

**AUTOMATED FAULT DIAGNOSIS OF CHEMICAL PROCESS PLANTS
USING MODEL-BASED REASONING**

by

FRANCIS ERIC FINCH

B.S. Chemical Engineering
University of California, Davis
(1984)

Submitted to the Department of Chemical Engineering in partial fulfillment of the requirements
for the degree of

DOCTOR OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 1989

© Massachusetts Institute of Technology 1989

Signature of
Author

Department of Chemical Engineering
August 18, 1989

Certified
by

Mark A. Kramer
Associate Professor, Chemical Engineering
Thesis Supervisor

Accepted
by

William M. Deen

Chairman, Departmental Committee on Graduate Students
**MASSACHUSETTS INSTITUTE
OF TECHNOLOGY**

OCT 3 1989

LIBRARIES

AUTOMATED FAULT DIAGNOSIS OF CHEMICAL PROCESS PLANTS USING MODEL-BASED REASONING

by

FRANCIS ERIC FINCH

Submitted to the Department of Chemical Engineering on August 18, 1989 in partial fulfillment of the requirements for the degree of Doctor of Science.

Abstract

This research has focused on the development of knowledge representations and reasoning strategies necessary to identify the root causes of disturbances in chemical and petrochemical plants. These plants are characterized by incomplete instrumentation; non-monotonic dynamic behaviors, and frequent plant modifications, making the diagnosis task resistant to automation.

The basis for diagnostic reasoning is a model of plant behavior. A major portion of the project was devoted to developing modeling formalisms capable of representing a wide variety of plant behavioral knowledge. The formalism devised, called event modeling, is able to integrate knowledge from qualitative causal models and quantitative constraint equation models in a single representation. Event models use transitions between plant states, rather than the states themselves, as the elemental model feature. Events are defined for qualitative changes in plant variables, constraint residuals, variable trends, and equipment status. Events are also defined for off-line test results and operator actions. Modeling plant dynamics, such as compensatory, inverse, and oscillatory response, is a natural outgrowth of this formalism. Event models can represent automatic controller responses, recycle loops, multiple malfunctions (including certain induced failures), sensor failure, off-line tests, transient malfunctions, false alarms, and out-of-order alarms.

A diagnostic reasoning algorithm was developed to interpret event models. The procedural knowledge contained in the algorithm is generic and will apply to any device for which an appropriate event model can be constructed. To diagnose a new plant, only a new event model need be developed. Diagnosis begins when an abnormal event is detected. The algorithm associates related events to produce a diagnosis postulating the minimum number of malfunctions necessary to explain observations. Diagnoses are presented as a list of possible root causes, ranked in order of relative likelihood. Likelihood calculations incorporate both the *a priori* probability of a malfunction and the weight of observed event evidence.

These concepts have been implemented in a prototype diagnostic aid for control room operators, called MIDAS. MIDAS generates an analysis of plant malfunctions by interpreting data collected from on-line sensors. MIDAS can also operate as an effective alarm filter by suppressing alarms caused by downstream propagation of an already diagnosed disturbance. In a case study of 76 malfunctions in a simulated chemical process, MIDAS produced an accurate diagnosis (i.e. included the true root cause in the list of candidates) after 98.9% of event detections, and gave highest rank to the true root cause in 82% of cases.

Thesis Supervisor: Dr. Mark A. Kramer
Associate Professor of Chemical Engineering

Acknowledgement

Far from being a solitary endeavor, this work has had many contributors whom I would like to thank, including:

My advisor, Dr. Mark Kramer, for providing the right mix of advice and enthusiasm, supervision and autonomy;

My thesis committee, Dr. Lawrence Evans and Dr. George Stephanopoulos of the Department of Chemical Engineering and Dr. Thomas Sheridan of the Department of Mechanical Engineering, whose incisive comments kept the work honest and on track;

My friend and colleague, Olayiwola Oyeleye, who provided major contributions to this project, and in whose abilities I have often put my faith and never been disappointed;

The members of my research group past and present -- James Leonard, Philippe Rose, and Bernard Palowitch -- who have proven a reliable sounding board for ideas; and

The members of the MIT Laboratory for Intelligent Systems in Process Engineering (LISPE) and the industrial sponsors who support it for their unbiased input that has helped keep this project firmly grounded in reality.

In a less direct but perhaps more important way, this work was made possible by the love and support of my family, including:

My wife Leslie, who has endured the sacrifices that this work entailed and provided balance to my life while at MIT; and

My parents, for always encouraging me to be curious, self-reliant, and ambitious.

This research was supported financially by the National Science Foundation, Duke Power Company, and Texaco. I am grateful for their support.

Table of Contents

Title Page	1
Abstract	2
Acknowledgement	3
Table of Contents	4
List of Figures	10
List of Tables	13
Notation and Abbreviations	15
D.0 Research Overview	18
D.1 Approaches to Automated Fault Diagnosis	22
D.2 SLD	23
D.3 MIDAS	25
D.3.1 Detection in MIDAS	26
D.3.2 MIDAS Plant Model	27
D.3.2.1 Causal Models	27
D.3.2.2 Constraint Models	28
D.3.2.3 Integrating Causal and Constraint Models	28
D.3.2.4 Representing the Plant Model	29
D.3.3 Diagnostic Reasoning in MIDAS	30
D.3.3.1 Linking Related Events	31
D.3.3.2 The Hypothesis Model	33
D.3.3.3 Event Clusters	33
D.3.3.4 Ranking Fault Candidates	34
D.4 MIDAS Case Study	35
D.5 Opportunities for Future Work	42
D.6 Overview References	43
1.0 Introduction	49

1.1	Problem Statement	52
1.2	Research Scope	54
2.0	Fault Diagnosis	59
2.1	Problem Formulation	59
2.1.1	Definitions	59
2.1.2	Basic Strategies	61
2.1.3	Basic Difficulties in Diagnosis	66
2.2	Human Diagnostic Methodologies	68
2.2.1	Human Diagnostic Reasoning	68
2.2.2	Mental Models	70
2.2.3	Evaluation of Human Diagnostic Reasoning	70
2.2.4	Human Diagnostic Performance and Automated Diagnostic Systems	72
2.3	Automated Detection and Diagnosis Methodologies	72
2.3.1	Parameter Estimation	73
2.3.1.1	Data Reconciliation	74
2.3.1.2	Filtering	75
2.3.2	Pattern Classifiers	75
2.3.3	Neural Networks	76
2.3.4	Control Charts	77
2.3.5	Expert Systems	77
2.3.6	Graph Methods	80
2.3.6.1	Causal Graphs	80
2.3.6.2	Fault Trees	88
2.3.7	Other Methods	88
2.3.8	Comparison of Methods	89
3.0	Process Modeling and Diagnosis in MIDAS	92
3.1	Qualitative Models	92
3.1.1	Extended Signed Directed Graphs (ESDG)	93
3.1.2	Systems-Level Diagnosis (SLD)	96
3.1.2.1	Systems Modeling	97
3.1.2.2	System Performance	99

3.1.2.3	Diagnosis using System Models	102
3.2	Quantitative Models	113
3.3	Event Models	116
3.3.1	Event Graphs	116
3.3.2	Simulation and Diagnosis using Event Graphs	118
3.4	Comparison of Modeling Techniques	126
4.0	Knowledge Representation and Interpretation in MIDAS	134
4.1	MIDAS Structure	134
4.1.1	Monitors	134
4.1.2	Event Interpreter	137
4.1.3	Process Model	139
4.1.4	Hypothesis Model	142
4.1.5	User Interface	143
4.2	Knowledge Representation	144
4.2.1	Definitions	144
4.2.2	Monitor Representation	145
4.2.3	Event Representation	149
4.2.4	Root Cause Representation	152
4.2.5	Link Representation	155
4.2.6	Other Objects	155
4.3	The MIDAS Inference Cycle	160
4.3.1	Event Creation	160
4.3.2	Search and Linkage	160
4.3.3	Evaluating Source Events	165
4.3.4	Evaluating Root Cause Evidence	166
4.3.5	Inference Cycle Examples	168
4.4	Failure Scenarios Interpreted by MIDAS	177
4.4.1	Sensor Failures	177
4.4.2	Dynamics	178
4.4.3	Variations in event Detection Order	180
4.4.4	Multiple Malfunctions	181

5.0	MIDAS User Guide	182
5.1	General Information	182
5.2	Installation	183
5.3	Diagnostics	185
5.3.1	MIDAS10D	196
5.3.2	Running a Diagnostic Session	203
5.3.2.1	LOGIN	204
5.3.2.2	ACTION SCREEN	204
5.3.2.3	EXIT SCREEN	210
5.3.2.4	Commands	210
5.4	Creating Data Files for MIDAS	222
5.5	Monitors	224
5.5.1	Learning Monitor Thresholds	225
5.5.2	Monitor Test Criteria	226
5.5.3	Drift Compensation	228
5.6	Hypothesis Likelihood Formulas	229
5.6.1	Relative Likelihood	229
5.6.2	Normalized Relative Likelihood	230
5.6.3	Probabilistic Likelihood	231
5.6.4	Normalized Probabilistic Likelihood	232
5.6.5	Conditional Probability	232
5.6.6	Normalized Conditional Probability	233
6.0	MIDAS Case Study	234
6.1.	Case Study Information	234
6.1.1	Process	234
6.1.2	Process Model	235
6.1.3	Simulation	241
6.1.4	Simulated Faults	249
6.2	Case Study Results	254
6.2.1	Event Distribution	255
6.2.2	Evaluation Times	256
6.2.3	Tier Performance	258

6.2.4	Resolution	261
6.2.5	Diagnosis Examples	263
6.3	Case Study Analysis	271
6.3.1	On-Time Performance	271
6.3.2	Diagnostic Performance	272
7.0	Ideas and Discussion	281
7.1	Scaling MIDAS to Real Plants	281
7.2	Implementation Alternatives	283
7.3	Evidence Evaluation	284
7.4	Augmented Causal Models	285
7.4.1	Full Event Model Paradigm	285
7.4.2	Semi-Quantitative Causal Models	288
7.4.3	Induced Failure Models	291
7.5	Alternate Reasoning Strategies	292
7.5.1	AND Worlds and OR Worlds	292
7.5.2	Breaking Causal Loops	294
7.6	Improved Detection	298
7.7	Explanations and Advice	299
8.0	References	300
A1.	APPENDIX: Process Model Examples	308
A1.1	POTENTIAL-EVENT Object	308
A1.2	COMPILED-LINK Object	309
A1.3	MEASURED-OBJECT Object	310
A1.4	POTENTIAL-ROOT-CAUSE Object	311
A1.5	TEST Object	312
A1.6	DATA-MONITOR Object	312
A1.7	User Interface Objects	314

A2.	APPENDIX: Jacketed CSTR Simulation	319
A3.	APPENDIX: SLD Example	340
A3.1	Functional Decomposition	340
A3.2	Process Graph	343
A3.3	Diagnosis Example	345
A3.4	Analysis	348

List of Figures

Figure D-1:	MIDAS Components	25
Figure D-2:	Diagnosis Intersection Effect	30
Figure D-3:	Diagnostic Inference Procedure	32
Figure D-4:	Linking Out-of-Order Events	32
Figure D-5:	Jacketed CSTR Process	36
Figure D-6:	Distribution of Resolution	38
Figure D-7:	Evaluation Time Distribution	40
Figure D-8:	Apparent Inverse Response	41
Figure 1-1:	Cause-based Control Strategy	51
Figure 1-2:	Symptom-based Control Strategy	51
Figure 2-1:	Hypothesize and Test Strategies	63
Figure 2-2:	Linear Process for Example 2-1	65
Figure 2-3A:	Shewhart Control Chart for Mean	79
Figure 2-3B:	Shewhart Control Chart for Range	79
Figure 2-4:	Single Stage Fractionator	84
Figure 2-5:	Digraph for Single Stage Fractionator	85
Figure 2-6:	Simulation of High Feed Flowrate	86
Figure 2-7:	Diagnosis of P High	87
Figure 3-1:	Tank Schematic	94
Figure 3-2A:	Tank SDG	95
Figure 3-2B:	Tank ESDG	96
Figure 3-3:	Hierarchical Diagnosis Strategy	97
Figure 3-4:	Control System Response	102
Figure 3-5:	Tank Event Graph	118
Figure 3-6:	Tank Dynamic Event Graph	122
Figure 3-7A:	Inverse Response	123
Figure 3-7B:	Event Model of Inverse Response	123
Figure 3-8:	Alternative Dynamic Event Graph for Tank	125
Figure 3-9:	CSTR with External Recycle	133
Figure 4-1:	MIDAS Global Structure	135

Figure 4-2:	Event Interpreter Cycle	138
Figure 4-3:	Process Model for Tank Process	141
Figure 4-4A:	Case 2 Event	162
Figure 4-4B:	Case 3 Event	163
Figure 4-4C:	Case 4 Event (Single Cluster)	164
Figure 4-4D:	Case 4 Event (Dual Cluster)	164
Figure 4-5:	Examples of Causal Loops	165
Figure 4-6:	Multiple Primary Symptoms	166
Figure 4-7:	Abstract Process Model	170
Figure 4-8:	Evolving Hypothesis Model for Inference Example 1	171
Figure 4-9:	Evolving Hypothesis Model for Inference Example 2	173
Figure 4-10:	Evolving Hypothesis Model for Inference Example 3	175
Figure 4-11:	Evolving Hypothesis Model for Inference Example 4	177
Figure 5-1:	MIDAS Structure	184
Figure 5-2:	MIDAS WELCOME Screen	206
Figure 5-3:	MIDAS LOGIN Page	207
Figure 5-4:	MIDAS ACTION Screen	208
Figure 5-5:	MIDAS EXIT Screen	209
Figure 6-1:	Jacketed CSTR Process	237
Figure 6-2:	Node Connectivity Distribution	241
Figure 6-3:	Distribution of Events	256
Figure 6-4:	Evaluation Time Distribution	257
Figure 6-5:	Time Elapsed per Event	258
Figure 6-6:	Tier of True Fault	260
Figure 6-7:	Average Normalized Rank of True Fault	261
Figure 6-8:	Distribution of Resolution	262
Figure 6-9:	Average Resolution	262
Figure 6-10:	Causal Model of Temperature Control	266
Figure 6-11:	Tier Formation	274
Figure 6-12:	Compensatory Response	277
Figure 6-13:	Apparent Inverse Response	278
Figure 6-14:	Apparent Oscillatory Response	278
Figure 6-15:	Temperature Control Causal Loop	280

Figure 7-1A: Normal System Trajectory	287
Figure 7-1B: Disturbed System Trajectories	289
Figure 7-2: Derivation of Waiting Period for a WCL	290
Figure 7-3: Induced Failure Link Example	292
Figure 7-4: AND World Diagnosis Example	294
Figure 7-5: Causal Loop	295
Figure 7-6A: Detection Range of PE-3	297
Figure 7-6B: Detection Range of PE-1	297
Figure A2-1: Serial and Parallel Flow Paths	319
Figure A2-2: Model Diagram for Jacketed CSTR Process	322
Figure A3-1: Process Graph for Jacketed CSTR Process	345
Figure A3-2: Malfunctional Subgraphs for Diagnosis of Jacket Leak	350

List of Tables

Table D-1: Overall Diagnostic Performance of MIDAS	39
Table D-2: Randomly Generated Faults for Case Study	45
 Table 2-1: Comparison of Fault Diagnosis Techniques	 91
 Table 3-1: System Interaction Classifications	 105
Table 3-2: Pair-Wise Single Fault Diagnosis of Passive Systems	106
Table 3-3A: Pair-Wise Single Fault Diagnosis of Passive and Control Systems (System 1 Passive, System 2 Control)	107
Table 3-3B: Pair-Wise Single Fault Diagnosis of Passive and Control Systems (System 1 Control, System 2 Passive)	108
Table 3-4: Pair-Wise Single Fault Diagnosis of Two Control Systems	109
Table 3-5A: Passive System Failure Modes used to Derive Pair-Wise Diagnostic Rules	114
Table 3-5B: Control System Failure Modes used to Derive Pair-Wise Diagnostic Rules	115
Table 3-6: Comparison of Modeling Techniques	128
Table 3-7A: DIEX and SLD Results for Reactor Temperature Sensor Bias	131
Table 3-7B: DIEX and SLD Results for Gradual Rise in Activation Energy	131
Table 3-7C: DIEX and SLD Results for CV-2 Stuck Closed	132
 Table 4-1: Process Model Objects	 140
Table 4-2: Hypothesis Model Objects	143
Table 4-3: DATA-MONITOR Slots	147
Table 4-4: POTENTIAL-EVENT Slots	150
Table 4-5: RECORDED-EVENT, EXPECTED-EVENT, and LATENT-EVENT Slots	151
Table 4-6: POTENTIAL-ROOT-CAUSE Slots	153
Table 4-7: HYPOTHESIZED-ROOT-CAUSE Slots	154
Table 4-8: COMPILED-LINK Slots	156
Table 4-9: TEST Slots	157
Table 4-10: MEASURED-OBJECT Slots	158
Table 4-11: INFERRRED-MALFUNCTION Slots	159
Table 4-12: Default POAD and POID Values	168
Table 4-13: Inference Example 1	171

Table 4-14: Inference Example 2	172
Table 4-15: Inference Example 3	174
Table 4-16: Inference Example 4	176
Table 5-1: Diagnostic Component Files	185
Table 5-2: MIDAS Diagnostic Programs	186
Table 6-1: Unit Failure Modes for Jacketed CSTR Process	243
Table 6-2: Simulated Failure Mode Effects on Jacketed CSTR Process	247
Table 6-3: Randomly Generated Faults for Case Study	250
Table 6-4: Overall Diagnostic Performance of MIDAS	259
Table 6-5: Diagnosis of Jacket Leak to Reactor	263
Table 6-6: Diagnosis of Feed Flowrate Transient	267
Table 6-7: Diagnosis of Induced Sensor Failure	268
Table 7-1: Bounding of Evidence Evaluation	285
Table A2-1: Values used for Case Study Fault Simulations	320
Table A3-1: Jacketed CSTR Process System Definitions	340
Table A3-2: Jacketed CSTR Process System Components	341
Table A3-3: Jacketed CSTR Process Malfunction Modes	344
Table A3-4: SLD Diagnosis Example of Jacket Leak to Reactor	346

Notation

Roman Symbols:

A	- area	G_1	- input transfer function matrix
A	- composite input transfer function matrix	G_2	- process transfer function matrix
A_1	- linear constraint matrix	G_f	- malfunction transfer function matrix
A_2	- linear constraint matrix	G_s	- sensor transfer function matrix
B	- composite malfunction transfer function matrix	H	- hypothesized malfunction vector
c	- average process model connectivity	I	- measured process input vector
C	- controlled variable measurement	I_s	- measured system input vector
C	- vector of controlled variable measurements	L	- liquid level
C_A	- concentration of species A	L_s	- level sensor signal
C_B	- concentration of species B	m	- size of malfunction set
C_C	- concentration of species C	M	- manipulated variable measurement
C_f	- flow controller output signal	M	- malfunction vector
C_l	- level controller output signal	M	- vector of manipulated variable measurements
C_p	- pressure controller output signal	M_e	- expected manipulated variable value
C_{sp}	- controlled variable setpoint	M_c	- vector of expected manipulated variable values
C_{sp}	- vector of controlled variable setpoints	M_{ss}	- vector of nominal steady-state manipulated variable values
f^c	- controller reference frame function	n	- number of measurements
f^d	- dynamic reference frame function	N	- number of events
f^m	- malfunction size function	p	- plausibility
f^s	- system reference frame function	P	- pressure
f^σ	- variance function	P_{eq}	- equilibrium pressure
F_{cw}	- cooling water flowrate	P_s	- pressure sensor signal
$F_{diagnosis}$	- number of faults in diagnosis	Q	- heat flux
F_f	- feed flowrate	r	- flow path resistance
F_h	- heavy ends flowrate	R	- constraint residual
F_{in}	- inlet flow (inflow)	R	- flow path resistance
F_l	- light ends flowrate	R	- ideal gas constant
F_{out}	- outlet flow (outflow)	s	- number of source event
F_s	- steam flowrate	s	- supportability
F_{total}	- number of possible faults	S_k	- system k
g	- gravitational acceleration	S_m	- system m
		t	- time
		t_{eval}	- event evaluation time

link	- link evaluation time	\mathbf{X}	- vector of state variable values
T_s	- temperature sensor signal or steam tube temperature	X_f	- feed mol fraction
\mathbf{u}_1	- vector of measured values	X_h	- heavy end mol fraction
\mathbf{u}_2	- vector of unmeasured determinable values	X_l	- light ends mol fraction
U	- heat transfer coefficient	\mathbf{X}^{ss}	- vector of nominal steady-state variable values
U_{ij}	- the jth unit function of unit i	\mathbf{Y}_m	- expected process output vector (model generated)
V	- vaporization rate	\mathbf{Y}_s	- measured process output vector
\mathbf{x}	- measurement vector		
\mathbf{X}	- liquid mol fraction		
\mathbf{X}	- process input vector		

Greek Symbols:

α	- probability of accurate detection	Φ	- performance parameter
α	- reaction rate preexponential factor	κ	- average length of causal path
β	- probability of false detection	μ	- sample mean
β	- reaction rate activation energy	μ_{ss}	- nominal steady-state sample mean
δ_c	- controlled variable deviation parameter	θ_c	- controlled variable state threshold
δ_m	- manipulated variable deviation parameter	θ_m	- manipulated variable state threshold
ΔP_{pump}	- pump differential pressure	ρ	- liquid density
ΔT_{In}	- log-mean temperature difference	σ	- sample variance
ϵ	- difference vector between expected and measured outputs	τ	- trajectory time constant
		ω	- vector of random measurement errors
		Ψ	- constraint equation function

Superscripts:

$'$	- modified value
$*$	- derived from model
-1	- matrix inverse

Abbreviations

AE	- All Events	MORT	- Management Oversight and Risk Tree
AI	- Artificial Intelligence	MV	- Manipulated Variable
ANN	- Artificial Neural Network	NPP	- Normalized Prior Probability
ATM	- Assumption Truth Maintenance	NR	- Normalized Rank
CCL	- Can Cause Link	NRL	- Normalized Relative Likelihood
CL	- Compiled Link	OE	- Opposing Events
CNT	- CoNTrol unit functions set	OOP	- Object-Oriented Programming
CP	- Conditional Probability	PE	- Potential Event
CRF	- Controller Reference Frame	PI	- Proportional-Integral (control)
CS	- Concentration Sensor	PL	- Probabilistic Likelihood
CSEN	- Controlled variable SENsor unit functions set	PM	- Process (plant) Model
CSTR	- Continuous Stirred Tank Reactor	POAD	- Probability of Accurate Detection
CV	- Controlled Variable	POID	- Probability of Inaccurate Detection
CW	- Cooling Water	PP	- Prior Probability
CWCS	- Cooling Water Controller Signal	PRC	- Potential Root Cause
CWF	- Cooling Water Flowrate	PRC	- PRoCess unit functions set
CWSP	- Cooling Water Setpoint	PS	- Pressure Sensor
DFR	- Dynamic Reference Frame	PSL	- Precursor/Successor Link
DIEX	- Diagnostic EXpert	RAM	- Random Access Memory
EDDIE	- Event Driven Diagnostic Inference Engine	RE	- Recorded Event
EE	- Expected Event	RF	- Reference Frame
EI	- Event Interpreter	RL	- Relative Likelihood
ESDG	- Extended Signed Directed Graph (digraph)	RT	- Reactor Temperature
FC	- Flow Controller	SCP	- Simple Causal Path
FMEA	- Failure Modes and Effects Analysis	SDG	- Signed Directed Graph (digraph)
FS	- Flow Sensor	SE	- Supporting Events
HAZOP	- HAZard and OPerability Study	SEN	- SENsor unit functions set
HM	- Hypothesis Model	SFA	- Single Fault Assumption
HRC	- Hypothesized Root Cause	SFE	- Single Fault Explanation
IFL	- Induced Failure Link	SISO	- Single Input Single Output
IHG	- Intelligent Hypothesis Generator	SLD	- Systems-Level Diagnosis
IM	- Inferred Malfunction	SRF	- System Reference Frame
LC	- Level Controller	SSRF	- Steady-State Reference Frame
LCL	- Local Cause Link	TC	- Temperature Controller
LE	- Latent Event	TS	- Temperature Sensor
LS	- Level Sensor	WCL	- Will Cause link
MIDAS	- Model-Integrated Diagnostic Analysis System		
MIMO	- Multiple Input Multiple Output		

D.0. Research Overview

The research described has explored techniques for automated fault diagnosis of industrial plants. Investigation has focused on continuous, steady-state processes in chemical plants and petroleum refineries, but the fundamental concepts apply to a broad range of complex systems and devices. The goal has been to find solutions to the diagnosis problem for an entire class of processes, rather than diagnose faults in any specific plant.

Fault diagnosis is defined as the problem of isolating and identifying the root causes of plant disturbances from knowledge of observable symptoms¹. Traditionally, this task has been within the realm of supervisory control and has been performed by human operators. Symptoms used for diagnosis can include visual inspection, smell, and sound, but remote sensors are the major source of information available to operators. Sensor signals are concentrated in a control room where values are displayed to operators through charts, gauges, or CRT displays. When a value deviates from the acceptable range, an alarm may alert the operator, but in general, operators are wholly responsible for recognizing and diagnosing symptoms. At present, human operators are the best diagnosticians available, but advances in computer technology, particularly in the area of artificial intelligence, are making computer-based diagnostic problem solvers a reality.

Effective fault diagnosis is crucial for plant economy and safety. Automated fault diagnosis has the potential to make fault diagnosis faster, more efficient, and more reliable than is possible with human operators alone. Automated systems benefit from large, infallible memories; consistent, verifiable diagnostic reasoning algorithms; and immunity to stress, boredom, or fatigue. Human diagnosticians have superior communication abilities, are extremely flexible problem solvers, and can learn from experience. The optimal mix of diagnostic skills will include both automated and human diagnosis. Therefore, this work is not aimed at replacing human operators, but seeks to compliment their skills by providing a competent diagnostic aid or advisor.

Fault diagnosis of large scale processes is a difficult problem that has resisted automation. Consequently, most existing diagnostic systems are plant specific and cannot be easily

¹ Unlike some authors, the terms "fault", "root cause", and "malfunction" are used interchangeably in this work to denote a failure or performance degradation of process equipment.

adapted to new processes. The difficulty of the problem derives from one fundamental question that all diagnostic systems must resolve: how to represent all the possible symptom combinations that may arise due to a fault. Simple representations, such as fault/symptom tables, are usually impractical for the following reasons:

- Extensive fault simulation must be performed to determine all possible fault symptoms,
- Complex process dynamics make symptoms time dependent and not easily predicted,
- Large numbers of potential symptoms can make table derivation and representation difficult on a plant-wide scale,
- Diagnostic information contained in symptom order may be lost during table compilation,
- The frequency of process modification and changeover common in chemical and petrochemical plants make plant specific tables prohibitively expensive, and
- The resulting diagnosis gives the operator little explanation of how it was derived or how to evaluate its quality.

The approach to diagnosis taken by this research seeks to solve a more general diagnosis problem. The broad goals of this project are to explore alternative knowledge representations and diagnostic reasoning procedures suitable for diagnosis of processes involving flow, pumping, mixing, heating, or cooling of pure or multi-component fluids with or without chemical reaction. Eight (8) specific project goals are listed below:

⇒ *Use existing process instrumentation to detect all symptoms.*

The set of observable symptoms should be determined by the availability of on-line sensors, rather than dictated by the requirements or limitations of the diagnostic procedure. Installation of the diagnostic system should not mandate additional sensors.

⇒ *Always produce an accurate diagnosis (defined as a diagnosis that includes the true fault in the set of fault candidates) with the highest fault resolution possible within the limitations of available instrumentation.*

An inaccurate diagnosis is worse than no diagnosis at all, because it will draw an operator's attention away from the true fault. Consequently, accuracy must be given highest priority. A diagnosis with poor resolution (i.e. a large set of possible fault

candidates) is roughly equivalent to no diagnosis at all, because it does not focus the operator's attention toward the true fault.

- ⇒ *Create a diagnosis that is useful to the operator.*

A useful diagnosis will provide the operator with information on how the diagnosis was derived, which fault candidates should receive the highest priority, and what tests can be performed to verify the diagnosis.

- ⇒ *Maximize utility and portability by minimizing the amount of information that must be supplied to diagnose a new process or modified process.*

The key to achieving this goal lies in developing general knowledge representations and generic reasoning procedures applicable to many processes. Generic knowledge can be used in multiple applications.

- ⇒ *Develop a framework for diagnosing multiple faults, specifically induced sensor failures and multiple independent failures.*

Many diagnostic systems rely on a single fault assumption (SFA) as part of their basic reasoning algorithm. Yet multiple malfunctions may occur concurrently in different sections of a large plant, and existing malfunctions can induce failure of additional equipment. Systems dependent on SFA logic may produce inaccurate diagnoses when multiple faults are present.

- ⇒ *Make the diagnosis robust to process dynamics, such as the compensating action of control systems and other non-monotonic behavior.*

A diagnosis should remain stable when a controlled variable returns to normal as a result of control action. This goal can only be achieved if both the initial and ultimate fault symptoms are included in the model of process behavior used for diagnosis. Ideally, a diagnostic system should also recognize transient disturbances and their dynamics.

- ⇒ *Develop a flexible modeling framework capable of representing different types of process knowledge, including quantitative equation models and qualitative causal models.*

All modeling formats have individual strengths and limitations which will be mirrored in any diagnostic system overly dependent on a single format. Rather than develop a detailed model in a single format, it is often preferable to develop a less detailed mixed model.

- ⇒ *Produce a system that uses symptom order in diagnostic reasoning, but is robust to false symptom detections or symptoms that occur "out-of-order".*

Useful information can be contained in the order of symptom detection. However, uncertainties in detection schemes can result in variations in observed symptom order. A diagnostic reasoning procedure that relies too heavily on detection order will be vulnerable to out-of-order symptoms.

This goal agenda is ambitious, yet all these goals have been achieved, to varying degrees, by adapting techniques developed by previous researchers and applying the following three (3) innovative ideas:

- ⇒ *Model potential process behaviors using qualitative events rather than qualitative states.*

Events are transitions that take place at a point in time, not persistent states. Events can be changes in the qualitative state of variables or constraint equations residuals, changes in qualitative trends, operator actions, or the results of off-line tests. The added flexibility of using events as diagnostic symptoms allows a natural integration of qualitative and quantitative models. An additional benefit is the ability to model dynamics using causal links to relate events acting on a single variable or constraint.

- ⇒ *Separate potential events and potential root causes from observed events and hypothesized root causes in the knowledge representation.*

The separation of the plant model from the "hypothesis model" allows diagnosis of selected multiple malfunctions and plays an important role in making the system robust to variations in event detection order.

- ⇒ *Use observed events to rank fault candidates by relative likelihood.*

Ranking fault candidates in order of relative priority increases diagnostic resolution and provides the operator with vital information on diagnosis quality.

These innovations have been incorporated in the Model-Integrated Diagnostic Analysis System (MIDAS), a computer-based diagnosis problem solver implemented in Common LISP on a PC compatible 386 computer. Tests of MIDAS on simulated faults have produced diagnoses with 98.9% accuracy. Resolution at or near the theoretical maximum imposed by on-line instrumentation has been achieved in 90% of test cases. Of cases tested, 40% exhibited control system compensation, 20% exhibited inverse response, and over 30% included out-of-order event detections. MIDAS has proven capable of diagnosing multiple independent malfunctions, induced sensor failures, false event detections, and certain process transients.

In the following paragraphs, several approaches to automated fault diagnosis are briefly discussed before the techniques developed in this thesis are described in more detail. First, a methodology developed to diagnose control system behaviors is described. This work provided the insights necessary for creation of MIDAS, which is discussed next. Discussion focuses on how the knowledge representation and diagnostic reasoning algorithm implemented in MIDAS achieve the previously stated goals. MIDAS test results are presented followed by recommendations for future research.

D.1. Approaches to Automated Fault Diagnosis

A variety of approaches to automated fault diagnosis have been explored by various researchers. No single method has proven universally superior. Himmelblau (1978) reviews techniques using pattern recognition, parameter estimation, and causal models.

Approaches can be broadly classified as *model-based* or *experience-based*. A model-based approach will incorporate a structured model of plant behavior based on fundamental engineering principles. Models can be quantitative, expressed in the form of numerical equations, or qualitative, expressed in the form of logical relationships. Experience-based approaches use examples or heuristics to represent possible plant behavior. Examples can be drawn from an archive of past fault episodes or generated by numerical simulation.

Model-based diagnostic systems have the ability to diagnose novel faults² and can be adapted to process modifications by selective alteration of the underlying model. An additional benefit is the ability to explain the logic behind diagnostic conclusions by referencing the fundamental principles in the model. The drawback of the model-based approach is the significant effort that must go into model development. Quantitative models often require a detailed model and extensive instrumentation. The Kalman filter and other parameter estimation techniques reviewed by Iscrmann (1984) use quantitative models. Qualitative models require less effort to develop but may contain ambiguities. Digraphs, discussed by Iri et al. (1976), and fault trees, discussed by Dhillon and Rayapat (1988), are examples of qualitative models that have been used for fault diagnosis.

Experience-based diagnostic systems require no modeling effort but are highly process specific and difficult to verify. Formulation of meaningful explanations is difficult because these systems are not based on fundamental principles. Diagnosis of novel malfunctions is impossible unless examples are produced using numerical simulation. Expert systems (Buchanan and Shortliffe, 1984) and neural networks (Cowan and Sharp, 1988) can be used to express diagnostic experience.

The work presented in this thesis has dealt exclusively with model-based diagnosis techniques. Although both qualitative and quantitative models are used, the plant models created have been based predominantly on qualitative causal models, supplemented by selected constraint equations. Existing methods based on causal models are predicated on the SFA and on snapshots³ of data. Symptom order information and diagnostic stability are sacrificed when snapshots are used for diagnosis, because snapshots have no memory of past states. The DIEX system developed by Palowitch (1987), in an attempt to move beyond snapshots, made extensive use of symptom detection order to improve resolution. The rigidity of the algorithm, however, made DIEX vulnerable to out-of-order symptoms.

D.2. SLD

In an attempt to develop a hierarchical integration of diagnosis methodologies using several levels of model abstraction, a methodology called Systems-Level Diagnosis (SLD) was

² A novel fault is a fault for which there is no past experience.

³ A snapshot contains only the most recent states of all measured variables.

developed to capture the behavior of control systems and use this knowledge in diagnosis. This model-based approach uses a functional process decomposition combined with causal information on control system dependencies to predict possible responses to different fault classes.

SLD captures the goal-directed behavior of control systems directly in models that are an order-of-magnitude less complex than digraph models of the same process. SLD relates *unit functions*, the tasks a unit performs, to control systems that rely on unit functions for proper operation. If a system is behaving abnormally, one or more of its unit functions have failed. Unit function failures are traced back to faults effecting unit operation. At a separate level, interactions between control systems are represented in a causal graph, similar to a digraph. This system graph is much less complex than a digraph, because most processes have far fewer systems than variables.

System performance is evaluated using qualitative system states. Three states represent system behavior when malfunctions are present: STRESSED, SATURATED, and UNCONTROLLED. A STRESSED system is adequately compensating a disturbance but is not operating at an optimal level. A system becomes SATURATED when a disturbance is outside the system's range of controllability, and a system with a failed controller or control valve is UNCONTROLLED. Diagnostic rules were developed to interpret system states. These rules were based on the ultimate steady-state response of system and are premised on the SFA and snapshots of system states.

In a case study presented by Kramer and Finch (1989), SLD produced diagnoses of approximately the same accuracy and resolution as the DIEX system utilizing digraphs. Moreover, the diagnoses were almost disjoint, meaning a substantially improved combined diagnosis could be derived by intersecting SLD and DIEX diagnoses. The fact that different models will produce diagnoses with distinctive characteristics and a substantially improved final diagnosis could be obtained if models were integrated was the impetus for subsequent development of MIDAS. At the current time, SLD models have not been fully integrated into MIDAS. Therefore, SLD stands as a separate approach to diagnosis. There is no conceptual barrier to integration of SLD models into the MIDAS event format, it is simply beyond the scope of a single thesis project and has been left to future research.

D.3. MIDAS

The culmination of this project is the Model-Integrated Diagnostic Analysis System (MIDAS). MIDAS is a model-based, general diagnostic system employing partially compiled plant models. MIDAS is implemented in Common LISP on a PC compatible computer using GoldWorksTM⁴ software.

MIDAS is comprised of three (3) basic components: a detection scheme, a plant model, and a diagnostic reasoning procedure. The relationship between components is illustrated in figure D-1. Each component is discussed below.

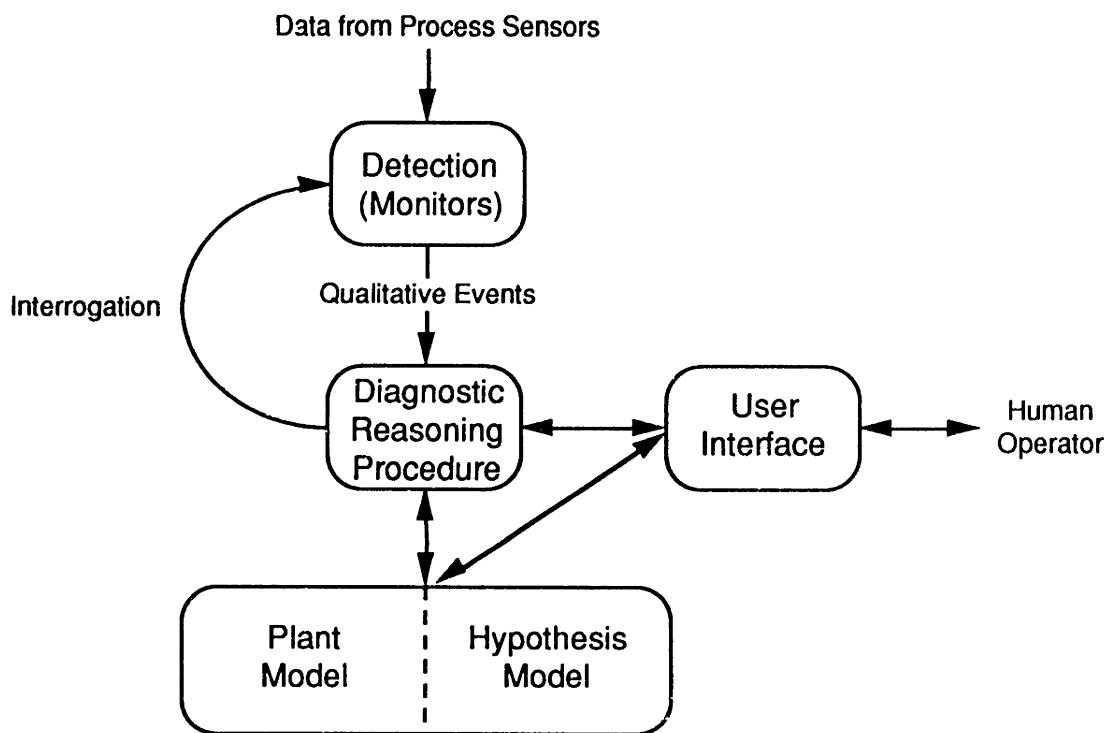


Figure D-1: MIDAS Components

⁴ Gold Hill Computers, Cambridge, MA.

D.3.1. Detection in MIDAS

A detection scheme is necessary to convert raw numerical plant measurements into symptoms useful for diagnosis. In MIDAS, the primary detection task is the classification of measurements and constraint residuals into one of three possible qualitative states: HIGH, NORMAL, or LOW. Nominal steady-state values are used as a frame of reference for this classification. If a measured variable value is significantly greater (less) than its nominal steady-state value, it is classified as being in a HIGH (LOW) state. When near its steady-state value, a variable is in the NORMAL state. Random errors in measurement values necessitate the use of statistical tests in determining the operating region.

In MIDAS, the symptoms used for diagnosis are *events*, defined as any significant process change. A wide variety of events can be detected. For example, the transition of a variable from the NORMAL to HIGH operating region is a state change event. Events can also be

- Changes in variable trends (e.g. STEADY to INCREASING),
- Results of an off-line test or visual inspection,
- Actions taken by operators, or
- Gross failure of equipment (particularly sensors).

For each measured variable or constraint equation in the plant model, MIDAS has a *monitor* responsible for detecting events associated with state changes, trends, or gross sensor failure. Monitors operate independently and do not account for correlations in measurement data.

Finch and Kramer (1989) show that a detection scheme cannot simultaneously ensure symptoms will be detected in a particular order and remain sensitive to all symptoms. This invariably results in cases of false and out-of-order event detections. A primary objective of MIDAS was to develop plant models and diagnostic reasoning procedures robust to imperfect detection schemes. Consequently, developing an advanced detection scheme, immune to these problems, was not a goal of this project. Detection in MIDAS is accomplished using a variation of the Shewhart control chart technique (Shewhart, 1931) used for statistical process control. This technique was selected because it is relatively simple to understand, implement, and adapt, and it is generally accepted in industrial practice.

D.3.2. MIDAS Plant Model

The plant model contains information relating observable symptoms to possible root causes. Model selection is a critical step in developing a diagnostic system, since the detection scheme and reasoning procedure must be developed to match the plant model. In MIDAS, the plant model (including the steady-state variable values needed for detection) is the only component that must be constructed for each new application.

The plant models used in MIDAS integrate existing quantitative and qualitative models into a single modeling framework. Because different aspects of process behavior are captured by each model type, integration produces a synergistic effect, resulting in a plant model superior to any component model.

D.3.2.1. Causal Models

Causal relationships between measured variables and parameters are represented by directed causal links between events. These causal relationships are similar to and can be derived from the digraph models described by Iri et al. (1979), Palowitch (1987), and Oyeleye (1989). An automatic procedure for converting digraph models to MIDAS plant models has been developed by Oyeleye. Using this procedure and a library of predefined modular unit digraphs, a basic MIDAS model can be created with minimal human interaction. The reduction in required modeling effort afforded by this procedure makes MIDAS responsive to plant modifications.

Specific diagnostic information can be attached to a causal link using a *condition*. Several types of conditions have been defined, but the most common is the NOT condition used to block propagation of a fault disturbance along a link. For example, the condition

(:NOT (TANK LEAK))

indicates that disturbances resulting from a tank leak can not propagate along any link to which the condition is attached.

Unlike digraph models, MIDAS models contain explicit information on process dynamics. The dynamic character of the models is a direct outgrowth of the use of the event modeling paradigm. For example, in a digraph model, tracing a disturbance from an observed HIGH

measured variable state to possible causes requires inferring states for unmeasured variables. If the variable subsequently returns to the NORMAL state, any assumptions made based on the previous HIGH state must be retracted. This is the problem of assumption truth maintenance (ATM) discussed by De Kleer (1986). Because events are transitions rather than states, truth maintenance is unnecessary. The transition of a variable from NORMAL to HIGH occurs at a specific time point and once detected becomes a fact. The transition from HIGH to NORMAL occurs at a later time point and does not abrogate the fact that the variable was previously observed in the HIGH state. Links connecting different events of the same variable model possible dynamics. The possible dynamics modeled in MIDAS are compensatory response (normal → high → normal), inverse response (normal → high → low), and oscillatory response. No previous qualitative model-based system has systematically modeled dynamics in its basic modeling representation.

