clear both windows used in sensor validation")
 (CLEARW VALIDWIN)
 (CLEARW VALRESULTWIN))

(CSRDLDB
 [LAMBDA NIL
 ("* Edited 9-Sep-89 12:57 by ")
 (*
 "function to create database user window with attached menu ")
 (SETQ CSRDLDBWIN (DECODE WINDOW ARG NIL 400 400 "DataBase User Window" 5))
 (CHANGEFONT BOLDFONT CSRDLDBWIN)
 (SETQ MENUNW (MENUWINDOW (create MENU ITEMS 
 (CLEAR-ALL-DATA (INIDATA))
 (CLEAR-ABSTRACTED-DATA (INABST))
 (INPUT-ALL-DATA (INPUTDATA))
 (MAKE-BATCH-DATAFILE (MAKEDATAFILE))
 (READ-DATA-FROM-FILE (READDATA))
 (CLEAR-WINDOW (CLEARW CSRDLDBWIN))
 (INFER-NORMAUTY (DATA-INFERENCE "INFERNORMAUTY" 'all)
 "This infers normality for all sensor data"
 (SUBITEMS (INFER-NORMAUTY-ALL (DATA-INFERENCE "INFERNORMAUTY" 'all)))
 (INFER-NORMAUTY-MANY (DATA-INFERENCE "INFERNORMAUTY" many))
 (INFER-NORMAUTY-ONE (DATA-INFERENCE "INFERNORMAUTY" one))
 (INFER-TREND (DATA-INFERENCE "INFERTREND" 'all)
 "This infers trend for all sensor data"
 (SUBITEMS (INFER-TREND-ALL (DATA-INFERENCE "INFERTREND" 'all)))
 (INFER-TREND-MANY (DATA-INFERENCE "INFERTREND" many))
)
(INFER-TREND-ONE (DATA-INFERENCE 'INFERTREND 'one)

TITLE " DATABASE FUNCTIONS "
CENTERFLG T
MENUROWS " 4
MENUFONT " BOLDFONT
MENUBORDERSIZE - 1
MENOUTLINESIZE - 2))
(ATTACHWINDOW MENUW CSRLDBWIN 'TOP))

(DATA-FETCH
(LAMBDA (classname ivname)
 (Ղ : *Edited 24-Jul-89 14:52 by shum-s")
 (* ; "function to get data from database for internal use without initiating validation")
 (if (type? class ($! classname))
 then (PROG ((dbvalue (GetValue ($! classname)
 ivname)))
 [if (EO dbvalue 'NIL)
 then (if (EO ivname 'value)
 then (SETQ dbvalue (INFERNORMALITY classname))
 elseif (EO ivname 'trend)
 then (SETQ dbvalue (INFERTREND classname))
 else (PROG (answer strlng1 string2)
 (SETQ strlng1 (CONCAT
 " The current entry for "
 (GetValue ($! classname)
 ivname 'question)
 "> > is "
 (GetValue ($! classname)
 ivname)))
 [SETQ strlng2 (CONCAT strlng1
 " Please give the new value (q for no changes) : 
 (GetValue ($! classname)
 ivname 'doc)
 (SETQ answer (PromptRead strlng2 CSRLDBWIN NIL))
 (if (EO answer 'q)
 then (RETURN)
 else (PutValue ($! classname)
 ivname answer)
 (SETQ dbvalue answer)]
 (RETURN dbvalue))
 else (PRINTOUT NIL (CONCAT classname " is not a class object. DATA-FETCH aborted. "]]

(DATA-INFERENCE
(LAMBDA (funcname flag)
 (Ղ : *Edited 27-Jul-89 11:49 by shum-s")
 (* ; "general data inferencing function for menu access")
 (if (EQUAL flag 'one)
 then (LET ((classname (PromptRead "Please give the name of the class: " CSRLDBWIN T)))
 (APPLICATION funcname classname)))
 [if (EQUAL flag 'many)
 then (LET ((classname (PromptRead "Please give the list of the classes: " CSRLDBWIN T)))
 (for classlist in classlist do (APPLICATION funcname classname))
 [if (EQUAL flag 'all)
 then (LET ((classname (PromptRead "Please give the name of the top most class of the database: "
 (CSRLDBWIN T)))
 classlist classname)
 [if (EO funcname 'INFERNORMALITY)
 then (SETQ classlist (APPEND (GetValue ($! topclass)
 'sensorclasses)
 (GetValue ($! topclass)
 'valuedasses)
 (for classname in classlist do (APPLICATION funcname classname)]
 (PRINTOUT CSRLDBWIN (CONCAT " >> > > Function " funcname " Completed << << " ])

(ATTACHWINDOW MENUW CSRLDBWIN 'TOP)
```
(DB-FETCH
  [LAMBDA (classname lname) (* ; "Edited 14-Aug-89 18:42 by ")
    (* ; "function to get data from database and initiate sensor validation")
    (if (type? class ($1 classname))
      then (PROG (dbvalue)
        (if (EQMEMB classname (APPEND (GetValue ($ CSTRDataBase) 'alarmclasses)
            (GetValue ($ CSTRDataBase) 'sensorclasses)
          then (VALIDATE classname))
            (SETQ dbvalue (GetValue ($1 classname) lname))
          (if (EQ dbvalue 'NIL)
            then (if (EQ lname 'value)
                  then (SETQ dbvalue (INFERNORMALITY classname))
                else (SETQ dbvalue (INFERTREND classname))
              else (PROG (answer strlng1 strlng2)
                (SETQ strlng1 (CONCAT "The current entry for " lname 
                    " >  is 
                    (GetValue ($1 classname) lname) 
                    
                    " Please give the new value (q for no changes) :
                    (GetValue ($1 classname) lname 'doc) 
          (SETQ answer (PromptRead strlng2 CSRLDBWIN NIL))
          if (EQ answer 'q)
            then (RETURN)
          else (PutValue ($1 classname) lname answer)
            (SETQ dbvalue answer)
          (RETURN dbvalue))
          else (PRINTOUT NIL (CONCAT classname " is not a database class. DB-FETCH aborted.")))]
    (DESTROYINSTANCES
      [LAMBDA NIL (* ; "Edited 18-Jun-89 18:41 by ")
        (* ; "function to destroy the instances of database classes")
        (OUTPUT CSRLDBWIN)
        (CHANGEFONT BOLDFONT CSRLDBWIN)
        (SETQ answer (PromptRead "Do you want to destroy a single instance (type s) or many instances (type m)" 
          CSRLDBWIN T))
          (* ; "equivalent to Interlisp (ASKUSER NIL NIL (Do you want to destroy a single instance (type s) or many instances (type m)) ((s %single instance%) (m %any instances%)) T NIL NIL")
        (SETQ answer (SELECTQ answer ((S s) T) ((m M) NIL) answer))
        (if (EQUAL answer T)
          then (PROG (classpointer instancepointer)
            (SETQ classpointer (PromptRead 
              "Please give the name of the class whose instance is to be destroyed : ") 
              CSRLDBWIN T))
            (SETQ instancepointer (PromptRead "Please give the name of the instance : 
              CSRLDBWIN T))
            (if ( (1 $1 instancepointer)
              InstOff ($1 classpointer))
              then (PROG NIL 
                (OUTPUT CSRLDBWIN))
```

(printout NIL (CONCAT Instancepointer ')))
    (_ ($1 Instancepointer)
        Destroy)
    (RETURN))
else (RETURN))
else (if (EQUAL answer NIL)
    then (PROG (objectpointer Instancepointer Instancellst)
        (SETQ objectpointer (PromptRead
            "Please give the name of the superspecialist of the data base :
            CSRLDBWIN T))
        [SETQ instancellst (APPEND (GetValue ($1 objectpointer)
            withdatainstances)
            (GetValue ($1 objectpointer)
                'alarminstances)
            (GetValue ($1 objectpointer)
                'sensorinstances)
        (SETQ num (LENGTH instancellst))
        (for I from 1 to num
            do (PROG (Instancepointer)
                (SETQ instancepolnter (CAR instancellst))
                (SETQ instancellst (CDR instancellst))
                (if (_ ($1 instancepolnter)
                        InstOfI
                        ($1 objectpointer))
                    then (PROG NIL
                        (OUTPUT CSRLDBWIN)
                        (printout NIL
                            (CONCAT Instancepolnter
                                ■/))
                        (_ ($1 Instancepolnter)
                            Destroy)
                        (RETURN))
                    else (RETURN
                        (PRINTOUT CSRLDBWIN "Destroy Instances Completed...")
                        (CHANGEFONT DEFAULTFONT CSRLDBWIN)
                        (OUTPUT TTY))
                )
        )
    else NIL)
    (PRINTOUT CSRLDBWIN "Destroy Instances Completed...")