D.3.2.2. Constraint Models

In addition to causal models, MIDAS incorporates constraint equations in its modeling framework. Constraints are formulated in the form

$$F(X) = R = 0 \quad (D-1)$$

where X is a vector of measured variables and R is the constraint residual. Constraints can be algebraic or differential equations. When the process is free of faults, the residual mean is zero. A shift in the residual mean is indicative of changes in one or more constraint parameters caused by a process malfunction. Constraint residuals are monitored identically to process variables. The transition of a residual from the NORMAL (zero) value to a HIGH (positive) or LOW (negative) value is a valid diagnostic symptom represented as an event. Any potential compensatory or inverse response exhibited by a constraint can be modeled by appropriate links between constraint events.

D.3.2.3. Integrating Causal and Constraint Models

By combining causal and constraint models in a single unified representation, MIDAS is able to take advantage of the different types of plant behaviors modeled in each format. Causal models represent local sensitivity relationships between process variables and parameters. Being qualitative, however, positive or negative feedback loops can produce ambiguities. Constraint equations can model global relationships, such as overall material and energy

balances, not represented in causal models. Because constraints are quantitative, they are not subject to ambiguities and can, in fact, be used to resolve ambiguities in causal models.

A MIDAS diagnosis can be expected to contain roughly the same number of fault candidates as a diagnosis produced by intersecting a digraph diagnosis with a constraint equation diagnosis. As shown in figure D-2, an intersected diagnosis will contain fewer candidates than any of the constituent diagnoses. Past experience has demonstrated that the diagnoses produced by causal and constraint models are moderately disjoint, resulting in a significant intersection effect.

Using the intersection effect, MIDAS can produce diagnoses superior to those produced by systems utilizing only a single model paradigm. Because MIDAS incorporates both knowledge sources in a single, nonexclusionary framework, however, it does not suffer the stability problems associated with intersecting diagnosis sets. That is, even if the causal portion of the MIDAS model fails to capture the true fault, an inaccurate diagnosis can be avoided if the true fault is captured by the constraint portion of the model. Simply intersecting two diagnoses will always produce an inaccurate diagnosis if any constituent diagnosis is inaccurate.

MIDAS allows a flexible combination of causal and constraint models. Case studies indicate that creation of a complete causal model, supplemented by several key constraint equations, is an effective modeling strategy. Another strategy is to create a complete constraint model, using causal relationships only to model possible dynamics. Determining the optimal model mix remains a subject for further research. Modeling effort is minimized by mixing models, because only the most important or easily obtainable portion of each model need be developed.

D.3.2.4. Representation of the Plant Model

MIDAS represents potential events and root causes as instances of generic object classes within the object-oriented programming environment provided in GoldWorks. Stefik and Bobrow (1986) provide an introduction to object-oriented programming.

The plant model consists of instances of the following three object classes:

- **POTENTIAL-EVENT (PE)** instances represent all the events detectable by the monitors,

- POTENTIAL-ROOT-CAUSE (PRC) instances represent all faults that could effect the process⁵, and
- COMPILED-LINK (CL) instances represent causal links connecting PEs.

PRCs and PEs are connected by a second type of link called a *local cause link*. A local cause link connects a PRC to PEs that are expected to occur whenever the fault represented by the PRC is present. Local cause links are represented using pointers and do not have a separate object class.

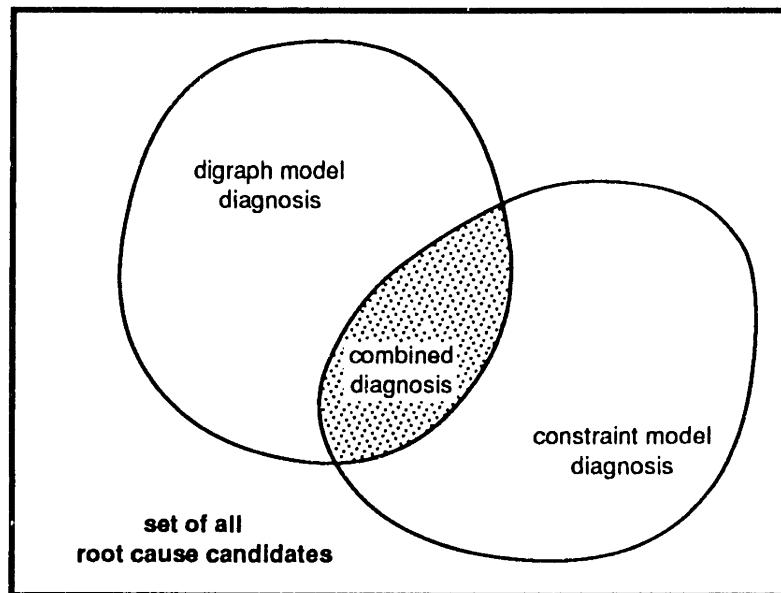


Figure D-2: Diagnosis Intersection Effect

D.3.3. Diagnostic Reasoning in MIDAS

The diagnostic reasoning procedure guides the process of tracing symptoms, detected by the monitors, back to possible causes using the information contained in the plant model. In general, the complexity of the reasoning procedure will depend on how direct the mapping of symptoms to causes is in the plant model. A compiled model (e.g. a fault table) requires a

⁵ The set of faults to be diagnosed must be determined at the time of model construction. MIDAS cannot diagnose a fault for which there is no PRC instance.

simple diagnostic procedure. In essence, all avenues of diagnostic reasoning are included in the model. An uncompiled model requires a complex diagnostic procedure, often including several levels of search and association.

The plant models employed by MIDAS are partially compiled -- some avenues of diagnostic reasoning are included in the model in the form of conditions. However, the compilation of MIDAS models falls short of the point at which event order information is lost. Partial compilation is desirable because it reduces the complexity of the reasoning procedure and shortens the time required for analysis of symptoms.

D.3.3.1. Linking Related Events

Because MIDAS models are not full compiled, the diagnostic reasoning procedure is an important component in the overall system. The primary objective of the procedure is to relate newly detected events to previously detected events and update the diagnosis correspondingly. Figure D-3 illustrates this procedure. Events can be related either through causal links or by common root causes, with link conditions guiding the diagnosis update when links are invoked. Between event detections, the inference procedure is not active.

The inference procedure is made robust to out-of-order events using predictions of future events. In figure D-4, for example, a new event is detected that cannot be linked directly to the previously detected event because of an intervening event that has not been detected. At this point, the inference procedure interrogates the monitor responsible for detecting the intervening event. The monitor is asked to predict whether the intervening event will be detected in the near future (i.e. it is a possible out-of-order event). The monitor makes the prediction by extrapolating past measurements and applying statistical tests of slightly lower significance than those used for primary event detection. If the monitor determines that the intervening event is out-of-order, MIDAS anticipates the correct event order by labeling the intervening event an "expected" event and completing the link between the new and existing event. For most diagnostic purposes, expected events are treated as if they have already been detected. Of course, it is possible for the prediction to be wrong, so MIDAS contains procedures to correct certain prediction errors when they become known.

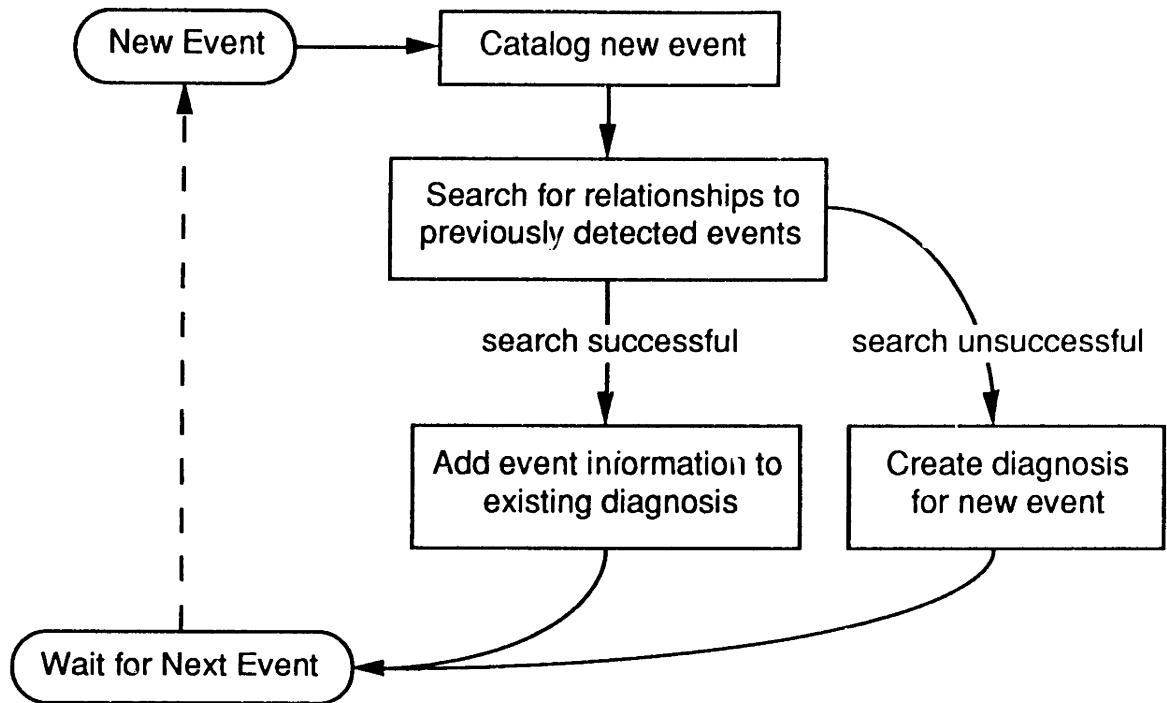


Figure D-3: Diagnostic Inference Procedure

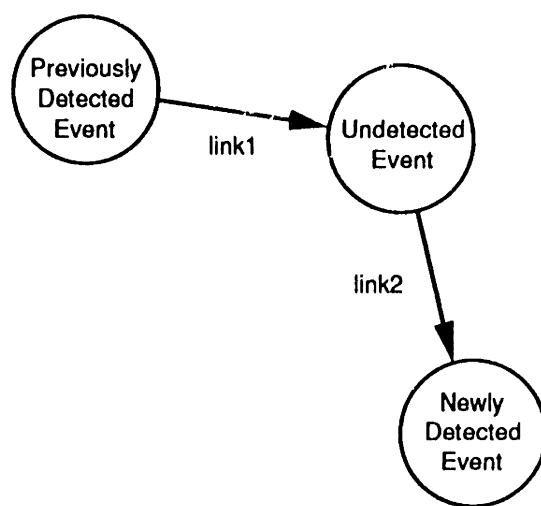


Figure D-4: Linking Out-of-Order Events

D.3.3.2. The Hypothesis Model

When MIDAS has identified possible links, it creates a mirror image of the pertinent section of the plant model, called the hypothesis model. The plant model contains all potential events and potential root causes, representing the full set of possible plant behaviors. The hypothesis models contains only events that have been observed or root causes that are part of a current diagnosis, representing the behavior of the current malfunction episode. The hypothesis model is empty before symptoms have been detected and is created on-line by the diagnostic reasoning procedure.

The hypothesis model uses different object classes than those used in the plant model:

- **RECORDED-EVENT (RE)** instances represent detected events,
- **EXPECTED-EVENT (EE)** instances represent events predicted to occur through monitor interrogation, and
- **HYPOTHESIZED-ROOT-CAUSE (HRC)** instances represent faults that are currently part of a candidate set.

Hypothesis model objects contain information related to a specific fault episode. For example, a RE instance would include information on the time of event detection and the detection certainty, not present in the analogous PE instance. In the hypothesis model, all links between REs, EEs, and HRCs are represented using pointers rather than objects.

D.3.3.3. Event Clusters

After multiple event detections, the hypothesis model is populated by clusters of related events. MIDAS assumes that all events in a cluster have a common root cause. This assumption embodies the philosophy that the minimum number of malfunctions necessary to explain observed events should be postulated -- a variation of the single fault assumption ubiquitous in other diagnostic systems. Unlike the SFA, however, MIDAS allows for multiple malfunctions so long as the symptoms of the malfunctions are sufficiently separated in the plant model that their events clusters do not interact. Only in cases where the symptoms of two malfunctions significantly interact is MIDAS in danger of failing to make a proper diagnosis. Tests have shown that for some cases of interacting malfunctions, MIDAS can still diagnose one of the malfunctions correctly.

Two important types of multiple malfunction are false symptom detection and induced sensor failures. The use of statistical tests for event detection make occasional false detections inevitable. MIDAS contains special logic to diagnose a transient event with no secondary symptoms as a false detection. Induced sensor failure is an example of dependent multiple malfunctions that can have a high conditional probability. MIDAS plant models are structured to successfully diagnose induced sensor failure, and MIDAS can continue to operate (albeit in a marginally degraded manner) after sensor failures.

Event clusters are dynamic in the sense that they can form, grow, coalesce, and disappear in response to event detections. A new event that cannot be related to any existing events will form a new cluster. Subsequent detection of related events will cause the cluster to grow. If a new event provides a bridge between two existing clusters, the clusters will merge to form a single supercluster, and if all abnormal symptoms associated with a cluster return to normal, the cluster is deactivated and archived for later review.

D.3.3.4. Ranking Fault Candidates

Each event in a cluster is a piece of evidence that can support a root cause candidate or oppose it. An event will support a candidate if the event is a primary symptom⁶ of the candidate or if a path of unblocked links can be traced from a primary symptom to the event. An event will oppose a candidate if no link path connects a primary symptom to the event, or if disturbance propagation is blocked by NOT conditions attached to links connecting the event to a primary symptom. By evaluating all events in a cluster, the cluster candidates can be ranked according to the relative weight of evidence supporting and opposing each candidate.

MIDAS performs this ranking and lists candidates in order of decreasing evidential weight. MIDAS can rank candidates using any of several formulas. By selecting different formulas, an operator can control the information included in the ranking. For example, one formula computes rankings that account for the prior probability of the fault candidate. Under this scheme, a rare fault would tend to be ranked lower than a frequent fault with equal supporting event evidence. Since operators use prior probability to rank candidates, this result should be similar to a diagnosis produced by an operator. Another formula, however, computes

⁶ A primary symptom is defined as the first symptom expected to be observed when a fault is present.

rankings based solely on the detection probabilities of events. Comparing the two diagnoses, the degree to which prior probability influences the diagnosis can be determined.

Ranking candidates gives MIDAS a tremendous advantage over systems that produce unranked candidate sets. Assuming candidates are examined and verified in order of decreasing rank, the absolute size of the candidate set is deemphasized. The crucial factor governing the quality of the diagnosis is how many candidates are ranked higher than the true fault. This factor, called the *resolution* of the diagnosis, determines the maximum number of candidates that must be examined before the true fault is discovered. In an unranked set, candidates can be listed and examined in any order. Therefore, the resolution is always equal to the absolute size of the candidate set. For example, if a cluster set includes twenty (20) root cause candidates, the unranked resolution of the diagnosis is twenty (20). If the true fault is the top ranked candidate, the resolution is one (1).

Ranking is also an important feature from the perspective of aiding the operator. It gives MIDAS the ability to list all viable candidates, including those likely to be overlooked or disregarded by the operator because of their low prior probability, without necessarily giving all candidates equal likelihood. In an unranked set, the operator is left to determine the order in which candidates will be examined. Operators often place undue weight on prior probability in making likelihood determinations (Tversky and Kahneman, 1974).

D.4. MIDAS Case Study

A case study has been performed to test MIDAS in a realistic plant environment. The results of the study verify the high performance expectations of MIDAS. Additional testing of MIDAS is currently underway. Imperial Chemical Industries Ltd. (ICI) has supplied a verified simulation of an actual process to serve as the basis of a second MIDAS case study. In addition, Duke Power Co. is currently using MIDAS as part of a diesel generator diagnostic system. Several other companies have received copies of MIDAS and are performing internal tests.

At this stage of testing, a simulated plant environment was chosen over a real plant to allow freedom in process design and to allow control over all case study parameters. The process selected for the study is shown in figure D-5. The essential features of the process are:

- A jacketed stirred tank reactor hosting an exothermic, first-order, irreversible chemical reaction,
- A cascade reactor temperature control system regulating flow from a cooling water utility,
- A reactor level control system regulating outlet flow, and
- Ten on-line sensors measuring flowrates, temperatures, pressures, and concentrations (the output signals of all controllers are also assumed available on-line).

This process, though limited in overall scope, provides sufficient diagnostic challenge to represent a reasonable test of MIDAS. The process includes several feedback loops, opportunities for compensatory and inverse response, and ample potential for out-of-order events. Based on these features, diagnostic performance on this process should provide an accurate indicator of MIDAS performance on a wide variety of similar processes.

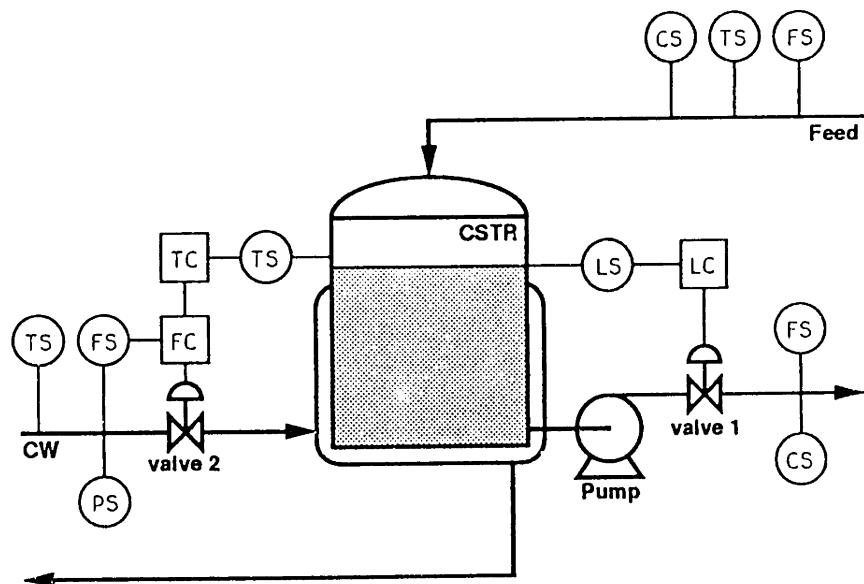


Figure D-5: Jacketed CSTR Process

A FORTRAN program, dynamically simulating one hundred and six (106) fault modes, was created for this process. Faults are simulated by changes in program variables and parameters. The ultimate extent, extent versus time trajectory, and time delay can be specified for up to three (3) separate faults per simulation run. Random errors are added to all sensor measurements to simulate signal noise. Sensor outputs are written to an output file in random order and with variable frequency to simulate different sensor sampling rates.

To generate fault cases, one hundred (100) faults were chosen randomly (with replacement) from a list of one hundred nine (109) possible faults modes. Each case was assigned a random ultimate extent (within a predetermined range) and a random extent versus time trajectory⁷. The faults generated by this random selection process are presented in Table D-2. Some of the fault cases were unusable because the faults were not included in the simulator, were undetectable with the given instrumentation, or were of insufficient extent to produce detectable events. Seventy six (76) simulated fault scenarios were used in the final study after unusable cases were discarded.

The causal portion of the plant model was created from unit digraphs using the automatic algorithm developed by Oyeleye. Creation required approximately four man-hours and eight computer-hours on a PC compatible 386/20 MHz computer. This portion of the model was used without further modification. Event detection thresholds in MIDAS monitors were established using an automatic tuning utility provided with the program.

Supplementing the causal model, four constraint equations were added to the model manually. These constraints were:

- An overall material balance for the reactor,
- A chemical species balance for the reactor,
- A pressure drop equation for the reactor outlet stream, and
- A pressure drop equation for the cooling water stream.

Several additional man-hours were required to add these constraints to the model.

⁷ All fault trajectories approached their ultimate extent asymptotically using an exponential function. Random trajectories were assigned by selecting a time constant for the exponential. Time constants varied over five (5) orders-of-magnitude.

The results of the case study showed MIDAS to be an excellent diagnostic system. MIDAS included the true fault in the candidate set in 98.9% of all passes through the inference cycle. In 82% of scenarios, the true fault was given top rank. In another 8% of scenarios, the true fault was ranked second. The average diagnostic resolution attained by MIDAS was 4.2 candidates (i.e. 97% of possible candidates were eliminated from consideration or ranked lower than the true fault). Figure D-6 shows the distribution of diagnostic resolution for all cases. From this figure, the median resolution can be calculated at two (2) faults. In almost a third of all cases, perfect diagnosis was achieved.

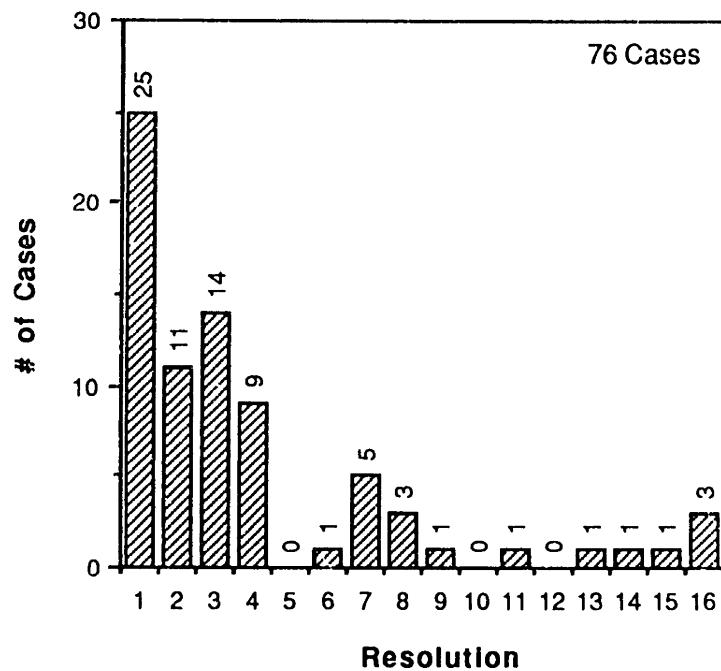


Figure D-6: Distribution of Resolution

Performance of MIDAS as a function of the number of events observed is presented in Table D-1. The accuracy of MIDAS is nearly constant. Resolution, measured by the performance parameter (Φ)⁸, achieves a maximum between five (5) and six (6) events observed, slowly declining as more than seven (7) events are observed.

⁸ The performance parameter (Φ) is the fraction of potential root causes eliminated from consideration or ranked lower than the true fault.

TABLE D-1: Overall Diagnostic Performance of MIDAS†

Events Observed	Accuracy (% of Cases)	Ranked in Top Tier (% of Cases)	Ranked in 1st or 2nd Tier (% of Cases)	Average Performance (Φ)
1	93	89	97	0.88
2	100	90	97	0.94
3	100	82	100	0.95
4	100	91	98	0.95
5	100	88	98	0.96
6	100	77	92	0.96
7	97	76	90	0.95
8	100	67	80	0.94
9	100	63	84	0.93
10	100	38	76	0.92
11	100	36	63	0.92

† Statistics compiled based on a sample of 76 cases.

The only significant performance blemish was related to cycle time. On average, 1.75 minutes per event detection were required to complete the inference cycle -- too long to consider MIDAS a practical real-time system in most applications. The distribution of evaluation times, shown in figure D-7, indicates the high average is a result of relatively few events requiring more than two (2) minutes. However, the long tail of the distribution indicates the unpredictable nature of event evaluation. The slow speed of MIDAS points to weaknesses in its implementation, not the concepts which form its foundation. Speed was not a paramount issue in development of the MIDAS prototype, and could be addressed in future

versions of MIDAS through several avenues, including faster platforms⁹, more efficient programming, and parallel processing.

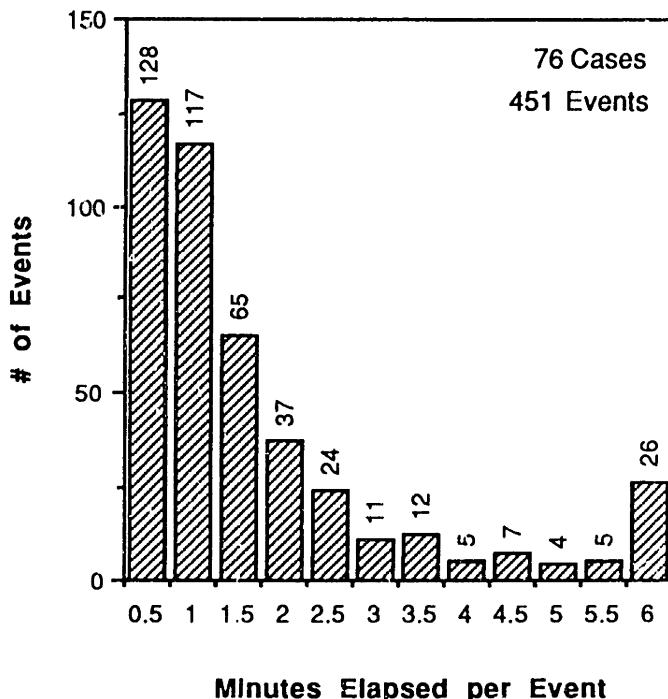


Figure D-7: Evaluation Time Distribution

In approximately 5% of scenarios, MIDAS ranked the true fault significantly lower than the top ranked candidate. These cases indicate the limitations of the qualitative causal models that form the basic MIDAS plant model. Three factors contribute to this suboptimal performance:

- Apparent inverse response,
- Causal loops in the plant model, and
- Ambiguities in the digraph models used to form the plant model.

⁹ A PC compatible 386/20MHz computer with 10MB RAM was used for all cases study scenarios.

Apparent Inverse Response

Erroneous detection of inverse response can occur whenever a control system exhibits underdamped behavior. In figure D-8, an idealized underdamped response is plotted. Event detection thresholds are shown indicating values which are considered HIGH and LOW. From the plot, the initial response is HIGH and the ultimate response is NORMAL. An erroneous LOW state (i.e. apparent inverse response) is detected at approximately 23 seconds. Inverse response is not included in the causal model, which assumes control systems produce clean compensatory response. Consequently, detection of inverse response can confuse the diagnosis, resulting in multiple event clusters or erroneous causal link paths. This problem may be addressed by more advanced trend modeling schemes currently under study by Cheung and Stephanopoulos (1988).

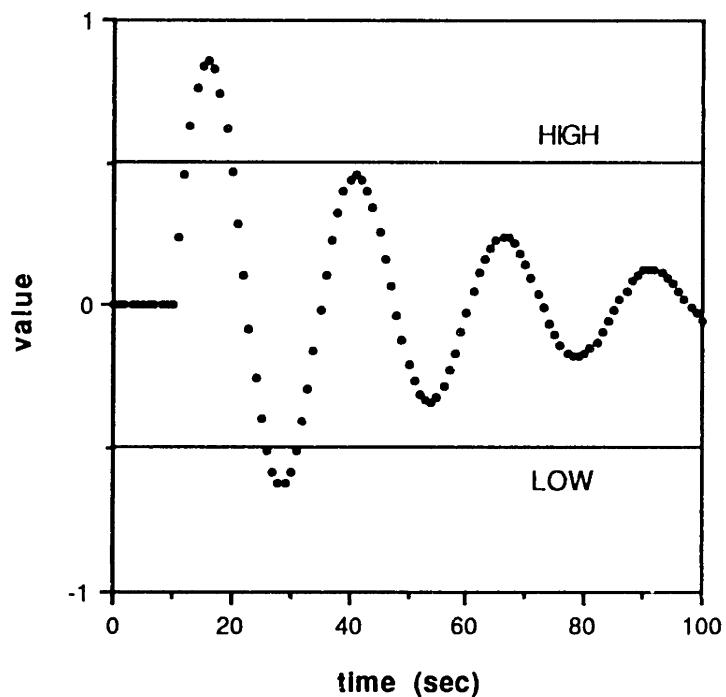


Figure D-8: Apparent Inverse Response

Causal Loops

A causal loop is a group of active links that connect two or more observed events in a cycle. Causal loops degrade the resolution of a diagnosis by destroying a clear event sequence. Under the tenet that events can occur out-of-order, any event in a causal loop could be the true

initial event (i.e. primary symptom) of the sequence. Therefore, in the diagnostic reasoning procedure, all events in the loop must be considered as possible initiating events. This uncertainty in proper event order is manifest as an increase in the absolute size of the root cause candidate set and a relative decrease in evidence for the true fault. This problem effects only the resolution of the diagnosis, not the accuracy. Causal loops are an example of the limitations of qualitative models. Breaking a causal loop requires addition of selected quantitative information to the plant model. Specifically, causal loops can be broken with knowledge of the transfer functions (i.e. gains and delays) associated with causal links.

Ambiguities

An example of digraph ambiguity is the effect on reactor temperature of a blockage at the CSTR inlet that reduces the feed flowrate. Low feed flowrate tends to increase reactor temperature by reducing the cooling effect of the relatively cold feed stream. Low feed flowrate can also tend to reduce reactor temperature by decreasing the reactant concentration and the reaction rate. The dominant effect cannot be determined without quantitative information on the specific process under study. Therefore, the qualitative model must entertain both possibilities, resulting in degraded diagnostic resolution.

The deleterious effects of these fundamental qualitative model limitations are observed in only a few pathological cases. In most cases, the qualitative model, supplemented by constraint equations, produced diagnoses near the theoretical maximum imposed by the available instrumentation. Moreover, the addition of quantitative information on the gain and delay associated with disturbance propagation along a causal link could break causal loops in a variety of situations, without resort to fully quantitative models.

D.5. Opportunities for Future Work

Several opportunities for future work are presented (in no particular order):

- ⇒ Adapt the control system modeling paradigm used by the SLD technique to an event format and integrate the resulting dynamic system models into the MIDAS plant model. SLD captures aspects of process behavior difficult to model using either digraph or constraint models. Therefore, a substantial intersection effect can be expected when system models are integrated with the existing MIDAS paradigms.

- ⇒ The event paradigm can be expanded to systematically include events for possible operator actions and changes in qualitative trends. This project would concentrate on the necessary concepts for modeling trends, which has received less attention than qualitative state modeling.
- ⇒ Semi-qualitative models can be developed that include gains and delays associated with disturbance propagation along causal links. This additional information can be used to break causal loops in the HM or allow the diagnostic reasoning procedure to include reasoning based on non-detection of events.
- ⇒ Advanced detection schemes, resistant to apparent inverse response, false event detection, and out-of-order event detection, can be developed.
- ⇒ Plant models with a different mix of qualitative and quantitative knowledge can be developed and tested. The performance of MIDAS using a model comprised entirely of constraint equations should provide additional insights into the relative merits of the two modeling formats.

D.6. References

Buchanan, B.G. and E.H. Shortliffe, Rule-Based Expert Systems, Addison-Wesley, Reading, MA (1984).

Cheung, J.T. and G. Stephanopoulos, "A Descriptive Representation of Process Trends," MIT Laboratory for Intelligent Systems in Process Engineering, Report LISPE-88-039 (1988).

Cowan, J.D. and D.H. Sharp, "Neural Nets and Artificial Intelligence," *Daedalus*, 117 (1), 85 (1988).

De Kleer, J., "An Assumption-Based TMS," *Artificial Intelligence*, 28 (2), 127 (1986).

Dhillon, B.S. and S.N. Rayapat, "Chemical System Reliability: A Review," *IEEE Trans. on Reliability*, 37 (2), 199 (1988).

Finch, F.E. and M.A. Kramer, "The Handling of Dynamics, Multiple Faults, and Out-of-Order Alarms in the MIDAS Diagnosis System," paper 37e, AIChE Spring National Meeting, Houston, TX (1989).

Himmelblau, D.M., Fault Detection and Diagnosis in Chemical and Petrochemical Processes, Elsevier, Amsterdam (1978).

Iri, M., K. Aoki, E. O'Shima and H. Matsuyama, "An Algorithm for Diagnosis of System Failures in the Chemical Process," *Comp. and Chem. Eng.*, **3**, 489 (1979).

Isermann, R., "Process Fault Detection Based on Modeling and Estimation Methods -- A Survey," *Automatica*, **20**, 387 (1984).

Kramer, M.A. and F.E. Finch, "Fault Diagnosis of Chemical Processes," in Tzafestas, S.G. (Ed.), Knowledge-Based Systems Diagnosis, Supervision and Control, Plenum, New York (1989).

Oyeleye, O.O., Sc.D. Thesis: Qualitative Modeling of Continuous Chemical Processes and Applications to Fault Diagnosis, Massachusetts Institute of Technology (1989).

Palowitch, B.L., Sc.D. Thesis: Fault Diagnosis of Process Plants using Causal Models, Massachusetts Institute of Technology (1987).

Shewhart, W.A., Economic Control of Quality in Manufactured Product, Van Nostrand, New York (1931).

Stefik, M. and D.G. Bobrow, "Object-Oriented Programming: Themes and Variations," *The AI Magazine*, **6** (4), 40 (1986).

Tversky, A. and Kahneman, D., "Judgement under Uncertainty: Heuristics and Biases," *Science*, **185**, 1124 (1974).

TABLE D-2: Randomly Generated Faults for Case Study

#	Mode	Extent (% of range)	Trajectory Constant	Description
1	8	95	0.1	CSTR_OUTLET_BLOCKAGE
2	21	99	1.0	PUMP_MOTOR_SPEED_HIGH
3	39	45	10.0	¹ VALVE_2_LOSS_OF_INSULATION
4	26	46	10.0	² VALVE_1_FIRE
5	6	42	0.1	REACTOR_FEED_TEMPERATURE_LOW
6	37	32	10.0	VALVE_2_BLOCKAGE
7	11	12	10.0	CSTR_CATALYST_1_DEACTIVATION
8	91	70	100.0	³ CONC_A_SENSOR_FAILED_LOW
9	39	40	100.0	¹ VALVE_2_LOSS_OF_INSULATION
10	102	79	1.0	³ CW_TEMP_SENSOR_FAILED_HIGH
11	87	34	0.1	³ PRODUCT_FLOW_SENSOR_FAILED_LOW
12	105	55	1.0	CW_TEMP_SENSOR_BIASED_LOW
13	12	70	0.01	⁵ CSTR_CATALYST_2_DEACTIVATION
14	35	25	10.0	JACKET_FEED_SOURCE_TEMP_LOW
15	72	55	0.01	FEED_TEMP_SENSOR_BIASED_HIGH
16	90	91	10.0	³ CONC_A_SENSOR_FAILED_HIGH
17	91	52	0.1	⁴ CONC_A_SENSOR_FAILED_LOW
18	83	52	100.0	³ REACTOR_LEVEL_SENSOR_FAILED_LOW
19	102	2	0.01	⁴ CW_TEMP_SENSOR_FAILED_HIGH
20	8	84	1.0	CSTR_OUTLET_BLOCKAGE
21	14	88	1.0	CSTR_LOSS_OF_INSULATION
22	108	10	10.0	CW_PRESSURE_SENSOR_BIASED_HIGH
23	2	66	0.01	REACTOR_FEED_CONC_A_LOW
24	26	35	1.0	² VALVE_1_FIRE
25	15	71	1.0	CSTR_OUTLET_BLOCKAGE
26	86	50	1.0	⁴ PRODUCT_FLOW_SENSOR_FAILED_HIGH
27	79	92	1.0	⁴ REACTOR_TEMP_SENSOR_FAILED_LOW
28	87	57	100.0	³ PRODUCT_FLOW_SENSOR_FAILED_LOW
29	29	33	100.0	VALVE_1_STUCK_OPEN
30	100	88	100.0	CW_FLOW_SENSOR_BIASED_HIGH
31	73	9	10.0	⁵ FEED_TEMP_SENSOR_BIASED_LOW
32	2	86	0.01	REACTOR_FEED_CONC_A_LOW

TABLE D-2: Randomly Generated Faults for Case Study (continued)

#	Mode	Extent (% of range)	Trajectory Constant	Description
33	61	100	0.01	CW_FLOW_CONTROLLER_FAILED_LOW
34	1	89	0.1	REACTOR_FEED_CONC_A_HIGH
35	8	24	0.01	CSTR_OUTLET_BLOCKAGE
36	49	22	100.0	LEVEL_CONTROLLER_FAILED_LOW
37	79	55	10.0	⁴ REACTOR_TEMP_SENSOR FAILED LOW
38	21	56	0.01	PUMP_HIGH_MOTOR_SPEED
39	88	84	1.0	PRODUCT_FLOW_SENSOR_BIASED_HIGH
40	86	54	0.01	³ PRODUCT_FLOW_SENSOR FAILED LOW
41	85	84	10.0	REACTOR_LEVEL_SENSOR_BIASED_LOW
42	89	15	0.01	PRODUCT_FLOW_SENSOR_BIASED_LOW
43	15	26	100.0	CSTR_OUTLET_BLOCKAGE
44	13	4	0.01	⁵ CSTR_FIRE
45	19	30	0.1	PUMP_LOW_MOTOR_SPEED
46	7	24	10.0	¹ CSTR_INLET_BLOCKAGE
47	41	73	100.0	VALVE_2_STUCK_OPEN
48	70	69	0.1	⁴ FEED_TEMP_SENSOR FAILED HIGH
49	51	99	100.0	LEVEL_CONTROLLER_BIASED_HIGH
50	59	53	0.01	TEMP_CONTROLLER_SETPOINT_LOW
51	6	45	0.1	REACTOR_FEED_SOURCE_TEMP_LOW
52	43	35	0.1	CSTR_JACKET_FOULING
53	67	27	0.1	⁴ FEED_FLOW_SENSOR FAILED LOW
54	41	99	1.0	VALVE_2_STUCK_OPEN
55	69	49	1.0	FEED_FLOW_SENSOR_BIASED_LOW
56	99	92	0.01	³ CW_FLOW_SENSOR FAILED LOW
57	9	98	1.0	PUMP_LEAK_TO_ENVIRONMENT
58	12	5	0.1	CSTR_CATALYST_1_DEACTIVATION
59	43	99	10.0	CSTR_JACKET_FOULING
60	109	7	0.1	CW_PRESSURE_SENSOR_BIASED_LOW
61	11	60	0.01	CSTR_CATALYST_1_DEACTIVATION
62	6	27	0.01	REACTOR_FEED_TEMP_LOW
63	37	73	0.01	VALVE_2_BLOCKAGE
64	18	42	0.1	PUMP_LEAK_TO_ENVIRONMENT

TABLE D-2: Randomly Generated Faults for Case Study (continued)

#	Mode	Extent (% of range)	Trajectory Constant	Description
65	99	89	0.1	⁴ CW_FLOW_SENSOR_FAILED_LOW
66	96	91	100.0	CONC_B_SENSOR_BIASED_HIGH
67	35	74	1.0	CSTR_JACKET_FEED_TEMP_HIGH
68	66	24	0.1	⁴ FEED_FLOW_SENSOR_FAILED_HIGH
69	57	8	100.0	⁵ TEMP_CONTROLLER_BIAS_LOW
70	109	0	0.1	⁶ CW_PRESSURE_SENSOR_BIAS_LOW
71	13	12	0.1	⁵ CSTR_FIRE
72	49	76	0.01	LEVEL_CONTROLLER_FAILED_LOW
73	92	7	10.0	⁵ CONC_A_SENSOR_BIASED_HIGH
74	78	94	100.0	³ REACTOR_TEMP_SENSOR_FAILED_HIGH
75	38	39	0.1	¹ VALVE_2_FIRE
76	57	44	10.0	TEMP_CONTROLLER_BIAS_LOW
77	20	96	1.0	² PUMP_OVERHEATING
78	32	81	1.0	EFFLUENT_SINK_PRESSURE_LOW
79	104	99	0.01	CW_TEMP_SENSOR_BIASED_HIGH
80	109	91	1.0	CW_PRESSURE_SENSOR_BIASED_LOW
81	3	5	1.0	REACTOR_FEED_PRESSURE_HIGH
82	73	0	0.1	⁶ FEED_TEMP_SENSOR_BIASED_LOW
83	95	75	0.1	CONC_B_SENSOR_FAILED_LOW
84	36	10	0.01	JACKET_FEED_SOURCE_TEMP_LOW
85	84	53	0.01	REACTOR_LEVEL_SENSOR_BIASED_HIGH
86	83	53	0.1	⁶ REACTOR_LEVEL_SENSOR_FAILED_LOW
87	29	37	10.0	VALVE_1_STUCK_OPEN
88	71	38	0.1	FEED_TEMP_SENSOR_FAILED_LOW
89	51	7	1.0	⁵ LEVEL_CONTROLLER_BIASED_LOW
90	38	48	100.0	¹ VALVE_2_FIRE
91	26	68	1.0	² VALVE_1_FIRE
92	101	67	1.0	CW_FLOW_SENSOR_BIASED_LOW
93	102	0	0.01	⁶ CONC_A_SENSOR_BIASED_LOW
94	85	63	0.1	REACTOR_LEVEL_SENSOR_BIASED_LOW
95	11	5	10.0	CSTR_CATALYST_1_DEACTIVATION
96	42	58	0.1	VALVE_2_STUCK_CLOSED

TABLE D-2: Randomly Generated Faults for Case Study (continued)

#	Mode	Extent (% of range)	Trajectory Constant	Description
97	39	35	1.0	¹ VALVE_2_LOSS_OF_INSULATION
98	46	25	0.1	⁵ JACKET_EFFLUENT_PRESSURE_HIGH
99	98	27	0.1	⁴ CW_FLOW_SENSOR_FAILED_HIGH
100	27	11	0.1	² VALVE_1_LOSS_OF_INSULATION

Notes: ¹ Fault not simulated.

² Fault undetectable with given instrumentation.

³ In-range sensor failure.

⁴ Out-of-range sensor failure.

⁵ No events detected

⁶ Unusable

1.0. Introduction

Using intelligence and ingenuity, mankind has created a technology comprised of complex systems and devices that require continuous monitoring and periodic adjustments to remain within operating limitations imposed by physical, economic, or safety considerations. How and when to make these adjustments is the problem of *control*. Transportation systems, energy generation and distribution systems, manufacturing plants, communication networks, and waste treatment facilities are all examples of complex systems that rely heavily on control technology. From these examples, it is clear that control failures have the potential to cause serious personal safety and environmental hazards as well as enormous social and economic disruption.

Scientists and engineers have devoted significant energy to develop efficient and reliable solutions to the control problem. These solutions range from automatic controllers executing complex mathematical algorithms to advanced simulators to train human operators. As device complexity increases and the tolerance for control error decreases, however, existing control technology is continually forced to improve.

When a system is disturbed by an equipment failure, a human error, or an exogenous influence, controls must be applied to compensate or correct the disturbance. At the global level, there are only two strategies, shown in figures 1-1 and 1-2, to accomplish this objective. In *symptom-based control*, control actions are formulated directly from observed symptoms of a disturbance. Disturbance compensation is accomplished by making adjustments to system parameters for which there exists built-in variability. In effect, a disturbance to a critical system variable is transferred to a non-critical variable, but the root cause of the disturbance is not removed. *Cause-based control* seeks to identify the source of a disturbance and remove it, if possible, by repairing failed equipment or correcting human errors.

In practice, neither of these strategies is sufficiently robust to adequately control all systems. Symptom-based control will fail if

- A single large disturbance or multiple small disturbances exhaust the variability of available control parameters,
- No control parameters exist able to compensate a disturbance,
- The control apparatus fails, or

- The disturbance moves the system sufficiently far from the optimum operating region to make further operation uneconomical.

Cause-based control will fail if

- The disturbance source cannot be identified or repaired,
- The symptoms represent such an immediate threat that there is no time for diagnosis and repair, or
- Repair becomes physically or economically impractical¹.

The deficiencies of each strategy usually force an approach that balances symptom- and cause-based control. Asbjornsen (1989) discusses these issues of plant controllability in greater detail.

The degree to which one control strategy is favored over the other is dependent on several factors. Symptom-based control, in its most common form as *feedback control*, has proven easier to express mathematically and is, therefore, easier to automate. In contrast, cause-based control has resisted automation due to the difficulty of the diagnosis problem and the diversity of possible control actions. Moreover, the types of disturbance that require cause-based control tend to be rare; whereas, the disturbances suited to symptom-based control are more common. Consequently, symptom-based control has seen wider industrial application because it is typically faster and less expensive than cause-based control. Continuous cause-based control has been limited to the largest, most expensive, and most dangerous applications, and even there, it is used primarily as a backup to automatic symptom-based controllers.

An overemphasis on symptom-based control can result in operating hazards if a system becomes uncontrollable due to insufficient parameter variability or if control actions, formulated without knowledge of underlying causes, intensify a disturbance. Unusual occurrences, such as equipment failures, are more likely to produce disturbances that exceed the capacity of automatic controllers or fool automatic controllers into inappropriate responses. When equipment fails, diagnosis and repair is usually the best strategy. Minimally, an

¹ Small disturbances of short duration produced by the natural variability of a process or the environment are not well suited for cause-based control because of the long delays involved in diagnosis and repair.

accurate diagnosis should be available to ensure that symptom-based controllers do not further the disturbance through ignorance.

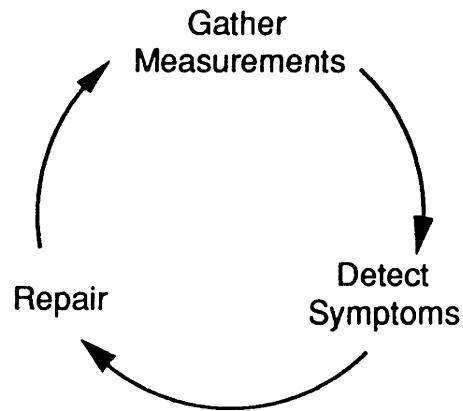


Figure 1-1: Cause-Based Control Strategy

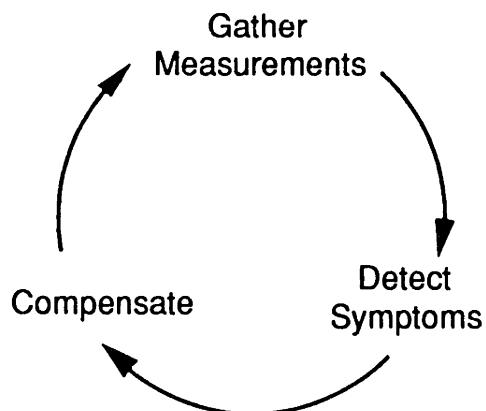


Figure 1-2: Symptom-Based Control Strategy

1.1. Problem Statement

Diagnosis has been an activity performed almost exclusively by human operators working from a central control room where information from a network of remote sensors is concentrated. This arrangement has worked well, primarily due to the unique qualities of human reasoning:

- Human operators can deal with incomplete and uncertain information,
- With experience, humans can learn to discern the key diagnostic features in large sets of data,
- Operators can communicate to pool knowledge and experience,
- Humans are flexible in their approach to problem solving, and
- Humans can innately integrate widely differing knowledge sources to form a global picture of a system and its operation.

Despite these strengths operators occasionally misinterpret symptoms and take inappropriate corrective actions. Misdiagnosis can result in safety hazards, poor quality products, waste of raw materials, environmental release of toxics, or equipment damage. The causes of misdiagnosis highlight the weaknesses in human reasoning and the potential danger in relying solely on human operators for diagnosis of complex systems. Some of these weaknesses are

- Limited, imprecise, and unreliable memory,
- Loss of cognitive ability when subjected to stress or fatigue,
- Inability to handle large quantities of data presented numerically,
- Wide variance in experience and training,
- Incomplete knowledge of system fundamentals, and
- Boredom and inattentiveness with repetitive tasks.

When systems must operate continuously, it is not possible to have the expert operators and supervisors available at all times. This motivates the desire to provide operator aids. Common aids include improved control room layouts, audible and visual alarms and warnings, automatic safety systems, enhanced and redundant instrumentation, and graphical presentation of data. These aids concentrate on the problem of the man-machine interface and can address only a subset of the human weaknesses listed.

An aid capable of addressing all the weaknesses of the human operator would need the capability to perform an *independent diagnostic analysis*. This type of advanced aid would address the weaknesses associated with human reasoning in addition to the weaknesses in the man-machine interface. Such an aid could provide the operator with additional malfunction hypotheses that may have been overlooked or forgotten, insights into the relationships between symptoms (derived from models of process behavior), and advice on how the malfunction should be handled (drawn directly from a training manual or technical specification). In a real sense, such an aid would be an operator assistant or alter ego. A computer based aid would have a large and infallible memory, would be available at all times, would not be influenced by boredom, fatigue, or stress, and could handle large amounts of numerical data.

The challenges to be surmounted in developing an automated operator assistant are enormous. It is not sufficient to address only the weaknesses of human operators, a useful system must also embody many of the strengths of human operators. It must communicate its reasoning and formulate advice in a manner that operators can understand. It must operate in the same environment of uncertain and incomplete information. It must be flexible and capable of integrating information into a global picture. Most importantly, it must produce a diagnostic analysis with the same speed and of the same caliber as the most proficient human diagnosticians.

Each of these requirements is a significant challenge. Traditional programming techniques have had difficulty meeting these challenges. Symbolic and artificial intelligence programming techniques show more promise. Even with these new programming tools, however, the essential questions to be answered remain conceptual:

- What knowledge of system behavior is required and how can this knowledge be acquired,
- How should knowledge be represented to enable flexible and reliable diagnostic reasoning,
- What rules and methodologies do human diagnosticians use to perform diagnosis and can these methodologies be adapted to use in an automated operator assistant,
- Are there any useful diagnostic methodologies not commonly used by human diagnosticians that could be implemented in the computer environment to enhance diagnosis, and
- How can an automated operator assistant receive system measurements and communicate results transparently to human operators.

The potential applications for an operator aid are numerous. Where human operators are currently used, an operator assistant would provide a useful backup, increasing the speed and reliability of diagnosis while decreasing the uncertainty and variation in operator performance. In applications where human operators are not currently used, either because of cost, accessibility, or a lack of available experts, an operator assistant could provide a cost effective means of performing on-site cause-based control. Additionally, such a system could be easily adapted to serve as a training aid to help human operators hone their diagnostic reasoning skills.

1.2. Research Scope

The overall goal of this research has been to develop the foundations for an operator aid capable of performing diagnostic analysis without human attention. It is hoped that this type of aid will lower the barriers to cause-based control and lead to more balanced and robust control schemes.

The concepts developed here apply to systems that involve the flow, pumping, mixing, heating, or cooling of pure or multi-component fluids with or without chemical reaction. To limit the scope of the problem and provide concrete case study examples. This research has been confined to studying diagnosis of continuous, nominal steady-state processes, primarily from the chemical and refinery industries. Initial investigation into the extension of this research to batch, semi-continuous, and non steady-state processes appears promising, but remains a problem for further research. Adapting the concepts presented to purely mechanical or electrical systems is straightforward and not discussed in detail.

Chemical and refinery processes present several significant diagnostic challenges not found in many electronic, mechanical, or computer systems. These challenges include:

Uncertain Models

Process designers and operators may not have a complete understanding of the reaction kinetics, physical properties, or transport phenomena involved in a particular process. This problem is compounded when a malfunction forces the process away from its normal operating regime where operating experience is available.

Uncertain Data

Process sensor data may be corrupted by random noise or systematic bias and, therefore, is not completely trustworthy. Gross sensor failure (often induced by other malfunctions) may deny access to vital measurements.

Incomplete Instrumentation

Typically only a small fraction of process variables are measured continuously. Other variables may be measured indirectly or periodically, but the majority of process variables are not measured at all or must be measured manually in response to an operator request.

Complex Responses

Automatic controllers, multiple disturbance propagation paths, and nonlinearities in the process may result in certain process variables exhibiting non-monotonic behavior, such as compensatory (normal → high → normal), inverse (normal → high → low), or oscillatory responses. Specifically, automatic symptom-based controllers can complicate diagnosis by masking certain malfunction symptoms and accentuating others.

Process Interactions

Recycle and reflux streams, heat recovery systems, and control systems all provide pathways by which process malfunctions can produce symptoms far from their root cause. In highly coupled systems, the diagnostic information content of measurements may actually be degraded by the propagation of a malfunction as symptoms and causes become confused.

Multiple Malfunctions

Several malfunctions may exist simultaneously in a large plant. Multiple malfunctions may be causally related or the result of coincidence. Diagnosis can be complicated if the symptoms of multiple malfunctions overlap.

Changing Processes

Over time, processes will be modified to increase efficiency or upgrade products. Normal maintenance will occasionally require that certain equipment be taken off-line temporarily, and product changeovers may alter process characteristics. These actions may make profound differences in system behavior in the presence of a malfunction.

To ensure a useful diagnostic aid, the following minimum requirements have been established as goals for this research:

Accuracy

If the diagnosis consists of a set of one or more malfunction candidates, then the true malfunction should be represented in the set at all times.

Resolution

The number of malfunction candidates in the diagnosis set should be minimized given the available information.

Stability

The set of malfunction candidates, though changing, should be fundamentally stable (i.e. an incremental increase in diagnostic information should produce an incremental change in the diagnosis).

Reliability

The system should be based on basic principles rather than case studies or examples, so that accurate and predictable diagnosis can be expected for all malfunctions, including novel malfunctions².

Speed

The system should perform diagnosis in real-time, or minimally, as quickly as the best human diagnostician.

Portability

The system should be structured so that the process model is separate from the diagnostic reasoning, thereby minimizing difficulties adapting the system to new or modified processes. Process models should be structured for ease of creation, modification, and understanding.

Robustness

The system must be able reason with uncertain and incomplete information, and must be robust to potential problems of symptom detection, including false alarms, out-of-order alarms, and latent alarms.

This research has had two major objectives: to develop useful diagnostic methodologies and modeling paradigms and to implement these methodologies in a working prototype diagnostic

² A novel malfunction is any malfunction for which there is no past experience.

system called MIDAS (for Model-Integrated Diagnostic Analysis System). It will be demonstrated in subsequent sections how this prototype satisfies all of the above criteria for a usable diagnostic aid, except speed³. Additional features of this prototype include the following:

Fixed Instrumentation

The system operates using any instrumentation scheme, and will continue to operate (albeit in a somewhat degraded manner) if sensors fail. Accordingly, implementing the system does not mandate installation of addition instrumentation.

No Operator Input

The system operates without human input, and performs an independent diagnostic analysis. An operator can consult the system for an unbiased alternative evaluation of plant performance.

Hypothesis Ranking

The system can rank malfunction candidates using both prior probabilities and the weight of available evidence (i.e. observed symptoms) supporting the candidate. This feature is useful for quickly narrowing the focus of attention, particularly when sparse instrumentation produces relatively large candidate sets.

Tests

The system recommends tests that can be performed to further narrow the diagnosis set. These tests are presented in order of relative importance.

Features not incorporated in the prototype, but desirable in a final version include:

Operator Advice

The current system does not provide advice to operators on how to proceed after diagnosis. This is viewed as a distinct problem separate from diagnosis and not necessarily best

3 The speed criterion could be satisfied if MIDAS were implemented on a more powerful platform, written in a faster language, or programmed more efficiently. Therefore, failure of the current version of MIDAS to meet the speed requirement cannot be considered a fundamental limitation.

addressed by the same techniques. Diagnosis is a first step toward the goal of formulating operator advice.

Instrumentation Scheme Evaluation

The system does not indicate in an explicit way whether a given instrumentation scheme is ideal for diagnosis, although instrumentation will have a profound effect on diagnostic performance. This feature could be useful in design, serving as a guide to sensor placement.

In subsequent sections, the MIDAS project is described in detail. In Section 2.0, an introduction and review of fault diagnosis methodologies is presented. Section 3.0 discusses specific modeling paradigms used in the MIDAS system. The overall structure and diagnostic reasoning algorithm used by MIDAS is described in Section 4.0. Section 5.0 provides a user guide to the MIDAS program. Section 6.0 presents the results of case studies performed using MIDAS to diagnose faults in a simulated process environment, and Section 7.0 summarizes the current status of the MIDAS project and discusses avenues for further extension of the concepts presented. All references are listed in Section 8.0.

2.0. Fault Diagnosis

In this chapter, the diagnosis problem is introduced, and solution strategies are explored. The first section deals with problem formulation, defining important terms and outlining general solution strategies. The next section explores human problem solving to gain insight into an operator's approach to diagnosis. The final section reviews the state-of-the-art in methodologies for automated diagnosis. Emphasis is always placed on concepts and ideas that will form the basis for subsequent chapters.

2.1. Problem Formulation

Before the relative merits of the various diagnosis methodologies can be discussed, it is necessary to formulate the diagnosis problem by defining a lexicon of important terms, framing basic objectives, and exploring general solution strategies. In the process of examining these issues, several of the inherent difficulties that must be overcome by any viable diagnosis methodology (model errors, measurement errors, measurement delays, and incomplete instrumentation) will be considered.

2.1.1. Definitions

Fault diagnosis can be defined as the problem of identifying the root cause or causes of a process disturbance. Potential causes include equipment failures (faults), human errors, and exogenous influences. Causes from any of these three categories are referred to as *process malfunctions*¹. A *diagnosis* is a solution of the identification problem. If the root cause cannot be uniquely distinguished using the available information or procedure, then a set of likely candidates constitutes a diagnosis.

The primary objectives of any diagnostic analysis are *accuracy* and *resolution*. An accurate diagnosis is one that includes the true malfunction in the set of malfunction candidates. Resolution refers to the ability to discriminate between the true malfunction from other malfunction candidates. A highly resolved diagnosis will contain the minimum number of

¹ This definition differs from the one used by some authors who prefer to use fault or malfunction to describe the symptoms of a failure rather than the root cause.

malfunction candidates possible, given the available information. Swets (1988) presents criteria for measuring diagnostic accuracy, assuming perfect resolution (i.e. one malfunction candidate). When multiple candidates are possible, there exists a trade-off between accuracy and resolution. Perfect accuracy can be achieved with no resolution by postulating all potential malfunctions as candidates. Perfect resolution can be achieved with no accuracy by postulating a single candidate at random. The optimum diagnosis exists somewhere between these two extremes. In this work, accuracy is considered the more crucial objective and resolution is sacrificed (albeit to the minimum extent necessary) to achieve an accurate diagnosis.

Success in diagnostic problem solving depends on the effective integration of knowledge from multiple sources, including: measurements of process variables (historical, current, and predicted), models of process behavior (including malfunction behavior), and past fault experiences. The engine for integrating these knowledge sources is a logical, coherent, and robust *diagnosis methodology*. This methodology can be viewed as another source of knowledge, but is procedural rather than factual. Ideally, the methodology should be process independent to allow portability between processes.

Measurements can be derived from *sensors* or *tests*. Sensors are measuring devices that report the values of process variables continuously (limited in some cases by sampling or analysis delays). Sensor measurements are typically reported automatically to operators, and are sometimes referred to as on-line measurements. Tests are off-line measurements, usually taken manually, that are reported only in response to a specific request. Test measurements are reported slower than sensor measurements and are assumed to have a non-zero cost. Therefore, another objective of an efficient diagnosis methodology is to minimize the number of tests necessary to achieve a satisfactory diagnosis.

A *process model* is essential for two reasons. First, it is needed for fault *detection*. Detection is the task of identifying abnormal states, trends, or patterns from raw measurement data. Fault symptoms can only be detected by comparing measurements with expectations derived from a process model (i.e. normal operating behavior must be known before abnormal behavior can be recognized). Second, it can be used for diagnostic reasoning. Some models can be "inverted" to estimate process inputs (including faults) given measured outputs. Models can also be used to test hypothesized fault scenarios to ensure that the observed behaviors correspond to those expected. Two or more distinct process models can be used to perform these tasks, depending on the methodology used.

Models can be classified as being either *qualitative* or *quantitative*. A quantitative model accepts inputs and produces outputs that are real numbers. In contrast, the inputs and outputs of a qualitative model are symbolic, perhaps representing lumped numerical values. For example, a quantitative model might represent a temperature as 400 C. A qualitative model might represent the same temperature as "HOT". Quantitative models are most often expressed as mathematical equations; whereas, qualitative models are usually expressed logically. There exists a gray region of semi-quantitative, semi-qualitative models that require fuzzy logics [Zadeh (1965)].

Another method of classifying models is by their function in the diagnosis methodology. A *simulation model* is a model formulated to calculate process outputs from a set of known process inputs and can be used in malfunction detection or testing. A *diagnostic model* calculates possible process inputs from a set of known outputs and can be used to generate malfunction candidates. Typically both a simulation and diagnostic model are needed to perform diagnosis, and ideally, the same model can be used for both functions. Creating model representations independent of function is a goal of artificial intelligence research.

Process models can also be classified by the range of situations in which they apply. Models of normal operation may be applicable in limited circumstances when the process is free of malfunctions. Since the processes studied here are nominal steady-state processes, models comprised largely of algebraic constraints serve as models of normal operation. Because malfunctions will typically produce a period of non steady-state behavior before a new steady-state is achieved, dynamic models may be required to simulate (or diagnose) malfunctions.

2.1.2. Basic Strategies

All diagnosis methodologies are based on some variant of the approach illustrated in figure 2-1. The following are key features of this approach:

- A process with known inputs (X) influenced by unknown malfunctions (M),
- A simulation model that generates expected process outputs (Y_m) from known process inputs and a malfunction hypothesis (H),
- A group of sensors and tests that sample process outputs and produce measured outputs (Y_s),

- A comparator or pattern recognizer to detect differences (ϵ) between expected and measured outputs, and
- A hypothesis generator to revise the malfunction hypothesis when differences between measurements and expectations are detected.

The form and sophistication of the simulation model will be largely dependent on the process being analyzed. Dynamic or nonlinear processes require more elaborate models than linear or static processes. For example, a model consisting of a list of normal values may be sufficient for detection of malfunctions in steady-state processes. Sensor readings may contain random errors (noise), systematic errors (bias), and delay. Sensors are also subject to gross failure. If sensors contain noise, statistical techniques must be used to detect deviations from normal behavior.

In figure 2-1, two separate diagnosis loops are depicted: the *simulation loop* and the *test loop*. The simulation loop provides a method of testing the current malfunction hypothesis with on-line measurements using the simulation model. The test loop provides a method of testing the malfunction hypothesis by requesting additional information from off-line measurements. Repeated use of either loop has been referred to as the *hypothesize and test strategy* by various authors.

The operation of these loops is best illustrated by stepping through the stages of diagnosis after a fault enters the process. Prior to introduction of the fault, it is assumed that $H = M$ (i.e. the malfunction hypothesis is accurate). The fault is modeled as a modified malfunction vector M' .

- Step 1)** The fault produces changes in the process outputs and corresponding changes in measured outputs.
- Step 2)** The comparator detects differences between measurements and expectations calculated by the model.
- Step 3)** The hypothesis generator uses the observed differences to request more information (test loop) or revise the malfunction hypothesis (simulation loop).
- Step 4)** Simulations or tests are performed. Then steps 2, 3, and 4 are repeated until a satisfactory diagnosis has been achieved.

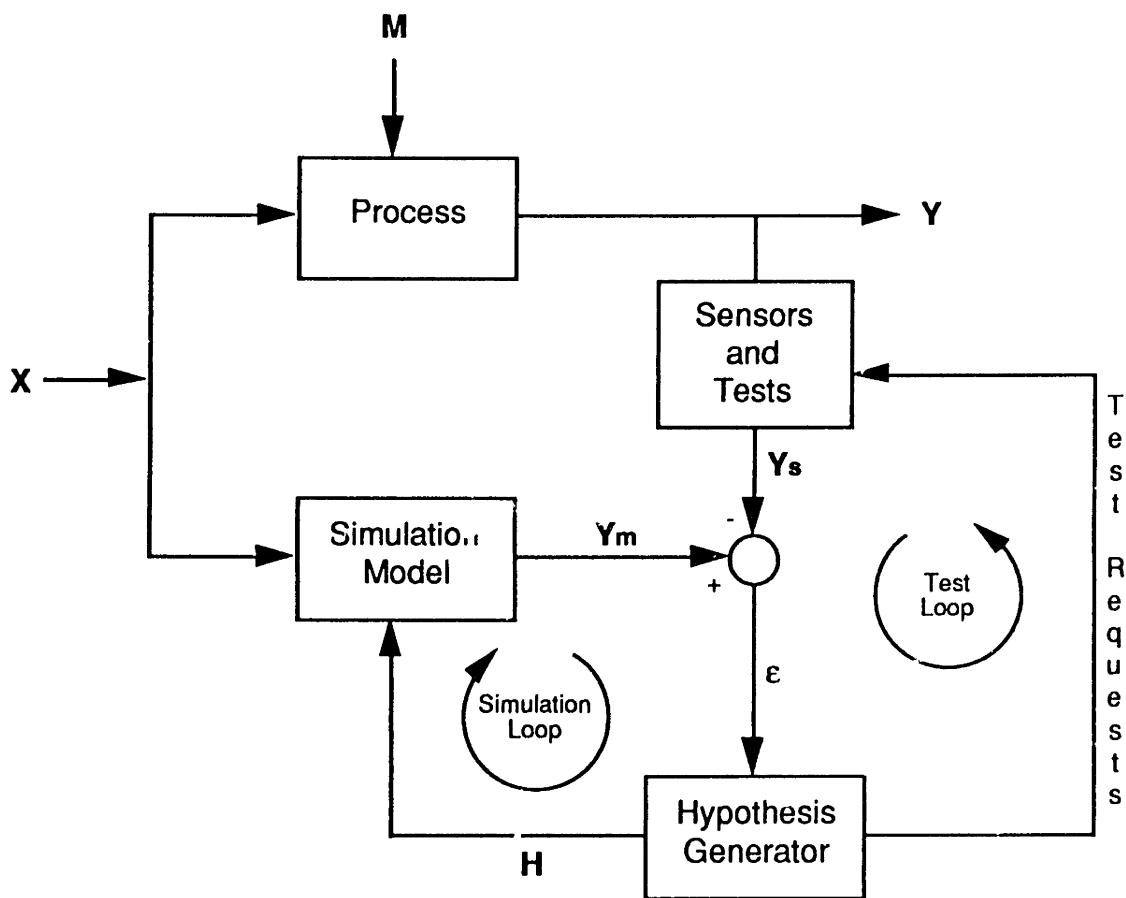


Figure 2-1: Hypothesize and Test Strategies

The goal of this procedure is to produce a revised malfunction hypothesis H' with the property:

$$Y_m(X, H') = Y_s(X, M') \quad (2-1)$$

A hypothesis that fits this criteria is called a *valid hypothesis* and can explain all observed measurements. A unique valid hypothesis is never guaranteed. Approximations in the simulation model, nonuniqueness of faults, or the lack of key sensors and tests may produce situations in which multiple valid hypotheses exist. Each valid hypothesis is a candidate for the true malfunction, and in these situations, the diagnosis problem cannot be considered solved until all valid hypotheses have been identified.

The design of the hypothesis generator is the paramount issue in fault diagnosis. A "dumb" hypothesis generator might need to simulate all possible malfunctions and use all available tests to locate the set of valid hypotheses. This approach necessitates many passes through the simulation and test loops. In contrast, a "brilliant" hypothesis generator would use a complete inverted process model to produce all valid hypotheses without any passes through the simulation loop and with minimal tests. A example of a brilliant hypothesis generator is developed below.

Example 2-1

In this example, a hypothesis generator is developed for the simple process illustrated in figure 2-2.

When a new malfunction M' enters the process, the following relations can be derived:

$$Y_s = AX + BM' \quad (2-2)$$

$$Y_m = A^*X + B^*M' \quad (2-3)$$

$$\epsilon = Y_m - Y_s \quad (2-4)$$

where

$$\begin{aligned} A &= G_s G_2 G_1 & A^* &= G_s^* G_2^* G_1^* \\ B &= G_s G_2 G_f & B^* &= G_s^* G_2^* G_f^* \end{aligned}$$

By solving equation (2-4) for M' , the following expression can be derived:

$$M' = B^{-1}(A^* - A)X + B^{-1}B^*H - B^{-1}\epsilon \quad (2-5)$$

Equation (2-5) assumes that an inverse exists for B . The process transfer functions can be replaced with the simulation model transfer function, assuming that the model is accurate. This substitution reduces equation (2-5) to the following:

$$H' = M' = H - B^{*-1}\epsilon \quad (2-6)$$

For this system, an exact revised malfunction hypothesis can be calculated using equation (2-6) from knowledge of the old malfunction hypothesis, the process model, and the difference vector. No simulation or testing is required.

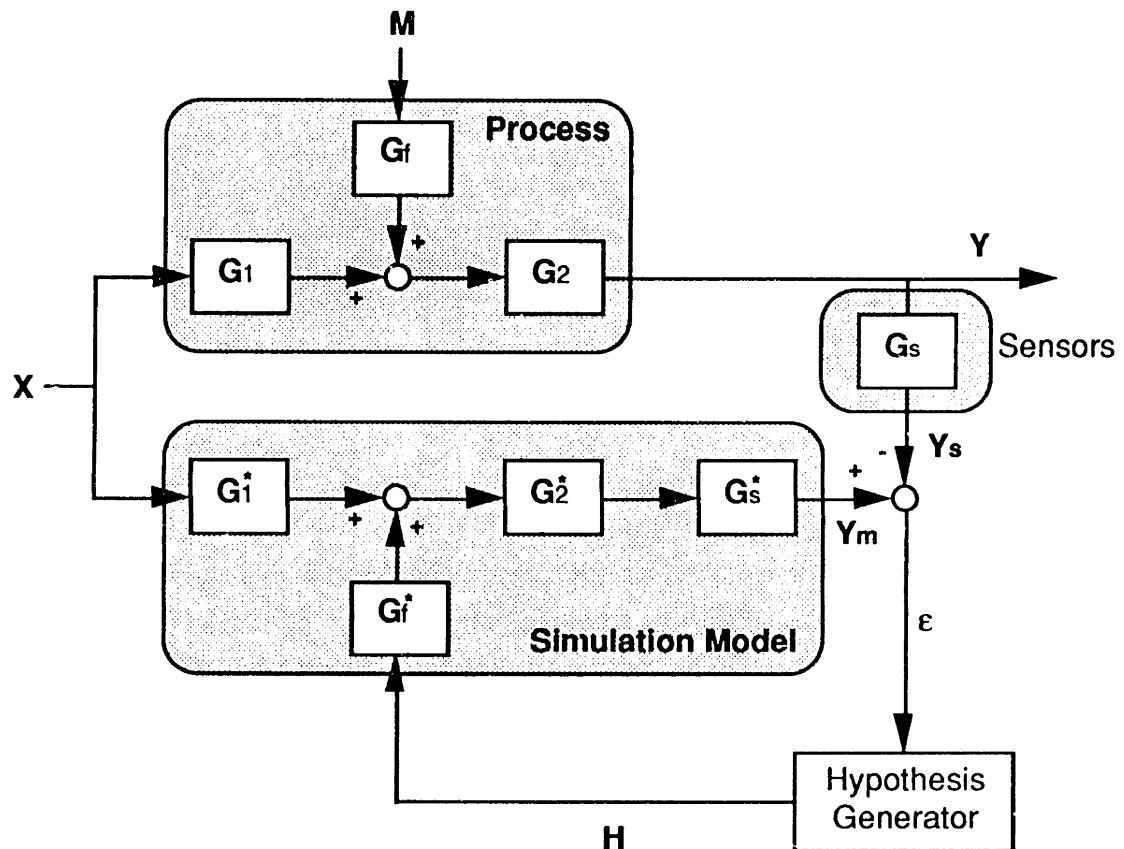


Figure 2-2: Linear Process for Example 2-1

Although the brilliant hypothesis generator is the ideal to which all automated fault diagnosis systems aspire, it is seldom achievable in real processes. More often, one must search for merely "intelligent" hypothesis generators that use approximate or incomplete inverted models or heuristics. Human operators fall in this category. Since intelligent hypothesis generators (IHGs) are by definition suboptimal, it should not be surprising that many competing approaches to designing IHGs have been proposed. All designs strive in different ways to overcome the inherent difficulties of the diagnosis problem.

2.1.3. Basic Difficulties in Diagnosis

What follows is a list of fundamental problems frequently encountered in the diagnosis of chemical and refinery processes:

- Nonuniqueness. Two or more possible malfunctions may give rise to the same set of observed symptoms. Alternately, a single type of malfunction (e.g. a pipe rupture) may produce remarkably different symptoms depending on the severity and speed of onset of the malfunction. This problem can be reduced (for a cost) by adding sensors, but the level of instrumentation necessary for its elimination is prohibitive in large scale chemical operations.
- Initial diagnosis must be performed with available on-line instrumentation, usually designed for normal process monitoring and control and not ideal for diagnosis. Critical shutdown decisions may need to be made before test measurements become available.
- Detailed models of process behavior may not exist, particularly in the presence of malfunctions. Models that do exist were probably developed for control or design, not diagnosis. Because faults are rare, past failure experience may not be available or may be outdated due to process modifications.
- For many processes, it may not be possible to vary inputs (e.g. introduce an pulse, sinusoid, or tracer) during operation solely for diagnostic purposes. For processes that allow active testing, the size of allowable pulses and the delay before onset of additional symptoms may not be ideal for on-line diagnosis. Restricting active testing modes reduces the options available for diagnosis and eliminates frequency response analysis as a tool for diagnosis of many systems.
- Sensors may contain systematic bias and random error and are subject to complete failure. Statistical techniques used to detect changes in process variables can produce false alarms (false positive indications) or latent alarms (false negative indications).
- Certain failures may produce a "cascade" of hundreds of alarms in a matter of minutes, making evaluation difficult [Kragt and Bonten (1983)]. One of the key tasks faced by any diagnostic system is to separate primary alarms (caused by the fault) from secondary

alarms (caused by downstream propagation of the resulting disturbance). This problem is often referred to as *alarm filtering*.

- Sensors and tests are subject to varying amounts of delay. For example, temperature may be reported 10 times per second; whereas, concentration may be reported once every 30 minutes. Sensors can catastrophically fail making certain measurements unavailable.
- Multiple malfunctions may exist simultaneously. Occasionally, multiple independent malfunctions will occur coincidentally. More often, certain malfunctions can be induced by the presence of other malfunctions. For example, sensor failures can often be induced by malfunctions that change process conditions and damage equipment.

Several of the difficulties encountered in diagnosis problems can be illustrated by examining the assumptions that were necessary to develop the brilliant hypothesis generator in example 2-1. One vital assumption used to develop equation (2-6) was that the simulation model transfer functions (A^* and B^*) were exactly equal to the process transfer functions (A and B). In practice, models will always contain errors and approximations that will defeat this assumption and introduce errors and uncertainty into the diagnosis. It was also assumed in equation (2-5) that the inverse matrix B^{-1} exists, implying the existence of inverses for the G_S , G_2 , and G_f matrices. The existence of these inverses is not guaranteed and is doubtful for most large scale chemical processes. Chemical processes are seldom fully instrumented; therefore, G_S can be expected to be rank deficient and lack an inverse. G_f may also lack an inverse, since several malfunctions may produce identical effects (i.e. the nonuniqueness problem).

Such serious problems with a very simple example portend the difficulties encountered with more realistic systems. The process in example 2-1 does include many features that can be considered essential in dealing with realistic problems. The most important of these features are noise in sensor measurements, sensor failures, malfunctions that change process transfer functions, multiple points at which malfunctions can enter the process, nonlinear processes, measurement of intermediate outputs, and extensive feedback and recycle. In the general case, these problems make the diagnosis problem sufficiently complex and that closed-form solutions such as equation (2-6) are impossible to derive. The next section begins an examination of competing approaches to the design of IHGs. The starting point for the search is the human operator.

2.2. Human Diagnostic Methodologies

Unfortunately, there are no detailed models of the human thought process, so research has been limited to examining input-output relations and postulating simple mechanisms. Despite a moderately extensive effort in this direction, the current understanding of the mechanisms of human diagnostic reasoning remains quite poor from an engineering perspective.

In this section, certain theories of human reasoning are examined in an attempt to determine the strengths and weaknesses of human diagnosis and identify techniques suitable for automated diagnosis systems. This examination will focus on theories relevant to the work presented in this thesis. Although the subjective study of the human reasoning process has had some impact of automated diagnosis systems, more concrete information is available on human diagnostic performance. The findings of these performance studies can be used to frame the objectives of automated systems designed to work in concert with human operators.

2.2.1. Human Diagnostic Reasoning

Can the mind understand its own reasoning processes? This seemingly philosophical question has real implications in the study of human diagnosis methodologies. Because existing technology does not provide an unobjective observer of the human thought process, researchers must rely on the comments and reactions of a subject to understand what and how he is thinking. Consequently, such research lacks the concrete results to which engineers are often accustomed. Despite such fundamental questions researchers continue to develop theories of human reasoning by examining human behavior, by making seemingly reasonable assumptions, and by tracing the apparent steps involved in accomplishing a mental task.

A plausible theory, proposed by Rasmussen (1981) (1983), that is useful in subsequent classification of automated fault diagnosis methodologies holds that human behavior derives from three different levels of reasoning: skill-based, rule-based, and knowledge-based. Skill-based reasoning is automatic and involves little conscious thought. Rather, it is the rote recitation of a practiced action or series of actions. In contrast, rule-based reasoning involves a conscious effort in identifying recognizable patterns in a possibly ill-defined set of signs and applying associations (rules) derived over time though experience. Knowledge-based reasoning is the most cognitively stressful level and involves deriving goals, planning, and making decisions using a *mental model* that explicitly represents the system under analysis.

Rasmussen contends that when confronted with a task, the human will prefer to use skill-based, rule-based, and knowledge-based reasoning, in that order. With training, a task performed by knowledge-based reasoning can become a rule-based task or a skill-based task. An implication of this theory is that humans are essentially "cognitively lazy" and will resort to mental models to solve problems only when no shortcuts methods are available.

An example is the process of learning a new language. At first, one must resort to a memorized translation model to convert phrases from a known language into the new language. With time, the phrasing of the new language may become more intuitive. At this stage the speaker is relying more on rule-based reasoning. Eventually, one begins to "think" in the new language. At this point, speaking has become a skill-based task.

It is unusual for diagnosis to attain the skill-based reasoning level, especially for complex systems, since failures are relatively rare. However, it is reasonable to assume that some (more experienced) operators will rely predominantly on heuristics and other (less experienced) operators will rely largely on mental models. Since experienced operators are usually more proficient than inexperienced operators, one is tempted to conclude that rule-based reasoning is better than knowledge-based reasoning when applied to diagnostic problems. This conclusion is premature. The differences between experienced and inexperienced operators can be explained in several ways:

- Experienced operators have more highly refined mental models,
- Experienced operators have two types of reasoning available, where inexperienced operators have only one, and
- Experienced operators may simply reason more quickly and with a higher level of confidence than inexperienced operators.

Even the most experienced operators must resort to mental models to diagnose novel faults where no rules apply.

This is not the only theory on human diagnostic reasoning. However, the concept of multiple levels of reasoning based on rules and models has been a basic principle of automated fault diagnosis research, and the relative merits of rules and models in human diagnosis can be used as a guide to the use of rules and models in automated systems.

Others actively studying human diagnostic reasoning include Rouse (1981) (1983) who has examined human strategies in solving abstract graph search problems, and Duncan (1981) (1987) who has evaluated the effectiveness of diagnosis training in process plants.

2.2.2. Mental Models

What kind of models do humans use to solve problems, how are these models represented, and can they be improved or adapted for automated systems? Again, it is difficult to treat these questions rigorously, but certain characteristics of mental models are well accepted:

- Mental models are incomplete representations of reality (like any model),
- Mental models are qualitative rather than quantitative in nature,
- Mental models are based on perceived connections between mental concepts or objects [Williams et al. (1983)],
- Mental models can be improved by experience or training, and
- Mental models can exist at several levels of abstraction with more advanced models generally containing more abstract concepts [DiSessa (1983)].

A significant effort has been undertaken to understand mental models in more detail. One approach has been to study how non-experts reason about simple physics problems. These so called "naive physics" experiments [Hayes (1979)] have spawned great interest in the qualitative modeling arena. De Kleer and Brown (1984) (1986), Forbus (1984), Iwasaki and Simon (1986), and Kuipers (1984) (1986) (1987) have all developed qualitative modeling techniques. How the qualitative models proposed by these authors are related to human mental models remains a controversial subject, however, their methods have proven useful in developing qualitative models for automated diagnosis systems.

2.2.3. Evaluation of Human Diagnostic Reasoning

Even without a complete understanding of the human reasoning process or mental models, it is possible to evaluate human diagnostic performance. It is generally accepted that with sufficient time and information, a human can solve almost any diagnostic problem. Of most interest is human performance in situations of minimal information and time limits like those encountered in diagnosis of chemical and refinery operations. Not surprisingly, humans

demonstrate wide variations in ability under these circumstances, but several consistent reasoning biases have been observed:

Probability Bias

Humans generally have poor intuition when asked to estimate probabilities [Tversky and Kahneman (1974)]. This deficiency stems from an insensitivity to prior probabilities, sample size, and dependencies between distributions. This difficulty in estimating probabilities results in two other phenomenon called *availability bias* -- the tendency to associate a higher probability with situations that are more easily remembered (i.e. more recent or memorable) -- and *false causal inference* -- the tendency to perceive a causal relationship between any two occurrences that are similar or occur close together in time or space regardless of whether a physical relationship exists.

Overconfidence

Humans are generally overconfident in their assessments. That is, they typically overestimate the plausibility of their initial hypotheses [Mehle et al. (1981)].

Confirmation Bias

Humans will preferentially seek evidence that confirms existing hypotheses and avoid evidence that may refute existing hypotheses [Arkes and Harkness (1980), Sheridan (1981)].

Negative Evidence Bias

Humans generally have difficulty using the absence of symptoms in diagnostic reasoning [Yoon and Hammer (1988)], when in many situations, the lack of symptoms may provide more diagnostic information than the presence of symptoms.

Self Limiting Hypothesis Sets

Humans will try to keep the set of active hypotheses as small as possible. Therefore, the likelihood of adding a new hypothesis to an existing set is inversely proportional to the size of the set [Gettys and Fisher (1979), Rouse and Hunt (1984)].

All these biases produce suboptimal performance and are present in varying degrees in both expert and novice diagnosticians. Whether the source of these biases is overreliance on heuristic rules or flawed mental models remains unclear, but these biases do represent specific problems to be guarded against in developing automated diagnosis systems. Lees (1974) provides an extensive list of additional research on human operator performance.

2.2.4. Human Diagnostic Performance and Automated Diagnostic Systems

Since there are two types of operators -- operators who rely on knowledge-based reasoning and mental models and operators who rely on rule-based reasoning -- there exist opportunities for two types of diagnostic aid. Because of its greater cognitive requirements, knowledge-based reasoning is slower than rule-based reasoning. Therefore, an aid for operators using mental models might be aimed at speeding diagnosis using rules derived from expert operators. An aid for operators using rules would have different objectives -- primarily guarding against human reasoning biases and errors caused by excessive reliance on heuristics.

As pointed out by O'Shima (1983), most serious operational failures are the result of operator errors. Failure to recognize and correctly diagnose problems or misdiagnosis resulting in inappropriate actions are the major sources of operator error. The objective of automated systems is to reduce operator error. To achieve this objective, an aid should be designed to compliment the operator in those areas where the human reasoning process is weak.

A third type of operator aid, useful for all operators, would synthesize operating procedures and provide operator advice. For an "operator advisor" to perform well, it will need to know the current state of the plant. Therefore, a reliable diagnosis is an essential prerequisite. Providing operator advice is beyond the scope of the work presented here and is viewed as a separate problem. It has been studied by other researchers, including Fusillo and Powers (1987), Yufik and Sheridan (1986), and Long (1984).

2.3. Automated Detection and Diagnosis Methodologies

In this section, the major techniques of automated fault detection and diagnosis are reviewed. Methods are grouped into six major categories -- parameter estimation, pattern classifiers, neural network, control charts, expert systems, and graph methods -- but the distinctions between categories are sometimes fuzzy and some methods incorporate features that cut across category boundaries. For each method, the type and format of the process model required (if any) will be briefly described and the method's primary function (i.e. detection or diagnosis) will be indicated. Himmelblau (1978) and Pau (1981) provide a broad introduction to this area.

2.3.1. Parameter Estimation

Parameter estimation refers to an entire class of methods that can be used for fault detection and diagnosis but have also seen wide application in traditional control schemes. The basic concept behind estimation is to use available measurements in combination with a model of the underlying process to estimate the values of uncertain or unmeasured process state variables or model parameters. Because measurements are assumed to be random variables, statistical tests and techniques are central to estimation methodologies. Estimation techniques are surveyed by Gertler (1986) and Isermann (1984).

Estimation can be used for detection or diagnosis. Effectiveness in these tasks depends in large part on the state variable or parameter chosen to estimate. For example, a scheme could be devised to estimate the total heat flux (Q) in a heat exchanger, given by equation (2-7).

$$Q = UA\Delta T_{ln} \quad (2-7)$$

A low heat flux would be indicative of a small temperature difference (ΔT_{ln}) possibly caused by a low hot stream temperature or a high cold stream temperature, a low heat transfer coefficient (U) caused by fouling, or plugged tubes which decrease available transfer area (A).

While a low heat flux can indicate the presence of a malfunction, it does not provide enough information for a complete diagnosis since there are three possible causes. A better choice is to estimate the heat transfer coefficient directly. A low value of this coefficient can be associated with a single cause -- fouling. In general, estimation of model parameters will provide more direct diagnostic information than estimation of state variables, although it may not always be possible due to a lack of necessary instrumentation or accurate models.

Parameter estimation techniques exist for models expressed as linear or nonlinear algebraic equations, ordinary or partial differential equations, or difference equations. Therefore, a process can be modeled in the most appropriate format and the estimation technique chosen to fit that format. For linear algebraic equation models, regression techniques (e.g. least-squares) can be used to estimate parameter values. For nonlinear algebraic systems, more general optimization algorithms (e.g. gradient methods) are needed. Differential equation models are typically solved using discrete or continuous filters or transfer function methods. These approaches will be discussed in more detail in the following sections.

2.3.1.1. Data Reconciliation

A persistent problem in process plants is the inability to close material and energy balances due to measurement errors. Data reconciliation is a technique that uses process measurements in conjunction with a quantitative process model, expressed as a series of constraint equations, to estimate true values for each measured variable. In this capacity, data reconciliation can be used for detection of certain process malfunctions. The reconciliation technique can be adapted to diagnose malfunctions that result in violation of model constraints by systematically identifying constraints that produce large inconsistencies and postulating the existence of malfunctions that can result in violation of those constraints. Most work in this area has focused on linear or linearized constraint systems, but extensions to nonlinear constraints are possible. An advantage of the technique is the ability to handle large sets of constraints, and therefore, diagnose on a plant wide scale.

The reconciliation problem, as it is usually formulated, is to estimate a vector \mathbf{u}_1 of measured and a vector \mathbf{u}_2 of unmeasured (but determinable) parameters. Measurements \mathbf{x} contain random errors ω

$$\mathbf{x} = \mathbf{u}_1 + \omega \quad (2-8)$$

where \mathbf{u} is subject to a process model comprised of linear constraints

$$\mathbf{A}_1\mathbf{u}_1 + \mathbf{A}_2\mathbf{u}_2 = \mathbf{0} \quad (2-9)$$

Here, \mathbf{A} is a matrix of known constants that are the coefficients for the linear constraints. The constraints provide "analytical redundancy" in certain measurements. Least-squares estimation is used to minimize the difference between model estimated and measured parameter values.