    (CHANGEFONT DEFAULTFONT CSRLDBWIN)
    (OUTPUT TTY))

(FT1-VALUE
  [LAMBDA NIL
    ("; "Edited 24-Jul-89 13:51 by shum-s")
    (";
        "function to estimate FT1 reading based on FT4, it is assumed that level control system is working properly")
    (DATA-FETCH ($ FT4)
        'numvalue))

(FT1-VP1
  [LAMBDA NIL
    ("; "Edited 24-Jul-89 14:04 by shum-s")
    (";
        "function to estimate FT1 reading based on VP1 reading")
    (LET ((position (DATA-FETCH ($ VP1)
            'numvalue))
        reading)
        (if (NUMBERP position)
            then (SETQ reading (TIMES 1.2 position))
            else (SETQ reading 'UNKNOWN))
    )

(FT2-VP2
  [LAMBDA NIL
    ("; "Edited 24-Jul-89 14:02 by shum-s")
    (";
        "function to estimate FT2 based on VP2 reading")
    (LET ((position (DATA-FETCH ($ VP2)
            'numvalue))
        reading)
        (if (NUMBERP position)
            then (SETQ reading (TIMES 2.4 position))
            else (SETQ reading 'UNKNOWN))
    )

(FT3-VP3
  [LAMBDA NIL
    ("; "Edited 24-Jul-89 14:07 by shum-s")
    (";
*function to estimate FT3 based on VP3 reading*

```lisp
(defun ft3-estimation (vp3)
  (let ((position (data-fetch ($vp3) 'numvalue))
        (reading)
        (if (numberp position)
            (setq reading (times 2.4 position))
            (setq reading 'UNKNOWN))
  (ft4-value (lambda nil (*))
    (* ; "Edited 24-Jul-89 13:51 by shum-s")
    (* ; function to estimate FT4 reading based on FT1. It is assumed that level control system is working properly")
    (data-fetch ($ ft1) 'numvalue))
  (ft4-vp4 (lambda nil (*))
    (* ; "Edited 24-Jul-89 14:04 by shum-s")
    (let ((position (data-fetch ($ vp4) 'numvalue))
           (reading)
           (if (numberp position)
                (setq reading (times 1.2 position))
                (setq reading 'UNKNOWN))
  (htex-test (lambda (objectname) (*))
    (* ; "Edited 2-Aug-89 12:50 by ")
    (let ((f2 (db-fetch ft2 'numvalue))
           (13 (db-fetch 'ft3 numvalue))
           (t4 (db-fetch tt4 'numvalue))
           (t6 (db-fetch 'tt6 'numvalue))
           (t7 (db-fetch 'tt7 'numvalue))
           (t8 (db-fetch ttb 'numvalue))
           (toi (getvalue ($) objectname) 'tolerance))
      (if (and (numberp 12) (numberp 13) (numberp t4) (numberp t6) (numberp t7) (numberp t8))
          (setq temp 1 (times 12 (difference 1814)))
          (setq call (plus temp (times 1.200613 (difference 1716))))
          (setq deki (difference 1416))
          (setq dek2 (difference 1817))
          (setq dm (quotient (difference deki dek2) (log (quotient deki del12))))
          (setq cal2 (plus dm (times 2.8931699 temp))))
      (if (and (leq (abs call) toi) (lteq (abs call2) toi))
        (setq test 'pass)
        (else (setq test 'unknown)))
    else (setq test 'unknown))
  (hypothesis-test (lambda (objectname) (*))
    (* ; "Edited 24-Jul-89 16:43 by shum-s")
    (progn (hypo (getvalue ($) objectname))
      "general function for evaluating malfunction hypothesis by calculation")
)
'hypothesis))
equation test)
(if (EQ hypo NIL)
    then (SETQ equation (GetValue ($! objectname)
        'function))
    (SETQ test (APPLY* equation objectname))
    (if (EQ test 'Pass)
        then (SETQ hypo 'Rejected)
    elseif (EQ test 'Fail)
        then (SETQ hypo 'Established)
    else (SETQ hypo 'UNKNOWN))
    (PutValue ($! objectname)
        'hypothesis hypo))
(RETURN hypo))

(INFERNORMALITY
MATCHA (classpointer) (* ; "Edited 27-Jul-89 13:27 by shum-s")
(* ;
    "function to assign normality value to sensor data")
(LET ((norm (GetValue ($! classpointer)
        'normal))
    (reading (DATA-FETCH classpointer 'numvalue))
    (hvalue hvvalue lvalue lvvalue normality string)
    (CONCAT " Abstracted normality value of "
        [ Abstracted normality value of "
        value normality")
    (PRINTOUT CSRLDBWIN string)
    (PutValue ($! classpointer)
        'value normality)

(INFERTREND
MATCHA (classpointer) (* ; "Edited 27-Jul-89 13:30 by shum-s")
(* ;
    "function to assign trend value to a list of data")
(LET ((del (GetValue ($! classpointer)
        'deltrend))
    (rate (LSTSQ (DATA-FETCH classpointer 'pastdata)
        trend string)
    (if (NUMBERP rate)
        then (COND
            ((GREATERP rate del)
                (SETQ trend 'UP))
            ((LESSP rate (MINUS del))
                (SETQ trend 'DOWN))
            (T (SETQ trend 'STEADY))
        else (SETQ trend 'UNKNOWN))
    (SETQ string (CONCAT " Abstracted trend value of "
        [ Abstracted trend value of "
        trends trend"]
        " is: " [ Abstracted trend value of "
        trend trend"]
        " value normality")
    (PRINTOUT CSRLDBWIN string)
    (PutValue ($! classpointer)
        'trend trend))
(INIABST)
[LAMBDA NIL
  (* ; "Edited 9-Sep-89 11:27 by ")
  (* ; "function to initialize abstracted sensor data to NIL")
  (LET ((sensorlist (GetValue ($ CSTRDataBase) ‘sensordasses))
         (valueclist)
         (PRINTOUT CSRLDBWIN " Sensor Abstracted Data Initialization In Progress ")
         (SETQ valuelist (APPEND (GetValue ($ CSTRDataBase) ‘valuedasses)
                                  . sensorlist)))
    (for classname in valuelist do (PutValue ($1 classname)
                                        ‘value NIL))
    (for classname in sensorist do (PutValue ($1 classname)
                                     ‘trend NIL))
    (PRINTOUT CSRLDBWIN " > > > Sensor Abstracted Data Initialization Completed < < < ")
)

(INIDATA)
[LAMBDA NIL
  (* ; "Edited 9-Sep-89 11:28 by ")
  (* ; "function to initialize the case dependent input data in database to NIL")
  (LET ((sensorlist (GetValue ($ CSTRDataBase) ‘sensordasses))
         (valueclist)
         (PRINTOUT CSRLDBWIN " Database Initialization In Progress ")
         (SETQ datalist (APPEND (GetValue ($ CSTRDataBase) ‘valuedasses)
                                  . sensorlist))
         (SETQ dasslist (APPEND (GetValue ($ CSTRDataBase) ‘alarmasses)
                                   . datalist))
         (for dassname in dasslist do ((SETQ ivs (GetValue ($1 dassname)
                                                   ‘ivs))
                                    (for iv in ivs
do (PutValue ($1 dassname)
                                              ‘value NIL))
                                    (for classname in datalist do (PutValue ($1 classname)
                                                               ‘value NIL))
                                    (for classname in sensorlist do (PutValue ($1 classname)
                                                         ‘trend NIL))
                                    (PRINTOUT CSRLDBWIN " > > > Database Initialization Completed < < < ")
)

(INIHYPOTHESIS)
[LAMBDA NIL
  (* ; "Edited 9-Sep-89 11:28 by ")
  (* ; "function to initialize the hypotheses evaluated by calculation to NIL")
  (LET ((objectlist (GetValue ($ Process-Ob) ‘hypothesesobjects))
         (PRINTOUT VALIDWIN " Malfunction Hypothesis Initialization In Progress ")
         (for objectname in objectlist do (PutValue ($1 objectname)
                                              ‘hypothesis NIL))
         (PRINTOUT VALIDWIN " > > > Malfunction Hypothesis Initialization Completed < < < ")
)

(INIVALIDITY)
[LAMBDA NIL
  (* ; "Edited 9-Sep-89 11:28 by ")
  (* ; "function to initialize all the validation test results to NIL ")
  (LET ((alarmlist (GetValue ($ Process-Ob) ‘alarmobjects))
         (sensorlist (GetValue ($ Process-Ob) ‘sensorobjects))
         (PRINTOUT VALIDWIN " Sensor Validation Initialization In Progress ")
         (for alarmobject in alarmlist do (PutValue ($1 alarmobject)
                                             ‘validity NIL))
         (for sensorobject in sensorlist do (PutValue ($1 sensorobject)
                                              ‘validity NIL)
                                              ‘status NIL)
                                              ‘reliability-test NIL))
(PutValue ($1 sensorobject) 'limtest NIL)
(PutValue ($1 sensorobject) 'ratetest NIL)
(PutValue ($1 sensorobject) 'devtest NIL))

(PRINTOUT VALIDWIN " >>> Sensor Validation Initialization Completed <<< ")

(INPUTDATA
[LAMBDA NIL
("* ; *Edited 24-Jul-89 15:22 by shum-s")
(LAMBDAS"function to update the entire database interactively by asking the user")
((topclass (PromptRead " Please give the name of the top node of the database: " CSRLDBWIN NIL))

string answer)

(setq string (CONCAT " Do you want to make any changes to the database " topclass " ? " ))

(setq answer (SELECTQ answer
(by Y yes YES)
(N N no No NO)

answer))

(if (EQUAL answer NIL)

then (PRINTOUT CSRLDBWIN " >>> Data Not Changed <<< ")

else (ASKDATALOOPS topdass))

(PRINTOUT CSRLDBWIN " >>> Initialization completed <<< ")

(UMITCHECK
[LAMBDA (dassname)
("* ; *Edited 1-Aug-89 13:53 by shum-s")
(LAMBDAS"function to perform limit checking on the data")
((highlimit (GetVaiue ($1 dassname) 'hlimit))
(lowlimit (GetVaiue ($1 dassname) 'llimit))
(reading (DATA-FETCH dassname 'numvalue))

icheck string objectname)

(if (NUMBERP reading)

then (if (OR (GREATERP reading highllimit)
(LESSP reading lowlimit))

then (SETQ icheck 'Fail)

else (SETQ Icheck 'Pass))

else (SETQ icheck 'UNKNOWN))

(SETQ string (CONCAT " [ Sensor " dassname " absolute limit checking result: " icheck " ] "

))

(PRINTOUT VALIDWIN string)

(if (NOT (EQ icheck 'Pass))

then (PRINTOUT VALRESULTWIN string))

(SETQ objectname (PACK* dassname '-Ob))

(PutValue ($1 objectname) 'limtest icheck))

LSTSQ
[LAMBDA (x y)
("* ; *Edited 24-Jul-89 15:27 by shum-s")
(LAMBDAS"function to perform linear regression on a list of data to find the slope")
((if (AND (LSTP x) (EQ y 'NIL))

then (LET ((num (LENGTH x)))

(setq y x)

(setq x NIL)

(for i from 1 to num do (SETQ x (CONS I x)))

(setq x (REVERSE x))

(SIGMA x y))

else (AND (LSTP x)

(LSTP y))

then (SIGMA x y)

else (PRINTOUT NIL " Function LSTSQ requires data list as input, which is not provided. "))
(LT1-VALUE
  (LAMBDA (classname)
    (let ((t2 (DATA-FETCH 'TT2 'numvalue))
          (t3 (DATA-FETCH 'TT3 'numvalue))
          (t4 (DATA-FETCH 'TT4 'numvalue))
          (t5 (DATA-FETCH 'TT5 'numvalue)))
      (if (and (numberp t2)
               (numberp t3)
               (numberp t4)
               (numberp t5))
        (if (geq (difference t2 t3) 3.5)
            (setq l1 1.75)
            (if (geq (difference t3 t4) 3.5)
                (setq l1 1.25)
                (if (geq (difference t4 t5) 3.5)
                    (setq l1 0.75)
                    (if (numberp (get-value ($$ TT4)
                        'normal))
                        (setq l1 2.25)
                        (setq l1 0.25)))
            (setq l1 'unknown))
        (makédataline
          (lambdā nil
            (let ((filestream (openstream cstr-dataline output nil nil))
                  (string (promptread "input data string (classname lvname value)" csrdlwline nil nil)))
              (print string filestream)
              (close filestream)))
      (makéinstancés
        (lambdā nil
          (let ((answer (promptread "do you want to make a single instance (type s) or many instances (type m)" csrdlwline) t))
            (setq answer (selectq answer
                            (s s)
                            (m m)
                            nil
                            answer))
            (if (equal answer t)
                (prog ((classpointer (promptread "please give the name of the database class" csrdlwline) t))
                      (instancepointer (promptread "please give the name of the instance : " csrdlwline) t)
                      inst)
                (setq inst (get-value ($$ objectpointer)
                                    'node)
                        'newinstancepointer))
                (return))
            (else (if (equal answer nil)
                        (prog ((objectpointer (promptread "please give the name of the top node of the database : " csrdlwline) t))
                              instancelist (classlist num)
                              (setq classlist (append (get-value ($$ objectpointer)
                                                          'classes)
                                             (get-value ($$ objectpointer)
                                                          'alarmclasses)
                                             (get-value ($$ objectpointer))))
                        (return)))
            (return)))))

(makefile
  (lambdā nil
    (let ((filestream (openstream cstr-data-file output nil nil))
          (string (promptread "input data string (classname lvname value)" csrdlwline nil nil)))
      (print string filestream)
      (close filestream)))

(makeinstances
  (lambdā nil
    (let ((answer (promptread "do you want to make a single instance (type s) or many instances (type m)" csrdlwline) t))
      (setq answer (selectq answer
                      (s s)
                      (m m)
                      nil
                      answer))
      (if (equal answer t)
          (prog ((classpointer (promptread "please give the name of the database class" csrdlwline) t))
                  (instancepointer (promptread "please give the name of the instance : " csrdlwline) t))
                inst)
          (setq inst (get-value ($$ objectpointer)
                              'node)
                  'newinstancepointer))
        (return))
      (else (if (equal answer nil)
               (prog ((objectpointer (promptread "please give the name of the top node of the database : " csrdlwline) t))
                     instancelist (classlist num)
                     (setq classlist (append (get-value ($$ objectpointer)
                                                  'classes)
                                              (get-value ($$ objectpointer)
                                              'alarmclasses)
                                              (get-value ($$ objectpointer))))
               (return)))
          (return)))))
"sensorclasses"

[SETQ instanceslist (APPEND (GetValue ($1 objectpointer) 'withdatainstances)
 (GetValue ($1 objectpointer) 'alarminstances)
 (GetValue ($1 objectpointer) 'sensorinstances)]

(SETQ num (LENGTH classlist))
(for 1 from 1 to num
 do (PROG ((classpointer (CAR classlist))
 (instancepointer (CAR Instanceslist))
 ninst)
 (SETQ classlist (CDR classlist))
 (SETQ Instanceslist (CDR Instanceslist))
 (printout CSRLDBWIN (CONCAT instancepointer '/))
 (SETQ nInst (_ ($1 classpointer))
 New Instancepointer))
 (* ninst Inspect)
 (RETURN)
 else NIL)
(PRINTOUT CSRLDBWIN "Make instances Completed...")