Several researchers have described methods of detecting and identifying gross errors (faults) using data reconciliation methods, including Crowe (1988), Rosenberg et al. (1987), Iordache et al. (1985), Tamhane and Mah (1985), Mah and Tamhane (1982), Romagnoli and Stephanopoulos (1981), and Mah et. al (1976).

2.3.1.2. Filtering

Filtering is an estimation technique used when model equations are linear or nonlinear differential equations. Kalman filters [Kalman (1960)] can be used to estimate states in linear ordinary equation models. Stanley and Mah (1977) describe the use of the Kalman filter in estimating flows and temperatures. Griffin and coworkers (1988) apply Kalman filtering for incipient fault detection. For nonlinear differential equation models, the extended Kalman filter or other nonlinear filters must be used. Watanabe and Himmelblau (1983a) (1983b) (1984) discuss the issues involved in applying filters to nonlinear process models. Kalman filters have seen wide application in control.

The result shown in equation (2-6) is essentially a discrete linear filter for estimating \mathbf{M} . The same problems associated with the filter in equation (2-6) confront other filters in general. That is, the high degree of instrumentation necessary makes application of this technique difficult on a plant-wide scale.

2.3.2. Pattern Classifiers

A *pattern*, in the context of detection and diagnosis, is a point in a generalized N dimensional space of possible measurements and observations. Pattern classifiers divide this space into various regions, with the boundaries between regions being referred to as *decision surfaces*. Decision surfaces are chosen so that all patterns within a given region share one or more specified characteristics. The differences between classifiers stem from the variety of techniques they employ to establish decision surfaces. Once decision surfaces are available, classifying patterns is a straightforward process of determining on which side of each decision surface a pattern lies. Duda and Hart (1973) provide an introduction to various classifier methods.

With sufficient knowledge of the problem domain (i.e. a detailed model or rules), decision surfaces can be established a priori. It is more common, however, for decision surfaces to be derived from an analysis of case examples. The use of examples eliminates the need for detailed process models or expertise. Classifier research is predominantly focused on developing efficient and effective means of case analysis.

Another means by which domain knowledge can be incorporated into the analysis is in the development of a specialized *feature space*. Rather than develop decision surfaces in measurement space where the surfaces may be highly convoluted, patterns can be transformed in a feature space where decision surfaces are less convoluted and easier to establish. This transformation can involve algebraic or other functional combinations of measurements.

Classifiers can be developed for detection or diagnosis. A detection classifier will transform numerical data into discrete symptoms that can be used to drive a separate diagnostic analysis such as an expert system or fault dictionary technique [Berenblut and Whitehouse (1977), Kramer (1987b)]. A diagnosis classifier will convert numerical data into fault hypotheses, essentially incorporating the fault dictionary in its decision surfaces.

2.3.3. Neural Networks

The driving concept behind neural networks is the emulation of the human brain. It has been observed that the human brain consists of a vast number of highly connected neurons that can stimulate and be stimulated by surrounding neurons. In a neural network, a large number of individual microprocessors are connected in such a way that they can communicate and interact with nearby processors. The algorithms such an architecture supports may have many advantages over conventional algorithms in solving certain basic computational problems, including the search problems encountered in diagnosis [Hillis (1985)].

At the present time, such massively parallel architectures are not commonly available. Therefore, the term neural network is usually applied to an artificial neural network (ANN) -- a parallel network simulated on a serial processor machine. All the purely computational advantages of true neural networks are lost in artificial neural networks. As currently implemented, ANNs simply represent an alternative algorithm for determining decision surfaces in multi-dimensional spaces. A detailed description of ANNs is beyond the scope of this discussion and the reader is referred to Cowan and Sharp (1988) for introduction to this area.

Applied to engineering systems, an ANN is a "black box" that receives inputs and produces outputs. ANNs can be trained from examples to recognize input patterns and respond with characteristic output patterns, making ANNs a new class of pattern classifier. The training algorithms for ANNs are discussed by McClelland and Rumelhart (1986) and Kramer and

Leonard (1989). Hoskins and Himmelblau (1988) discuss the application of neural networks to diagnosis problems.

2.3.4. Control Charts

Control charts are a graphical means of charting process data to detect changes in operating characteristics and have a long history of application in manufacturing and quality control. MacGregor (1988) provides review of control chart techniques.

The primary task of a control chart is to test the null hypothesis $\mu = \mu_{ss}$ versus the alternative hypotheses $\mu > \mu_{ss}$ and $\mu < \mu_{ss}$. There are two common types of control chart: Shewhart [Shewhart (1931)] and Cusum. Each type has certain special characteristics, but for most purposes are functionally identical.

Shewhart charts plot the mean value and range of process data samples. The null hypothesis test uses control limits that define a confidence region of known statistics. If a sample mean or range fall outside the control limits, it is concluded that the null hypothesis is no longer valid. Other tests for significant trends, changes in noise characteristics, and excessive process variability are also commonly performed. Figure 2-3 is a typical Shewhart chart.

Because of the nature of random error, a certain fraction of sample means will fall outside the control limits even when the null hypothesis is true. This is the fundamental trade-off inherent in control charts: the tighter the limits placed on the process the more false indications. Himmelblau (1978) discusses methods of determining sample size and control limits to optimize this trade-off. Chan and coworkers (1988) examine the robustness on Shewhart charts when the underlying statistics are unknown.

2.3.5. Expert Systems

The name "expert system" derives from the concept that human expertise can be captured in an automated system. The expertise in question is the set of heuristic rules humans develop from experience. An introduction to expert systems is provided by Buchanan and Shortliffe (1984).

The standard rule format used by expert systems is

IF <pattern> THEN <action> (2-10)

where <pattern> (the rule *antecedent*) is a set of conditions that must be matched and <action> (the rule *consequent*) is a procedure to be initiated or a conclusion to be drawn when the antecedent is satisfied. In complex systems, it is necessary to relate the information stored in a large set of rules. Standard search algorithms (forward and backward rule chaining) have been developed for this purpose. An expert system "shell" is an computer programming environment that contains all the processing facilities of an expert system without any rules. Rules can be added to a shell to produce an expert system.

Rules provide an attractive alternative to other paradigms because they are reasonably transparent and easily understood, do not require additional modeling, and provide a convenient method of decomposing complex problems. Expert systems can be created incrementally (one rule at a time, if desired) as knowledge becomes available. Because rule patterns are usually qualitative, expert systems must be used in conjunction with a detection scheme to convert numerical data into qualitative patterns. However, an advantage of rule based systems is the lack of a detailed mathematical process model.

Clancey (1983) points out the limitations of the simple rule paradigm and suggests a more elaborate environment consisting of rules, objects, and procedures. In the latest commercial expert system shells, the basic rule paradigm has been extended and augmented by procedures, and facilities for object-oriented knowledge-bases. In these shells, rules and procedures can be viewed as a new high level programming language.

In addition to the general shells available commercially, certain shells -- notably PICON and its successor G2 -- have been specifically designed for process control applications [Rosenof et al. (1989), Moore et al. (1984)]. These shells include real time schedulers and other features for dynamic environments.

Extracting rule-based knowledge from humans is a difficult empirical process. Moreover, the knowledge extracted may be incomplete, limited to a single application, flawed, or tainted by the human reasoning biases previously discussed. More significantly, even if the transference of knowledge is successful, the expert system will never perform better than the humans from which its expertise is drawn.

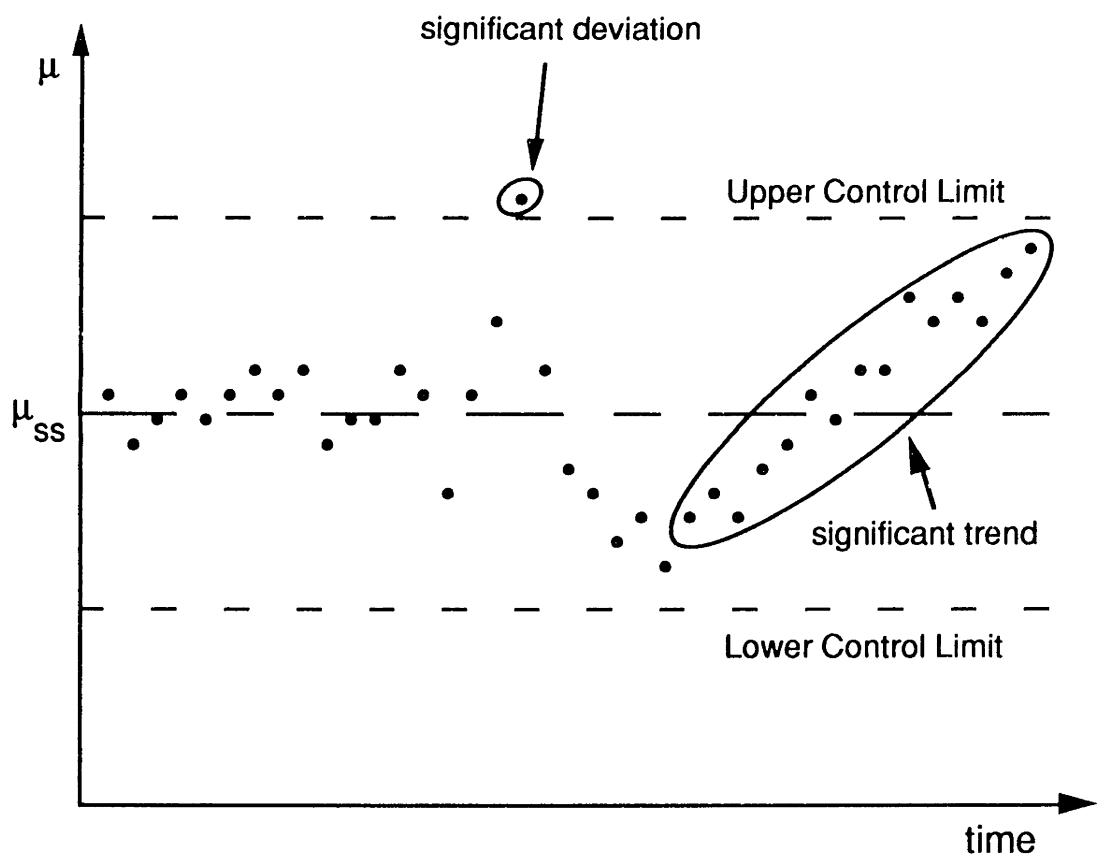


Figure 2-3A: Shewhart Control Chart for Mean

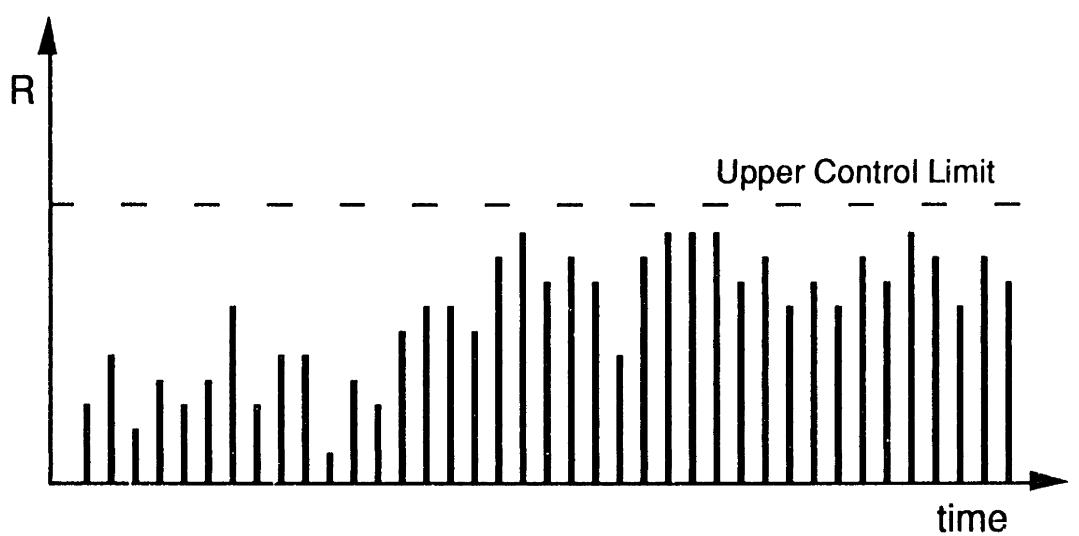


Figure 2-3B: Shewhart Control Chart for Range

Due to these difficulties, some developers attempt to bypass human experts and devise rules from the underlying models of process behavior. For example, Rich and Venkatasubramanian (1987) describe MODEX, a system that derives rules on demand from an underlying process model. The terms *shallow reasoning* and *deep reasoning* have been coined to indicate systems based on heuristics or models, respectively. The name "expert system" is applied to both types of system despite the fact that deep reasoning rules are not derived from human experts. Kramer (1986) discusses the limitations of shallow knowledge expert systems.

There has been an explosion of expert systems for fault diagnosis in recent years, so a complete listing is not possible here. Fink and Lusth (1987) describe an expert system for diagnosis of electrical and mechanical systems. Sachs and Paterson (1986) have developed ESCORT, a real time diagnostic expert system. Scarl et al. (1987) developed the LES and KATE systems. Bylander and coworkers (1983) describes CSRL, a expert system development language used by Shum and coworkers (1988) to attack diagnosis problems in the process industries. Rowan (1988) (1986) and Dhurjati (1987) describe their experiences developing and implementing a diagnostic expert system in a commercial process plant. Chung and Modarres (1986) describe an expert system to implement goal tree models. Newquist (1987), Harmon (1986), and Pau (1986) survey existing expert systems in greater detail.

2.3.6. Graph Methods

As implied by the name, graph methods utilize process models represented as graphs². Graph methods use qualitative models (quantitative models are not well represented as graphs). These methods are purely diagnosis methods and cannot be used for detection. As a result graph methods must be used in conjunction with a fault detection scheme (such as state estimation or pattern classification) to convert numerical data to qualitative patterns.

2.3.6.1. Causal Graphs

One type of graph model that has been used in fault diagnosis is the causal graph. These

² These models can also be represented as incidence matrices.

graphs are used to model causal relationships between different process objects (i.e. variables, systems, or units).

The most common form of causal graph is the signed directed graph (SDG) or "digraph". SDGs model causal relationships between process variables and parameters and consist of two primary components: nodes and arcs. A node represents a process variable or parameter and can have qualitative states of HIGH (+), NORMAL (0), or LOW (-). An arc connects two nodes to represent a local causal relationship. Arcs have signs, magnitudes, and delays that convey information about the causal interaction. If a pair of nodes is connected by an arc, the upstream node is referred to as the *initial node* and the downstream node is referred to as the *terminal node*.

The sign of an arc indicates whether the causal relationship between the initial and terminal node is positive (indicated by a "+") or negative (indicated by a "-"). In a positive relationship, the terminal node will tend to change in the same direction as the initial node (i.e. a HIGH initial node state will lead to a HIGH terminal node state). In a negative relationship, the initial and terminal nodes will change in opposite directions (i.e. a HIGH initial node state will lead to a LOW terminal node state).

The magnitude of an arc is a value (0 or 1) that indicates the qualitative gain of the transfer function between the nodes. A zero magnitude indicates that, although a causal relationship may exist, changes in the initial node will never effect the state of the terminal node. A non-zero (1) magnitude indicates that disturbance propagation along causal pathways can be important. In diagnostic reasoning, zero magnitude arcs can be ignored.

The arc delay is a qualitative assessment (0 or 1) of the propagation time between observing a change in the initial node and seeing the resulting change in the terminal node. A zero delay can be used to indicate instantaneous transmission along the arc. A non-zero (1) delay indicates that there is a positive delay between effects. Zero delay arcs should be used cautiously in process modeling since delays are sometimes factored into diagnostic reasoning. The most common use of zero delay arcs is to indicate equality relationships between variables. For example, if a digraph contained nodes for both liquid level and liquid volume in a constant area tank, these nodes could be connected by a zero delay arc.

Only a fraction of the nodes in a typical digraph represent measured variables. For these nodes, the variable value can be measured directly (or estimated if the measurements contain

random error) and converted to a qualitative state. For unmeasured nodes, states must be inferred by the diagnostic reasoning process.

Diagnosis is accomplished using a graph search algorithm to locate all nodes (measured or unmeasured) that through causal propagation could explain all abnormal states of measured nodes. Implicit in this algorithm is the single fault assumption (i.e. only one fault is present at any given time). The goal is to find a single root node that will explain the states of all nodes. More often, however, several root node candidates will be identified that cannot be further discriminated due to a lack of instrumentation.

Associated with each root node candidate will be a list of faults capable of producing the indicated deviation from normality. The final diagnosis consists of the union of the fault lists for all root node candidates. Example 2-2 illustrates the use of digraph for reasoning about process disturbances.

Digraphs are not unique representations of a process. The differences that can exist between two digraphs representing the same process include the number of nodes (variables) modeled and the manner nodes are connected. Different digraphs may exhibit distinct characteristics in diagnosing certain malfunctions. It is also possible for a malfunction to change the process in such a way that the digraph normally used for diagnosis no longer applies. Such transitions between qualitative regimes can be difficult to detect.

Digraphs have been used for diagnosis by O'Shima and coworkers [Iri et al. (1979), Umeda et al. (1980), Shiozaki et al. (1985)]. Palowitch (1987) describes digraph modeling in detail, discusses rules for deriving digraph relations from equation models, provides several examples of SDGs for common chemical process equipment, and presents the Diagnostic Expert (DIEX) a LISP algorithm for identifying root causes from digraph models. Oyeleye (1988) describes extensions of the basic SDG paradigm and presents additional guidelines for developing digraph models.

Example 2-2

Figure 2-4 is a flowsheet for a single stage fractionator process. Figure 2-5 shows a simplified digraph model for this process.

To better illustrate the use of digraphs, consider the effect of the disturbance: High Feed Flowrate.

- Effect 1) Feed flowrate (F_f) is related to liquid level (L) via a positive arc.
Therefore, level is expected to increase in response to a high feed flowrate.
- Effect 2) Level has positive arcs directed at level sensor signal (L_s) and bottoms flowrate (F_h). These variables are also expected to increase.
- Effect 3) Level sensor signal has a positive arc to level controller signal (C_l) which also increases.
- Effect 4) Level controller signal has a positive effect on bottoms flowrate, further increasing its value.
- Effect 5) High bottoms flowrate has a negative effect on level, tending to decrease its value.

This chain of disturbance propagation is shown in figure 2-6 and can be used to illustrate several characteristics of digraph models:

- Digraphs model local causal relationships. To find the output effects of an input disturbance it is necessary to follow a causal pathway through the digraph.
- Digraphs model only the initial response of a variable. Because digraphs use only first derivative models, second order effects such as inverse or compensatory response are lost.
- Digraphs produce ambiguities. In this example, there are both positive and negative effects on liquid level with no way of determining whether the final level is high, lower, or equal to its prior value.
- Digraphs are not well suited to modeling control systems. In general, it is difficult to model feedback effects using digraphs because of the ambiguities produced and the restriction to first order responses.

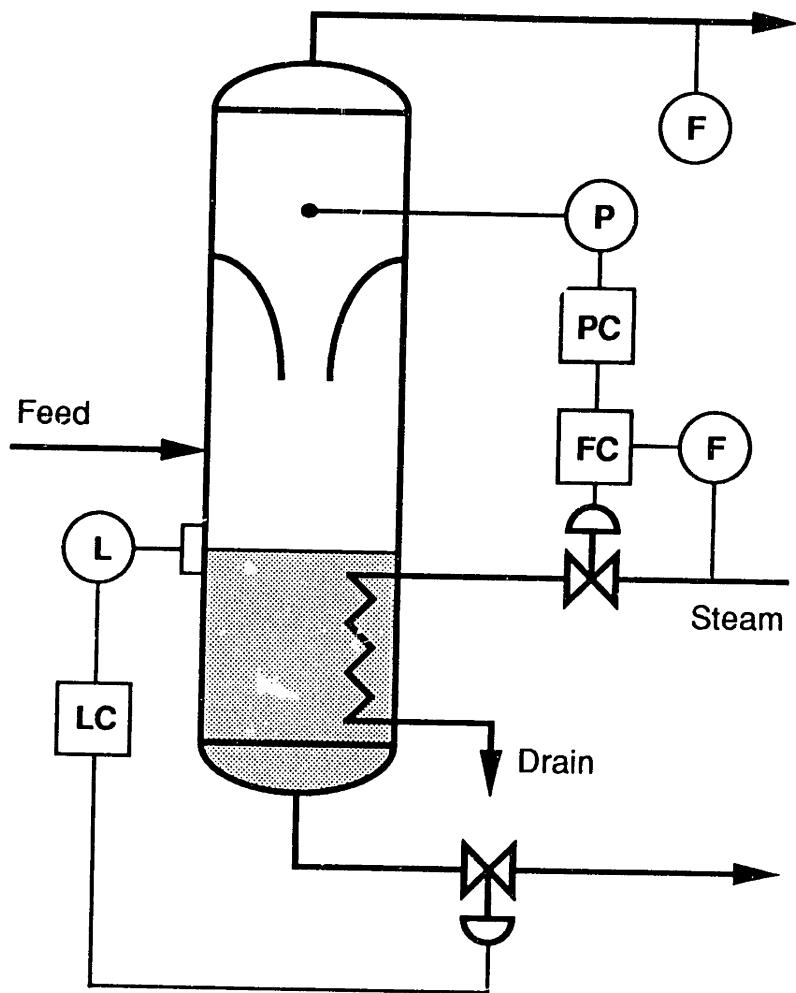


Figure 2-4: Single Stage Fractionator

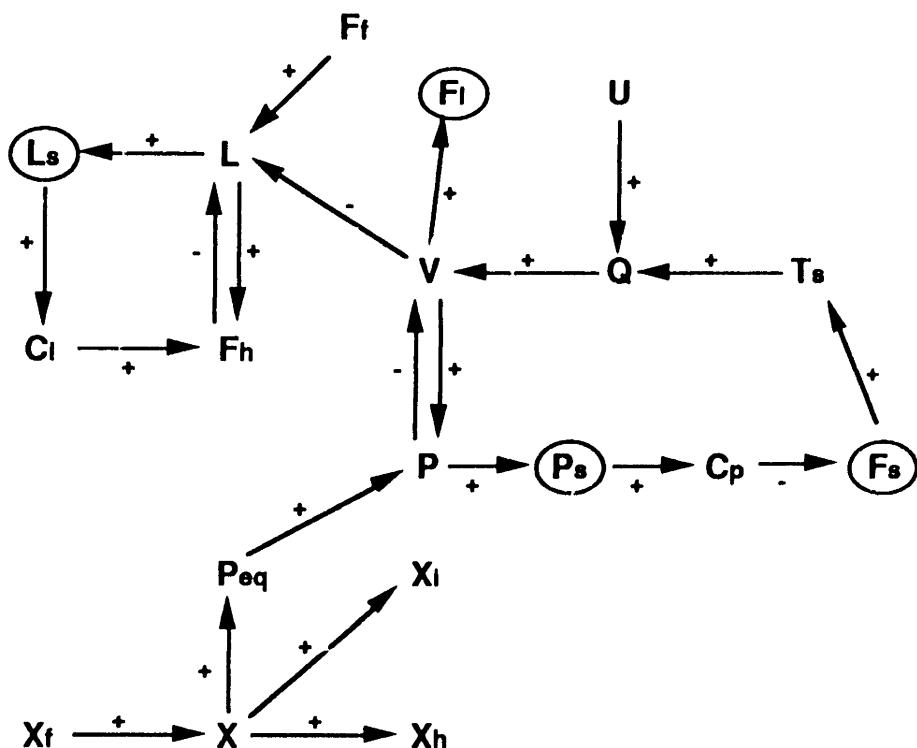


Figure 2-5: Digraph for Single Stage Fractionator

C_l	— Level Controller Signal	P_{eq}	— Equilibrium Pressure
C_p	— Pressure Controller Signal	P_s	— Pressure Measurement
F_f	— Feed Flowrate	Q	— Heat Flux
F_h	— Heavy Ends Flowrate	T_s	— Steam Tube Temperature
F_l	— Light Ends Flowrate	U	— Heat Transfer Coefficient
F_s	— Steam Flowrate	V	— Vaporization Rate
L	— Liquid Level	X	— Liquid Mol Fraction
L_s	— Level Measurement	X_f	— Feed Mol Fraction
P	— Pressure	X_h	— Heavy Ends Mol Fraction
		X_i	— Light Ends Mol Fraction

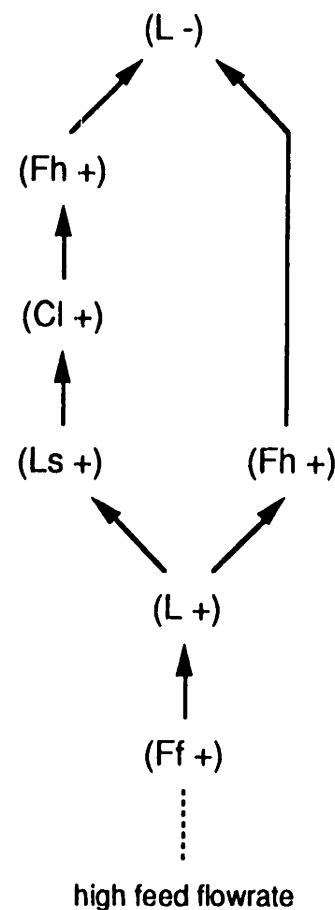


Figure 2-6: Simulation of High Feed Flowrate

Example 2-2 showed how digraphs could be used for simulating the effects of a disturbance. In example 2-3, the procedure is reversed to illustrate the use of digraphs in diagnosis.

Example 2-3

If it is observed that the vapor pressure in the fractionator shown in figure 2-4 is abnormally high, the diagnostic procedure illustrated in figure 2-7 can be used to trace backward to the root causes.

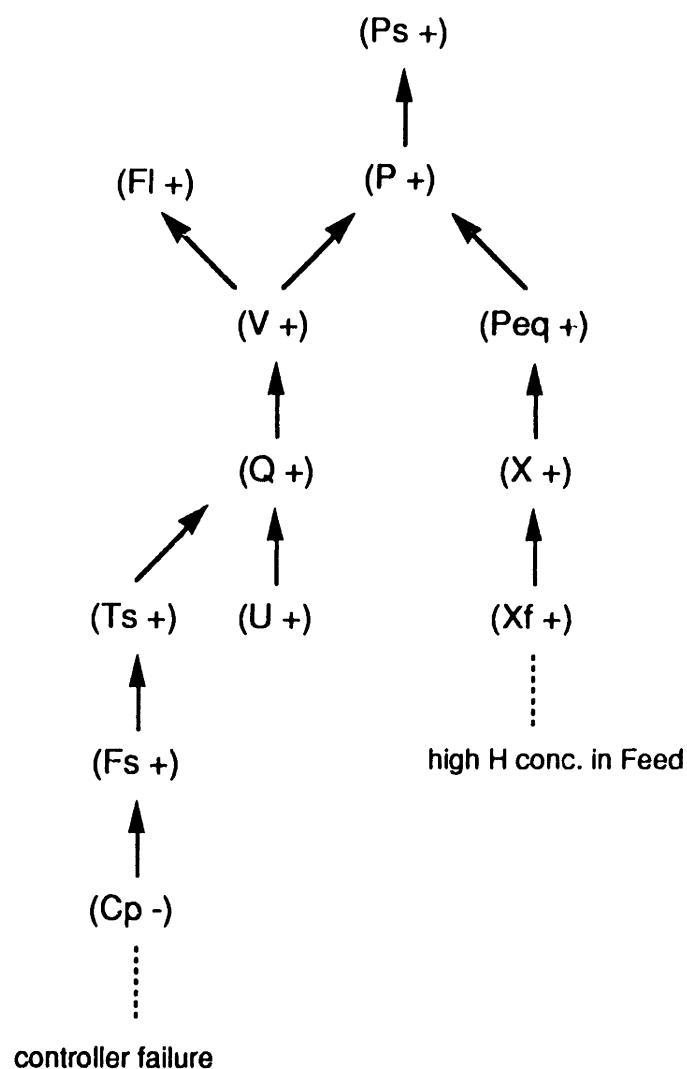


Figure 2-7: Diagnosis of P HIGH

The causes are pressure controller failure and abnormal feed concentration. Using the simulation loop, a controller failure is expected to produce a high light-ends flowrate (Fl). If this effect is observed, then controller failure is indicated. If high flowrate is not observed, then abnormal feed concentration is indicated.

2.3.6.2. Fault Trees

Fault trees are logic diagrams used to analyze the possible failure modes of complex systems. Fault trees are developed in stages by choosing a top level (observable) failure and exploring all combinations of faults, failures, and human errors that will result in the top level failure. The result is a hierarchical graph with several intermediate stages based on AND and OR logic. Andow (1985) describes how fault tree models could be implemented in an intelligent knowledge-base system. A review is provided by Dhillon and Rayapat (1988).

Fault trees have been used primarily in process design to identify high risk accident scenarios and develop safe operating procedures. Yoon (1982) and Ulerich (1988) have applied these graphs to diagnosis of faults. Modarres and Cadman (1986) have applied a modified fault tree (called a goal tree) to diagnosis.

In the diagnosis task, fault trees have the advantage of explicitly representing multiple failure scenarios. However, because many multiple failure scenarios are extremely unlikely, this feature is often of questionable value. Moreover, for large scale systems, the number of possible failure scenarios can be prohibitively large for real time analysis when all multiple failure scenarios are considered.

Another difficulty in using fault trees for diagnosis is the problem of developing the fault tree itself [Andow (1980)]. The experience of the nuclear power industry is that complete fault trees require many man-years of commitment. A related problem is fault tree maintenance which is also an important concern. In an attempt to simplify the development of fault trees, Lapp and Powers (1977), Shafaghi et al. (1984), and Kumamoto and Henley (1986) have developed methods of converting digraph models to fault trees. These efforts testify to the fact that digraphs are easier to build and maintain than fault trees.

2.3.7. Other Methods

In addition to the methods already discussed, there are several other diagnostic analysis tools in common practice. These tools are used primarily for accident analysis or risk assessment but are occasionally mentioned in the context of fault diagnosis. None are particularly suited for the on-line diagnosis problem.

Two closely related methods are FMEA (Failure Modes and Effects Analysis) and HAZOP (Hazard and Operability Study) both described by Himmelblau (1978). Each is in essence a "what if" analysis of the process, resembling the results of a combined digraph/fault tree analysis. MORT (Management Oversight and Risk Tree) is a similar technique described by Pitblado and Lake (1987). Cause-Consequence Analysis combines elements of fault tree and event tree analysis. Applications of this technique are discussed by Lees (1983).

2.3.8. Comparison of Methods

What criteria can be used to guide the selection of a diagnosis methodology? This section will attempt to answer this question by examining and comparing the characteristics of each methodology. It will be shown that each method has certain favorable and unfavorable characteristics, summarized in Table 2-1, that can make the choice of methodology difficult. No single methodology listed is universally superior in all applications.

The process knowledge necessary for diagnosis can be derived from three possible sources: process models, operating experience, and malfunction examples. In Table 2-1, methods requiring examples or experience do not require a model, and methods requiring a process model do not require examples³.

General characteristics of experience-based methods are

- Diagnosis is performed more quickly than model-based methods since no complex search algorithms are required,
- Applications are specific to the single process from which the experience or examples were derived and may not be easily transportable to other processes, and
- Explanation facilities are poor due to a lack of first principles reasoning⁴.

³ All qualitative model-based methods require a supplementary malfunction detection scheme, often a pattern classifier, neural network, or control chart. Here, methods are compared based on their diagnostic characteristics.

⁴ The exception to this rule is the shallow expert system, capable of producing good explanations if sufficient information was provided by the domain expert during system construction.

These characteristics derive largely from the fact that these methods contain no explicit deductive reasoning component. Of course, if no operating experience or malfunction examples are available for a particular process, use of the these methods is not possible.

In contrast, model-based methods typically include a general reasoning component that can be applied to many similar processes and will provide explanations of the reasoning procedure used to create a diagnosis. Also, because there is no need for operating experience, model-based methods can be developed for new processes before they become operational and are capable of handling new situation for which there is no experience or examples. The cost associated with these features is the effort required to create the model and slower on-line performance.

If a model-based methodology is chosen, the developer must decide on the modeling paradigm to be used. The first choice is whether to model the process using quantitative equations or qualitative graphs. If equation models are preferred, algebraic or differential equations (or both) can be used. Similarly, there are several competing qualitative modeling techniques that must be evaluated if qualitative models are preferred. In all cases, the developer must consider the accuracy and range of applicability of the model.

In subsequent chapters, the diagnosis methodologies presented are exclusively model-based. Focus is placed on techniques to integrate different modeling paradigms, eliminating the need to choose a single paradigm at the outset of the modeling task. Supplementary detection is accomplished using control chart techniques.

TABLE 2-1: Comparison of Fault Diagnosis Techniques

Technique	Type of Model Req'd ¹	Examples or Experience Req'd	Supplementary Detection Req'd	Modeling Effort Req'd	Explanations
Data Reconciliation	E	No	No	High	Fair
Kalman Filters	E	No	No	High	Fair
Pattern Classifiers	None	Yes	No	Low	Poor
Neural Networks	None	Yes	No	Low	Poor
Control Charts	None	Yes	No	Low	Poor
Expert Systems (shallow)	None	Yes	Yes	Low	Fair to Good
Expert Systems (deep)	E or Q	No	Yes	Moderate to High	Good
Causal Graphs	Q	No	Yes	Moderate	Good
Fault Trees	Q	No	Yes	High	Good

¹ E = Equation Model
Q = Qualitative (graph) model

3.0. Process Modeling and Diagnosis in MIDAS

This section describes in detail the fundamental process modeling techniques and diagnosis methodologies used in the MIDAS prototype diagnostic system. The discussion will build on the basic modeling and diagnosis concepts introduced in section 2.0.

The primary goal of modeling in MIDAS is to integrate the results of earlier efforts in a single unified representation that can be analyzed using a general diagnostic algorithm. Specifically, the focus has been on the integration of quantitative constraint equation models and qualitative causal graph models. In the following sections, the characteristics of each important MIDAS modeling paradigm are discussed, a unified modeling format is described, and results demonstrating the power of model integration are presented.

3.1. Qualitative Models

The primary modeling paradigm used in MIDAS is the qualitative causal model. But why use qualitative models? This question is often asked by scientists and engineers more accustomed and comfortable with fully quantitative models and must be answered before a serious discussion of qualitative modeling begins.

In science and engineering, a model is judged primarily by its predictive power -- to what degree it can predict future happenings. When judged solely by this criterion, qualitative models are inferior. Qualitative models lack predictive power and may produce ambiguities that can only be resolved by resorting to more quantitative models. Predictive power, however, should not be the sole criterion used in evaluation of engineering models. Other important criteria are the ease with which the model can be developed and modified, the generality of the model, and the ease with which the model can be used and understood. In these areas, qualitative models excel.

Diagnostic models can be expected to change relatively frequently in response to process modifications or changeovers and must be reasonably well understood by a variety of individuals to be accepted. These requirements tend to support the use of qualitative models in diagnosis. But do qualitative models provide sufficient predictive power to solve the diagnosis problem? To answer this question, one need look no farther than the mental models used for diagnosis by humans. These models are certainly qualitative rather than quantitative,

and with training, humans are excellent diagnosticians. Therefore, the assumption that qualitative models, properly formulated, are suitable for even the most difficult diagnostic tasks seems justified.

The argument remains, however, that if quantitative models are available why would one throw away information by using a qualitative model? The answer to this question lies in the concept of *minimal modeling*. Minimal modeling is the tenet that the minimum amount of information necessary for problem solving should be used in modeling. Any additional information simply wastes resources without significantly improving the final solution. For example, one would not normally perform a series of mathematical operations carrying twelve decimal places if the final answer was to be rounded to the nearest integer. The information provided by the extra decimal places is lost, and time and memory was wasted to compute them. Human reasoning seems to follow this tenet. Humans are continually trying to simplify their mental models through abstraction or by converting them to rules. In diagnosis problems, where the objective is to locate a malfunction or at least isolate it to a specific unit or subsystem, qualitative models can provide sufficient information for solution. If it is later necessary to analyze the malfunction (e.g. determine its extent or other detailed characteristics), purely quantitative methods can be employed.

3.1.1. Extended Signed Directed Graphs

The extended signed directed graph (ESDG) augments the basic SDG paradigm by including additional non-physical arcs to model certain non-local relationships. Because of their added robustness, ESDGs are used to model causality in MIDAS. What follows is a brief description of the ESDG paradigm. A complete treatment is given by Oyeleye (1988) (1989).

Three features of ESDG modeling, not present in SDGs, are self cycles, delta branches, and ESDG branches. A digraph cycle is a causal pathway that begins and ends at a single node. Cycles exist in SDGs with positive or negative composite signs¹. Negative cycles are negative feedback loops, and positive cycles are positive feedback loops. In ESDG modeling, self cycles (i.e. cycles comprised of a single arc) can have "zero" sign to indicate nodes that exhibit integrating behavior. For example, in figure 3-2A, a SDG for a simple tank,

¹ The composite sign of a causal path is computed by multiplying the signs of all arcs in the path.

illustrated in figure 3-1, is shown with a zero self cycle on level (L). The self cycle on level is included, because over time, small undetectable changes in tank inlet or outlet flow will be integrated to produce measurable changes in level. SDGs do not model this effect. In addition to modeling the effects of natural integrators such as tank level, zero self cycles are necessary to model the effects of integrating controllers. Oyeleye (1988) describes how self cycles can be derived at the time of SDG development from an analysis of underlying model equations. Final ESDG models do not include self cycles. Rather, self cycles are used as a guide for creating the delta branches described below.

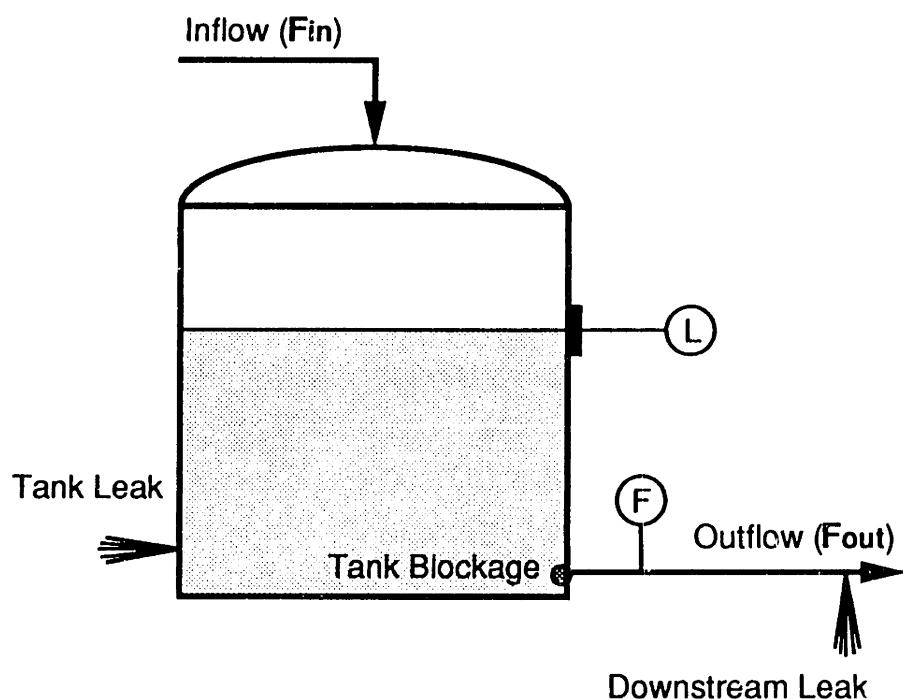


Figure 3-1: Tank Schematic

Delta branches provide a mechanism for disturbances to "jump" over apparently normal nodes. In SDG models, it is assumed that a node can have an abnormal (+ or -) state only if a fault is directly influencing the node or an adjacent node has an abnormal state that is propagating along a causal path. Oyeleye calls this the simple causal path (SCP) assumption. Nodes with zero self cycles can violate the SCP assumption. In figure 3-2B, a delta branch has replaced the zero self cycle in the SDG in figure 3-2A to produce a digraph that correctly predicts the integrating effect of level while satisfying the SCP assumption. The interpretation

of this branch is that any malfunction effecting outlet resistance can result an abnormal level without necessarily producing an abnormal outflow. The magnitude of the branch is a qualitative Dirac delta function of outlet flow (i.e. the branch is active only when outlet flow is in the normal or zero state). Note that the delta branch does not represent a separate physical causal relationship, but rather, has been added to correctly model the causal relationships already present in the original graph.

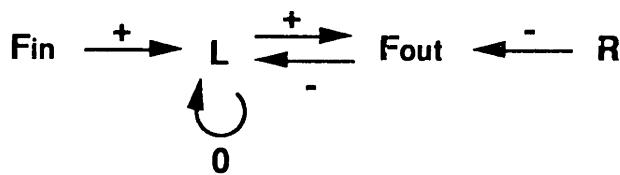


Figure 3-2A: Tank SDG

Fin	— Inflow	L	— Level
Fout	— Outflow	R	— Outlet Resistance

ESDG branches are identical to normal SDG arcs in all respects except how they are derived. ESDG branches are non-physical in that they do not represent a cause-and-effect relationship. Rather, ESDG branches are included to allow the digraph to model possible inverse response (delta branches model possible compensatory response). ESDG branches are derived from a global analysis of positive and negative SDG cycles. Oyeleye (1989) presents an algorithm for computing delta and ESDG branches from a qualitative analysis of the SDG.

ESDG models have several advantages over SDG models:

- ESDGs eliminate certain spurious qualitative behaviors predicted by SDG models,
- ESDG models can represent nonlinear behaviors such as inverse or compensatory response,
- ESDGs capture behaviors of control systems and other feedback mechanisms not possible in SDG models,

- ESDGs can predict the ultimate response of a process (not just the initial response).

These characteristics of the ESDG make it a more appropriate tool for dynamic process diagnosis than SDG models. The same diagnostic algorithms used with SDG models can be used with ESDG models.

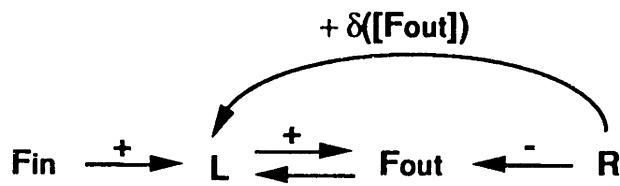


Figure 3-2B: Tank ESDG

3.1.2. Systems-Level Diagnosis

Systems-level diagnosis (SLD) [Finch and Kramer (1988)] is a causal diagnosis methodology based on a higher level of model abstraction than SDG or ESDG modeling. The objective is to model a process functionally, concentrating on the goal-directed nature of process control systems. Davis (1984) argues the necessity of functional modeling in a complete description of device behavior.

As originally envisioned, SLD would form the first stage in hierarchical diagnosis strategy aimed at narrowing diagnosis from a plant wide scale to an individual unit or subsystem before applying other methods to isolate the malfunction. Such a hierarchical structure, shown in figure 3-3, has been proposed by Rasmussen (1985). Further investigation [Kramer and Finch (1989)] discovered that the SLD modeling paradigm provided significant additional information not captured by SDG models and could be used in parallel with these methods to provide increased diagnostic resolution. This phenomenon was predicted by Milne (1987), who classifies different levels of diagnostic reasoning, including behavioral (SDG) reasoning and functional (SLD) reasoning.