(MEAN
[ LAMBDA (dataset) (* ; "Edited 24-Jul-89 16:18 by shum-s")
 (*;
 "function to calculate the mean of a list of data")
 (if (LISTP dataset)
 then (LET ((num (LENGTH dataset))
 (total (SUM dataset))
 (avg)
 (SETQ avg (FQUOTIENT total num)))
 else (PRINTOUT NIL (CONCAT dataset " is not a data list")))

(NOISECHECK
[ LAMBDA (classname) (* ; "Edited 1-Aug-89 13:51 by ")
 (*;
 "function to check for the presence of noise in the data")
 (LET ([dev (STDEV (DATA-FETCH classname pastdata)]
 (dlim (GetValue ($1 classname)
 'dlim))
 dcheck string objectname)
 (if (NUMBERP dev)
 then (if (GREATERP (ABS dev)
 dlim)
 then (SETQ dcheck 'Pass)
 else (SETQ dcheck 'Fail))
 else (SETQ dcheck 'UNKNOWN)
 (printout VALWIN string)
 (if (NOT (EQ dcheck 'Pass))
 then (printout VALRESULTWIN string))
 (SETQ objectname (PACK* classname 'Ob))
 (PutValue ($1 objectname)
 'devtest dcheck))

(NORMAL
[ LAMBDA NIL (* ; "Edited 7-Jun-89 17:38 by ")
 (*;
 "function to change font to normal font")
 (CHANGEFONT DEFAULTFONT (TTYDISPLAYSTREAM))

(NORMALITYCHECK
[ LAMBDA (classname) (* ; "Edited 1-Aug-89 10:23 by ")
 (*;
 "function to check the normality of data")
 (PROG (normality (INFERNORMALITY classname))
 ncheck string)
 (if (EQ normality 'N)
(SETQ ncheck 'Pass)
else (SETQ ncheck 'Fall))
(PRINTOUT VALIDWIN string)
(if (EQ ncheck 'Fall)
then (PRINTOUT VALRESULTWIN string)))
(RETURN ncheck)]

(PRINTCSRLSYSTEM
[LAMBDA (BCSRsystemname) (* ; "Edited 16-Jul-89 17:48 by shum-s")
(* function to print out the BCSRL specialist definitions in a file called CSRLPRINTOUT)
(PrintSpecialistsinSystem BCSRLsystemname (OPENSTREAM "CSRLPRINTOUT "OUTPUT NIL NIL))

(PUMP-TEST
[LAMBDA (objectname) (* ; "Edited 26-Jul-89 14:11 by shum-s")
(* function to test the pump malfunction hypothesis by calculation)
(LET ((f2 (DB-FETCH FT2 'numvalue))
  (f4 (DB-FETCH FT4 'numvalue))
  (valve2 (DB-FETCH 'VP2 'numvalue))
  (valve4 (DB-FETCH 'VP4 'numvalue))
  (tol (GetValue ($1 objectname)
    'tolerance))
  delflow2 delflow4 test)
  (if (AND (NUMBERP f2)
    (NUMBERP f4)
    (NUMBERP valve2)
    (NUMBERP valve4))
    then (SETO delflow2 (DIFFERENCE (TIMES 2.4 valve2) f2))
    (SETO delflow4 (DIFFERENCE (TIMES 1.2 valve4) f4))
    (if (AND (GREATERP delflow2 tol)
      (GREATERP delflow4 tol))
      then (SETO test 'Fall)
      else (SETO test 'Pass))
    else (SETO test 'UNKNOWN))

(RATECHECK
[LAMBDA (classname) (* ; "Edited 1-Aug-89 13:53 by ")
(* function to perform rate of change limit checking on a list of data *)
(LET ((rate (LSTSQ (DATA-FETCH classname 'pastdata)
    (limit (GetValue ($1 classname)
      'ratiolimit))
    rcheck string objectname)
    (if (NUMBERP rate)
      then (if (GREATERP rate rlimit)
        then (SETO rcheck 'Fail)
        else (SETO rcheck 'Pass))
      else (SETO rcheck 'UNKNOWN))
    (SETO string (CONCAT "[ Sensor " classname " rate of change limit checking result: " rcheck "]"))
    (PRINTOUT VALIDWIN string)
    (if (NOT (EQ rcheck 'Pass))
      then (PRINTOUT VALRESULTWIN string)))
    (SETQ objectname (PACK* classname -Ob))
    (PutValue ($1 objectname)
      'ratetest rcheck))

(READDATA
[LAMBDA NIL (* ; "Edited 3-Aug-83 14:40 by ")
(* function to read data from a file and enter into the classes")
(LET ((fname (PromptRead 
  "Please give the name of the datafile: " CSRLDBWIN NIL)))
  fstream datalist num)
  (SETO fstream (OPENFILE fname 'INPUT NIL NIL))
  (SETO datalist (READ fstream NIL NIL))
(DEFINITION num (LENGTH datalist))
(for i from 1 to num
  do (LET ((data (CAR datalist))
        (classname lvname value))
      (SETQ classname (CAR data))
      (SETQ lvname (CADR data))
      (SETQ value (CADDR data))
      (SETQ datalist (CDR datalist))
      (PRINTOUT CSRLDBWIN (CONCAT " (classname " lvname " value " ))
      (PutValue ($1 classname)
               lvname value))))
(PRINTOUT CSRLDBWIN *)

>>> Number of data read: "num" <<<")
(CLOSEF fstream])

(RELIABILITYCHECK
  [LAMBDA (classname)
  (* ; "Edited 1-Aug-89 10:23 by ")
  (* ; function to check the reliability ranking of sensor")
  (PROG ((objectname (PACK* classname '-Ob))
         rel rellimit relcheck string)
    (SETQ rel (GetValue ($1 objectname)
                     'reliability))
    (if (GEQ rel 6)
      then (SETQ relcheck Pass)
      else (SETQ relcheck 'Fail))
    (PutValue ($I objectname)
               'reliability-test relcheck)
    (SETQ string (CONCAT "[ Sensor " classname " reliability ranking test result: " relcheck " ]")
    (PRINTOUT VALIDWIN string)
    (if (EQ relcheck 'Fail)
      then (PRINTOUT VALRESULTWIN string))
    (RETURN relcheck)]

(SIGMA
  [LAMBDA (xy)
  (* ; "Edited 24-Jul-89 16:29 by shum-s")
  (* ; function called by LSTSQ to perform linear regression")
  (LET ((m (LENGTH X))
         (sumx (SUM x))
         (sumy (SUM y))
         sqsumx sumxy sumxsq d a)
    (SETQ sqsumx (TIMES sumx sumx))
    (SETQ sumxy 0.0)
    (SETQ sumxsq 0.0)
    (for i from 1 to m do [SETQ sumxy (PLUS sumxy (TIMES (CAR x)
                   (CAR y))
      (SETQ sumxsq (PLUS sumxsq (TIMES (CAR X)
                   (CAR x))
      (SETQ d (DIFFERENCE (TIMES m sumxsq)
                   sqsumx))
      (SETQ b (QUOTIENT (DIFFERENCE (TIMES sumxsq sumy)
                   (TIMES sumx sumxy))
                   d))
      (SETQ a (QUOTIENT (DIFFERENCE (TIMES m sumxy)
                   (TIMES sumx sumxy))
                   d))
      (STDEV
        [LAMBDA (dataset)
        (* ; "Edited 27-Jul-89 13:35 by shum-s")
        (* ; function to calculate the standard deviation of a list of data")
        (if (LISTP dataset)
          then (LET ((num (LENGTH dataset))
                     (avg (MEAN dataset))
                     (var 0.0))
          else (RETURN var)))]

(PRINTOUT CSRLDBWIN *)
for I from 1 to num do ((SETQ dif (DIFFERENCE (CAR dataset)
   (avg))
   (SETQ var (PLUS var (TIMES dif dif)))))
(SETQ dev (SORT (QQUOTIENT var (DIFFERENCE num 1))
   else (PRINTOUT NIL (CONCAT dataset
   " is not a data list. Standard deviation not calculated. ")))

(SUM
[LAMBDA (dataset) .
   (* ; "Edited 20-Jul-89 17:21 by shum-s")
   (*;
   "function to calculate the sum of a list of data")
(if (LSTP dataset)
   then (EVAL (APPEND '(PLUS)
      dataset))
   else (PRINTOUT NIL (CONCAT dataset '  Is not a data list')))

(TIME-VALIDATION
[LAMBDA NIL
   (* ; "Edited 9-Sep-89 11:26 by ")
   (*;
   "function to reroute the window printout to files for tracing case run")
(if (TSTP dataset)
   then (EVAL (APPEND '(PLUS)
      dataset))
   else (PRINTOUT NIL (CONCAT dataset '  Is not a data list')))

(TIME-VALIDATION-STOP
[LAMBDA NIL
   (* ; "Edited 9-Sep-89 11:33 by ")
   (*;
   "function to return the printout to the interface windows")
(CLOSEF CSRLDBWIN)
(CSLDB)
(CLOSEF VAUWIN)
(CLOSEF VALRESULTWIN)
(VALIDATION))

(TRENDCHECK
[LAMBDA (classname) (* ; "Edited 1-Aug-89 10:23 by ")
   (*;
   "function to check the trend of a list of data")
(PROG ((trend (INFERTREND classname))
tcheck string)
   (if (EQ trend 'STEADY)
      then (SETO tcheck 'Pass)
      else (SETO tcheck 'Fail))
   (SETQ string (CONCAT '  [ Sensor * classname * trend test result: '  tcheck '  ] ' )))
   (PRINTOUT VAUWIN string)
   (if (EQ tcheck 'Fail)
      then (PRINTOUT VALRESULTWIN string))
   (RETURN tcheck))

(TT1-VALUE
[LAMBDA NIL
   (* ; "Edited 3-Aug-89 16:32 by ")
   (*;
   "function to estimate the value of feed temp. measurement for validating TT1")
(LET ((ca2 (DATA-FETCH AT2 'numvalue)))
   (t1 (DATA-FETCH 'T1 'numvalue))
   (t2 (DATA-FETCH 'T2 'numvalue)))
   (if (AND (NUMBERP ca2)
            (NUMBERP t1)
            (NOT (EQUAL t1 0))
            (NUMBERP t2)
            (NUMBERP t4)
            (NUMBERP t6))
      then (SETO t1 (DIFFERENCE (PLUS t4 (QUOTIENT (TIMES (DIFFERENCE t4 t6) (QQUOTIENT t6 (DIFFERENCE t1 t6)))))
   (RETURN t1))
   else (PRINTOUT NIL (CONCAT '  Is not a data list. Standard deviation not calculated. ")))

232

f1))

(QUOTIENT (TIMES 502.192 I) (TIMES ca2 ca2)
(Antilog (QUOTIENT (MINUS 1421.632)
(PLUS I4 273.16

else (SETQ 11 'UNKNOWN)

(TT2-COM
LAMBDA NIL
(* ; "Edited 4-Sep-89 15:51 by ")

"function to estimate TT2 value for comparison with the sensor reading"

(LET ((t2 (DATA-FETCH 'TT2 'numvalue))
(t3 (DATA-FETCH 'TT3 'numvalue))
(t4 (DATA-FETCH 'TT4 'numvalue))
(t5 (DATA-FETCH 'TT5 'numvalue))
(if (AND (NUMBERP I2)
(NUMBERP t3))
then (if (GEQ (DIFFERENCE t3 t2) 3.5)
then (SETQ t2 (TIMES t3 0.95))
else (SETQ t2 t3))
elseif (AND (NUMBERP t2)
(NUMBERP t4))
then (if (GEQ (DIFFERENCE t4 t2) 3.5)
then (SETQ t2 (TIMES t4 0.95))
else (SETQ t2 t4))
elseif (AND (NUMBERP t2)
(NUMBERP t5))
then (if (GEQ (DIFFERENCE t5 t2) 3.5)
then (SETQ t2 (TIMES t5 0.95))
else (SETQ t2 t5))
else (SETQ t2 'UNKNOWN)

(TT2-VALUE
LAMBDA NIL
(* ; "Edited 3-Aug-89 12:07 by ")

"function to estimate the value of reactor vapor phase temp. for validating TT2"

(LET ((t1 (DATA-FETCH 'LT1 'numvalue))
(t3 (DATA-FETCH 'TT3 'numvalue))
(t4 (DATA-FETCH 'TT4 'numvalue))
(t5 (DATA-FETCH 'TT5 'numvalue))
(t2)
if (NUMBERP I1)
then (if (GREATERP I1 2.0)
then (if (NUMBERP I3)
then (SETQ t2 t3)
elseif (NUMBERP t4)
then (SETQ t2 t4)
elseif (NUMBERP t5)
then (SETQ t2 t5)
else (SETQ t2 'UNKNOWN))
elseif (AND (NUMBERP I1 1.5)
(NUMBERP I3))
then (SETQ t2 (TIMES 1.95 I3))
elseif (AND (GREATERP I1 1.9)
(NUMBERP I4))
then (SETQ t2 (TIMES 1.95 I4))
elseif (AND (GREATERP I1 0.5)
(NUMBERP I5))
then (SETQ t2 (TIMES 0.95 I5))
else (SETQ t2 'UNKNOWN)
else (SETQ t2 'UNKNOWN)

(TT3-COM
LAMBDA NIL
(* ; "Edited 3-Aug-89 13:18 by ")

"function to estimate TT3 value for comparison with the sensor reading"
(LET ((t2 (DATA-FETCH 'TT2 'numvalue))
  (t3 (DATA-FETCH 'TT3 'numvalue))
  (t4 (DATA-FETCH 'TT4 'numvalue))
  (t5 (DATA-FETCH 'TT5 'numvalue))
  (if (AND (NUMBERP t2) (NUMBERP t3) (NUMBERP t4) (NUMBERP t5))
    (if (GEO (DIFFERENCE t3 t2) 3.5)
      (if (NUMBERP t4)
        (if (GEQ (DIFFERENCE t3 t4) 3.5)
          then (SETQ t3 (QUOTIENT t3 0.95))
          else (SETQ t3 'UNKNOWN))
        else (SETQ t3 'UNKNOWN))
    else (SETQ t3 'UNKNOWN))

(TT3-VALUE
 [LAMBDA NIL
  (* "EDHEX 3-Aug-89 12:27 by ")
  (* :
    "Function to estimate TT3 value for validating the sensor")
  (LET ((f1 (DATA-FETCH 'LT1 'numvalue))
    (t2 (DATA-FETCH 'TT2 'numvalue))
    (t4 (DATA-FETCH 'TT4 'numvalue))
    (t5 (DATA-FETCH 'TT5 'numvalue))
    t3)
    (IF (NUMBERP f1)
      (IF (GREATERTP t1 1.5)
        then (IF (NUMBERP t4)
          then (SETQ t3 (QUOTIENT t3 0.95))
          else (SETQ t3 'UNKNOWN))
        else (SETQ t3 'UNKNOWN))
    else (SETQ t3 'UNKNOWN))
)

(TT4-VALUE
 [LAMBDA NIL
  (* "EDHEX 3-Aug-89 13:04 by ")
  (* :
    "Function to estimate TT4 value for validating the sensor reading")
  (LET ((f1 (DATA-FETCH 'LT1 'numvalue))
    (t2 (DATA-FETCH 'TT2 'numvalue))
    (t3 (DATA-FETCH 'TT3 'numvalue))
    (t5 (DATA-FETCH 'TT5 'numvalue))
    t4)
    (IF (NUMBERP f1)
      (IF (GREATERTP t1 1.0)
        then (IF (NUMBERP t5)
          then (SETQ t4 (QUOTIENT t4 0.95))
          else (SETQ t4 'UNKNOWN))
        else (SETQ t4 'UNKNOWN))
    else (SETQ t4 'UNKNOWN))
)
then (SETQ t4 13)
else (NUMBERP t5)
then (if (GREATERP t1 0.5)
then (SETQ t4 (TIMES 0.9 5 15))
else (SETQ t4 t5))
else (SETQ t4 'UNKNOWN])

else (SETQ t4 'UNKNOWN])

(TT5-VALUE
[FUNCTION NIL (* ; "Edited 3-Aug-89 13:04 by ")

"function to estimate TT5 value for validating the sensor reading"

(LET ((II (DATA-FETCH LT1 'numvalue))
(t2 (DATA-FETCH TT2 'numvalue))
(t3 (DATA-FETCH 'numvalue))
(t4 (DATA-FETCH TT4 'numvalue))
(t5))
(if (NUMBERP t1)
then (if (NUMBERP t2)
then (SETQ t5 t2)
else (SETQ t5 (QUOTIENT t2 0.95)))
else (NUMBERP t1)
then (if (GREATERP II 1.5)
then (SETQ t5 t3)
else (SETQ t5 (QUOTIENT t3 0.95)))
else (NUMBERP t4)
then (if (GREATERP t4 1.0)
then (SETQ 15 (QUOTIENT 14 0.95)))
else (SETQ 15 UNKNOWN])

else (SETQ 15 UNKNOWN])

(TT6-VALUE
[FUNCTION NIL (* ; "Edited 3-Aug-89 16:29 by ")

"function to estimate coolant inlet temp. for validating TT6 sensor reading"

(LET ((f2 (DATA-FETCH FT2 'numvalue))
(f3 (DATA-FETCH 'numvalue))
(f5 (DATA-FETCH TT5 'numvalue))
(f7 (DATA-FETCH TT7 'numvalue))
(f8 (DATA-FETCH TT8 'numvalue)))
16)
(if (AND (NUMBERP f2)
then (if (NUMBERP f3)
then (SETQ 16 (DIFFERENCE 17 (QUOTIENT (TIMES 0.8329 f2 (DIFFERENCE 15 18))
3)))
else (SETQ 16 'UNKNOWN])

else (SETQ 16 'UNKNOWN])

(TT7-VALUE
[FUNCTION NIL (* ; "Edited 3-Aug-89 16:29 by ")

"function to estimate coolant outlet temp. for validating TT7 sensor reading"

(LET ((f2 (DATA-FETCH FT2 'numvalue))
(f3 (DATA-FETCH FT3 'numvalue))
(f5 (DATA-FETCH TT5 'numvalue))
(f6 (DATA-FETCH TT6 'numvalue))
(f8 (DATA-FETCH TT8 'numvalue)))
\(\text{(t7)}\)
\(\text{(if (AND (NUMBERP f2) (NUMBERP f3) (NOT (EQUAL f3 0)) (NUMBERP tS) (NUMBERP t7))}\)