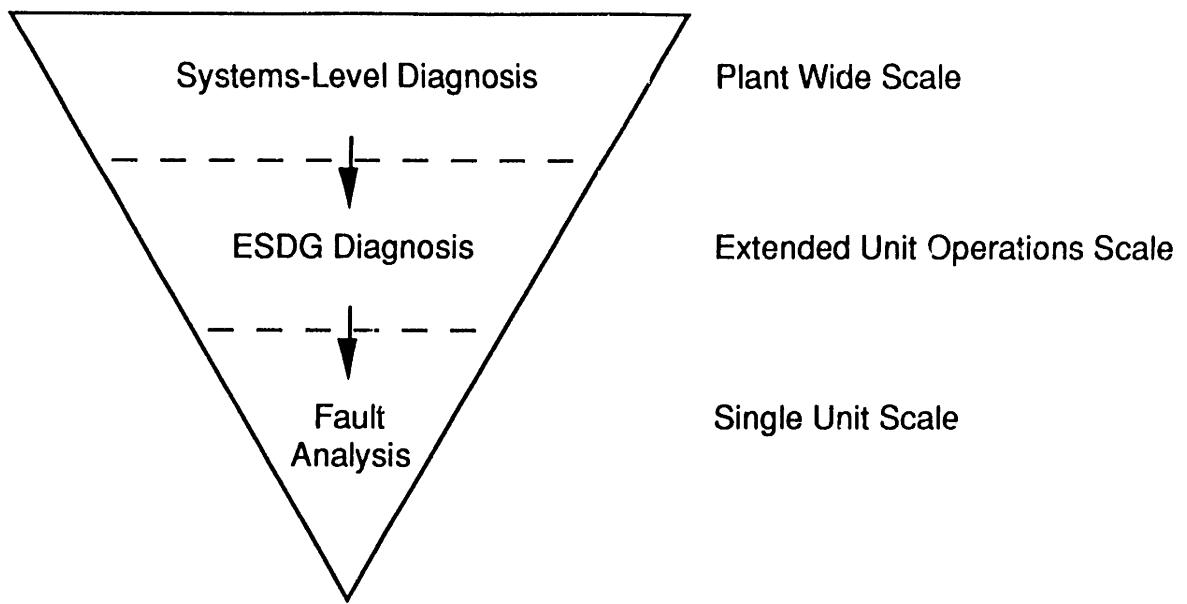


Figure 3-3: Hierarchical Diagnosis Strategy

3.1.2.1. System Modeling

Within the context of SLD, a system (or subsystem) is defined as any goal-directed set of *unit functions*, as defined in equation 3-1.

$$S_k = \{U_{ij} \mid U_{ij} \text{ vital to achieving system goals}\} \quad (3-1)$$

Here, S_k is system k and U_{ij} is the jth unit function of unit i. A unit function is any of the responsibilities a unit performs. For example, a heat exchanger has five major unit functions: heat transfer, shell-side fluid containment, tube-side fluid containment, shell-side fluid conductance, and tube-side fluid conductance. The unit functions of any given unit can belong to different systems. Physical unit boundaries are not relevant in functional modeling.

The unit function sets of various systems are not disjoint. That is, systems may share certain unit functions as expressed in equation (3-2).

$$S_k \cap S_m = \{U_{ij} \mid U_{ij} \in S_k \text{ and } S_m\} \quad (3-2)$$

When a malfunction occurs that effects a shared unit function, all systems that rely on the unit function are affected. Therefore, the notion of a single primary deviation used in digraph models is not valid in SLD modeling.

Malfunctions can affect certain unit functions while leaving others unaffected. For example, a tube blockage in a heat exchanger degrades the tube-side conductance and heat transfer, leaving other functions unaffected. A tube leak degrades tube-side containment but may enhance the heat transfer. Exchanger fouling primarily effects heat transfer, although conductance may be effected if fouling is severe. Since a system depends on all its unit functions to varying degrees to meet its goals, malfunctions degrading the effectiveness of a unit function can result in a system failing to achieve its objectives -- a system failure.

The goal of a system is to maintain one or more specified process variables (called the *controlled variables*) at desired (setpoint) values. Systems can be classified by the strategy they employ to achieve this objective. *Passive (open-loop) systems* achieve their goals by relying solely on the proper operation of their constituent unit functions. *Control (closed-loop) systems* rely on their unit functions, but can also vary the values of one or more *manipulated variables* to achieve their objectives. Maintaining all manipulated variables within a regime of normal operation can be considered a secondary goal of a control system, achievable only when the system is undisturbed. If unable to achieve both primary and secondary goals, a control system will pursue its primary goal exclusively. The work described below has focused entirely on SISO control systems and single output passive systems. In a MIMO control system, there will be multiple primary and secondary goals that may be pursued in a flexible manner.

Because of their additional capabilities, control systems are significantly more robust than passive systems. Control systems can continue to meet their primary objectives in the presence of malfunctions (assuming the malfunctions do not exceed the capacity of the system to compensate); whereas, a passive system will fail if any malfunction affecting its unit functions is present.

The two level mapping of malfunctions to affected unit functions and unit functions to systems defines one part of the SLD model. This mapping is constructed by performing a functional decomposition. The second part of the SLD model is an interaction diagram modeling intersystem dependencies. The *process graph* is a causal network of nodes and arcs similar to a digraph. Unlike a digraph, however, nodes represent the process systems and

arcs represent modes of system interaction. A process graph can be derived from a digraph model given system definitions. A process graph example is presented in Appendix 3.

Most process systems can be influenced by other systems; therefore, the ability of one system to achieve its goals may effect the success of other systems. A system S_k is said to be *dependent* on another system S_m if failure of S_m to meet its goals can effect S_k . In the process graph, the dependency relationship between two systems is represented as an arc with the dependent system as the terminal node. If the initial node in a dependency relationship is a control system, then two modes of dependency are possible. A primary or controlled variable dependency exists when the terminal node depends on the primary goal of the initial node. A secondary or manipulated variable dependency exists when the terminal node depends on the secondary goal of the initial node. A system that depends on both the primary and secondary goals of another system is termed unconditionally dependent.

In summary, the ability of a system to achieve its objectives can be effected by two factors: the ability of its own unit functions to perform their respective responsibilities and the success of other systems (on which the system in question is dependent) in meeting their goals.

3.1.2.2. System Performance

System performance is evaluated by examining the degree to which system goals are being achieved. For all systems, a controlled variable deviation parameter δ_c can be defined by

$$\delta_c = |C - C_{sp}| \quad (3-3)$$

where C is the measured or estimated value of the controlled variable and C_{sp} is the desired (setpoint) value. Due to random errors in measured values, δ_c is a positive random variable².

For control systems it is possible to define a manipulated variable deviation parameter δ_m .

² In the current formulation of SLD, the sign of the deviation is not used, hence the use of absolute values in calculating deviations. Broadening the method to include deviation sign is left as a possible extension of the basic methodology.

$$\delta_m = |M - M_e| \quad (3-4)$$

Here, M is the measured or estimated value of the manipulated variable and M_e is an expected value computed from a quantitative model. The choice of model used to calculate M_e depends on the reference frame (RF) selected for diagnosis. The choice of reference frame can have a significant impact on the resolution of the final diagnosis.

Setpoint values determined by operators (or designers in the case of passive systems) define an absolute reference frame in which to judge controlled variable performance. The manipulated variables of control systems do not have an absolute reference frame and several are possible, being designed to vary in response to process disturbances. Similarly, controlled variables whose setpoints are determined by other control systems, as in cascade control systems, will not have an absolute reference frame.

For steady-state processes, the steady-state reference frame (SSRF) can be used. In the SSRF, manipulated variables are expected to remain near their normal steady-state values as shown in equation (3-5).

$$M_e = M_{ss} \quad (3-5)$$

M_e is the vector of expected manipulated variable values and M_{ss} is a vector of steady-state values. Any disturbance entering the process that requires control system compensation will produce a deviation of δ_m if this reference frame is used.

If a nominal steady-state does not exist or is not practical, a dynamic reference frame (DRF) can be used. This frame uses a dynamic quantitative process model to calculate expected manipulated variable values as shown in equation (3-6).

$$M_e = f^d[C, C_{sp}, M, I] \quad (3-6)$$

Here C is the vector of all controlled variable measurements, C_{sp} is the vector of all controller setpoints, M is the vector of all manipulated variable measurements, and I is the vector of all other measured process inputs. This reference frame will not produce a deviation of δ_m in response to disturbances in measured input variables.

In addition to the two global reference frames described above, local reference frames can also be defined. A system reference frame (SRF), shown in equation (3-7), can be used to isolate malfunctions that effect a specific system.

$$M_e = f^s[C, C_{sp}, I_s] \quad (3-7)$$

Here, I_s is a vector of system inputs which may include the controlled or manipulated variables of other systems. Alternately, a controller reference frame (CRF), shown in equation (3-8), can be defined.

$$M_e = f^c[C, C_{sp}] \quad (3-8)$$

The CRF function $f^c[C, C_{sp}]$ reproduces the controller logic and will produce deviations only when the controller itself is malfunctioning.

Aldanondo and Sheridan (1986) and Kramer (1987) describe other diagnosis schemes that exploit the power of alternate reference frames. Because the processes examined in this work are nominal steady-state processes, the SSRF has been used exclusively.

Qualitative system states are defined based on the values of δ_c and δ_m in relation to defined thresholds, θ_c and θ_m . For passive systems, two states are defined:

- 1) Functional -- if $\delta_c \leq \theta_c$
- 2) Saturated -- if $\delta_c > \theta_c$

For control systems four states are defined:

- 1) Functional -- if $\delta_c \leq \theta_c$ and $\delta_m \leq \theta_m$
- 2) Stressed -- if $\delta_c \leq \theta_c$ and $\delta_m > \theta_m$
- 3) Uncontrolled -- if $\delta_c > \theta_c$ and $\delta_m \leq \theta_m$
- 4) Saturated -- if $\delta_c > \theta_c$ and $\delta_m > \theta_m$

The stressed, saturated, and uncontrolled states can all be considered dysfunctional since at least one system goal is not being achieved.

Although these states provide information on system performance, even more information can be garnered from plotting system trajectories in δ_c and δ_m space. Figure 3-4 shows an idealized trajectory for a system being subjected to a large disturbance. The system is ultimately saturated, a condition that usually implicates the control apparatus failure as a possible root cause. From the trajectory, however, it is clear that the system was attempting to compensate for the disturbance when the available variability of the manipulated variable was exhausted. Therefore, controller failure and control element (valve) failure can be eliminated as possible malfunctions. This work has concentrated on the use of system states in diagnosis. As a result, the use of trajectories has not been thoroughly investigated and remains for future research.

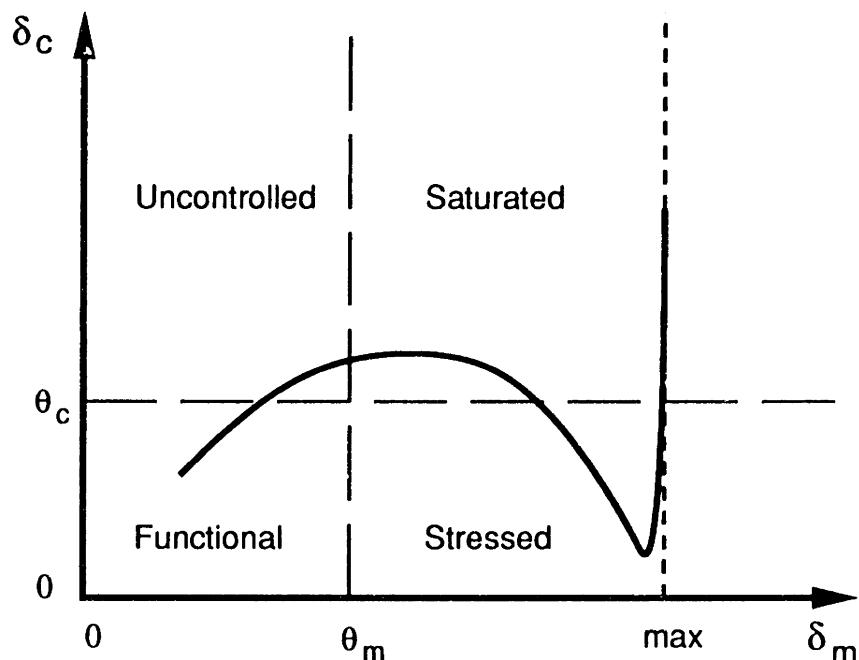


Figure 3-4: Control System Response

3.1.2.3. Diagnosis using System Models

Finch and Kramer (1988) present one possible procedure for using system models in diagnosis. The assumptions implicit in this procedure are

- 1) Only a single fault is present at any given time, and
- 2) Controlled and manipulated variables are measured for all systems³.

Diagnosis proceeds in two stages:

- 1) Location of source systems, and
- 2) Application of compiled diagnostic rules for system interactions.

A *source system* is a system malfunctioning due to the direct action of a fault on one or more of the system's unit functions, and is therefore analogous to the root node of a digraph. Source systems must be identified by searching the process graph.

Although a formal algorithm for graph search is not presented, the following procedure is used:

- 1) Identify dysfunctional subgraphs.

Malfunctioning systems in the process graph will form clusters of connected stressed, uncontrolled, and saturated systems. Each cluster is a dysfunctional subgraph and must contain at least one source system.

- 2) Within each subgraph identify definite source systems.

A *definite source system* is a malfunctioning system that is causally "upstream" of all other malfunctioning systems within a dysfunctional subgraph (i.e. there are no dependency arcs directed toward it from other malfunctioning systems and therefore no uncertainty that the system is a source system). The existence of a definite source system is not guaranteed and will depend on the connectivity of the process graph. If a definite source system is identified it is possible to proceed directly to step (4).

- 3) If no definite source systems were found, then identify possible source systems.

³ Park and Himmelblau (1987) show that this level of instrumentation is necessary to diagnose system faults.

The possibility of multiple source systems can produce severe degradation of the diagnosis when a definite source system cannot be identified since in a highly connected graph almost every malfunctioning system can be a possible source system. The only manner in which non-source systems can be discriminated from possible source systems is to examine the unit function sets of all malfunctioning systems in detail and apply the criteria that all source systems must share a common unit function. Such a detailed examination can produce a minimum set of possible source systems.

4) Apply diagnostic rules to identify malfunctioning unit functions.

Pair-wise diagnostic rules for isolation of single faults have been developed and are presented in Tables 3-1 to 3-4. These rules were derived by performing a qualitative simulation of the effects of the specified failure and applying the diagnostic principles described by Milne (1985). To use these rules:

- Identify all systems with which the source system being analyzed interacts (each system, in conjunction with the source system, constitutes a system "pair")
- Use Table 3-1 to classify the mode of interaction that exists between the system pair (label the individual systems as system 1 or system 2 as necessary), and
- Apply the rules from Tables 3-2, 3-3, or 3-4 depending on the system types. If a source system interacts with more than one system, the diagnostic rules must be applied for each pair and the results combined to produce a diagnosis.

If a definite source system was identified, the rules can be applied for that system and the malfunction is guaranteed to be captured. A more advantageous situation occurs when multiple definite source systems are identified since the diagnostic rules can be applied to each source system and the results intersected to produce a final diagnosis. However, if no definite source systems were identified, the rules must be applied to all possible source systems and the results combined via a union operator to produce a diagnosis.

The rules in Tables 3-2 to 3-4 indicate the types of unit functions that may be responsible for a system failure. Unit functions are divided into four categories:

- 1) Control functions (CNT) involved in proper operation of the controller and final control element (valve),

- 2) Sensing functions (SEN) involved in measuring process variables,
- 3) Control sensing functions (CSEN) involved in measuring controlled variables and sending signals to controllers (these functions are a subset of the SEN functions), and
- 4) Process function (PRC) which include all other unit functions.

When the diagnostic rules implicate one or more unit function types, all unit functions of each type associated with the indicated system are possible malfunction sources. Finch and Kramer (1987) show how these rules could be expressed in an expert system.

TABLE 3-1: System Interaction Classifications:

#	Classification	Description
1	Independent	A system that does not depend on the any other system
2	CV Dependence (one-way)	A system that depends on the primary (CV) goal of another system
3	MV Dependence (one-way)	A system that depends on the secondary (MV) goal of another system
4	CV Dependence (two-way)	Two systems, each dependent on the other's primary (CV) goal
5	MV Dependence (two-way)	Two systems, each dependent on the other's secondary (MV) goal
6	CV-MV Dependence	A system that depends on the primary (CV) goal of another system that in turn depends on the secondary (MV) goal of the first system
7	MV-CV Dependence	A system that depends on the secondary (MV) goal of another system that in turn depends on the primary (CV) goal of the first system
8	Unconditional Dependence	A system that is dependent on both the primary (CV) and secondary (MV) goal of another system

An example of the SLD method applied to a simple chemical process is presented in Appendix 3. The SLD approach is advantageous in that it models large scale processes at an appropriate level for initial diagnosis and it captures an aspect of process behavior not easily captured by other models. The disadvantages of the method are the relatively poor resolution that can be produced when no definite source systems can be identified and the non-dynamic character of the diagnostic rules which were derived based on ultimate steady-state behavior not system trajectories.

TABLE 3-2: Pair-wise Single Fault Diagnosis of Passive Systems

System 1 State	System 2 State	System 1 Interaction Classification		
		1	2	4
Functional	Functional	No Fault	No Fault	No Fault
Functional	Saturated	PRC2 SEN2	PRC2 SEN2	PRC2 SEN2
Saturated	Functional	PRC1 SEN1	PRC1 SEN1	PRC1 SEN1
Saturated	Saturated	PRC1 \cap PRC2	PRC2	PRC1 PRC2

Definitions for Tables 3-2, 3-3, and 3-4

- No Fault — No faults present
- PRC1 — Fault in process unit functions of System 1
- PRC2 — Fault in process unit functions of System 1
- PRC1 \cap PRC2 — Fault in shared unit functions of Systems 1 and 2
- SEN1 — Fault in sensing functions of System 1
- SEN2 — Fault in sensing functions of System 2
- CSEN1 — Fault in controlled variable sensing functions of System 1
- CSEN2 — Fault in controlled variable sensing functions of System 2
- CNT1 — Fault in controller or control element of System 1
- CNT2 — Fault in controller or control element of System 2
- No SFE — No single fault explanation
(multiple faults present or control systems not at steady state)

TABLE 3-3A: Pair-wise Single Fault Diagnosis of Passive and Control Systems (System 1 Passive, System 2 Control)

		System 1 Interaction Classification									
		1		2		3 (8)†		4		7	
System 1 State	System 2 State	Functional	No Fault	Functional	No Fault	Functional	No Fault	Functional	No Fault	Functional	No Fault
Functional	Functional	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2
Functional	Stressed	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2	PRC2 SEN2 CNT2
Functional	Uncontrolled	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2
Functional	Saturated	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2
Saturated	Functional	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2
Saturated	Stressed	PRC1 \cap PRC2	PRC1 \cap PRC2	PRC1 \cap PRC2 CSEN2 CNT2	PRC1 \cap PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC1 CSEN2 CNT2	PRC1 CSEN2 CNT2	PRC1 CSEN2 CNT2	PRC1 CSEN2 CNT2
Saturated	Uncontrolled	No SFE	No SFE	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2
Saturated	Saturated	PRC1 \cap PRC2	PRC1 \cap PRC2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC1 CSEN2 CNT2	PRC1 CSEN2 CNT2	PRC1 CSEN2 CNT2	PRC1 CSEN2 CNT2

† Unconditional Dependence can be treated as MV dependence in diagnosis.

TABLE 3-3B: Pair-wise Single Fault Diagnosis of Passive and Control Systems (System 1 Control, System 2 Passive)

		System 1 Interaction Classification							
		1		2		4		6	
System 1 State	System 2 State	No Fault	No Fault	No Fault	No Fault	No Fault	No Fault	No Fault	No Fault
Functional	Functional	PRC2 SEN2	PRC2 SEN2	PRC2 CSEN1 SEN2	PRC2 SEN1 CNT1	PRC1 SEN1 CNT1	PRC1 SEN1 CNT1	PRC1 PRC2 CSEN1 CNT1	PRC1 PRC2 CSEN1 CNT1
Functional	Saturated	PRC2 SEN2	PRC2 SEN2	PRC2 CSEN1 SEN2	PRC2 SEN1 CNT1	PRC2 SEN1 CNT1	PRC2 CSEN1 CNT1	PRC1 PRC2 CSEN1 CNT1	PRC1 PRC2 CSEN1 CNT1
Stressed	Functional	PRC1 SEN1 CNT1	PRC1 SEN1 CNT1	PRC1 CSEN1 SEN2	PRC1 SEN1 CNT1	PRC1 SEN1 CNT1	PRC1 SEN1 CNT1	PRC1 PRC2 CSEN1 CNT1	PRC1 PRC2 CSEN1 CNT1
Stressed	Saturated	PRC1 \cap PRC2	PRC2	PRC2 CSEN1 CNT1	PRC2 SEN1 CNT1	PRC2 SEN1 CNT1	PRC2 CSEN1 CNT1	PRC1 PRC2 CSEN1 CNT1	PRC1 PRC2 CSEN1 CNT1
Uncontrolled	Functional	CNT1	CNT1	CNT1	CNT1	CNT1	CNT1	CNT1	CNT1
Uncontrolled	Saturated	No SFE	No SFE	No SFE	CNT1	CNT1	CNT1	No SFE	No SFE
Saturated	Functional	PRC1 CSEN1 CNT1	PRC1 CSEN1 CNT1	PRC1 CSEN1 CNT1	PRC1 CSEN1 CNT1	PRC1 CSEN1 CNT1	PRC1 CSEN1 CNT1	PRC1 PRC2 CSEN1 CNT1	PRC1 PRC2 CSEN1 CNT1
Saturated	Saturated	PRC1 \cap PRC2	PRC2	PRC2 CSEN1 CNT1	PRC2 SEN1 CNT1	PRC2 SEN1 CNT1	PRC2 CSEN1 CNT1	PRC1 PRC2 CSEN1 CNT1	PRC1 PRC2 CSEN1 CNT1

TABLE 3-4: Pair-wise Single Fault Diagnosis of Two Control Systems

System 1 State	System 2 State	System 1 Interaction Classification						
		1	2	3 (8)†	4	5	6	7
Functional	Functional	No Fault	No Fault	No Fault	No Fault	No Fault	No Fault	No Fault
Functional	Stressed	PRC2 SEN2	PRC2 SEN2	PRC2 SEN2	PRC2 SEN2	PRC2 SEN2	PRC2 SEN2	PRC2 SEN2
Functional	Uncontrolled	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2
Functional	Saturated	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN1 CSEN2	PRC2 CSEN1 CSEN2	PRC2 CSEN1 CSEN2	PRC2 CSEN1 CSEN2

† Unconditional Dependence can be treated as MV dependence in diagnosis.

TABLE 3-4: Pair-wise Single Fault Diagnosis of Two Control Systems (continued)

System 1 State		System 2 State		System 1 Interaction Classification				
		1	2	3 (8)†	4	5	6	7
Stressed	Functional	PRC1 SEN1	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2 CNT2	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2 CNT2	PRC1 SEN1 CSEN2	PRC1 SEN1 CSEN2 CNT2
Stressed	Stressed	PRC1 \cap PRC2	PRC1 \cap PRC2 CSEN2	PRC2 CSEN2	PRC1 \cap PRC2 CSEN1 CSEN2	PRC1 PRC2 CSEN2 CSEN2	PRC1 CSEN1 CSEN2	PRC2 CSEN1 CSEN2
Stressed	Uncontrolled	No SFE	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2
Stressed	Saturated	PRC1 \cap PRC2	PRC2 CSEN2 CNT2	PRC1 CSEN1 CSEN2 CNT2	PRC1 PRC2 CSEN1 CSEN2 CNT2	PRC1 PRC2 CSEN1 CSEN2 CNT2	PRC1 PRC2 CSEN1 CSEN2 CNT2	PRC1 PRC2 CSEN1 CSEN2 CNT2

† Unconditional Dependence can be treated as MV dependence in diagnosis.

TABLE 3-4: Pair-wise Single Fault Diagnosis of Two Control Systems (continued)

System 1 State	System 2 State	System 1 Interaction Classification					
		1	2	3	4	5	6
Uncontrolled	Functional	CNT1	CNT1	CNT1	CNT1	CNT1	CNT1
Uncontrolled	Stressed	No SFE	No SFE	No SFE	CNT1	No SFE	No SFE
Uncontrolled	Uncontrolled	No SFE	No SFE	No SFE	No SFE	No SFE	No SFE
Uncontrolled	Saturated	No SFE	No SFE	No SFE	CNT1	No SFE	No SFE

TABLE 3-4: Pair-wise Single Fault Diagnosis of Two Control Systems (continued)

		System 1 Interaction Classification						
System 1 State	System 2 State	1	2	3	4	5	6	7
Saturated	Functional	PRC1 CSEN1 CNT1	PRC1 CSEN1 CSEN2 CNT1	PRC1 CSEN1 CSEN2 CNT1 CNT2	PRC1 CSEN1 CSEN2 CNT1 CNT2	PRC1 CSEN1 CSEN2 CNT1 CNT2	PRC1 CSEN1 CSEN2 CNT1 CNT2	PRC1 CSEN1 CSEN2 CNT1 CNT2
Saturated	Stressed	PRC1 \cap PRC2 CSEN2	PRC1 \cap PRC2 CSEN2	PRC2 CSEN2	PRC1 CSEN1 CNT1	PRC1 CSEN1 CSEN2 CNT1	PRC1 CSEN1 CSEN2 CNT1	PRC1 CSEN1 CSEN2 CNT1 CNT2
Saturated	Uncontrolled	No SFE	CNT2	CNT2	CNT2	CNT2	CNT2	CNT2
Saturated	Saturated	PRC1 \cap PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC2 CSEN2 CNT2	PRC1 PRC2 CSEN1 CSEN2 CNT1 CNT2	PRC1 PRC2 CSEN1 CSEN2 CNT1 CNT2	PRC1 PRC2 CSEN1 CSEN2 CNT1 CNT2	PRC1 PRC2 CSEN1 CSEN2 CNT1 CNT2

3.2. Quantitative Models

The quantitative models used in MIDAS are constraint equations expressed in the form

$$\Psi(\mathbf{X}) = \mathbf{R} \quad (3-9)$$

where \mathbf{X} is the vector of state variables and \mathbf{R} is the constraint residual. $\Psi(\mathbf{X})$ can be a linear, nonlinear, algebraic, differential, or integral equation, but must be determinable (i.e. all equation variables are measured or estimated). Constraint functions are formulated so that \mathbf{R} has zero mean during normal operation. The residual is treated as a pseudo state variable that can be monitored and estimated like any other state variable. The qualitative state of the residual is used for diagnosis. The types of constraint equations that can be formulated for process plants include: mass, heat, or momentum balances; reaction rate expressions; pressure drop relations; and many others.

Although constraint residuals are treated as state variables by MIDAS, it should be noted that residuals will display certain unique characteristics. Because residuals are calculated by combining two or more measured variables, the residual variance σ_r will be greater than the variance of any individual measured variable during nominal steady-state operation⁴.

$$\sigma_r = f\sigma(\sigma_1, \sigma_2, \dots, \sigma_n) \geq \sigma_i \quad \text{for any } 1 \leq i \leq n \quad (3-10)$$

Consequently, detection threshold may be established for the constraint that are less sensitive to small disturbances than the detection thresholds established for the variables used to calculate the constraint. Similarly, the number of malfunctions that can effect a residual m_r will be greater than the number of malfunctions affecting any individual measurement used to compute the residual.

$$m_r = f^m(m_1, m_2, \dots, m_n) \geq m_i \quad \text{for any } 1 \leq i \leq n \quad (3-11)$$

Given that residuals may be less sensitive to certain disturbances and can be associated with more malfunctions, what are the advantages of using constraint residuals? There are several:

⁴ This assumes that random measurement errors are uncorrelated.

- Constraints can be used to augment a poor instrumentation scheme by providing additional "virtual sensors".
- Constraints can capture aspects of process behavior that cannot be expressed in other modeling formats. Diagnosis based on constraints will produce malfunction candidate sets disjoint from the candidate sets produced by an ESDG model that cannot capture global behaviors such as material balance relationships.
- Constraints can model dynamic processes better than ESDG models. To illustrate this point, it can be shown that measured variables can be expressed as a special form of constraint

$$X_i - X_i^{ss} = R_i \quad (3-12)$$

where X_i^{ss} is the nominal steady state value of X_i . This constraint is only valid when a process has a nominal steady-state. For processes with no steady state, only general dynamic constraints such as balances and pressure drops will be valid.

TABLE 3-5A: Passive System Failure Modes used to Derive Pair-Wise Diagnostic Rules

System(s) Containing Effected Functions	Type of Unit Functions Effected	Designation	Description of Failure Mode
1	Process	PRC1	Uncontrollable Failure
2	Process	PRC2	Uncontrollable Failure
1&2	Process	PRC1 \cap PRC2	Uncontrollable Failure
1	Sensing	SEN1	CV Sensor Failed Normal
1	Sensing	SEN1	CV Sensor Failed Abnormal
2	Sensing	SEN2	CV Sensor Failed Normal
2	Sensing	SEN2	CV Sensor Failed Abnormal

TABLE 3-5B: Control System Failure Modes used to Derive Pair-Wise Diagnostic Rules

System(s) Containing Effected Functions	Type of Unit Functions Effected	Designation ¹	Description of Failure Mode
1	Process	PRC1	Controllable Failure
1	Process	PRC1	Uncontrollable Failure
2	Process	PRC2	Controllable Failure
2	Process	PRC2	Uncontrollable Failure
1&2	Process	PRC1 \cap PRC2	Controllable Failure
1&2	Process	PRC1 \cap PRC2	Uncontrollable Failure ²
1	Control	CNT1	Controller not Responding
1	Control	CNT1	Controller Responding Incorrectly
1	Control	CNT1	No Input to Controller
1	Control	CNT1	No Output from Controller
2	Control	CNT2	Controller not Responding
2	Control	CNT2	Controller Responding Incorrectly
2	Control	CNT2	No Input to Controller
2	Control	CNT2	No Output from Controller
1	Control	CNT1	Valve Stuck in Normal Position
1	Control	CNT1	Valve Stuck in Abnormal Position
2	Control	CNT2	Valve Stuck in Normal Position
2	Control	CNT2	Valve Stuck in Abnormal Position
1	Sensing	SEN1	MV Sensor Failed Normal
1	Sensing	SEN1	MV Sensor Failed Abnormal
2	Sensing	SEN2	MV Sensor Failed Normal
2	Sensing	SEN2	MV Sensor Failed Abnormal
1	Sensing	SEN1/CSEN1	CV Sensor Failed Normal
1	Sensing	SEN1/CSEN1	CV Sensor Failed Controllably
1	Sensing	SEN1/CSEN1	CV Sensor Failed Uncontrollably
2	Sensing	SEN2/CSEN2	CV Sensor Failed Nonnal
2	Sensing	SEN2/CSEN2	CV Sensor Failed Controllably
2	Sensing	SEN2/CSEN2	CV Sensor Failed Uncontrollably

¹ Note: CSEN is a subset of SEN² Three cases: 1) Both systems uncontrollable, 2) System 1 controllable, System 2 uncontrollable, and 3) System 1 uncontrollable, System 2 controllable

3.3. Event Models

The models that MIDAS uses for diagnosis are called *event models*⁵. Event models are graphical association models that represent causal relationships similar to digraph and process graph models. Oyeleye (1989) shows that an ESDG model can be converted into event model form and presents a computer algorithm that performs this conversion automatically.

The advantage of event modeling over other causal graph methods is the flexibility of the event paradigm. In the following sections, it will be shown how event models can incorporate constraint equations and operator actions into the basic causal framework. In this way, the benefits of causal and constraint modeling can be realized within a single representational structure. Section 3.4 illustrates the synergies that can be derived from this unification of different modeling paradigms.

3.3.1. Events Graphs

In its broadest sense, an *event* is defined as any observable discrete occurrence that carries significant diagnostic information. Examples of the types of events important in chemical process modeling and diagnosis are

- Reactor Temperature was NORMAL and is now HIGH,
- Furnace Oxygen Consumption was INCREASING and is now STEADY,
- Recycle Flow Control System was STRESSED and is now SATURATED,
- Reflux Drum Level Sensor was OK and has now FAILED,
- Mass Balance was SATISFIED and is now VIOLATED,
- Pump Test was performed with NEGATIVE results, and
- Operator has initiated an emergency SHUTDOWN.

Notice that events can be changes in the qualitative state of a process variable, constraint equation residual, or system; changes in the prevailing trend of a variable or constraint equation residual; changes in equipment status; the results of tests; or operator actions.

⁵ Event modeling in this context should not be confused with event trees used in reliability and risk analysis.

In defining an event, it is often necessary to explicitly declare the prior and consequent state, trend, or status to avoid confusion with other events. For example, the prior trend must be declared in the following events to indicate that these are two separate events:

- Furnace Oxygen Consumption was INCREASING and is now STEADY
- Furnace Oxygen Consumption was DECREASING and is now STEADY

An *event graph* is a pictorial representation of an event model. A simple event graph for the tank in figure 3-1 is shown in figure 3-5. Nodes in the event graph represent the different qualitative states of the modeled process variables. Only one node can be active at any given time for a single process variable (i.e. a variable cannot have both high and low nodes active simultaneously). An event is a transition between two nodes that activates the new (consequent) node and deactivates the old (prior) node. Nodes can be connected by *precursor/successor links* (PSLs) that depict causal relationships between nodes. PSLs can be classified as being *intravariable* or *intervariable*, depending on whether they connect nodes that represent different states of the same variable or states of different variables, respectively. All the PSLs in figure 3-5 are intervariable.

Root causes are the different malfunctions that could effect the process. For example, in figure 3-5, **High Inflow** and **Level Sensor High Bias** are root causes. In the event graph, root causes are linked to nodes via a second type of link called a *local cause link* (LCL). To differentiate LCLs from PSLs graphically, LCLs are depicted as dashed arrows and PSLs are depicted as solid arrows. Any node linked to a root cause is referred to as a *primary symptom* of the root cause. The primary symptom is the first symptom expected when the root cause is present. Any given cause may have one or more primary symptoms (if a root cause has more than one primary symptom, the primary symptoms could appear in any order). In the example, **Level High** is the primary symptom of the root cause **High Inflow**.

The similarities between digraphs and event graphs are evident. PSLs and LCLs indicate potential avenues of disturbance propagation like digraph arcs. For example, in figure 3-5, **Flow High** is causally related to **Level High**. Despite the apparent similarities, however, there are several important differences between event models and digraph models. In an event graph, each possible qualitative state is represented explicitly by a separate node (e.g. in general, representing one digraph node requires three event graph nodes); all nodes represent observable symptoms (i.e. there are no unmeasured event graph nodes); links are directed but

not signed; and nodes are not limited to process variables states but can also represent trends, constraint residuals, systems, equipment status, and operators actions.

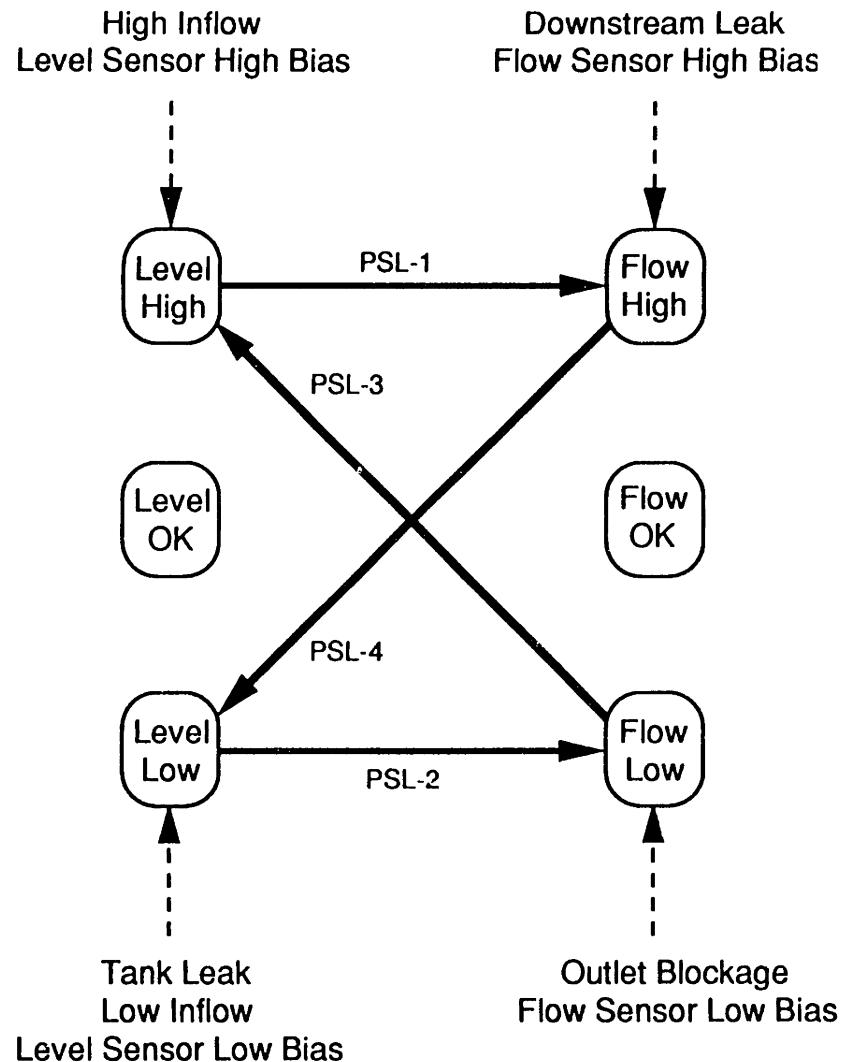


Figure 3-5: Tank Event Graph

3.3.2. Simulation and Diagnosis using Event Graphs

Event models can be used for simulation or diagnosis. In the simulation direction, figure 3-5 indicates **Downstream Leak** has a primary symptom of **Flow High** which can in turn

result in the secondary symptom **Level Low** via PSL-4. This type of reasoning is not sufficient in all situations. For example, another root cause capable of producing a **Flow High** state is **Flow Sensor High Bias**, but a sensor bias would never produce a secondary symptom unless it was included in a control loop. In this situation the event model in figure 3-5 gives the erroneous result that **Flow Sensor High Bias** can result in a **Level Low** state.

To correct this problem, information must be added to PSL-4 to indicate that it will transmit a disturbance caused by a downstream leak but not a sensor bias. This information is added by attaching one or more *conditions* to the link.

Of the various types of conditions that can be used to refine the event model, the NOT condition is the most common. NOT conditions block propagation of a malfunction disturbance along a PSL. An example of a NOT condition is shown in equation 3-13.

$$(:NOT (fault_1 \ fault_2 \ ... \ fault_N)) \quad (3-13)$$

The essential features of the condition are an initial :NOT keyword indicating the condition type and a list of faults that are blocked by the condition. Other types of link conditions are also possible. ADD conditions, illustrated in equation 3-14, can be used to augment the set of root cause candidates. If two or more events are necessary to indicate a fault (i.e. there is no single primary symptom), an ADD condition can be used rather than LCLs.

$$(:ADD (fault_1 \ fault_2 \ ... \ fault_N)) \quad (3-14)$$

The ONLY-IF condition shown in equation 3-15 blocks propagation of all faults except those indicated in the list following the keyword.

$$(:ONLY-IF (fault_1 \ fault_2 \ ... \ fault_N)) \quad (3-15)$$

RETAIN conditions and REPLACE conditions preserve a subset of malfunction candidates or replace them with another list, respectively.

$$(:RETAIN (fault_1 \ fault_2 \ ... \ fault_N)) \quad (3-16)$$

$$(:REPLACE (fault_{11} \ fault_{12} \ ... \ fault_{1N}) (fault_{21} \ fault_{22} \ ... \ fault_{2N})) \quad (3-17)$$

NOT and ADD conditions form a elemental set of operators into which other conditions can be decomposed. Tremendous modeling flexibility that can be derived from the use of link conditions. Other specialized link conditions can undoubtedly be developed to accommodate specialized modeling needs.

Another type of link condition necessary to produce a dynamic event model is the ONLY-IF-TRANSIENT condition. This condition appears only on intravariable PSLs directed from abnormal to normal states.

(:ONLY-IF-TRANSIENT) (3-18)

When attached to a link, this condition indicates that two states can be causally related only when the malfunction or disturbance responsible for the original abnormal state has been corrected or has disappeared spontaneously (e.g. a transient disturbance or false alarm).

Figure 3-6 is a revised dynamic event graph for the tank process. With conditions attached to PSLs, the diagnostic utility of the event model becomes more apparent. In figure 3-6, it is now possible to discriminate between **Downstream Leak** and **Flow Sensor High Bias**. If **Flow High** is detected, both **Downstream Leak** and **Flow Sensor High Bias** must be considered as possible root causes⁶. However, if **Level Low** is subsequently observed, **Flow Sensor High Bias** can be eliminated as a possible cause and **Downstream Leak** becomes the sole root cause candidate.

In addition to the intervariable links that have been discussed, event models support intravariable links that can be used to create dynamic models. Certain process variables that exhibit compensatory or inverse response may experience several changes of state or trend during a single malfunction episode (controlled variables are an example of this class of variables). Other variables, not capable of exhibiting nonlinear response, can experience only a single state or trend change in a single malfunction episode. For all variables, multiple state changes form a variable trajectory that can be used in diagnosis.

⁶ Because the event graph is qualitative, it does not model the gain or delay along a PSL. Without this additional information, it is not possible to use the absence of a **Level Low** observation in diagnostic reasoning.

To model and extract diagnostic information from variable trajectories, PSLs can be directed between different states of the same variable as illustrated in figure 3-6. Here, links from **Flow High** to **Flow OK** and **Flow Low** to **Flow OK** model the fact that outflow is a compensatory variable and can be expected to return to normal when a new steady-state level is achieved (assuming inflow is unchanged and there is no tank leak). **ONLY-IF-TRANSIENT** PSLs from **Level High** to **Level OK** and **Level Low** to **Level OK** indicate that level will return to normal only when the root cause of the disturbance has been removed.

Simulation of **Outlet Blockage** using the model in figure 3-6 indicates that **Flow Low** is expected first, followed by **Level High** and **Flow OK**. No particular order can be assigned to **Level High** and **Flow OK** because link delays are not modeled.

Modeling inverse response is slightly more complex than modeling compensatory response and requires the use of *activatable PSLs*. The PSLs discussed thus far have been permanent features of the event model, but it is also possible to create PSLs that are active only in certain situations. These activatable PSLs can be turned on or off as required. When turned on, the link behaves like any other PSL. When turned off, the link disappears from the model until it is activated again.

Figure 3-7 illustrates the use of activatable links to model inverse response. Figure 3-7A, shows the value of a variable X exhibiting inverse response. The event sequence in figure 3-7A is **X OK** → **X High** → **X OK** → **X Low**. Normally, one would not create a link from **X OK** to **X Low**, since **X Low** does not have a causal relationship with **X OK** except in the special circumstance of inverse response when **X High** has previously been observed. In fact the presence of a permanent PSL from **X OK** to **X Low** could result in abnormal states being explained as normal behavior. Instead, an activatable PSL is created from **X OK** to **X Low**. The link is turned on whenever **X High** is observed and turned off when the malfunction episode is complete. The event graph for X is shown in figure 3-7B.

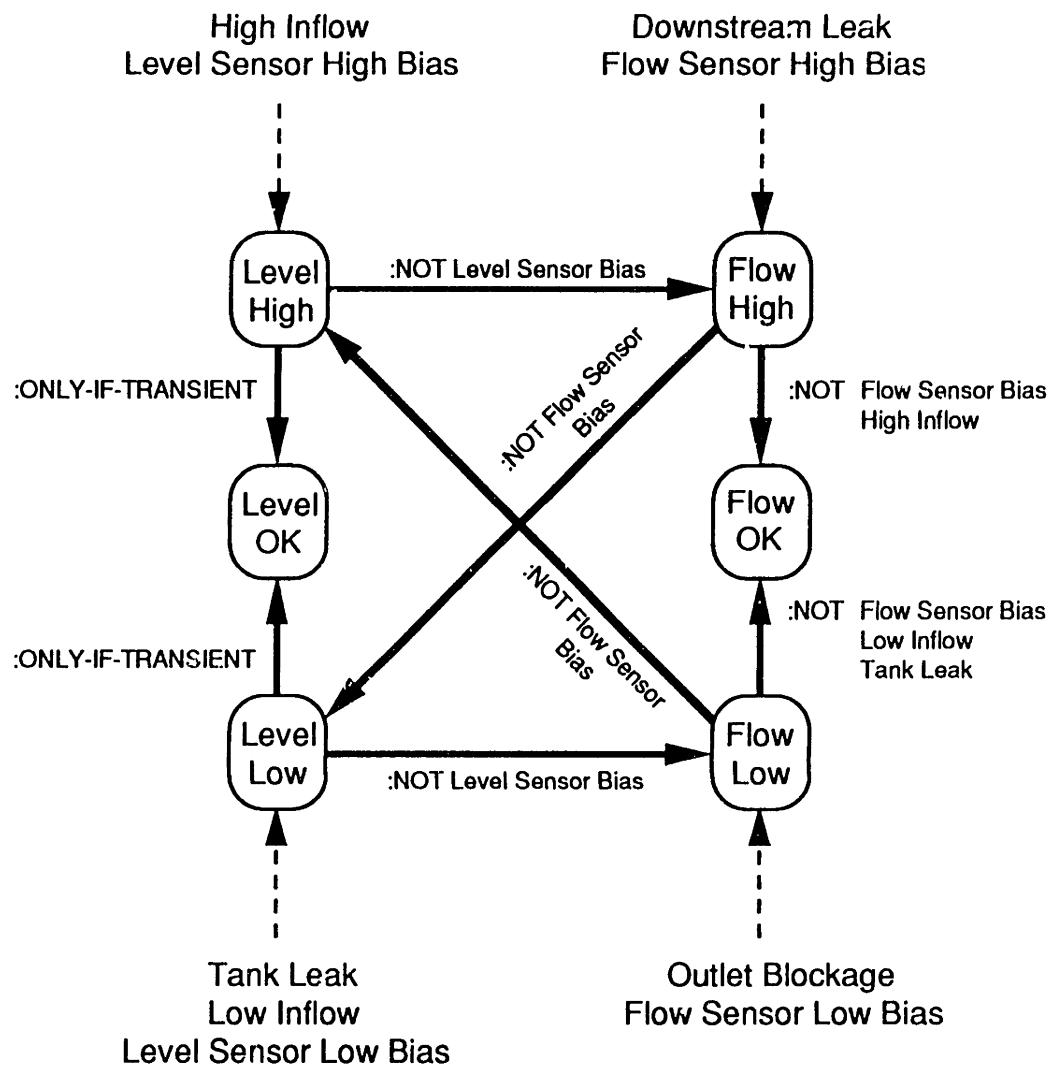


Figure 3-6: Tank Dynamic Event Graph

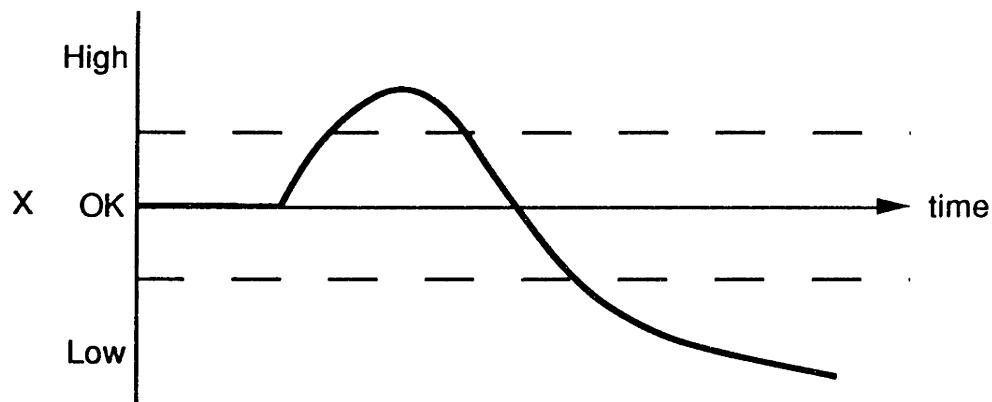


Figure 3-7A: Inverse Response

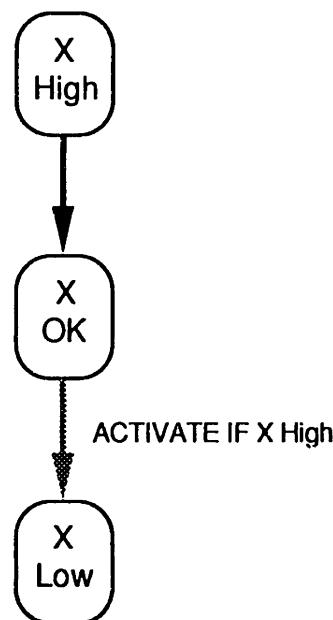


Figure 3-7B: Event Model of Inverse Response

Modeling inverse response is only one of the potential applications of activatable PSLs. Other applications include:

- Modification of the event model in response to changes in the process such as an operator action, a process changeover, or equipment taken off-line for maintenance,
- Augmenting the model when faults create new causal pathways, or
- As a vehicle for expressing alternate modeling styles.

The idea of alternate modeling styles is not new to qualitative modeling. Palowitch (1987), while attempting to formalize the conversion of equation models to SDG models, concedes that there is no single correct SDG for any process. Analogously, there is no unique solution to the event modeling problem, and in event modeling, the problem may be even more pronounced because of the tremendous flexibility of the paradigm. To illustrate the point, figure 3-8 is a dynamic event model for the tank process equivalent to the model in figure 3-6. In figure 3-8, however, the possible compensatory response of outflow is modeled by two activatable intervariable PSLs rather than two permanent intravariabale PSLs.

Because they are not unique, establishing criteria to judge event models is an important concern. Several basic criteria are discussed below.

Generality

The wider the range of situations modeled in an event model, the greater the reliability of the model when used for diagnosis.

Causal Ordering

Event models should attempt to maintain, to the maximum degree possible, the causal ordering of the original ESDG.

Connectivity

The complexity of the event model is largely dependent on the number of PSLs originating and terminating at an average node. The impact of high levels of complexity on diagnosability have been described by Karpman and Dubuisson (1985) and Henneman and Rouse (1984). In event models, every attempt should be made to keep the average connectivity as low as possible⁷.

⁷ Event model connectivity will typically be much higher than the connectivity of the original EDSG model.

Size

If two otherwise equivalent event models differ in size (i.e. number of nodes, PSLs, or root causes) then the smaller model should be selected. In such cases, the larger model probably contains redundant information.

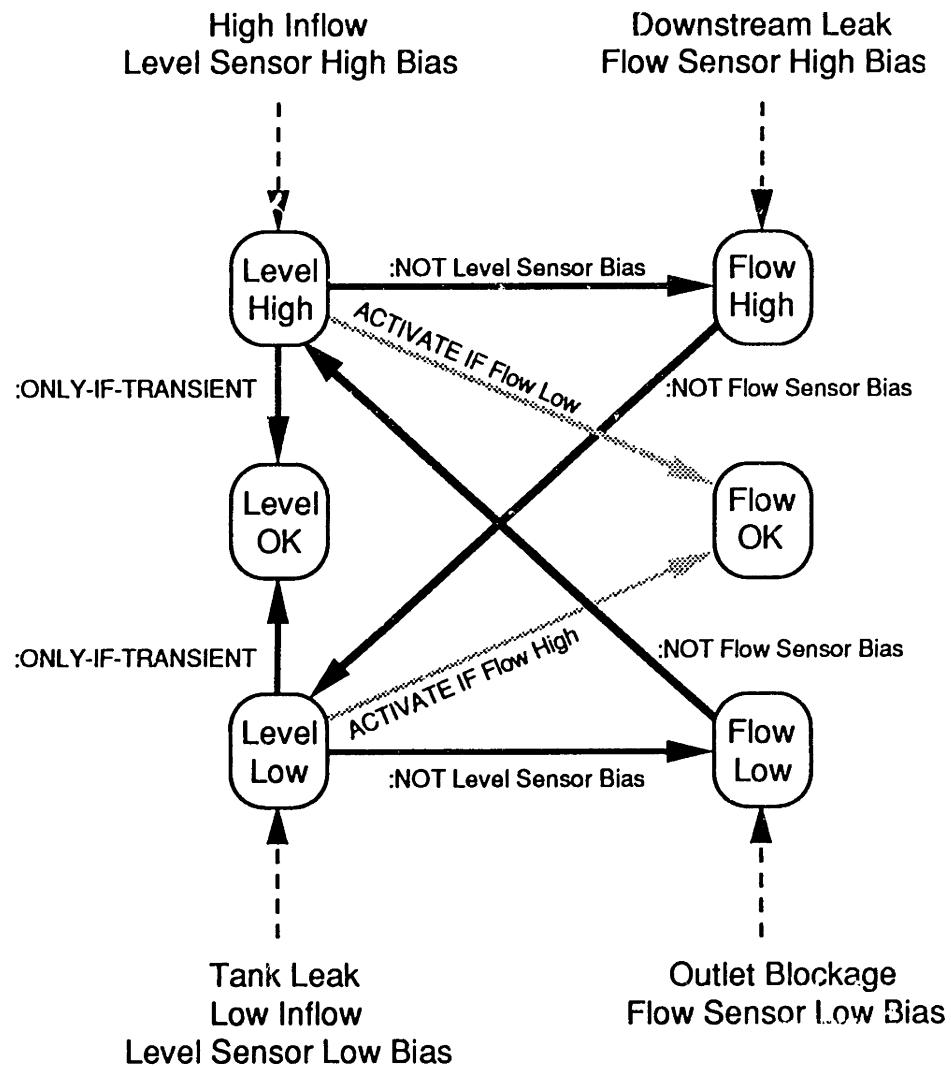


Figure 3-8: Alternative Dynamic Event Graph for Tank

A process event model can be created manually or generated automatically from an existing digraph using Oyeleye's algorithm. Event models produced from digraphs will contain only

those events related to state changes. Other event types (such as constraint residuals) must be added manually. At present, only variable state changes and constraint residuals have been examined in detail. Test result events have been added to some event models, although a special reasoning procedure was necessary to accommodate these events (see section 4.2.6).

At present, no event models have been created incorporating system state changes, trends, operator actions, or equipment status events. The primary purpose of operator action and equipment status change events is to activate and deactivate PSLs. These event types result in changes in the structure or behavior of the process that must be accompanied by changes in the event model. In fact, one could envision certain operator actions, such as SHUTDOWN INITIATED, resulting in the entire network of PSLs active during normal operation being turned off and a new network of shutdown PSLs being activated.

Trend events have not been included because trend modeling is less advanced, relative to state modeling. Kuipers (1986) presents a formalism for reasoning with trends in the QSIM algorithm. However, there exists no trend equivalent of a digraph model.

System state changes could be included in current event models, except that more work is required to determine how PSLs (particularly intravariable PSLs) should be structured to model possible system trajectories and capture the diagnostic information contained in the trajectories. As was shown in figure 3-4, system trajectories are more complex than variable trajectories. Because the SLD methodology is not dynamic, it can provide only the starting point for developing system PSLs.

Event models can be naturally represented in an object-oriented knowledge base. Object definitions for one such representation will be presented in section 4.0.

3.4. Comparison of Modeling Techniques

Table 3-6 summarizes the essential features of the modeling techniques discussed in sections 3.1, 3.2, and 3.3. Three major model features are considered:

- Range of applicability (dynamic or steady-state),
- Range of behaviors predicted (initial or ultimate), and
- Ability to model control actions.

In reality, all three features are intimately related and are derived from a common source: the ability to model process feedback.

A diagnostic method based on a modeling paradigm that is not dynamic will typically require a "reset" after each malfunction episode when the fault has been corrected and the process returns to nominal steady-state behavior. This weakness stems from a lack of model information on the correct interpretation of intravariable events. For the same reasons, non-dynamic models may have difficulty interpreting false alarms, transients, or malfunctions that produce complex responses such as compensatory or inverse response.

These limitations of non-dynamic methods can be partially overcome by limiting the memory of the diagnostic method. Typically, the shorter the memory, the better the performance of the method in the face of complex dynamics. However, limiting memory to one episode by including a reset after each malfunction may not be sufficient to assure reasonable diagnostic performance. It may be necessary to perform diagnosis using a series of process "snapshots". A snapshot contains only the current process states and does not include any information on past states or the time order in which state changes were detected. A method based on snapshots has no memory of past diagnoses and should be moderately robust to unmodeled dynamics. However, snapshots may reduce the effectiveness of the diagnostic method by eliminating trajectory information that could be useful in diagnosis.

Not all non-dynamic methods are equal with regard to interpreting intravariable events. SDG methods are usually poor performers with regard to dynamics. SDGs predict only the initial response of the process; therefore, if the ultimate response differs from the initial response (usually the case in systems with controllers), the SDG model will fail. Palowitch (1987) tried to overcome this inherent limitation of the SDG using heuristics to help model control system behaviors.

The SLD method suffers the opposite problem -- predicting only the ultimate process response. Because the rules for system diagnosis were developed using the ultimate behavior of goal directed controllers, the rules may give erroneous diagnoses early in a malfunction episode before controllers have attained their ultimate response. To be assured that the process has attained its ultimate response requires waiting for some specified period after the last detected state change. This requirement is at odds with the desire to produce a timely diagnosis. Because of the high level of abstraction inherent in SLD models, misdiagnosis of initial response has only been an occasional problem in case studies [Finch and Kramer

(1989)], where performance has been superior to that produced by SDG models. Often the user is alerted to the fact that systems have not achieved their ultimate response when the SLD rules produce a "no single fault explanation" diagnosis.

TABLE 3-6: Comparison of Modeling Techniques

Technique	Dynamic	Behavior Predicted	Control Action and Feedback Predicted
SDG	No	Initial	No
ESDG	Yes	Initial and Ultimate	Yes
Constraints	Yes ¹	Initial and Ultimate	Yes ¹
SLD	No	Ultimate	Yes
Events	Yes	Initial and Ultimate	Yes

¹ Dependent on equation formulation

By incorporating both initial and ultimate process behavior, ESDG models escape the problems of both SDG and SLD methods and can be expected to give reliable diagnoses when using snapshots. ESDG models do not, however, explicitly model intravariable transitions, making dynamic reasoning difficult.

Event models can be used to circumvent the problems encountered by non-dynamic models. Intravariable events are included as part of the basic modeling paradigm. Diagnostic methods based on event models never require a reset and can maintain unlimited memories. In addition to the ability to handle false alarms, transients, complex responses, and corrected malfunctions, diagnostic methods based on event models can take advantage of information contained in the time order of events when creating a diagnosis. Kramer (1987a) and Washio et al. (1987) discuss the importance of this trajectory knowledge in diagnosis.

Although event models using only a single class of event can be created, the driving force behind development of the event model paradigm was the benefit to be derived from the

integration of different type of models (variable, constraint, and system). Given equivalent information, each model type is capable of producing a set of possible malfunction candidates. Because different models capture different aspects of process behavior, the sets produced will not be identical. Therefore, assuming each model produces an accurate diagnosis, intersecting the candidate sets produced by different methods will produce an accurate diagnosis with resolution greater than or equal to that of the best individual diagnosis. In certain fault situations, experience has shown that the candidate sets produced by different models can be largely disjoint, making the combined diagnosis dramatically better than any individual diagnosis.

Intersection of fault sets provides a simple method of combining diagnoses produced by different models. However, if any individual diagnosis is inaccurate, the combined diagnosis will also be inaccurate. Occasionally, this situation may be indicated by an empty set intersection set that alerts the user to the problem. An empty intersection is not guaranteed, however and in its absence, inaccuracies can not be detected a priori. The result is that the combined diagnosis will be much less stable than even the worst individual diagnosis. If non-dynamic models are used then performance of the intersection methodology will suffer the problems associated with such models. Event models benefit from the effects of combining model types without suffering the deleterious effects of actually intersecting malfunction candidate sets. The process knowledge contained in each model is combined during the diagnostic procedure rather than after diagnosis is complete. An example of combined diagnosis based on intersection of candidate set is presented in example 3-2.

Example 3-2

Tables 3-7A-C compare the individual performance of DIEX [Palowitch (1987)], an SDG based methodology modified to use snapshots, and SLD. Also included is the performance of the combined diagnosis produced by intersecting the candidate sets produced by each method.

Diagnostic performance is measured using a performance parameter (Φ) calculated using by equation 3-19.

$$\Phi = (\text{accuracy}) \times (\text{resolution}) \quad (3-19)$$

where accuracy = 1 (if diagnosis is accurate) or 0 (if diagnosis is inaccurate)

$$\text{resolution} = (F_{\text{total}} - F_{\text{diagnosis}})/(F_{\text{total}} - 1)$$

Here, F_{total} is the total number of possible malfunction candidates and $F_{\text{diagnosis}}$ is the number in the current diagnosis set. Performance varies from zero to one, with zero indicating a useless diagnosis and one indicating a perfect diagnosis.

Data for these examples was produced by dynamic numerical simulation of the process illustrated in figure 3-9 -- a CSTR performing an exothermic reaction cooling by an external recycle. Of the one hundred fourteen (114) fault modes modeled, three cases were chosen arbitrarily: a +2C bias in the reactor temperature sensor, a 10% increase in reaction activation energy, and control valve 2 stuck 60% closed. The process graph, system definitions, and unit functions for the process are given by Finch and Kramer (1989). Palowitch (1987) provides the digraph for this process.

Tables 3-7A-C show both the enhanced resolution and the instability of the combined diagnosis. Both SLD and DIEX produce occasional inaccurate diagnoses (indicated by a zero performance value) because of their non-dynamic models. In certain cases, the combined diagnosis is able to remain accurate (albeit with reduced resolution) even when one of the individual methods is inaccurate⁸. In most cases, the performance of the combined diagnosis is near the theoretical maximum (1.0).

⁸ When the intersection of the two diagnosis sets is empty, the combined diagnosis is computed by forming the union of the two sets. This procedure will produce an accurate diagnosis unless both methods are inaccurate -- unlikely since DIEX errors are the result of failures to model ultimate response and SLD errors are the result of failure to model initial response.

TABLE 3-7A: DIEX and SLD Results for Reactor Temperature Sensor Bias (+2C)

Time (sec)	Observed Symptom	Performance (Φ)		
		DIEX	SLD	Combined
1	Reactor Temperature High	0.62	0*	0.58
5	Cooling Water Setpoint High	0.63	0.76	0.93
11	Cooling Water Flow High	0*	0.76	0.60
18	CV-3 Excessively Open	0.71	0.76	0.93
178	Product Concentration Low	0.71	0.76	0.93
236	Reactant Concentration High	0.71	0.71	0.92
323	Reactor Temperature Normal	0.71	0.88	0.99
∞		0.71	0.88	0.99

* Zero performance caused by inaccurate diagnosis.

TABLE 3-7B: DIEX and SLD Results for Gradual Rise in Activation Energy (10%)

Time (sec)	Observed Symptom	Performance (Φ)		
		DIEX	SLD	Combined
136	Product Concentration Low	0.65	----	0.65
157	Cooling Water Setpoint Low	0.70	0*	0*
168	Reactant Concentration High	0.82	0.88	0.98
180	Reactor Temperature Low	0.63	0.71	0.91
182	Cooling Water Flow Low	0*	0.88	0*
209	CV-3 Excessively Closed	0.74	0.88	0.91
472	Reactor Temperature Normal	0.74	0.88	0.97
∞		0.74	0.88	0.97

* Zero performance caused by inaccurate diagnosis.

TABLE 3-7C: DIEX and SLD Results for CV-2 Stuck Closed (60%)

Time (sec)	Observed Symptom	Performance (Φ)		
		DIEX	SLD	Combined
1	Recycle Flowrate Low	0.97	0.65	0.97
1	CV-2 Excessively Closed	0.97	0.65	0.97
22	Reactor Temperature High	0.97	0*	0.97
31	Cooling Water Setpoint High	0.97	0.65	0.97
48	Cooling Water Flowrate High	0.97	0.65	0.97
50	CV-3 Excessively Open	0.97	0.65	0.97
92	Reactant Concentration Low	0.97	0.65	0.97
101	Product Concentration High	0.97	0.65	0.97
∞		0.97	0.65	0.97

* Zero performance caused by inaccurate diagnosis.

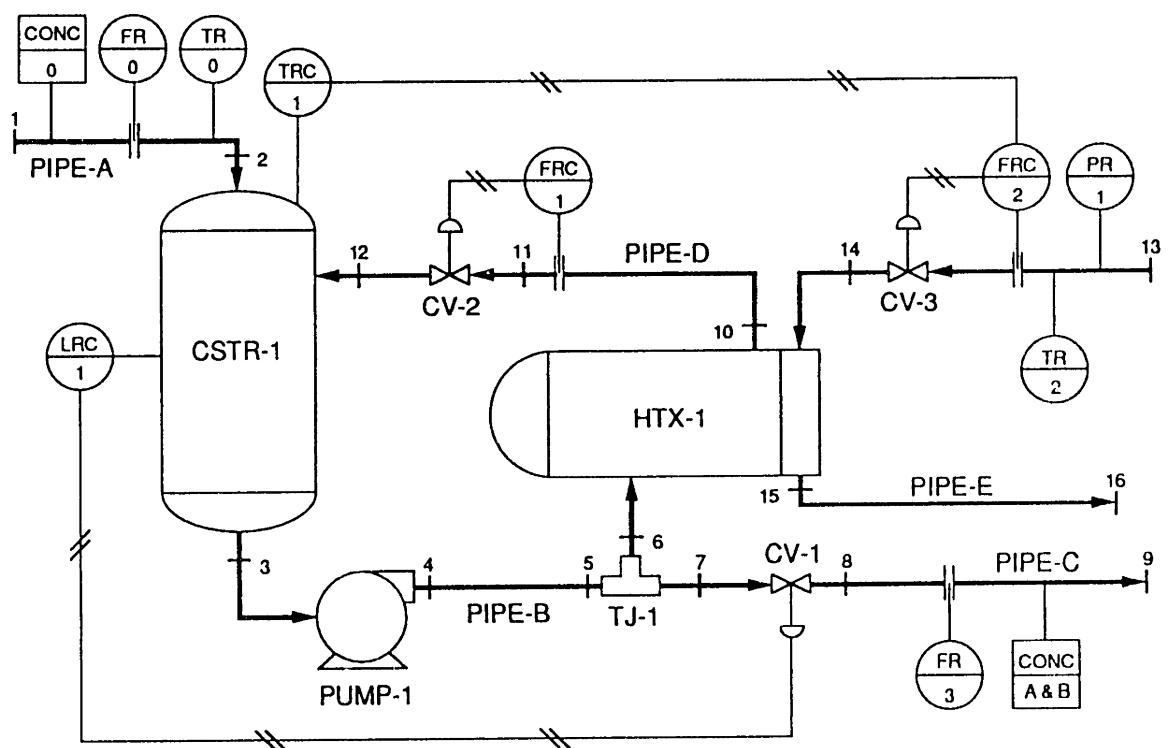


Figure 3-9: CSTR with External Recycle

4.0. Knowledge Representation and Interpretation in MIDAS

This section describes the MIDAS prototype diagnostic aid in detail. The first section outlines the major components of the system and their relationships. The second section discusses the types of object structures used for knowledge representation and their relationship to the event models described in section 3.0. The third section examines the inference procedure used to create a diagnosis, and summarizes the special features and characteristics of MIDAS. An early prototype of MIDAS called DEA is described by Kramer (1988). Both MIDAS and DEA share certain characteristics with AFS, an alarm filtering system described by Corsberg (1986) (1987). The emphasis of the section is the conceptual basis for MIDAS. Instructions for using MIDAS are presented in section 5.0.

4.1. MIDAS Structure

MIDAS is comprised of two types of knowledge: factual knowledge stored in an object-oriented knowledge-base and procedural knowledge contained in a set of inference algorithms coded in Common LISP.

The global structure of MIDAS is illustrated in figure 4-1. Five (5) major MIDAS component are defined:

- Monitors,
- Event Interpreter (EI),
- Process Model (PM),
- Hypothesis Model (HM), and
- User Interface.

4.1.1. Monitors

The primary task of a MIDAS monitor is to collect plant data for a specific measured variable or constraint residual and perform a statistical analysis to detect qualitative events. The types of events that can be detected in the current monitor implementation are state changes (including changes of state of constraint residuals), changes in trend, and gross sensor failures. The monitors are pattern recognition devices that provide the supplementary

symptom detection required by the qualitative models used in MIDAS, but are not directly involved in the diagnostic inference cycle.

The primary method of detection currently employed by MIDAS is based on the Shewhart control chart technique described in section 2.3.4. However, there is nothing to prevent other pattern recognition techniques from being used to perform the same function. Neural networks or pattern classifiers, for example, could be trained to detect events from data. Liu and Gertler (1987) discuss one alternate monitoring scheme. The relative merits of these alternatives are not explored here, since detection techniques are studied elsewhere and not considered the major intellectual contribution of the MIDAS system. The Shewhart technique gives reasonable performance, is relatively easy to understand and modify, and is generally accepted in industrial practice.

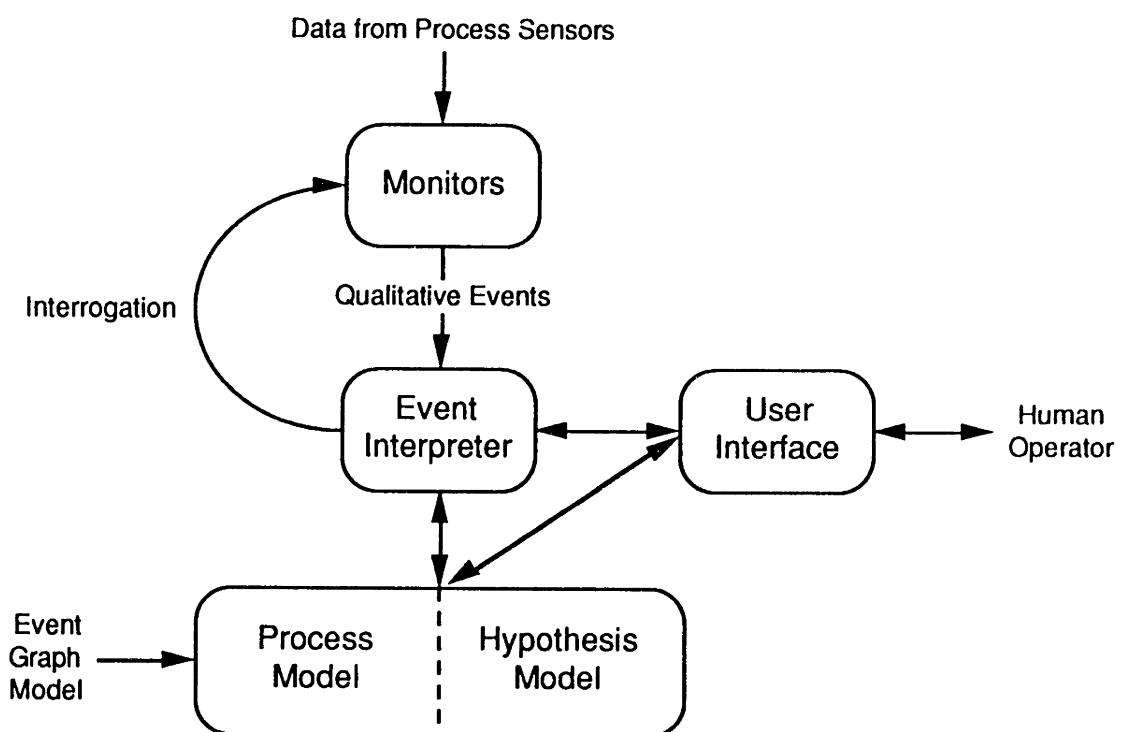


Figure 4-1: MIDAS Global Structure

To detect events from process data, a monitor must know both the nominal values or trajectories of all measured variables and constraints and the normal process variability produced by measurement noise and ordinary transients¹. Since the processes being considered are nominal steady-state processes, these nominal values are the simulation model necessary for symptom detection.

Monitors collect data in queues. When a queue reaches a specified sample length, the mean, standard deviation, and range of the data sample are computed and added to data queues containing past sample statistics. These sample statistics are subjected to statistical tests designed to detect events.

Monitors detect events related to gross sensor failures using sensor range and noise limit checks. Violation of these checks results in a type of event that in the inference process is linked directly with sensor failure. If range and noise tests are passed, the sensors are not automatically validated -- bias and in-range failure are still considered in subsequent reasoning.

The monitors apply statistical criteria to detect state change and trend events. Two types of analysis are performed: *appraisal* and *prediction*. Appraisal examines a vector of sample means and applies tests of known significance to identify abnormal trends or shifts in values. These tests will indicate a new state event if one sample mean is outside a control limit, or two consecutive sample means are outside a warning limit. A trend event is indicated if a run of five or more sample means is detected. The significance of each test and the control and warning limits can be easily modified to optimize monitor performance.

Prediction entails making a forecast of future measurement values by extrapolating from a fit of smoothed past data². Many features of prediction are user modifiable, such as the fitting function and the forecast time horizon (measured in data sampling intervals). Forecast values are subjected to tests similar, but of lower significance, to those used in appraisal to determine if an event can be expected to be observed in the future. Prediction is used primarily as a robustness feature in conjunction with monitor interrogation, discussed later.

¹ Ordinary transients are small random disturbances not associated with any identifiable equipment failure.

² Data smoothing is accomplished using a first-order filter.

For every process sensor or constraint equation there is a monitor that exists as an object in the MIDAS knowledge base. In addition, monitors exist for each defined process system. However, there are no monitors for operator actions or tests, which must be entered manually via the User Interface³.

The Monitors contain both factual knowledge in the form of data queues and limits and procedural knowledge in the LISP functions used to analyze data. The monitors must be carefully tuned for each process. Failure of the monitors to detect an event renders further diagnosis impossible. The monitors tuning is discussed in section 5.5.

4.1.2. Event Interpreter

The event interpreter is the name given to the procedural algorithms MIDAS uses to create a diagnosis from event observations. The EI consists of approximately one hundred (100) layered, interactive LISP functions and routines.

The procedural knowledge in the EI is generic and completely portable. The EI provided in MIDAS can be used, without modification, to any diagnose malfunctions in any process for which an event model and monitors exist. The EI is not limited to diagnosis of chemical or refinery processes, but can be used to diagnose any physical system for which an event model can be constructed.

The generality of the EI is achieved using the abstraction provided by the event paradigm. For every event detected, a similar sequence of three fundamental tasks must be performed:

- 1) The event must be cataloged in the active knowledge-base.
- 2) The interpreter must search for possible associations or relationships that might exist between the new event and previously detected events.

³ All events, including state changes, trend changes, and sensor failures, can be entered manually from the User Interface. To realize the full benefits of the inference algorithm, however, this mode of event entry is not recommended.

- 3) The previously existing diagnosis must be revised or a new diagnosis created to incorporate the additional information provided by the new event.

This set of essential tasks, illustrated in figure 4-2, is the same for all events, regardless of whether the event represents a state change, a change in trend, an operator action or any other type of significant process phenomenon.

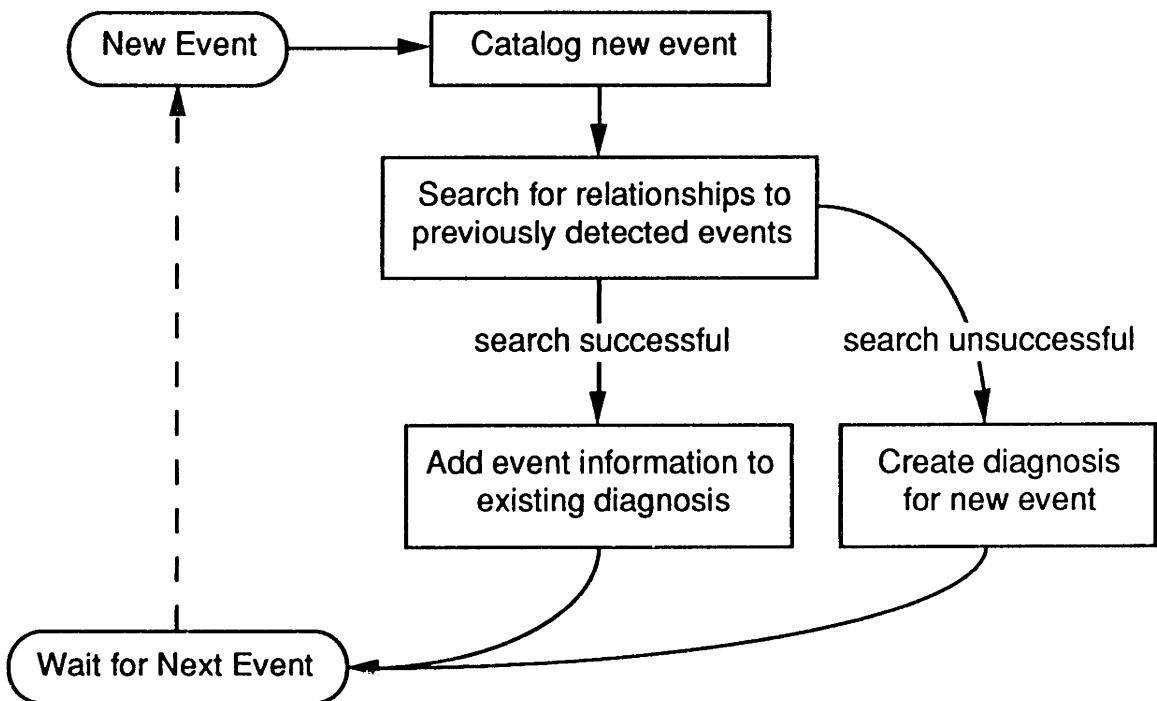


Figure 4-2: Event Interpreter Cycle

The EI interacts primarily with the process and hypothesis models, where events are cataloged, searches are conducted, and diagnoses are logged. Interaction with the monitors is mostly one-way -- monitors sending new events to the EI. The one exception to this rule is *interrogation*. Interrogation is a procedure that allows the interpreter to query a monitor for additional event information. The need for interrogation arises during the search phase of the inference cycle. Often, the interpreter will discover a situation where the diagnosis can be greatly simplified with the addition of one or more events not detected by the monitors. In

these situations, the EI will query the monitors responsible for detected the events of interest and ask for a prediction of possible future events. If the monitor predicts the designated event will be detected in the near future, the EI adds the event to the inference procedure, acting as if the event had already occurred. This robustness feature allows MIDAS to overcome problems associated with variations in the order of event detection and increases the stability of the diagnosis. At present, only state change events can be predicted.

An expiration time can be attached to the predicted event so that it can be deleted if the prediction was erroneous and the event is never detected, but the lack of a real-time operating system prevents the current implementation of MIDAS from using event expiration time. In the present system a predicted event cannot be retracted if it does not occur within the allotted time period.

The event interpreter is event driven, and between event detections, is not active. Thus, the system is always focused on explaining interesting events and system resources are freed for other tasks (i.e. monitoring incoming data) when the inference cycle has been completed for all detected events. The details of the inference cycle are discussed at length in section 4.3.

4.1.3. Process Model

The process model and hypothesis model represent two different sections of the factual knowledge-base. The process model (or static knowledge-base) contains knowledge of process behavior used to interpret events and is created prior to using MIDAS. The process model is the plant specific component of MIDAS; a new process model must be developed whenever MIDAS is used on a new process.

The process model is essentially an event graph represented as a set of objects. The relationship between event graph components and process model objects is summarized in Table 4-1. The word "POTENTIAL" added to objects representing events and root causes differentiates these objects from other objects (contained in the hypothesis model) that represent events or root causes that have been observed or hypothesized. The process model contains the complete set of events and root causes included in the event model, and thus represents all observable events and possible root causes, not necessarily those that have been observed or hypothesized during any given malfunction episode.

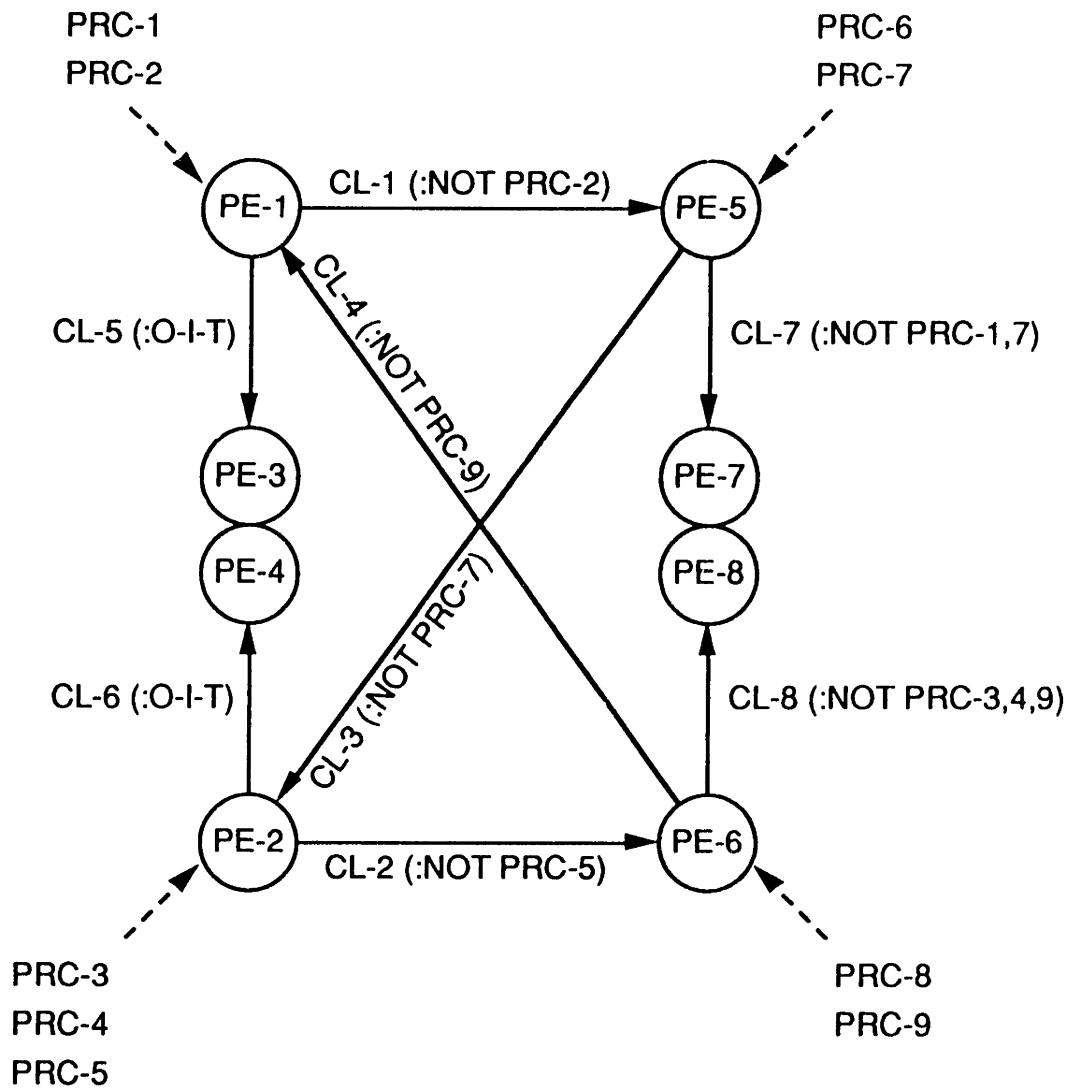
TABLE 4-1: Process Model Objects

Event Graph Component	Corresponding Process Model Object Class
Event	POTENTIAL-EVENT (PE)
Root Cause	POTENTIAL-ROOT-CAUSE (PRC)
LC Link	None ¹
PS Link	COMPILED-LINK (CL)

¹ LC Links are not represented as separate objects, but are modeled via pointers between root causes and events.

When the process model is to be depicted pictorially, it is convenient to represent it as a set of connected PEs and PRCs, as illustrated in figure 4-3 (representing a process model for the event graph shown in figure 3-5). The advantage of this representation is that it explicitly shows the causal relationship between events. This process model can be largely derived from the event graph by replacing event graph nodes with the PEs that result in node activation. For example, the node Level High is replaced by the PE representing the transition: Level OK → Level High. This representation does not exactly match the manner in which the process model is stored in MIDAS, but is a powerful conceptual tool.

Oyeleye (1989) provides an algorithm for automatic construction of a process model from an ESDG. Alternately, process model objects can be created manually from an event graph model, although this is not recommended due to the size and complexity of typical process models. In either case, events associated with constraint residuals, trends, operator actions, and tests must be created manually. Oyeleye's algorithm creates a process model consisting solely of state change events.



PE-1 — Level OK → Level High
 PE-2 — Level OK → Level Low
 PE-3 — Level High → Level OK
 PE-4 — Level Low → Level OK
 PE-5 — Flow OK → Flow High
 PE-6 — Flow OK → Flow Low
 PE-7 — Flow High → Flow OK
 PE-8 — Flow Low → Flow OK

PRC-1 — High Inflow
 PRC-2 — Level Sensor High Bias
 PRC-3 — Tank Leak
 PRC-4 — Low Inflow
 PRC-5 — Level Sensor Low Bias
 PRC-6 — Downstream Leak
 PRC-7 — Flow Sensor High Bias
 PRC-8 — Outlet Blockage
 PRC-9 — Flow Sensor Low Bias

Figure 4-3: Process Model For Tank Process

Oycleye's algorithm creates an ESDG model for an entire plant by linking together small SDG models for each of the plant's units using the global plant topology as a guide. Because SDGs are qualitative, generic SDG unit models can be created for a wide variety of process units, such as pipes, pumps, valves, heat exchangers, and tanks. These unit models are stored in a Model Library, eliminating the need model every unit individually and greatly reducing the overall modeling effort. In section 6.0, results are presented demonstrating a process model for a 22 elementary unit process can be created with four (4) man-hour of effort if unit models exist for all units.

4.1.4. Hypothesis Model

The hypothesis model (or active knowledge-base) contains a record of event observations and diagnoses and is created on-line by the event interpreter. The HM is empty when MIDAS is initially started.

The objects in the hypothesis model represent events and root causes that have been observed or hypothesized. These objects, listed in Table 4-2, have a direct correspondence to the potential events and root causes in the process model. The clarity of MIDAS is improved by the separation of potential and observed objects. The result of the dual object representation is that objects in the hypothesis model create a "mirror image" of certain subsections of the process model. The process model acts as a template for creating associations between objects in the hypothesis model.