\(\text{then (SETQ t7 (PLUS tS (QUOTIENT (TIMES 0.8329 (2 (DIFFERENCE tS tS)) f3))))}\)
\(\text{else (SETQ t7 "UNKNOWN" )}\)

\[\text{TTB-VALUE}\]
\[\text{[LAMBDA NIL] (\text{"Edited 3-Aug-89 16:27 by "})}\]
\(\text{(LET ((f2 (DATA-FETCH 'FT2 'numvalue)) (f3 (DATA-FETCH 'FT3 'numvalue)) (t5 (DATA-FETCH 'TTS 'numvalue)) (t7 (DATA-FETCH 'TT7 'numvalue)) (t8))}\)
\(\text{(if (AND (NUMBERP f2) (NOT (EQUAL f2 0)) (NUMBERP f3) (NUMBERP t5) (NUMBERP t6) (NUMBERP t7))}\)
\(\text{then (SETQ t8 (DIFFERENCE t5 (QUOTIENT (TIMES 1.2005 f5 (DIFFERENCE t7 tS)) m))}\)
\(\text{else (SETQ t8 "UNKNOWN")}\)

\[\text{UPDATE-VALIDITY}\]
\[\text{[LAMBDA (classlist)] (\text{"Edited 6-Sep-89 11:42 by "})}\]
\(\text{(*; \text{function to set the validity of sensors used in a calculation when the calculated value matches with measured value")}\)
\(\text{(LET ((sensorclass (GetValue ($ CSTRDataBase) 'sensordasses)) validlist sensorobject validlist rel val) (SETQ validlist NIL) (for classnamen in classlist do (if (EQMEMB classnamen sensorclass) then (SETQ sensorobject (PACK* classnamen 'Ob)) (SETQ rel (GetValue ($ sensorobject) 'reliability)) (SETQ val (GetValue ($ sensorobject) 'validity)) (if (AND (GEO rel 6) (EQ val Nil)) then (PutValue ($ sensorobject) 'validity 'Valid) (PutValue ($ sensorobject) 'status 'End) (SETQ validlist (CONS classnamen validlist)) (if (NOT (EQ validlist Nil)) then (PRINTOUT VALRESULTWIN (CONCAT " [ Sensors " validlist " are set to be valid ] "))))\)

\[\text{VALIDATE}\]
\[\text{[LAMBDA (classnamen)] (\text{"Edited 6-Sep-89 14:12 by ")}\]
\(\text{(*; \text{function to validate sensor and alarm data")}\)
\(\text{(LET ((sensors (GetValue ($ CSTRDataBase) 'sensorclasses)) (ctrlsensors (GetValue ($ CSTRDataBase) 'controlsensors)) (alarms (GetValue ($ CSTRDataBase) 'alarmclasses)) (if (EQMEMB classnamen (APPEND alarms sensors)) then then)\)\)}

(LET ((objectname (PACK* classname 'Ob))
  val)
  (SETQ val (GetValue ($) objectname)
    'validity))
(if (EQ val NIL)
  then (if (EQMEMB classname alarms)
    then (VALIDATE-ALARM classname objectname)
    else (LET ((simplecheck (VALIDATE-SIMPLE classname objectname))
        othercheck numvalue)
      .
      (if (EQ simplecheck 'invalid)
        then (PutValue ($) val)
          'numvalue
          'UNKNOWN)
          (PutValue ($) val)
          'trend
          'UNKNOWN)
          (PutValue ($) val)
          'pastdata
          'UNKNOWN)
          (SETQ numvalue (VALUE-REPLACE classname objectname))
      else (if (EQMEMB classname ctrlsensors)
        then (SETQ numvalue (VALUE-COMPARE classname objectname))
        else (SETQ othercheck (VALIDATE-REST classname objectname))
        (if (EQ othercheck 'Probable)
          then (SETQ numvalue
            (VALUE-COMPARE classname objectname))
          else (PRINTOUT VALIDWIN (CONCAT * [ Class '* classname
            " Is not a sensor or alarm. Validation not performed. * ] )
        )
      )
  )
else (PRINTOUT VALIDWIN (CONCAT * [ Sensor * classname
  " reading passed all tests in validation module. ] * )
)

(VALIDATE-ALARM
 [LAMBDA (classname objectname)]
 (* ; "Edited 14-Aug-89 18:45 by *

  (* ; "function to validate alarm state"

  (LET ((alarm (DATA-FETCH ($) objectname)
    'activated))
    (sensordass (GetValue ($) objectname)
      'sensor))
      reading norm highlow highlimit lowlimit alarmactivated val)
      (SETQ reading (DB-FETCH sensordass numvalue))
      (SETQ norm (GetValue ($) sensordass)
        'normal))
      (SETQ highlow (VALUE-COMPARE classname objectname))
      (SETQ highlimit (PLUS norm highlow))
      (SETQ lowlimit (DIFFERENCE norm highlow))
      (if (NUMBERP reading)
        then (if (OR (GREATERP reading highlimit)
          (LESSP reading lowlimit))
          then (SETQ alarmactivated 'T)
          else (SETQ alarmactivated 'F))
        else (SETQ alarmactivated 'UNKNOWN))
      (if (EQUAL alarmactivated alarm)
        then (SETQ val 'Valid)
        else (PRINTOUT VALRESULTWIN (CONCAT * [ Alarm * classname * is valid ] * ))
      )
      else (NOT (EQUAL alarmactivated 'UNKNOWN))
    (if (EQ val 'Invalid)
      then (PRINTOUT VALRESULTWIN (CONCAT * [ Alarm * classname * failed but validated ] * ))
      else (SETQ val 'UNKNOWN)
    )
  )
)
(PRINTOUT VALRESULTWIN (CONCAT "[ Alarm * classname * cannot be validated ]"))
(PutValue ($! objectname)
  validity val))

(VVALIDATE-REST
  [LAMBDA (classname objectname)
   (* ; "Edited 30-Aug-89 15:43 by ")
      (* ; function to perform the remaining tests in the algorithm and return validity condition")
     (LET (val string)
       (if (EQ (NORMALITYCHECK classname)
         "Fail")
         (SETQ val "Probable")
       (PRINTOUT VALRESULTWIN (CONCAT "[ Alternate * classname * value being calculated to confirm ]"))
       (elseif (EQ (TRENDCHECK classname)
         "Fail")
         (SETQ val "Probable")
       (PRINTOUT VALRESULTWIN (CONCAT "[ Alternate * classname * value being calculated to confirm ]"))
       (elseif (EQ (RELIABILITYCHECK classname)
         "Fail")
         (SETQ val "Probable")
       (PRINTOUT VALRESULTWIN (CONCAT "[ Alternate * classname * value being calculated to confirm ]"))
       (else (SETQ val Through)
         (PutValue ($! objectname)
           'status
           'End))
       (PutValue ($! objectname)
         'validity val)]
)

(VVALIDATE-SENSORLIST
  [LAMBDA (sensorlist)
   (* ; "Edited 21-Jun-89 15:57 by ")
      (* ; function to initiate the validation for a list of sensors")
     (for sensorclass in sensorlist do (VVALIDATE sensorclass)])

(VVALIDATE-SIMPLE
  [LAMBDA (classname objectname)
   (* ; "Edited 30-Aug-89 15:45 by ")
      (* ; function to perform simple data checks and return validity condition")
     (LET (val string)
       (if (NOT (EQ (LIMITCHECK classname)
         "Pass"))
         (SETQ val 'invalid)
       (PRINTOUT VALRESULTWIN (CONCAT "[ Alternate * classname * value being calculated to replace ]"))
       (elseif (NOT (EQ (RATECHECK classname)
         "Pass"))
         (SETQ val 'invalid)
       (PRINTOUT VALRESULTWIN (CONCAT "[ Alternate * classname * value being calculated to replace ]"))
       (elseif (NOT (EQ (NOISECHECK classname)
         "Pass"))
         (SETQ val 'invalid)
       (PRINTOUT VALRESULTWIN (CONCAT "[ Alternate * classname * value being calculated to replace ]"))
       (else (SETQ val 'Probable)
         (PutValue ($! objectname)
           'status
           'Midpoint)
       (PutValue ($! objectname)
         'validity val))
)

(VALIDATION
  [LAMBDA NIL
   (* ; "Edited 9-Sep-89 13:38 by ")
      (* ; function to create sensor validation user window with attached menu and perform initialization")