Besides improving the clarity of MIDAS, the dual representation has the additional advantage of virtually eliminating the need for elaborate truth maintenance. Truth maintenance, discussed by De Kleer (1986a) (1986b) (1986c), is the task of managing assumptions, so that when an assumption is retracted, the conclusions derived from the assumption will also be retracted. In real-time expert systems, truth maintenance takes on the additional element of time dependency. Conclusions may need to be retracted after the expiration of a "validity interval" based on the characteristic time scale for changes in underlying variables. Complicated agenda mechanisms are required to schedule data acquisition in coordination with validity intervals. MIDAS, however, avoids the problems associated with dynamic truth maintenance by representing facts in the event format. Once deduced, a MIDAS event such as "REACTOR TEMPERATURE SENSOR HIGH AT 13:00 HOURS" remains true forever. The event is a transition that occurred at a specific past time point and cannot be undone. If

reactor temperature subsequently returns to normal, another event -- "REACTOR TEMPERATURE SENSOR NORMAL AT 13:10 HOURS" -- is deduced. Both events can exist without conflict because they occurred at different time points. In fact, MIDAS can associate multiple events involving the same variable to produce a variable trajectory useful in diagnosis.

TABLE 4-2: Hypothesis Model Objects

Event Graph Component	Corresponding Hypothesis Model Object Class
Event	RECORDED-EVENT (RE) EXPECTED-EVENT (EE) LATENT-EVENT (LE)
Root Cause	HYPOTHESIZED-ROOT-CAUSE (HRC)
LC Link	None ¹
PS Link	None ²

¹ LC Links are not represented as separate objects, but are modeled via pointers between root causes and events.

² PS Links are not represented as separate objects, but are modeled via pointers between events.

After a malfunction episode, MIDAS retains a record of the disturbance in the form of a sequence of observed event and hypothesized root cause objects. This record can later be reviewed to follow the complete course of the malfunction and track the MIDAS reasoning procedure.

4.1.5. User Interface

The user interface allows a user to view or modify information contained in the knowledge-base or control certain aspects of event interpretation. The interface is comprised of a set of windows and menus and is mouse controlled. A detailed description of the user interface and its features is deferred to section 5.0.

4.2. Knowledge Representation

All factual knowledge in MIDAS is represented as objects in the GoldWorksTM⁴ programming environment. Charniak and McDermott (1985) and Stefik and Bobrow (1986) provide an introduction to the basic concepts of artificial intelligence and object-oriented programming (OOP).

OOP is an extremely versatile applications development environment. Within the OOP environment, rules, procedures, and data can exist in any proportion. Stephanopoulos et al. (1987) discusses other OOP applications in process engineering.

Examples of many of the objects described in sections 4.2.2 to 4.2.6 are presented in Appendix I.

4.2.1. Definitions

The object-oriented environment provided by GoldWorks contains two types of objects associated with information storage: *frames* and *instances*.

Frames are templates for other objects, allowing the user to define object characteristics. The primary task involved in creating a frame is to specify the *slots* in which information will be stored. Slots can be designated to allow single values, multiple values, or lists; numeric, symbolic, or user specified values; and can have attached "demons" that execute a LISP procedure whenever the contents of the slot are modified. Frames are generally arranged in an inheritance tree in which frames lower in the tree inherit slots from frames higher in the tree. Frames can also be referred to as "object classes".

Instances are instances of a frame. That is, objects created using the template provided by a frame. There can be any number of instances of a given frame, each sharing the same set of slots (but not the same slot values). An instance is completely analogous to a database entry, the data fields (slots) being defined by the frame template. In subsequent discussion, the term "object" generally refers to instance objects. Fikes and Kehler (1985) discuss the features of frame-based knowledge representation.

⁴ Gold Hill Computers, Cambridge, MA.

The GoldWorks environment also includes rules, procedures, and assertions. MIDAS does not make use of rules or assertions. GoldWorks procedures are Common LISP functions. Winston and Horn (1984) provide an introduction to the Common LISP programming language.

4.2.2. Monitor Representation

Each measured variable, constraint residual, or process system has an associated monitor object. Variable and constraint monitors are called DATA-MONITORS. Systems are monitored by SYSTEM-MONITORS. Both monitor classes inherit slots from the top level MONITORS object class.

Whereas, DATA-MONITORS collect and analyze numerical process data, SYSTEM-MONITORS collect and analyze the qualitative states produced by one or more DATA-MONITORS. Therefore, SYSTEM-MONITORS represent a higher level of monitor abstraction (essentially monitoring other monitors). Since systems are not fully implemented in the present event model paradigm, subsequent discussion will focus exclusively on DATA-MONITORS.

Each monitor is independent, collecting and analyzing data for a single source. All statistical analysis is also independent, so any correlation in the data is not considered in the current monitor implementation. A more sophisticated statistical treatment, including data correlation, is left as a possible avenue for monitor improvement.

A description of the slots defined in the DATA-MONITOR object class is presented in Table 4-3. There are five (5) major slot types:

- Analysis control,
- Analysis results,
- Bookkeeping,
- Data queues, and
- Threshold values,

Numerical information is stored in the data queues and compared to threshold values according to the current analysis control parameter values. Current qualitative state, trend, and status are placed in the analysis results slots. The analysis results slots contain demons that activate an event whenever the state, trend, or status change. Threshold values and analysis parameters can be modified from the user interface (see section 5.0). Bookkeeping slots are filled during model creation or by MIDAS routines and should not be modified.

TABLE 4-3: DATA-MONITOR Slots

Slot Name	Type ¹	Data Type	Description
ANALYSIS-HORIZON	A	integer	Maximum length of sample statistics data queues
ASSOCIATED-SYSTEM-MONITORS	B	symbolic	SYSTEM-MONITOR instances that monitor the DATA-MONITOR slots
ASSOCIATION	B	symbolic	Instance of MEASURED-OBJECT monitored by the DATA-MONITOR
DRIFT-CONSTANT	A	real [0,1]	Constant used in the drift filter
DRIFT-HORIZON	A	integer	Maximum length of DRIFT-VECTOR
DRIFT-VECTOR	D	list	Data queue produced by smoothing raw data using the drift filter
FILTER-CONSTANT	A	real [0,1]	Constant used in the data filter
FILTERED-MEAS	D	real	Most recent filtered data point
FILTERED-MEAS-VECTOR	D	list	Queue of filtered data points
FILTERED-SAMPLE-SIZE	A	integer	Maximum length of FILTERED-MEAS-VECTOR
ID	B	symbolic	Name of DATA-MONITOR instance
LAST-UPDATE	B	string	MIDAS time stamp of last data point
MAXIMUM-VALUE	T	real	Maximum credible raw data value
MEAN-LCL	T	real	Lower control limit for sample mean
MEAN-LWL	T	real	Lower warning limit for sample mean
MEAN-NOMINAL	T	real	Nominal mean of sample means
MEAN-UCL	T	real	Upper control limit for sample mean
MEAN-UWL	T	real	Upper warning limit for sample mean
MEAS-TIME	D	real	Measurement time of last measurement
MEAS-TIME-VECTOR	D	list	Queue of measurement times
MEAN-VECTOR	D	list	Queue of sample means
MINIMUM-VALUE	T	real	Minimum credible raw data value
MONITOR-EVENTS	B	symbolic	POTENTIAL-EVENT instances associated with change in status
MONITOR-TYPE	B	symbolic	Indicates the type of object monitored (SENSOR or CONSTRAINT)
PERCEIVED-MONITOR-STATUS	R	symbolic	Status determined by last data analysis
PERCEIVED-STATE	R	symbolic	State determined by last data analysis
PERCEIVED-TREND	R	symbolic	Trend determined by last data analysis

TABLE 4-3: DATA-MONITOR Slots (continued)

Slot Name	Type ¹	Data Type	Description
POLYNOMIAL-ORDER	A	integer	Order of the fitting polynomial used for prediction
POPUP	B	symbolic	Name of an associated popup window (MIDAS use only)
PREDICTED-STATE	R	symbolic	Predicted future state
PREDICTED-STATUS	R	symbolic	Predicted future status
PREDICTED-TREND	R	symbolic	Predicted future trend
PREDICTION-CONFIDENCE	R	symbolic	Level of confidence in predictions (HIGH, LOW, or NONE)
PREDICTION-HORIZON	A	integer	Number of time steps into future to compute predictions
PREDICTION-TIME-VECTOR	D	list	Queue of predicted measurement times
PREDICTION-VALID	B	symbolic	Set to NO when prediction expires
PREDICTION-VECTOR	D	list	Queue of predicted measurements
RANGE-LCL	T	real	Lower control limit for sample ranges
RANGE-NOMINAL	T	real	Nominal mean of sample ranges
RANGE-UCL	T	real	Upper control limit for sample ranges
RANGE-VECTOR	D	list	Queue of sample ranges
RAW-MEAS	D	real	Value of last measurement
RAW-MEAS-VECTOR	D	list	Queue of measurement values
RAW-SAMPLE-SIZE	A	integer	Maximum length of RAW-MEAS-VECTOR
SDEV-LCL	T	real	Lower control limit for sample standard deviations
SDEV-LWL	T	real	Lower warning limit for sample standard deviations
SDEV-NOMINAL	T	real	Nominal mean of sample standard deviations
SDEV-UCL	T	real	Upper control limit for sample standard deviations
SDEV-UWL	T	real	Upper warning limit for sample standard deviations
SDEV-VECTOR	D	list	Queue of sample standard deviations
STATE-EVENTS	B	symbolic	POTENTIAL-EVENT instances associated with change in state
TREND-EVENTS	B	symbolic	POTENTIAL-EVENT instances associated with change in trend

¹ A: Analysis Control B: Bookkeeping D: Data Queue R: Analysis Results T: Threshold

4.2.3. Event Representation

There are two basic event object classes: POTENTIAL-EVENTS contained in the process model and RECORDED-EVENTS contained in the hypothesis model. Two additional classes -- EXPECTED-EVENTS and LATENT-EVENTS -- represent events created as a result of interrogation and are basically identical to recorded events.

Table 4-4 lists the slots of the POTENTIAL-EVENT object class. When a monitor detects an event, it searches for an instance of POTENTIAL-EVENT with the appropriate prior and consequent state, status, and trend and inserts YES into the ACTIVE slot. This action initiates the inference cycle. The other slots of a PE object are used for bookkeeping purposes.

The event paradigm in MIDAS supports compound events. That is, events that involve two or more significant process changes. For example, a PE could be defined with prior state normal, prior trend steady, consequent state high, and consequent trend increasing. Presently, this situation is modeled by two separate events -- one for the change of state and another for the change in trend. An exploration of the uses of compound events, if any, remains a topic for future research.

Table 4-5 lists the slots of the RECORDED-EVENT, EXPECTED-EVENT, and LATENT-EVENT object classes. Because these events represent actual observations, slots such as TIME-OF-DETECTION and PROBABILITY-OF-ACCURATE-DETECTION. These slots contain information unique to the specific observation of the event represented by the object. Other slots store information on related objects in the hypothesis model. The HM has no equivalent of a PSL; therefore, the associations between REs, EEs, and LEs are stored in CAUSE-OF and CAUSED-BY slots.

TABLE 4-4: POTENTIAL-EVENT Slots

Slot Name	Data Type	Description
ACTIVE	symbolic	When set to YES, activates demons that initiate the inference cycle for the event (default is NO)
CONSEQUENT-STATE	symbolic	State that exists after event detection (NIL if event is not a change of state)
CONSEQUENT-STATUS	symbolic	Status that exists after event detection (NIL if event is not a change of status)
CONSEQUENT-TREND	symbolic	Trend that exists after event detection (NIL if event is not a change of trend)
EXCLUSIVE-EVENTS	symbolic	IDs of other POTENTIAL-EVENT instances with the same OBJECT
ID	symbolic	Name of the POTENTIAL-EVENT instance
OBJECT	symbolic	ID of the instance of MEASURED-OBJECT effected by the event
PRIOR-STATE	symbolic	State that exists prior to event detection (NIL if event is not a change of state)
PRIOR-STATUS	symbolic	Status that exists prior to event detection (NIL if event is not a change of status)
PRIOR-TREND	symbolic	Trend that exists prior to event detection (NIL if event is not a change of trend)
RECORD-OF-LAST-OCCURRENCE	symbolic	ID of RECORDED-EVENT instance created the last time the event was detected
SPECIAL-CONDITIONS	list	Special diagnostic conditions attached to event
TIME-OF-LAST-DETECTION	string	Time stamp of raw data point resulting in event detection
TIME-OF-LAST-OCCURRENCE	string	Time stamp of last RECORDED-EVENT
TYPE	symbolic	Type of MEASURED-OBJECT effected (VARIABLE, CONSTRAINT, or SYSTEM)

TABLE 4-5: RECORDED-EVENT, EXPECTED-EVENT, and LATENT-EVENT Slots

Slot Name	Data Type	Description
ACTIVE	symbolic	A YES indicates the event is active in diagnostic inference, otherwise, set to NO
CAUSED-BY	list	Other observed events linked to the event via causal links terminating at this event
CAUSE-OF	list	Other observed events linked to the event via causal link initiating at this event
CLASSIFICATION	symbolic	Classification of this event in the causal network (SOURCE or CONSEQUENCE)
DESCRIPTION	list	Event in MIDAS description format
EVENT	symbolic	Corresponding POTENTIAL-EVENT instance
EVENTS-EXPLAINED	symbolic	Other observed events in the cluster that can be explained by this event via causal propagation
EVENTS-NOT-EXPLAINED	symbolic	Other observed events in the cluster that cannot be explained by this event via causal propagation
EXCLUSIVE-EVENTS	symbolic	The IDs of other POTENTIAL-EVENT instances with the same OBJECT
EXPIRATION	symbolic	Time period event is valid
ID	symbolic	Name of the EVENT instance
OBJECT	symbolic	ID of the instance of MEASURED-OBJECT effected by the event
POPUP	symbolic	An associated popup window (MIDAS use only)
PROBABILITY-OF-ACCURATE-DETECTION	real [0,1]	Probability of true event detection (defaults: 0.95 for REs, 0.40 for EEs and LEs)
PROBABILITY-OF-INACCURATE-DETECTION	real [0,1]	Probability of false event detection (defaults: 0.05 for REs , 0.60 for EEs and LEs)
REALIZED-BY	symbolic	ID of RECORDED-EVENT instance that replaces an EXPECTED-EVENT or LATENT-EVENT.
SPECIAL-CONDITIONS	list	Special diagnostic conditions attached to event
STATE	symbolic	CONSEQUENT-STATE of event
STATUS	symbolic	CONSEQUENT-STATUS of event
SUPERSEDED-BY	symbolic	ID of a RECORDED-EVENT that has replaced the event as the most current effecting the indicated OBJECT

TABLE 4-5: RECORDED-EVENT, EXPECTED-EVENT, and LATENT-EVENT Slots

Slot Name	Data Type	Description
SYMPTOM-OF	symbolic	ID of the INFERRED-MALFUNCTION instance associated with this event
TIME-OF-CESSATION	string	Time stamp of the EVENT superseding this event
TIME-OF-DETECTION	string	Time stamp of the data point responsible for event creation
TIME-OF-OCCURRENCE	string	Time stamp of the EVENT (RECORDED-EVENT)
TIME-OF-POSTULATION	string	Time stamp of the EVENT (EXPECTED-EVENT and LATENT-EVENT)
TIME-OF-REALIZATION	string	Time stamp of the RECORDED-EVENT replacing an EXPECTED-EVENT or LATENT-EVENT
TREND	symbolic	CONSEQUENT-TREND of event
TYPE	symbolic	Type of MEASURED-OBJECT effected (VARIABLE, CONSTRAINT, or SYSTEM)

4.2.4. Root Cause Representation

The slots of the POTENTIAL-ROOT-CAUSE object class are listed in Table 4-6. The PRIMARY-DEVIATION slot is the most critical, listing all the primary symptoms of the root cause. TESTS that could be used to either confirm or eliminate the root cause can be listed in the APPLICABLE-TESTS slot. The PRIOR-PROBABILITY slot can be used to assigned a relative probability to the root cause used in likelihood ranking (see section 5.6.). This value should represent the normalized relative frequency of occurrence of the fault.

HYPOTHESIZED-ROOT-CAUSE objects, listed in Table 4-7, contain considerably more information than PRCs. During the inference cycle, all events and tests that support or oppose a particular HRC accumulate in slots of the HRC instance. Several likelihood rankings for the HRC are computed based on the relative weight of evidence supporting and opposing the HRC.

TABLE 4-6: POTENTIAL-ROOT-CAUSE Slots

Slot Name	Data Type	Description
APPLICABLE-TESTS	symbolic	IDs of TEST instances containing diagnostic conditions useful in diagnosis of the root cause
FUNCTION-CLASS	symbolic	The class of the unit function effected by the root cause (PRC, SEN, CSEN, CNT)
ID	symbolic	Name of the POTENTIAL-ROOT-CAUSE instance
NECESSARY-CONDITIONS	list	Conditions necessary for inclusion in diagnostic inference (not currently supported)
POPUP	symbolic	An associated popup window (MIDAS use only)
PRIMARY-DEVIATION	list	All primary symptoms of the root cause (listed in MIDAS event description format)
PRIOR-PROBABILITY	real [0,1]	A relative probability factor indicating the a priori root cause likelihood compared to other root causes

TABLE 4-7: HYPOTHEZIZED-ROOT-CAUSE Slots

Slot Name	Data Type	Description
ACTIVE	symbolic	A YES indicates the root cause is active in diagnostic inference, otherwise, set to NO
CONDITIONAL-PROBABILITY	real [0,1]	A likelihood ranking based on prior probabilities
DIAGNOSIS-TYPES	symbolic	Types of events supporting the root cause as a candidate (VARIABLE, CONSTRAINT, or SYSTEM)
HYPOTHESIZED-BY	symbolic	ID of EVENT instances that are primary symptoms of the root cause
ID	symbolic	Name of the HYPOTHESIZED-ROOT-CAUSE instance
INFERRRED-MALFUNCTION	symbolic	ID of the INFERRRED-MALFUNCTION instance associated with the root cause
NORMALIZED-CONDITIONAL-PROBABILITY	interval [0,1]	Evidential interval based on prior probabilities
NORMALIZED-PROBABILISTIC-LIKELIHOOD	interval [0,1]	Evidential interval based on detection probabilities
NORMALIZED-RELATIVE-LIKELIHOOD	interval [0,1]	Evidential interval based on evidence weights
OPPOSING-TESTS	symbolic	IDs of active TEST instances opposing the root cause candidate
POPUP	symbolic	An associated popup window (MIDAS use only)
PROBABILISTIC-LIKELIHOOD	real [0,1]	A likelihood ranking based on detection probabilities
RELATIVE-LIKELIHOOD	real	A likelihood ranking based on evidence weights
ROOT-CAUSE	symbolic	ID of the POTENTIAL-ROOT-CAUSE instance corresponding to this root cause.
STRONGLY-OPPOSING-EVIDENCE	symbolic	IDs of RECORDED-EVENT instances opposing the root cause candidate
STRONGLY-SUPPORTING-EVIDENCE	symbolic	IDs of RECORDED-EVENT instances supporting the root cause candidate
SUPPORTING-TESTS	symbolic	IDs of active TEST instances supporting the root cause candidate
WEAKLY-OPPOSING-EVIDENCE	symbolic	IDs of EXPECTED-EVENT and LATENT-EVENT instances opposing the root cause candidate
WEAKLY-SUPPORTING-EVIDENCE	symbolic	IDs of EXPECTED-EVENT and LATENT-EVENT instances supporting the root cause candidate

4.2.5. Link Representation

COMPILED-LINK objects are the MIDAS representation of a PSL, and are present only in the process model. Table 4-8 lists the slots of a CL object. Information on the initial and terminal nodes for the link is stored in the SOURCE-DESCRIPTION and RESULT-DESCRIPTION slots. Included are slots for all the types of conditions that could be attached to the link and the conditions under which the link should be activated or deactivated.

4.2.6. Other Objects

TEST events are represented as a separate object class in MIDAS and do not share the same inference cycle used for REs, EEs, and LEs. TEST inference is driven entirely by the conditions attached to the TEST instance⁵. The slots of a TEST object are listed in Table 4-9.

Other object classes, not introduced as part of the basic event model paradigm, are included in MIDAS for bookkeeping purposes. The MEASURED-OBJECT object class stores information on the current, postulated, and past states, trends, and status of a measured variable, constraint residual, or system. MEASURED-OBJECT instances also play a role in the process model by providing pointers between PE, PRC, and CL objects. Table 4-10 lists the slots of the MEASURED-OBJECT class.

The INFERRRED-MALFUNCTION (IM) object class stores information pertaining to an entire cluster of related REs, EEs, LEs, HRCs, and TESTs. Because not all observed events may be explainable by a single source, there may be multiple clusters of events, and therefore, multiple IMs. The name "inferred malfunction" derives from the fact that one malfunction is inferred to exist for each cluster of connected events. Table 4-11 lists the slots of the INFERRRED-MALFUNCTION object class.

An IM object maintains a list of all events associated with the cluster, all root cause candidates, all tests that have been performed, and the qualitative classification of the malfunction. The possible IM classifications are PERSISTENT, OSCILLATORY, SPURIOUS, ONGOING-TRANSIENT, COMPLETED-TRANSIENT, and CORRECTED.

⁵ The set of possible TEST conditions is the same as the set of conditions that can be attached to PSLs.

Definitions of these classifications are given in section 4.4.2. IM classification is stored in the STATUS slot.

TABLE 4-8: COMPILED-LINK Slots

Slot Name	Data Type	Description
ACTIVATE-IF	list	Conditions that trigger link activation (NIL if always active)
ACTIVE	symbolic	A YES indicates the link is active in diagnostic inference, otherwise, set to NO
ADD-CONDITIONS	list	:ADD conditions attached to link
DEACTIVATE-IF	list	Conditions that trigger link deactivation (NIL if always active)
DELAY	any	Time delay associated with disturbance propagation along link (not currently supported)
ID	symbolic	Name of the COMPILED-LINK instance
NOT-CONDITIONS	list	:NOT conditions attached to link
ONLY-IF-CONDITIONS	list	:ONLY-IF conditions attached to link
POPUP	symbolic	An associated popup window (MIDAS use only)
REPLACE-CONDITIONS	list	:REPLACE conditions attached to link
RESULT-DESCRIPTION	list	DESCRIPTION of terminal EVENT
RESULT-NODE	symbolic	ID of MEASURED-OBJECT instance effected by terminal event
RESULT-STATE	symbolic	CONSEQUENT-STATE of terminal event
RESULT-STATUS	symbolic	CONSEQUENT-STATUS of terminal event
RESULT-TREND	symbolic	CONSEQUENT-TREND of terminal event
RETAIN-CONDITIONS	list	:RETAIN conditions attached to link
SOURCE-DESCRIPTION	list	DESCRIPTION of initial EVENT
SOURCE-NODE	symbolic	ID of MEASURED-OBJECT instance effected by initial event
SOURCE-STATE	symbolic	CONSEQUENT-STATE of initial event
SOURCE-STATUS	symbolic	CONSEQUENT-STATUS of initial event
SOURCE-TREND	symbolic	CONSEQUENT-TREND of initial event
STRENGTH	any	Disturbance propagation gain along link (not currently supported)

TABLE 4-9: TEST Slots

Slot Name	Data Type	Description
ACTIVE	symbolic	A YES indicates the test is active in diagnostic inference, otherwise, set to NO
ADD-CONDITIONS	list	:ADD conditions attached to TEST instance
EXCLUSIVE-TESTS	symbolic	IDs of mutually exclusive TEST instances
ID	symbolic	Name of the TEST instance
NOT-CONDITIONS	list	:NOT conditions attached to TEST instance
ONLY-IF-CONDITIONS	list	:ONLY-IF conditions attached to TEST instance
POPUP	symbolic	An associated popup window (MIDAS use only)
PROBABILITY-OF-ACCURATE-DETECTION	real [0,1]	Probability of true test result (default: 1.0)
PROBABILITY-OF-INACCURATE-DETECTION	real [0,1]	Probability of false test result (default: 0.0)
REPLACE-CONDITIONS	list	:REPLACE conditions attached to TEST instance
RETAIN-CONDITIONS	list	:RETAIN conditions attached to TEST instance
TIME-LAST-PERFORMED	string	Time stamp of last activation of TEST instance

TABLE 4-10: MEASURED-OBJECT Slots

Slot Name	Data Type	Description
CURRENT-STATE	symbolic	CONSEQUENT-STATE of last related RECORDED-EVENT instance
CURRENT-STATUS	symbolic	CONSEQUENT-STATUS of last related RECORDED-EVENT instance
CURRENT-TREND	symbolic	CONSEQUENT-TREND of last related RECORDED-EVENT instance
EVENT-HISTORY	symbolic	IDs of all related EVENT instances listing the MEASURED-OBJECT as OBJECT
EVENT-SET	symbolic	IDs of all POTENTIAL-EVENT instances listing the MEASURED-OBJECT as OBJECT
ID	symbolic	Name of the MEASURED-OBJECT instance
LAST-UPDATED	string	Time stamp of last related observed EVENT instance
LOCAL-CAUSES	symbolic	All POTENTIAL-ROOT-CAUSE instances listing the MEASURED-OBJECT as a PRIMARY-DEVIATION
POPUP	symbolic	An associated popup window (MIDAS use only)
POSTULATED-STATE	symbolic	CONSEQUENT-STATE of last related EXPECTED-EVENT or LATENT-EVENT instance
POSTULATED-STATUS	symbolic	CONSEQUENT-STATUS of last related EXPECTED-EVENT or LATENT-EVENT instance
POSTULATED-TREND	symbolic	CONSEQUENT-TREND of last related EXPECTED-EVENT or LATENT-EVENT instance
PRECURSOR-LINKS	symbolic	IDs of all COMPILED-LINK instances listing the MEASURED-OBJECT as a RESULT-NODE
STATE-HISTORY	symbolic	List of all past CURRENT-STATES
SUCCESSOR-LINKS	symbolic	IDs of all COMPILED-LINK instances listing the MEASURED-OBJECT as a SOURCE-NODE
TREND-HISTORY	symbolic	List of all past CURRENT-TRENDS
TYPE	symbolic	Type of MEASURED-OBJECT (VARIABLE, CONSTRAINT, SYSTEM)

TABLE 4-11: INFERRED-MALFUNCTION Slots

Slot Name	Data Type	Description
ACTIVE	symbolic	Set to YES whenever active in diagnostic inference, otherwise, set to NO
APPARENT-SOURCE-EVENTS	symbolic	IDs of associated RECORDED-EVENT instances classified as SOURCE events
APPLICABLE-TESTS	symbolic	IDs of TEST instances that could supply addition diagnostic resolution
CREATED-BY	symbolic	ID of the first RECORDED-EVENT instance associated with the INFERRED-MALFUNCTION
EVENTS-EXPLAINED	symbolic	IDs of all EVENT instances associated with the INFERRED-MALFUNCTION
HYPOTHESIZED-ROOT-CAUSES	symbolic	IDs of all HYPOTHESIZED-ROOT-CAUSE instances associated with the INFERRED-MALFUNCTION
ID	symbolic	Name of the INFERRED-MALFUNCTION instance
LAST-EVALUATION-TIME	integer	Number of seconds elapsed during evaluation of last associated RECORDED-EVENT instance
LEVEL-OF-EVALUATION	integer	Current maximum network evaluation depth
POPUP	symbolic	An associated popup window (MIDAS use only)
POSSIBLE-LATENT-SOURCES	symbolic	IDs of associated LATENT-EVENT instances
ROOT-CAUSE-DISPLAY	symbolic	Contents of the ROOT-CAUSE-LIST slot formatted for display (MIDAS use only)
ROOT-CAUSE-LIST	symbolic	List of POTENTIAL-ROOT-CAUSE instances corresponding to the instances listed in the HYPOTHESIZED-ROOT-CAUSE slot
STATUS	symbolic	Qualitative description of INFERRED-MALFUNCTION (PERSISTANT, OSCILLATORY, ONGOING-TRANSIENT, COMPLETED-TRANSIENT, SPURIOUS, CORRECTED)
TESTS-APPLIED	symbolic	IDs of active TEST instances associated with the INFERRED-MALFUNCTION
TIME-OF-CREATION	string	Time stamp at INFERRED-MALFUNCTION creation
TIME-OF-removal	string	Time stamp at INFERRED-MALFUNCTION deactivation

4.3. The MIDAS Inference Cycle

The MIDAS inference cycle is a set of tasks performed whenever a new RE is detected. The goal of the inference cycle is to integrate the new event into the existing hypothesis model and update the diagnosis.

The inference cycle consists of four major phases:

- 1) Event Creation,
- 2) Search and Linkage,
- 3) Source Evaluation, and
- 4) Evidence Evaluation.

At the end of the inference cycle, MIDAS waits for a new event detection, having garnered all available information from the last event.

4.3.1. Event Creation

Whenever a monitor detects a change of state, trend, or status, it checks the contents of its STATE-EVENTS, TREND-EVENTS, and MONITOR-EVENTS slots for POTENTIAL-EVENT instances that match the detected change. Inserting "YES" in the ACTIVE slot of the PE initiates the inference cycle, starting with the creation of a new RECORDED-EVENT instance and the updating of the associated MEASURED-OBJECT instance. At this stage, most of the slots of the new RE will be empty. Only basic information, copied from the PE, and detection information such as TIME-OF-DETECTION and PROBABILITY-OF-ACCURATE-DETECTION will be available.

4.3.2. Search and Linkage

The next stage of the inference cycle is to search the hypothesis model for existing IM clusters to which the new RE can be linked. The process model is used to guide for the search. Based on the results of the search, one of ten (10) diagnostic actions are performed. Diagnostic actions create a network of causal links between events in the hypothesis model,

create necessary HYPOTHEZIZED-ROOT-CAUSE instances, and evaluate the evidence contained in the causal network.

Before a general search begins, all previously postulated EEs and LEs are checked. It is possible that the new RE could be the realization of a previous event postulation. If this is the case, DIAGNOSTIC-ACTION-1 is performed, replacing the existing EE or LE with the new RE but otherwise leaving the existing event network undisturbed. The EE or LE is stamped as having been realized and is removed from the hypothesis model. If the new event is not the realization of a previously postulated EE or LE, a general search begins, looking for causal links to other events.

Search is a depth-first progression through the process model starting at the PE corresponding to the new RE. Depending on the direction of search (forward or backward) a CL that originates or terminates at the PE is selected and the PE at the far end of the link (dubbed the "far PE") is checked. If the far PE has an active corresponding RE, EE, or LE then a link between the new RE and the existing observed event is established in the hypothesis model. If the search was in the forward direction, the new RE can explain the existing event. If the search was backward, the new RE can be explained by the existing event.

If the far PE does not have an active corresponding RE, EE, or LE, the search along that CL is terminated or an interrogation is performed. Interrogation seeks to determine if an EE or LE corresponding to the far PE can be created and added to the hypothesis model. If the monitor responsible for detecting the far PE predicts that it will be observed in the near future, the EE or LE is created⁶ and the search can continue from the far PE until a previously existing event is found or an undetected event that fails interrogation is encountered. In this way, a new RE can be linked to existing events separated by several intervening undetected events. The maximum depth of the search can be bounded (see section 5.3.2.4).

There are four possible outcomes of the search procedure:

- 1) The new RE cannot be linked to any existing events,
- 2) The new RE can be explained by previously existing events,
- 3) The new RE can explain previously existing events, or

⁶ Usually, if the search is in the backward direction a LE is created; if the search is in the forward direction an EE is created.

- 4) The new RE can explain certain existing events and can be explained by other events.

Each of these cases is discussed below.

Case 1

If a new RE cannot be linked to any existing events in the hypothesis model, it becomes the first event in a new IM cluster. DIAGNOSTIC-ACTION-2 performs this function.

Case 2

If the new RE can be explained by existing events, as shown in figure 4-4A, it is the result of downstream propagation of a fault disturbance. The overall diagnosis will change little in this case. Downstream propagation is expected and tends to confirm the existing diagnosis. DIAGNOSTIC-ACTION-3 or DIAGNOSTIC-ACTION-5 create the necessary causal links and update the IM. DIAGNOSTIC-ACTION-5 is used when the event can be linked to two or more separate IM clusters.

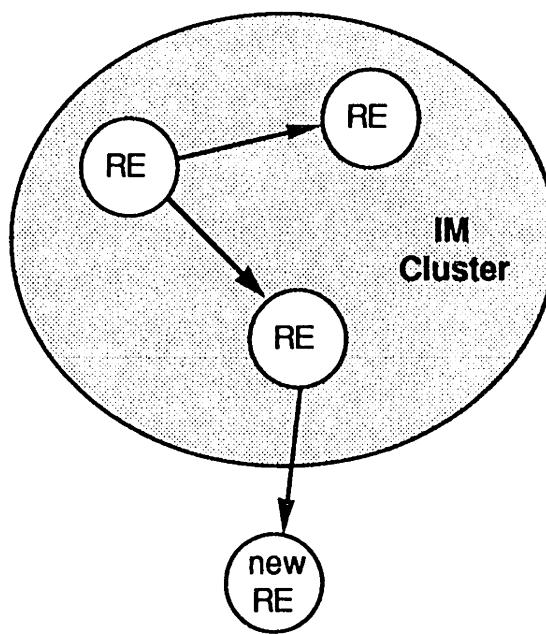


Figure 4-4A: Case 2 Event

Case 3

If the new RE explains other existing events it is a previously undetected source event (see section 4.3.4.). Figure 4-4B illustrates the case. Major revisions to the existing diagnosis can be expected whenever a previously undetected source is discovered because the existing diagnosis cannot explain the new event. DIAGNOSTIC-ACTION-4 or DIAGNOSTIC-ACTION-6 establish links and update the IM. DIAGNOSTIC-ACTION-6 is used when the event is the source of two or more separate IM clusters, in which case, the clusters are coalesced into a single cluster.

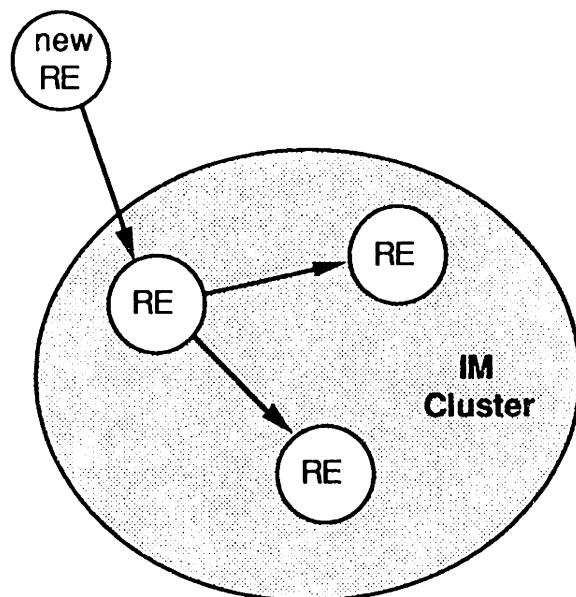


Figure 4-4B: Case 3 Event

Case 4

The situation where a RE explains certain events and is explained by others can arise if the new RE completes an alternate causal path within a single IM cluster, illustrated in figure 4-4C, or if the new RE constitutes a bridge between two separate IM clusters, illustrated in figure 4-4D. If the later is the case, the two IM clusters will be coalesced into a single IM cluster (i.e. one cluster will be absorbed by the other). DIAGNOSTIC-ACTION-7, DIAGNOSTIC-ACTION-8, DIAGNOSTIC-ACTION-9, or DIAGNOSTIC-ACTION-10 treat this case, depending on the number of IM clusters involved.

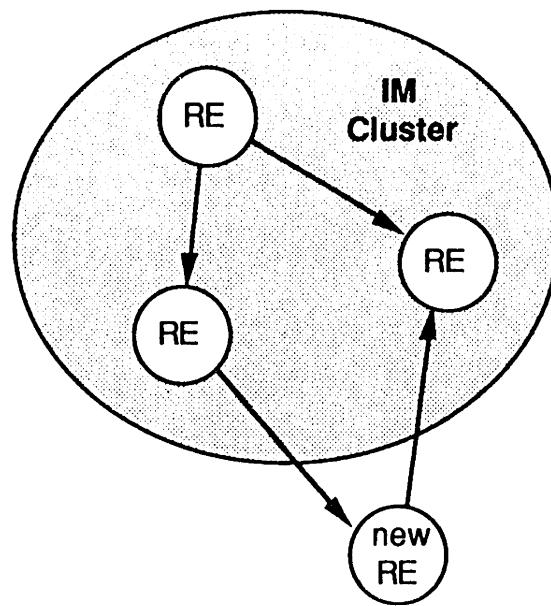


Figure 4-4C: Case 4 Event (Single Cluster)

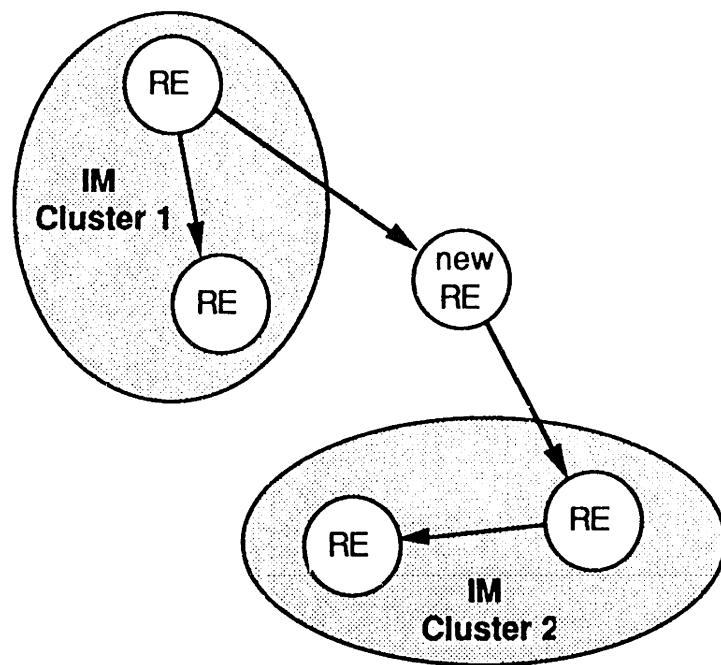


Figure 4-4D: Case 4 Event (Dual Cluster)

4.3.3. Evaluating Source Events

All events in an IM cluster are classified as *source events* or *consequence events*. A source event is defined as an event that can explain through causal pathways all other events in its IM cluster or minimally all consequence (i.e. non-source) events in its IM cluster. In subsequent diagnostic reasoning, source events are assumed to be primary symptoms of the root cause. Within each IM cluster there must be at least one source event. Source events are located by analyzing the causal network of an IM cluster to determine which events can be explained by other events in the network.

The first event in a new IM cluster is always a source event. Multiple source events can arise if subsequent events form causal loops, shown in figure 4-5, or if the fault has multiple primary symptoms, as illustrated in figure 4-6.

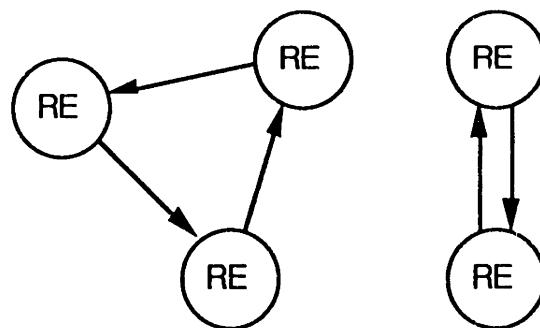


Figure 4-5: Examples of Causal Loops

In the case of causal loops, there is no way to distinguish the source event from the consequence events, therefore, all events must be considered as possible sources⁷.

⁷ The order of event detection could be used to classify source events in a causal loop (i.e. the first event could be considered the source event), but such a classification strategy would lead to inaccurate diagnoses if events are not detected in order (often the case in real plants).

If a conflict arises such that no events satisfy the criteria for source events, MIDAS will retain the source events of the previous inference cycle. These conflicts are usually resolved when more events are detected.

HRC instances are created for all PRCs that are *local causes*⁸ of the source events of the IM. Therefore, determining source events is a critical step in developing the final diagnosis. If an event is classified as a source event and later reclassified as a consequence event, any HRCs created as a result of the event being a source are removed.

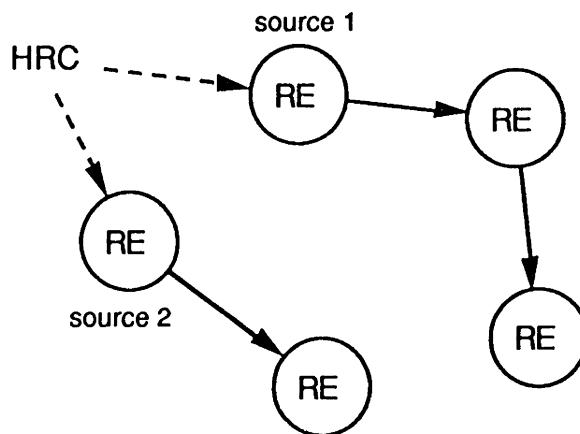


Figure 4-6: Multiple Primary Symptoms

4.3.4. Evaluating Root Cause Evidence

When a new RE has been linked to an IM, classified as source or consequence, and all HRCs have been created, the causal network of the IM cluster is evaluated to determine which events support and oppose various HRCs.

Each event in the IM cluster is a piece of evidence that can support a HRC (i.e. increase its relative likelihood ranking) or oppose a HRC (i.e. decrease its relative likelihood ranking).

⁸ A root cause is a local cause of an event if the event is a primary symptom of the root cause.

MIDAS sorts the set of root cause candidates associated with an IM cluster in order of relative rankings, with the highest ranked HRC representing the most candidate.

An event will support a HRC if

- 1) The event is a source event that is a primary symptom of the HRC, or
- 2) A free causal pathway⁹ exists from a primary symptom of the HRC to the event.

An event will oppose a HRC if

- 1) The event is a source event that is not a primary symptom of the HRC,
- 2) No causal pathway exists from any primary symptom of the HRC to the event, or
- 3) All causal pathways from the primary symptoms of the HRC to the event are blocked by NOT or ONLY-IF conditions.

During evidence evaluation, all IMs are considered separately. That is, events from one IM cluster are not considered evidence supporting or opposing the HRCs of another cluster. Test events can also provide evidence supporting or opposing HRCs, but these events are evaluated using a different procedure.

Three levels of evidence are currently supported in MIDAS. Test results provide the strongest evidence for or against a particular HRC. In fact, the default settings of MIDAS assume that tests are infallible. RECORDED-EVENTS are strong evidence for or against a particular HRC because the statistical tests involved in detection of a RE carry a high level of significance. EXPECTED-EVENTS and LATENT-EVENTS are considered weak evidence for or against a HRC. These events are created in response to interrogation, which uses tests of much lower significance than REs.

These three levels of evidence are reflected in the values of the PROBABILITY-OF-ACCURATE-DETECTION (POAD) and PROBABILITY-OF-INACCURATE-DETECTION (POID) slots of TESTS, REs, EEs, and LEs. These slot values are used to compute likelihood rankings for each HRC according to the formulas presented in section 5.6. Default values are given in Table 4-12.

⁹ A free causal pathway is an unbroken, unblocked chain of causal links from one event to another.

TABLE 4-12: Default POAD and POID Values†

Event Type	POAD	POID
TEST	1.0	0.0
RECORDED-EVENT	0.95	0.05
EXPECTED-EVENT	0.40	0.60
LATENT-EVENT	0.40	0.60

† In general: $0.0 \leq (\text{POAD} + \text{POID}) \leq 1.0$

Rankings need not be based on only three levels of evidence. If the monitors were capable of actually computing POAD and POID values for an event based on the which statistical tests are violated rather than using defaults based on event type, there would be infinite levels of evidence strength. Enhancing the monitors to include this capability is left for future research.