     (SETQ VALRESULTWIN (DECODE WINDOW ARG NIL 400 200 "Sensor Validation Important Results Window" 10 T))
)
(SETQ VALIDWIN (DECODE.WINDOW.ARG NIL 400 240 "Sensor Validation Information Window" 5))
(SETQ VALUSERWIN (DECODE.WINDOW.ARG NIL 390 400 "Sensor Validation User Input Window" 10 T))
(DSPSCROLL T VALRESULTWIN)
(DSPSCROLL T VALIDWIN)
(DSPSCROLL T VALUSERWIN)
(CHANGEFONT BOLDFONT VALIDWIN)
(CHANGEFONT BOLDFONT VALRESULTWIN)
(CHANGEFONT BOLDFONT VALUSERWIN)
(SETQ MENU)
(MENUWINDOW (create MENU
ITEMS "[(INITIALIZE-VALIDATION INVALIDITY))
  (INITIALIZE-HYPOTHESIS INYHYPOTHESIS))
  (CLEAR-WINDOWS CLEAR-VALIDATION-WINDOWS))
  [SENSOR-VALIDATION (DATA-INERENCE VALIDATE 'all)
    "This validates all sensor data"
    (SUBITEMS (SENSOR-VALIDATION-ALL (DATA-INERENCE
      "VALIDATE
      'all))
      (SENSOR-VALIDATION-MANY (DATA-INERENCE
        'VALIDATE
        'many))
      (SENSOR-VALIDATION-ONE (DATA-INERENCE
        'VALIDATE
        'one))
    [LIMIT-CHECKING (DATA-INERENCE 'LIMITCHECK 'all)
      "This checks absolute limit for all sensor data"
      (SUBITEMS (LIMIT-CHECKING-ALL (DATA-INERENCE
        'LIMITCHECK
        'all))
        (LIMIT-CHECKING-MANY (DATA-INERENCE 'LIMITCHECK
          'many))
        (LIMIT-CHECKING-ONE (DATA-INERENCE 'LIMITCHECK
          'one))
    [RATE-OF-CHANGE-CHECKING (DATA-INERENCE 'RATECHECK
      'all)
      "This checks absolute limit for all sensor data"
      (SUBITEMS (RATE-OF-CHANGE-CHECKING-ALL (DATA-INERENCE
        'RATECHECK
        'all))
        (RATE-OF-CHANGE-CHECKING-MANY (DATA-INERENCE
          'RATECHECK
          'many))
        (RATE-OF-CHANGE-CHECKING-ONE (DATA-INERENCE
          'RATECHECK
          'one))
    [NOISE-CHECKING (DATA-INERENCE 'NOISECHECK 'all)
      "This checks for presence of noise for all sensor data"
      (SUBITEMS (NOISE-CHECKING-ALL (DATA-INERENCE
        'NOISECHECK
        'all))
        (NOISE-CHECKING-MANY (DATA-INERENCE 'NOISECHECK
          'many))
        (NOISE-CHECKING-ONE (DATA-INERENCE 'NOISECHECK
          'one))
    [RELIABILITY-CHECKING (DATA-INERENCE 'RELIABILITYCHECK
      'all)
      "This checks reliability history for all sensor data"
      (SUBITEMS (RELIABILITY-CHECKING-ALL (DATA-INERENCE
        'RELIABILITYCHECK
        'all))
        (RELIABILITY-CHECKING-MANY (DATA-INERENCE
          'RELIABILITYCHECK
          'many))
        (RELIABILITY-CHECKING-ONE (DATA-INERENCE
          'RELIABILITYCHECK
          'one))
    TITLE "VALIDATION FUNCTIONS"
    CENTERFLAG T
    MENUROWS 4
    MENUFONT _BOLDFONT
(ATTACHWINDOW VALRESULTWIN VALIDWIN BOTTOM)
(ATTACHWINDOW MENUP VALIDWIN TOP)
(INVALIDITY)
(INHYPOTHESIS)
(PRINTOUT VALIDWIN * >> Sensor Validation & Hypothesis Evaluation Module ready <<
>> Please proceed with diagnosis <<*)

(VALUE-ASKUSER
[LAMBDA (classname objectname) (* ; "Edited 9-Sep-89 13:50 by *

("function to ask user for input when the validation cannot be completed")
(PROG ((numvalue (GetValue ($1 classname)
'numvalue))
answer string1 string2 input)
(OPENW VALUSERWIN)
(FLASHWINDOW VALUSERWIN)
(SETQ answer (PromptRead (CONCAT " [ Do you want to Input new value for sensor "
'classname
"] ? ]
) VALUSERWIN NIL))
(SETQ answer (SELECTQ answer
((Y Y yes Yes YES)
T)
((N N no No NO)
NIL)
answer))

(if (EQUAL answer 'NIL)
then (PRINTOUT VALUSERWIN (CONCAT " [ Value of "
'classname
"] not changed ]) )
else (SETQ string1 (CONCAT " The current entry for "
'classname
"> >> is "
numvalue
"] 
"question)

[SETQ string2 (CONCAT string1 " Please give the new value (q for no change) : ")
(GetValue ($1 classname)
'numvalue
'doc)
(SETQ input (PromptRead string2 VALUSERWIN NIL))
(if (NOT (EQUAL input 'q))
then (SETQ numvalue input)
(PutValue ($1 classname)
'numvalue numvalue)
(PutValue ($1 objectname)
'status
'End)
(RETURN numvalue))

(VALUE-COMPARE
[LAMBDA (classname objectname) (* ; "Edited 24-Jul-89 14:26 by shum-s")

("function to obtain an alternate value to a sensor reading so a comparison can be made")
(LET ((equation (GetValue ($1 objectname)
'function))
numvalue)
(if (LISTP equation)
then (SETQ numvalue (VALUE-COMPARE-LIST classname objectname equation))
else (SETQ numvalue (VALUE-COMPARE-ONE classname objectname equation))

(VALUE-COMPARE-LIST
[LAMBDA (classname objectname equation) (* ; "Edited 9-Sep-89 12:11 by *

("function called by VALUE-COMPARE when more than one calculation is possible")
(PROG ((num (LENGTH equation))
(oldvalue (GetValue ($1 classname)
'numvalue))
(flag 1)
(flag 2 1)
tol method band numvalue diff diff2 value1 value2 templist sensorlist second)
resetoop
(SETQ tol (GetValue ($1 objectname)
numvalue))
numvalue
'tol
'diff
'diff2
'value1
'value2
'templist
'sensorlist
'second

VALUE-COMPARE-ONE
(LAMBDA (classname objectname equation) (* ; "Edited 9-Sep-89 12:11 by *

("function called by VALUE-COMPARE when only one calculation is possible")
(PROG ((num (LENGTH equation))
(oldvalue (GetValue ($1 classname)
'numvalue))
(flag 1)
(flag 2 1)
tol method band numvalue diff diff2 value1 value2 templist sensorlist second)
resetoop
(SETQ tol (GetValue ($1 objectname)
numvalue))
numvalue
'tol
'diff
'diff2
'value1
'value2
'templist
'sensorlist
'second

resetoop

loop
(SETQ method (CAR equation))
(SETQ equation (ODR equation))
(SETQ band (CAR toi))
(SETQ toi (CDR tol))
(SETQ numvalue (APPLY* method clasname))
(if (NUMBERP numvalue)
then (SETQ diff (ABS (DIFFERENCE oldvalue numvalue)))
else (SETQ diff 10000))
(if (EQ flag 1)
then (if (GREATERP diff band)
then (SETQ value1 numvalue)
(SETQ flag (PLUS flag 1))
(GO loop)
else (PRINTOUT VALRESULTWIN (CONCAT 'Sensor ' clasname
'measurement is validated '))
(PutValue ($1 objectname)
'status 'End)
(PutValue ($1 objectname)
'validity 'Valid)
(SETQ numvalue oldvalue)
(SETQ templist (GetValue ($1 objectname)
'sensors-used))
(SETQ sensorlist (CAR templist))
(UPDATE-VALIDITY sensorlist))
else
(if (NUMBERP numvalue)
then (from 2 to flag
do ((SETQ tempvalue (EVAL (PACK* 'value (DIFFERENCE flag 1]
(if (NUMBERP tempvalue)
then (SET (PACK* 'diff flag)
(ABS (DIFFERENCE numvalue tempvalue)))
else (SET (PACK* 'diff flag)
10000))
(if (LEO (EVAL (PACK* 'diff flag))
band)
then (PRINTOUT VALRESULTWIN (CONCAT
'Two calculated values for sensor ' clasname
'match each other but not the original sensor reading. The calculated value Is chosen ')
(PutValue ($1 clasname)
'numvalue numvalue)
(INFERNORMALITY clasname)
(PutValue ($1 clasname)
'pastdata 'UNKNOWN)
(PutValue ($1 clasname)
'trend 'UNKNOWN)
(PutValue ($1 objectname)
'status 'End)
(PutValue ($1 objectname)
'validity 'invalid))
(SETQ flag2 'Match]
(if (NOT (EQ flag2 'Match))
then (if (LEO diff band)
then (PRINTOUT VALRESULTWIN (CONCAT 'Sensor ' clasname
'measurement is validated '))
(PutValue ($1 objectname)
'status 'End)
(PutValue ($1 objectname)
'validity 'Valid)
(SETQ numvalue oldvalue)
(SETQ templist (GetValue ($1 objectname)
'sensors-used))
(SETQ sensorlist (CAR templist))
(if (EQ flag 2)
then (SETQ goodlist (CAR (CDR templist))))
else (from 3 to flag
do ((SETQ templist (CDR templist))
(SETQ sensorlist (CAR templist))
(SETQ templist (CDR templist))
(SETQ sensorlist (UNION second sensorlist))
(SETQ goodlist (CAR templist))
(VALIDATE-SENSORLIST sensorlist)
else (if (LESSP flag num)
then (SETQ (PACK* value flag)
numvalue)
(SETQ flag (PLUS flag 1))
(GO loop)
elself (NOT (EQ flag2 'Done))
then (SETQ templist (GetValue ($1 objectname)
'sensors-used))
(SETQ sensorlist (CAR templist))
[from 2 to flag
do ((SETQ templist (CDR templist))
(SETQ sensorlist (CAR templist))
(SETQ templist (CDR templist))
(SETQ sensorlist (UNION second sensorlist))
(VALIDATE-SENSORLIST sensorlist)
(SETQ equation (GetValue ($1 objectname)
'function))
(SETQ flag2 'Done)
(GO resetloop)
elexe (PRINTOUT VALRESULTWIN
(CONCAT
" [ The conflict between calculated and measured values of sensor "
classname
* cannot be resolved ] ")
(PutValue ($1 objectname)
'status
'Conflict)
(SETQ rel (GetValue ($1 objectname)
'reliability))
(if (LESSP rel 6)
then (PRINTOUT VALRESULTWIN
" [ Since sensor reliability is low, UNKNOWN is used as final value ] ")
(PutValue ($1 objectname)
'numvalue
'UNKNOWN)
(PutValue ($1 objectname)
'value
'UNKNOWN)
(PutValue ($1 objectname)
'trend
'UNKNOWN)
(PutValue ($1 objectname)
'pastdata
'UNKNOWN)
else (PRINTOUT VALRESULTWIN
" [ Since sensor reliability is high, the original measured value is used ] ")
)
(SETQ numvalue (VALUE-ASKUSER classname
different name)
(RETURN numvalue))
(VALUE-COMPARE-ONE
(LAMBDA (classname objectname equation) (* ; "Edited 9-Sep-89 12:07 by ")
"function called by VALUE-COMPARE when only one calculation is possible"
(PRQ ((toi (GetValue ($1 objectname) 'tol' tolerance))
(sensorlist (GetValue ($1 objectname) 'sensors-using))
(oldvalue (GetValue ($1 classnamename) 'numvalue))
(flag 'Start)
(numvalue diff calculation sensors rel)
loop
(SETQ numvalue (APPLY* equation classnamename))

"classname is optional, depending on the function"
(if (NUMBERP numvalue)
then (SETQ diff (DIFFERENCE oldvalue numvalue))
else (SETQ diff 10000))
(if (GREATERP (ABS diff) toi)
then
(if (NOT (EQ flag 'Done))
then (PRINTOUT VALIDWIN (CONCAT " Calculated value for sensor " classnamename " is not within tolerance band of measurement. "
"Since only one calculation is available, "
"the data used in calculation are being validated ] "
))
isValidSerNsenlist sensorlist)
(SETQ flag 'Done)
(GO loop)
else
(SETQ calculation 'Valid)
(SETQ sensors (GetValue ($ CSTRDataBase) 'sensorsclass))
[for sensorclass in sensorlist
  do (if (EQMEMB sensorclass sensors)
    then (LET [(sensorobject (PACK* sensorclass '-Ob]
    (if (NOT (EQUAL (GetValue ($1 sensorobject) 'status) 'End))
    then (SETQ calculation 'Invalid)
    (if (NOT (EQUAL calculation 'Invalid))
    then (PutValue ($1 objectname) 'status 'End)
    (PutValue ($1 objectname) 'validity 'Invalid)
    (PutValue ($1 classnamename) 'numvalue numvalue)
    (INFERNORMALITY classnamename)
    (PRINTOUT VALRESULTWIN (CONCAT " The calculated value of sensor " classnamename " is chosen as final value ] "")
else (PRINTOUT VALRESULTWIN (CONCAT " [ The conflict between calculated and measured values of sensor "
"classnamename " cannot be resolved ] "")
    (PutValue ($1 objectname) 'status 'Conflict)
    (SETQ rel (GetValue ($1 objectname) 'reliability))
    (if (LESSP rel 6)
      then (PRINTOUT VALRESULTWIN " [ Since sensor reliability is low, UNKNOWN is used as final value ] "
    (PutValue ($1 classnamename) 'numvalue 'UNKNOWN)
    (PutValue ($1 classnamename) 'value 'UNKNOWN)
    (PutValue ($1 classnamename)"
'trend
'UNKNOWN)
(PutValue ($1 classname)
'pastdata
'UNKNOWN)
else (PRINTOUT VALRESULTWIN
  " [ Since sensor reliability is high, the original measured value is used ] ")
)

(SEQ numvalue (VALUE-ASKUSER classname objectname)
else (PRINTOUT VALRESULTWIN (CONCAT " [ Sensor " classname
  " measurement is validated ] " ))

(PutValue ($1 objectname)
'status
'End)
(PutValue ($1 objectname)
'validity
'Valid)
(UPDATE-VALIDITY sensorlist)
(SEQ numvalue oldvalue)
(RETURN numvalue)

(VALUE-REPLACE
  [LAMBDA (classname objectname) (* ; "Edited 9-Sep-89 12:01 by ")
  (* ; "function to replace a sensor reading which has failed one of the limit
or noise checks. The data used in the calculation are first validated")
  (PROG ((equation (GetValue ($1 objectname)
    'function))
  (sensorist (GetValue ($1 objectname)
    'sensors-used))
  flag method numvalue locheck sensorlist templist backuplist second goodlist)
  (if (LISTP equation)
    then (SEQ flag (LENGTH equation))
    (SEQ method (CDR equation))
    (SEQ equation (CAR equation))
    (SEQ sensorist (CDR sensorist))
    (SEQ sensorist (CAR sensorist))
  else (SEQ flag 1))
  loop
  (VALIDATE-SENSORLIST sensorist)
  (SEQ numvalue (APPLY* equation classname)
    "classname is optional, depending on the function")
  (if (NUMBERP numvalue)
    then (PRINTOUT VALRESULTWIN (CONCAT " [ Alternate value for sensor " classname
      " is obtained and used ] ")
    (PutValue ($1 classname)
      'numvalue numvalue)
    (PutValue ($1 objectname)
      'status
      'End)
  elseif (NOT (EQ flag 1))
    then (SEQ equation (CAR method))
    (SEQ method (CDR method))
    (SEQ sensorist (CDR sensorist))
    (SEQ sensorist (CAR sensorist))
    (SEQ flag (DIFFERENCE flag 1))
    (GO loop)
  else (PRINTOUT VALRESULTWIN (CONCAT " [ Alternate value for sensor " classname
    " cannot be obtained. It is set to UNKNOWN ] " ))
  (PutValue ($1 objectname)
    'status
    'Failed)
  (SEQ numvalue (VALUE-ASKUSER classname objectname)))
  (RETURN numvalue)

(VALUE-TEST
  [LAMBDA (objectname) (* ; "Edited 26-Jul-89 13:10 by shum-s")
  "function to test the control valve malfunction hypothesis by calculation using controller action")
  (LET ((vp (GetValue ($1 objectname)
    'positioner)))

(control (GetValue ($1 objectname) 'action))
(toi (GetValue ($1 objectname) tolerance))
(position maction calculated test)
(SETQ position (DB-FETCH ($1 vp) numvalue))
(SETQ maction (DB-FETCH ($1 control) numvalue))
(if (AND (NUMBERP position)
 (NUMBERP maction))
 then (SETQ calculated (EXPT 50.0 (MINUS maction)))
 if (LEQ (ABS (DIFFERENCE position calculated))
 toi)
 then (SETQ test 'Pass)
 else (SETQ test 'Fall))
 else (SETQ test 'UNKNOWN))

('PUTPROPS:CASTROBJECT.MPLOOPS:MAKEFILE:ENVIRONMENT (READTABLE "INTERLISP"; PACKAGE "INTERLISP")
(DECLARE% DONTCOPY

STOP