MIDAS does not actually remove HRCs for which there is opposing evidence as was done in the DEA system [Kramer (1988)] because subsequent events may increase the ranking of a low ranked HRC. However, HRCs with low likelihood rankings may not be displayed.

4.3.5. Inference Cycle Examples

In this section, four (4) examples are presented to illustrate the results of the diagnostic inference procedure applied in a variety of situations. The process model for these examples is shown in figure 4-7. It is an abstract model, not meant to represent any particular process, but is representative of process models for a variety of plants¹⁰. For clarity, activatable PSLs have not been included in this model; therefore, this process model does not change during diagnosis.

¹⁰ A process model for a real plant would be much larger than the one presented in figure 4-7. The model in figure 4-7 might represent a portion of a real process model.

In figure 4-7, PEs are represented as clear lettered circles and PRCs are represented as clear numbered diamonds. In figures 4-8 to 4-11, REs are represented as dark shaded circles with letters identical to the corresponding PE. EEs and LEs are also represented as circles, but are not shaded as darkly. HRCs are represented as shaded diamonds with numbers identical to the corresponding PRC. CLs are represented as solid arrows. Selected CLs have NOT conditions attached, indicating blocked PRCs. The primary symptoms of a PRC are indicated by dashed lines.

In all examples, the quoted likelihood rankings are those produced by the CONDITIONAL-PROBABILITY function discussed in section 5.6.5. All PRCs are assumed to have equal prior probabilities.

Inference Example 1

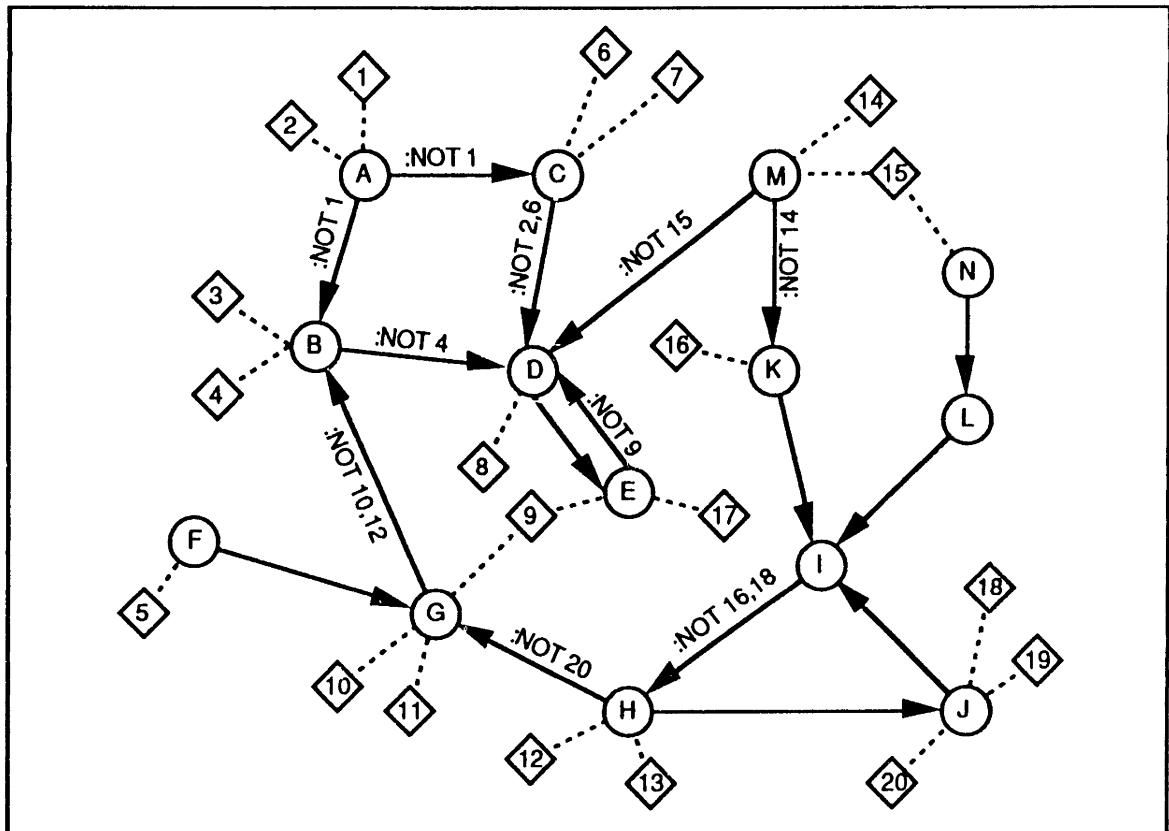
This example demonstrates the use of conditions and multiple primary symptoms in generating a diagnosis. The results of this example are presented in Table 4-13. The hypothesis model after detection of each event is shown in figure 4-8.

Stage 1) The first event detected corresponds to PE-M. HRCs are created for the two local causes of this event -- PRC-14 and PRC-15. Because the IM cluster has only one event that equally supports both HRCs, the likelihood ranking of each HRC is 0.5.

Stage 2) The second event detected (RE-K) can be linked to the first by a CL. However, the propagation of HRC-14 along the link is blocked by a NOT condition. Therefore, HRC-14 is supported by RE-M but opposed by RE-K. HRC-15 is supported by both REs. The rankings of the two HRCs reflect the greater support for HRC-15.

Stage 3) The third detected event (RE-N) cannot be linked to the existing cluster by CLs, but joins the cluster because it is an alternate primary symptom of HRC-15. As a primary symptom, RE-N supports HRC-15 but opposes HRC-14. The IM cluster now has two parallel sources.

Stage 4) The fourth event, corresponding to PE-L, can be linked to RE-N via a CL. The link has no conditions, so RE-L supports all HRCs supported by RE-N and opposes all HRCs opposed by RE-N. With the addition of this event, the evidence in favor of HRC-15 becomes overwhelming.



- — POTENTIAL-EVENT Object
- ◇ — POTENTIAL-ROOT-CAUSE Object
- — COMPILED-LINK Object

Figure 4-7: Abstract Process Model

TABLE 4-13: Inference Example 1

Stage	Events Detected	Source Events	Hypothesized Root Causes
1	M	M	14 (ranking = 0.500) 15 (ranking = 0.500)
2	M,K	M	14 (ranking = 0.050) 15 (ranking = 0.950)
3	M,K,N	M,N	14 (ranking = 0.003) 15 (ranking = 0.997)
4	M,K,N,L	M,N	14 (ranking = 0.000) 15 (ranking = 1.000)

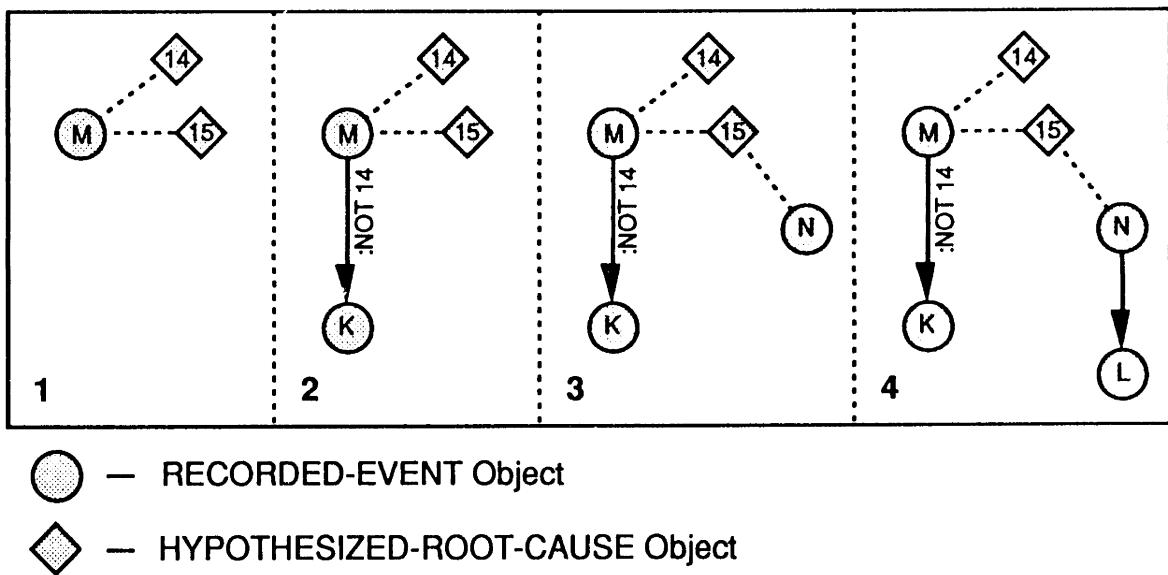


Figure 4-8: Evolving Hypothesis Model for Inference Example 1

Inference Example 2

This example demonstrates how multiple IM clusters can be created and coalesced. The results of this example are presented in Table 4-14. Figure 4-9 illustrates the evolving hypothesis model.

- Stage 1) The first event detected corresponds to PE-B and forms the a new IM cluster containing one event. As in example 1, both HRCs created have equal likelihood ranking.
- Stage 2) The second event (RE-C) cannot be linked to RE-B and forms a second IM cluster (i.e. the interrogation of PE-A failed to detect a sub-threshold event). The interpretation of two clusters is that two separate faults exist in the process -- one responsible for each cluster. Both RE-B and RE-C are considered sources in their respective clusters. The possible root causes are HRC-3 and HRC-4, equally likely to be responsible for the first cluster, and HRC-6 and HRC-7, equally likely to be responsible for the second cluster.
- Stage 3) The appearance of the third event (RE-A) provides a common source of both clusters. The two clusters are coalesced into a single cluster and RE-B and RE-C are reclassified as CONSEQUENCE events. The local causes of RE-A, the sole remaining SOURCE event, are added to the list of candidates and the local causes of RE-B and RE-C are removed. Because of the NOT conditions on the links connecting the REs, the weight of evidence is overwhelmingly in favor of HRC-2.

TABLE 4-14: Inference Example 2

Stage	Events Detected	Source Events	Hypothesized Root Causes	
1	B	B	3	(ranking = 0.500)
			4	(ranking = 0.500)
2	B,C	B (IM-1)	3	(ranking = 0.500)
		C (IM-2)	4	(ranking = 0.500)
3	B,C,A	A	6	(ranking = 0.500)
			7	(ranking = 0.500)
			1	(ranking = 0.003)
			2	(ranking = 0.997)

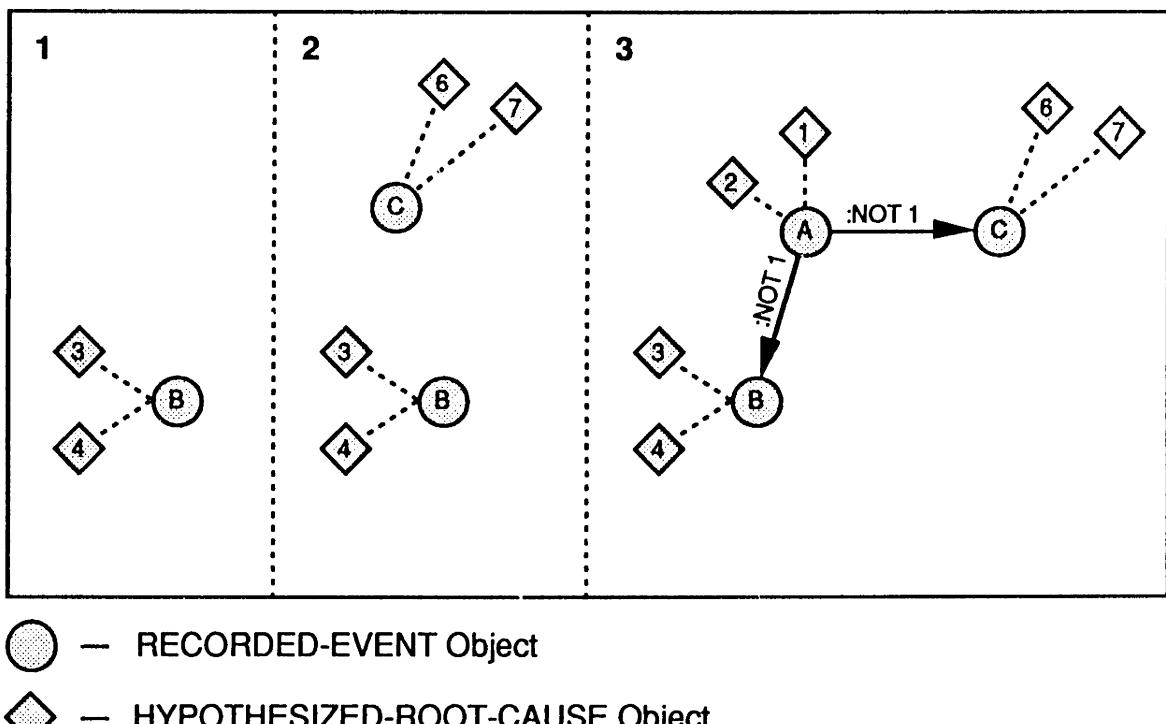


Figure 4-9: Evolving Hypothesis Model for Inference Example 2

Inference Example 3

This example demonstrates the role of EXPECTED-EVENTS and LATENT-EVENTS in the inference procedure. The results of this example are presented in Table 4-15 and the evolving hypothesis model is shown in figure 4-10.

Stage 1) The first event detected is RE-G, but monitor interrogation indicates that PE-F is forecast to be detected in the near future. A latent event (LE-F) is created to account for this prediction¹¹. Both RE-G and LE-F are considered as possible SOURCE events since LE-F has not actually been observed and may be erroneous. HRC-5 is supported by LE-F and RE-G. HRC-9, HRC-10, and HRC-11 are supported by RE-G but opposed by LE-F. Because a LE provides only weak evidence, however, HRC-5 is ranked lower in the overall diagnosis.

¹¹ A LATENT-EVENT is created for PE-F because when detected, it will a SOURCE event.

Stage 2) The second event detected is RE-D which can only be linked to the existing cluster through PE-B which has not been observed. Interrogation of the monitor responsible for PE-B indicates it can be expected to be observed in the near future. Consequently, an EXPECTED-EVENT (EE-B) is created to complete the link¹². EE-B and RE-D provide support for all HRCs except HRC-10 which is blocked by a NOT condition.

TABLE 4-15: Inference Example 3

Stage	Events Detected	Source Events	Hypothesized Root Causes	
1	G (F latent)	G,F	5	(ranking = 0.182)
			9	(ranking = 0.273)
			10	(ranking = 0.273)
			11	(ranking = 0.273)
2	G,D (F latent) (B expected)	G,F	5	(ranking = 0.245)
			9	(ranking = 0.368)
			10	(ranking = 0.020)
			11	(ranking = 0.368)
3	G,D,B (F latent)	G,F	5	(ranking = 0.250)
			9	(ranking = 0.375)
			10	(ranking = 0.001)
			11	(ranking = 0.375)
4	G,D,B,F	F	5	(ranking = 1.000)

Stage 3) The third event is the realization of EE-B. The new RE-B takes the place of EE-B in the hypothesis model. Although RE-B supports and opposes the same HRCs as EE-B did previously, RE-B provides stronger evidence and HRC-10 slips further in the likelihood rankings.

¹² An EXPECTED-EVENT is created for PE-B because when detected it will be a CONSEQUENCE event.

Stage 4) The last event is the realization of LE-F. Now RE-F becomes the sole SOURCE event for the cluster and HRC-5 becomes the sole root cause candidate.

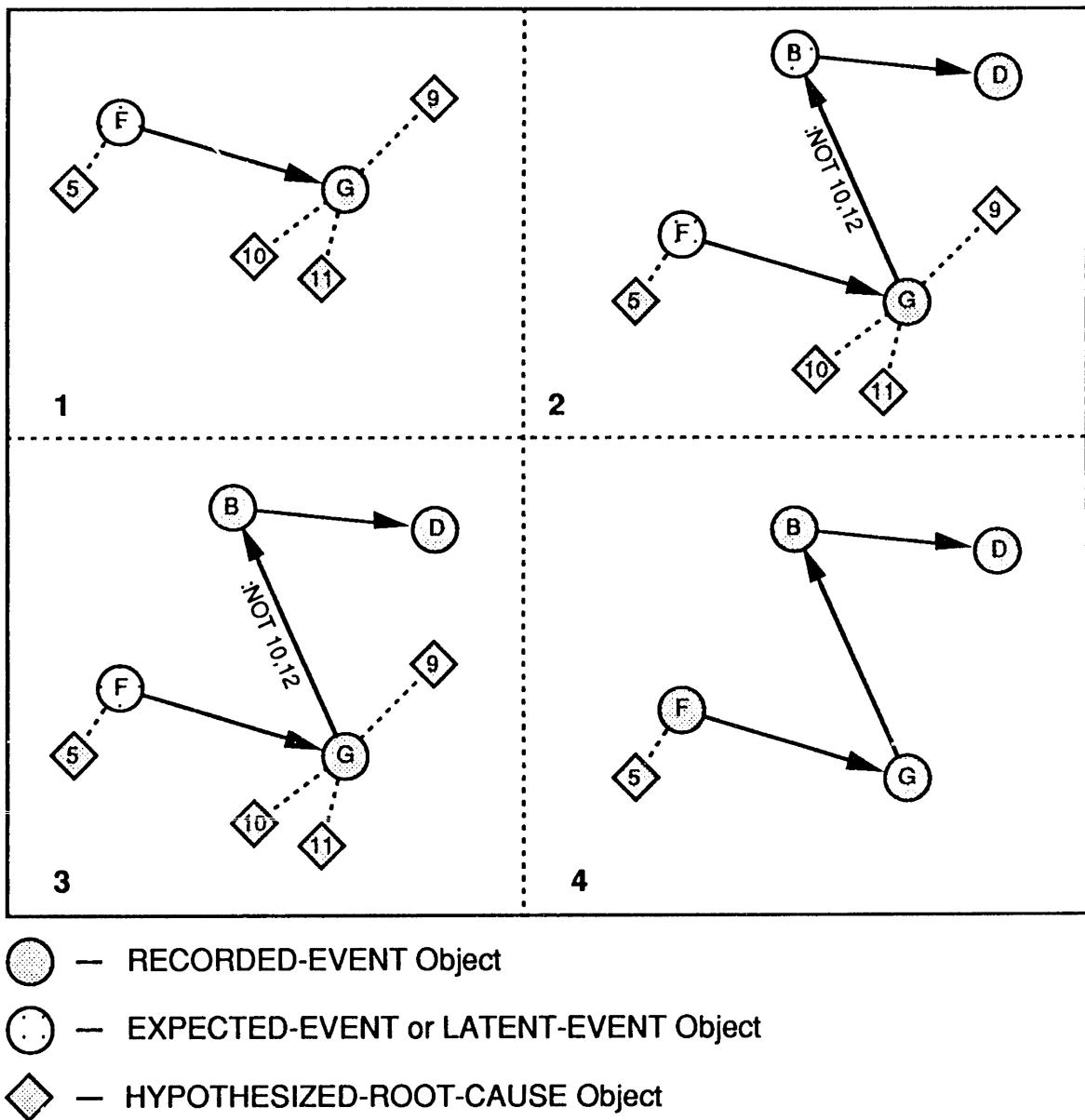


Figure 4-10: Evolving Hypothesis Model for Inference Example 3

Inference Example 4

This example demonstrates how it is possible for the diagnosis to degrade when causal loops are encountered. Table 4-16 presents the results of this example, while figure 4-11 illustrates the evolving hypothesis model.

Stage 1) The first event is RE-H which creates two HRCs -- HRC-12 and HRC-13.

Stage 2) The second event is RE-J which is linked to RE-H and classified as a CONSEQUENCE event. The likelihood rankings of HRC-12 and HRC-13 remain unchanged.

Stage 3) The third event is RE-I which completes a three event causal loop. It is now impossible to determine which event is the true source, so all REs are classified as SOURCE events. HRC-18, HRC-19, and HRC-20 are added to the candidate set and the likelihood rankings of all HRCs decrease. Because one of the links in the causal loop has a NOT condition that blocks HRC-18, HRC-18 is ranked significantly lower than the other candidates.

TABLE 4-16: Inference Example 4

Stage	Events Detected	Source Events	Hypothesized Root Causes
1	H	H	12 (ranking = 0.500) 13 (ranking = 0.500)
2	H,J	H	12 (ranking = 0.500) 13 (ranking = 0.500)
3	H,J,I	H,J,I	12 (ranking = 0.248) 13 (ranking = 0.248) 18 (ranking = 0.013) 19 (ranking = 0.248) 20 (ranking = 0.248)

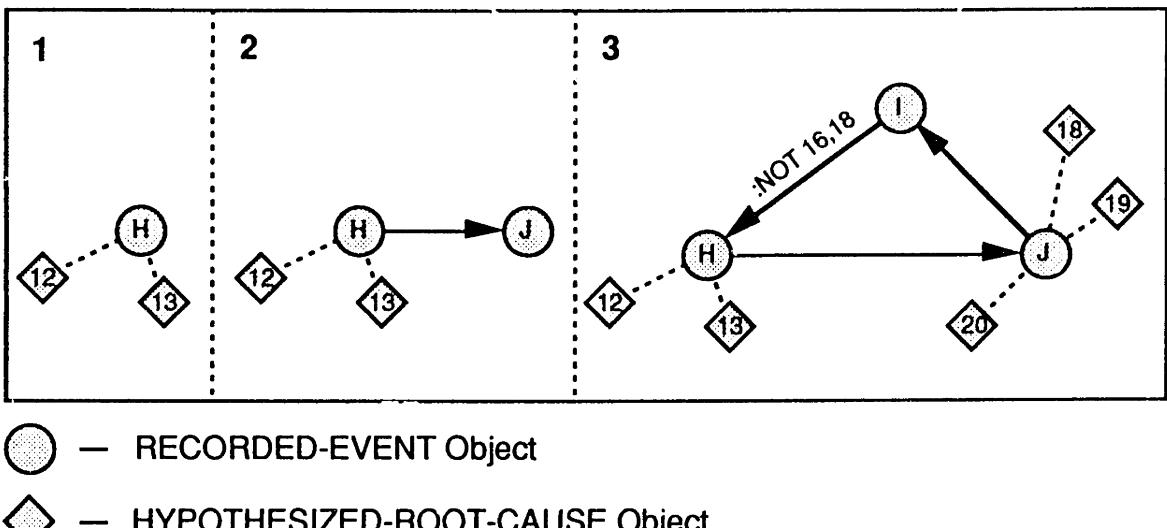


Figure 4-11: Evolving Hypothesis Model for Inference Example 4

4.4. Failure Scenarios Interpreted by MIDAS

This section describes in more detail the types of scenarios correctly interpreted by MIDAS. In the inference examples of the preceding section, several specific scenarios were presented. This section builds on that foundation, seeking to expand those examples into broad classes of diagnosis problems treatable by MIDAS.

MIDAS employs a number of techniques to solve different diagnosis problems. Some faults can be detected immediately by the monitors. Other faults are addressed through features of the inference algorithm or the process model. In all cases, MIDAS remains a general diagnostic system. If MIDAS can diagnose a specific failure scenario, it can diagnose the entire class of scenarios from which the example was drawn.

4.4.1. Sensor Failures

MIDAS is designed to recognize two types of sensor failure: *fixed failure* and *bias*. A fixed failure is failure of a sensor to a given value regardless of the true value of the measured variable. This failure mode may be identified by a sudden jump in the sensor measurement or

by a change in the noise characteristics of the sensor. However, fixed failure of a sensor need not be sudden and it is not always advisable to diagnose sensor failures based on changes in noise characteristics¹³. A sensor bias or miscalibration results in a sensor reporting a value slightly above or below the true value. In real plants, all sensors contain small biases that are considered normal. Here, a bias must be significantly greater than normal to be defined as a failure.

Sensor failures can also be classified as: *in-range*, *out-of-range*, or *induced*. An in-range sensor failure produces sensor measurements within the range of believable values. An out-of-range sensor failure produces sensor values outside the range of believable values. Out-of range failure can be detected immediately using monitor range checks. In-range failures may not be detectable at the monitor level and must be included in the process model. An induced sensor failure is a gross failure (i.e. a fixed failure) caused by another malfunction. A thermocouple melting in a fire is an example of induced sensor failure. Induced sensor failure is a special form of multiple malfunctions, discussed in section 4.4.4.

4.4.2. Dynamics

The problem of complex dynamics is treated through the process model. Since the event model paradigm which forms the basis of the process model is dynamic and includes provisions for modeling compensatory and inverse response, transient malfunctions and false alarms are easily interpreted. The interpretation of a malfunction's dynamics is stored in the STATUS slot of the appropriate INFERRED-MALFUNCTION object. Six (6) interpretations are possible:

Persistent

The malfunction does not appear to be a transient or false alarm and will require corrective action.

¹³ Sensors may display different noise characteristics when measured variables are in different portions of the sensor's range. Additionally, certain malfunctions may change the variability of the process, and therefore, the apparent noise statistics of the sensor. The monitors in MIDAS will perform optional noise checks only when sensor measurements are near their normal values.

On-Going Transient

All abnormal events¹⁴ classified as SOURCE events of the malfunction have been superseded by "return to normal" events¹⁵, but one or more abnormal CONSEQUENCE events remain active. This is indicative of a transient disturbance progressing through the process. All process variables should return to normal conditions without intervention after the disturbance has passed.

Completed Transient

All events have been superseded by return to normal events. The transient has passed and the process has returned to its normal operating regime.

Oscillatory

The SOURCE events of the malfunction appear, are superseded by return to normal events, and appear again. This situation can arise if a failure disturbs a process variable to a value just above or below the detection threshold where random variations produce an unstable detection. This situation can also result from certain process failures. For example, a badly tuned controller that alternately fully opens and then fully closes a valve may produce oscillatory behaviors. These behaviors must be modeled in the process model for MIDAS to be successful in interpretation.

Spurious

If an abnormal event appears and is superseded by a return to normal event without any CONSEQUENCE events, the malfunction is classified as spurious. This situation is usually caused by a false event detection.

Corrected

Whenever all abnormal events associated with a malfunction have been superseded, the malfunction can be tagged as having been "corrected". This is usually done just prior to archiving the malfunction and removing it from the active hypothesis model (see section 5.3.2.4).

¹⁴ An "abnormal" event is any event that leaves the process in an abnormal operating condition.

¹⁵ A "return to normal" event is any event that leaves the process in a normal operating condition.

Because MIDAS can interpret return to normal events as easily as any other event, there is no need for snapshots or resets. MIDAS will run continuously and maintain an infinite memory, if desired. As a matter of practicality, however, it is usually desirable to limit memory to a single malfunction episode by removing malfunctions (i.e. flagging them as corrected) after all events have returned to normal and sufficient time has passed to indicate a recurrence of symptoms is not imminent.

4.4.3. Variations in Event Detection Order

As was illustrated in inference example 3 in section 4.3.6, events need not be detected in the same order that the disturbance propagates through the system. Finch and Kramer (1989) present proof that in the general case, detection schemes cannot guarantee in-order detection without sacrificing sensitivity. Out-of-order detection is also possible when sensors have different sampling rates. For example, a causally downstream temperature probe with a 1 second sampling delay will likely detect a disturbance before an upstream concentration analyzer with a 15 minute delay.

To deal with variations in event detection order, MIDAS uses two strategies: *anticipation* and *correction*. First, MIDAS tries to anticipate the correct (i.e. causal) event order using monitor interrogation to identify sub-threshold events. EXPECTED-EVENTS and LATENT-EVENTS are created to fill the gaps of a propagation pathway or capture the undetected sources of an IM cluster. The weakness of anticipation is that EEs and LEs cannot be retracted if the indicated events are not detected within the specified time period. This weakness could be corrected if MIDAS were implemented with a real-time operating system.

If interrogation fails to identify a sub-threshold event (as in the case of event RE-A in inference example 2), correction is required. Correction refers to the entire set of actions MIDAS takes to reconfigure the diagnosis when two IM clusters coalesce or a previously undetected SOURCE event is discovered. Correction ensures that for a given set of events, the same diagnosis will be created regardless of the order of event detection. The weakness of this scheme is the degradation of the diagnosis when causal loops are encountered. Intuitively, it is reasonable to assume that the first event detected in a causal loop has a higher probability of being the SOURCE event than later events. This is especially true when the causal loop contains several events or takes a long time to close. Due to the lack of a real-time operating system and model information on propagation gains and delays, however, MIDAS

is unable to incorporate this intuition into its diagnostic reasoning. A possible extension of the current modeling paradigm to address this situation is presented in section 7.5.2.

4.4.4. Multiple Malfunctions

Whenever MIDAS cannot link an event to any existing IM cluster, it creates a new IM cluster, in essence, diagnosing multiple malfunctions. There is no limit to the number of IM clusters active at any time, thus in theory, MIDAS can diagnose any number of multiple, simultaneous malfunctions, assuming their events cannot be linked through the process model. On the other hand, if two or more malfunctions produce symptoms that can be linked through causal paths in the process model, MIDAS will interpret the symptoms as the result of a single malfunction. In this case, zero, one, or several of the underlying malfunctions may be present in the root cause candidate set. In summary, MIDAS will diagnose multiple malfunctions separated in time or space but will fail to diagnose multiple malfunctions whose symptom overlap.

This failure to diagnose interacting malfunctions should not be considered a flaw in MIDAS. Part of the philosophy built into the diagnostic inference algorithm is to explain observed events with the fewest number of postulated malfunctions. This limits MIDAS to the set of possible OR worlds and excludes all AND worlds within a single IM cluster (see section 7.5.1 for further explanation of OR and AND worlds). In most instances, this philosophy is correct. The probability of multiple, closely related, simultaneous, independent malfunctions is usually extremely low. The exception is dependent (i.e. induced) failures.

Induced failures are not explicitly modeled in the event graph paradigm, although a possible extension of the paradigm to include induced failures is discussed in section 7.4.3. At the present time, only induced sensor failure is reliably diagnosed by MIDAS. Because out-of-range sensor failure is modeled as an event with no initiating or terminating CLs, whenever an out-of-range failure is detected it is always interpreted as a separate malfunction.

5.0. MIDAS User Guide

This section provides users of the MIDAS software with instructions relating to the features and operation of the program. Because MIDAS was conceived as a research tool, a high degree of flexibility has been built into the program to allow experimentation with various aspects of the basic algorithm. Default settings, applicable for most problems, are provided. This section focuses on the details of setting up and running MIDAS, not the underlying concepts which were discussed in sections 3.0 and 4.0.

5.1. General Information

MIDAS is a computer-based diagnostic reasoning system designed to identify malfunctions in continuous, nominal steady-state chemical and refinery processes using a plant-independent reasoning strategy based on qualitative and quantitative process models. MIDAS specifically addresses problems not treated in past systems, including: process dynamics and control system responses, multiple faults, induced failures, out-of-order alarms, and false alarms.

The MIDAS software bundle contains both the programs necessary to perform diagnosis and the programs used to develop process models. MIDAS is partitioned into three major components: Diagnostics, Model Builder, and Model Translator. Each component is an independent program, and simultaneous use of multiple components is not recommended unless specifically indicated.

The overall structure of MIDAS is illustrated in figure 5-1. The Diagnostics Component contains the Diagnostics, the User Interface, and Data Interface subsections. The Model Builder contains interactive programs to guide users through the construction of a process model by linking together individual unit models contained in the Model Library. The Model Translator consists of an automatic procedure to convert process models developed using the Model Builder into the representation used for diagnosis. The Model Builder, Model Translator, Linker Interface, and Model Library were developed by Oyeleye (1989) and are not discussed here.

The minimum requirements necessary for successful use of MIDAS are the following:

Platforms:

- IBM PC/AT, PS/2 Model 70, PS/2 Model 80,
- Compaq 286 or 386,
- PC/AT or 386 Compatibles.

Hardware:

- 10+ MB RAM (16 MB recommended),
- 30 MB Storage Capacity,
- EGA or VGA Monitor,
- 1 HD Floppy Drive.

Software:

- MS-DOSTM¹ (version 3.00 or higher),
- GoldWorksTM²,
- DBASE IIITM³ (optional).

5.2. Installation

To install MIDAS, it is first necessary to load the GoldWorks software using the instructions provided. A batch program -- MINSTAL.BAT -- contained on MIDAS Disk #1 is provided to install MIDAS files. To use this program,

- 1) Set the current drive to the drive on which GoldWorks is installed,
- 2) Place MIDAS Disk #1 in drive A:,
- 3) Enter A:MINSTAL.BAT and follow the on-screen instructions.

This program automatically creates directories (subdirectories of the GoldWorks directory tree) and transfers files. Before using this program, check to insure that no directories exist named: MIDAS, DIAGNOST, BUILDER, or TRANSLAT.

¹ Microsoft Corporation, Seattle, WA.

² GoldWorks is an expert system development environment from Gold Hill Computers, Cambridge, MA.

³ Ashton-Tate, Torrance, CA.

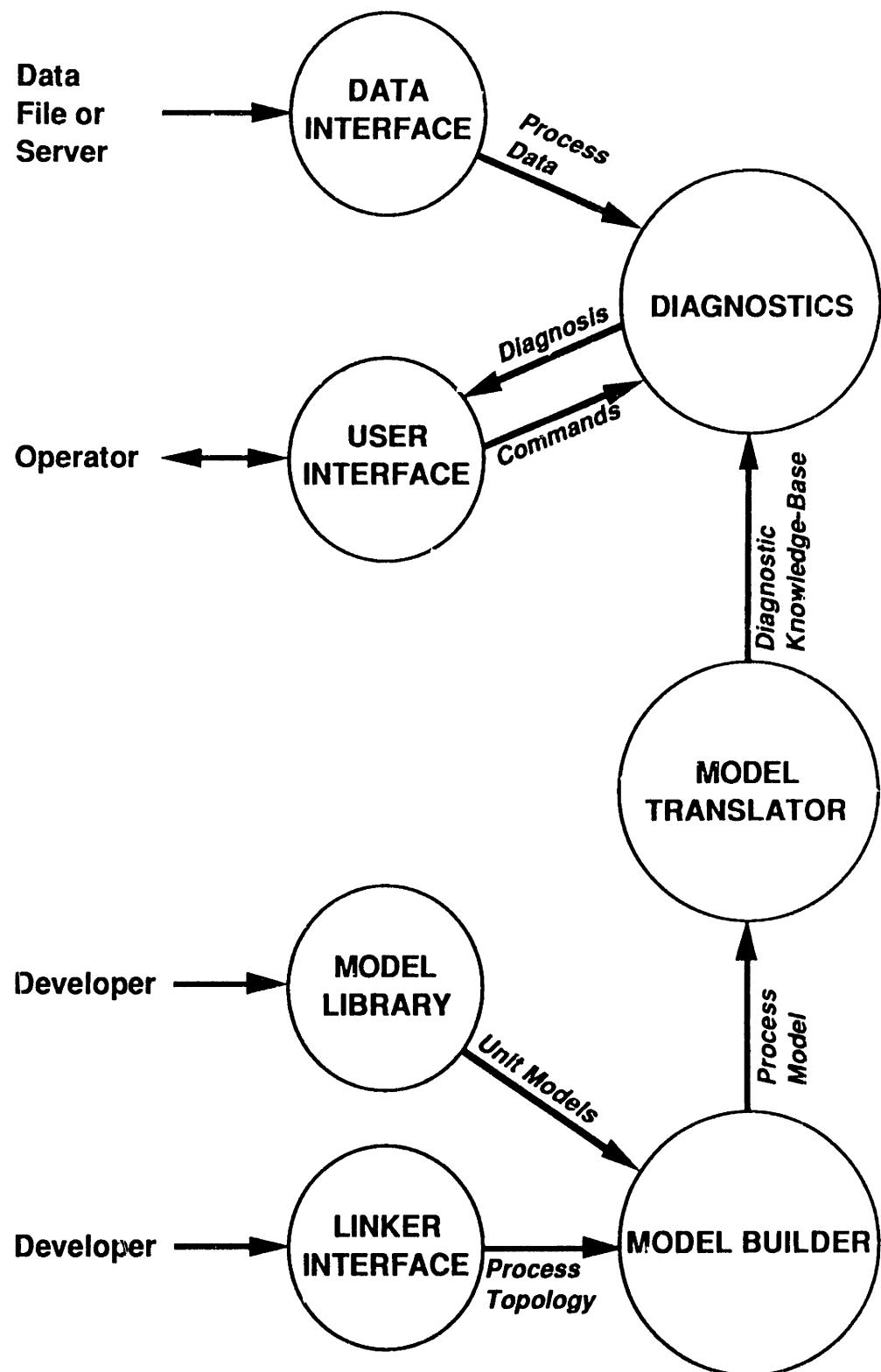


Figure 5-1: MIDAS Structure

All files conform to the MIDAS naming convention. The first five characters constitute the filename, the sixth and seventh characters indicate the version number (e.g. "10" for version 1.0), and the eighth character indicates the MIDAS component (e.g. "D" for Diagnostics, "B" for Builder, "T" for Translator, or "L" for Library). The three character suffix indicates the type of code (i.e. "LSP" for source code and "FAS" for compiled code). Within each file, programs are arranged alphabetically by name.

5.3. Diagnostics

The diagnostic component of MIDAS consists of six (6) files: five (5) files containing the diagnostic code and one (1) file containing utility programs to assist the user in loading, running, and compiling diagnostic code. File names and descriptions are shown in Table 5-1.

TABLE 5-1: Diagnostic Component Files

Name	Description
OBDEF10D	Frame and user interface definitions
DEMON10D	Demon, handler, and message passing routines
POPUP10D	Routines for creating popup windows
MONIT10D	Data acquisition (monitor) routines
DIFUN10D	Diagnostic reasoning routines
MIDAS10D	Interactive diagnostic component utilities

These files contain the two hundred and four individual subprograms that comprise the Diagnostics Component of MIDAS. A listing of all subprograms, including brief descriptions, is provided in Table 5-2. More detailed documentation is provided in the source code.

TABLE 5-2: MIDAS Diagnostic Programs

#	Name	File	Description
1	ACTIVATE-CHOSEN-EVENT	DEMON10D	Activates a PE instance entered from the user interface
2	ACTIVATE-CONFIRM-INTERROGATION-POPUP	POPUP10D	Activates a CONFIRM-INTERROGATION popup window
3	ACTIVATE-EMPTY-HYPOTHESIS-POPUP	POPUP10D	Activates an EMPTY-HYPOTHESIS popup window
4	ACTIVATE-EVENT-DETECTION-POPUP	POPUP10D	Activates an EVENT-DETECTION popup window
5	ACTIVATE-MALFUNCTION-ARCHIVE-ERROR-POPUP	POPUP10D	Activates an UNCORRECTED-MALFUNCTION popup window
6	ACTIVATE-MONITOR-CHANGE	DEMON10D	Activates a CHANGE-MONITOR popup window
7	ACTIVATE-MONITOR-ERROR-POPUP	POPUP10D	Activates a MONITOR-ERROR popup window
8	ACTIVATE-POPUP	DEMON10D	Activates a popup window instance
9	ACTIVATE-POTENTIAL-EVENT	DEMON10D	Activates a PE instance
10	ACTIVATE-TEST	DEMON10D	Activates a TEST instance
11	ADD-LOCAL-CAUSES-TO-HYPOTHESIS	DIFUN10D	Creates HRC instances for all LOCAL-CAUSES of an event
12	ADD-ZEROS	MONIT10D	Appends zeros to a list
13	ANALYZE-DATA	MONIT10D	Performs statistical tests on DATA-MONITOR data
14	ANALYZE-SYSTEM-MONITOR	MONIT10D	Performs qualitative tests on a SYSTEM-MONITOR
15	APPEND-BLANK-SPACE	MONIT10D	Adds blank spaces to a string
16	APPLY-SPECIAL-CONDITIONS	DIFUN10D	Applies diagnostic conditions in the SPECIAL-CONDITIONS slot of an event instance
17	ARCHIVE-CORRECTED-MALFUNCTION	DEMON10D	Archives all instances associated with an IM instance
18	ARCHIVE-DATA-TO-FILE	DIFUN10D	General data archive routine
19	ARCHIVE-EVENT	DEMON10D	Archives a RE instance
20	ARCHIVE-HYPOTHESIS	DEMON10D	Archives a HRC instance
21	ARCHIVE-MALFUNCTION	DEMON10D	Archives an IM instance
22	BUILD-CAUSAL-NETWORK	DIFUN10D	Builds a causal linkage between two events
23	CHANGE-DATA-MONITOR-POPUP	POPUP10D	Creates a popup widow for modification of a DATA-MONITOR

TABLE 5-2: MIDAS Diagnostic Programs (continued)

#	Name	File	Description
24	CHANGE-SYSTEM-MONITOR-POPUP	POPUP10D	Creates a popup widow for modification of a SYSTEM-MONITOR
25	CHECK-ABNORMAL	DIFUN10D	Determines if an event is NORMAL or ABNORMAL
26	CHECK-EVENT	DIFUN10D	Determines if an event with a given description already exists
27	CHECK-EVIDENCE-CONSISTENCY	DIFUN10D	Verifies that evidence in HRC slots is consistent
28	CHECK-EXCLUSIVE-EVENT	DIFUN10D	Determines if any exclusive events of a given event are currently active
29	CHECK-EXPECTED-EVENT	DIFUN10D	Determines if an EE with a given description already exists
30	CHECK-FOR-TRANSIENT	DIFUN10D	Evaluates the STATUS of a given IM
31	CHECK-HYPOTHESIZED-CAUSE	DIFUN10D	Determines if a HRC exists for a given PRC
32	CHECK-LATENT-EVENT	DIFUN10D	Determines if a LE with a given description already exists
33	CHECK-LEARNING-MODE	MONIT10D	Checks if MONITORS are set to learn and initiates learning procedure if indicated
34	CHECK-LOGIN-FILE	DEMON10D	Verifies user name and password
35	CHECK-PATH	DIFUN10D	Checks a list for an occurrence of a given element
36	CHECK-RECORDED-EVENT	DIFUN10D	Determines if a RE with a given description already exists
37	CHECK-VECTOR-LENGTH	DIFUN10D	Prunes a data queue to a given length
38	CLONE-EVENT-BOOKKEEPING	DIFUN10D	Performs bookkeeping associated with cloning an event
39	COALESE-MALFUNCTIONS	DIFUN10D	Merges two or more IMs
40	COMPENSATE-FOR-DRIFT	MONIT10D	Recomputes limits for drifting MONITORS
41	COMPUTE-NORMALIZED-LIKELIHOOD	DIFUN10D	Computes the NRL for all HRCs associated with an IM
42	COMPUTE-PREDICTION-VECTOR	MONIT10D	Extrapolates MONITOR data to make a forecast of future state and trend
43	COMPUTE-PROBABILISTIC-LIKELIHOOD	DIFUN10D	Computes the PL and NPL for all HRCs associated with an IM
44	COMPUTE-RELATIVE-LIKELIHOOD	DIFUN10D	Computes the RL for all HRCs associated with an IM
45	CONDITION-PROBABILISTIC-LIKELIHOOD	DIFUN10D	Computes the CP and NCP for all HRCs associated with an IM

TABLE 5-2: MIDAS Diagnostic Programs (continued)

#	Name	File	Description
46	CONVERT-DBASE-DATE	MONIT10D	Transforms a date stamp from DBASE to MIDAS format
47	CONVERT-DBASE-TIME	MONIT10D	Transforms a time stamp from DBASE to MIDAS format
48	CONVERT-EXPECTED-EVENT-TO-LATENT-EVENT	DIFUN10D	Converts an EE instance to a LE instance
49	CONVERT-LATENT-EVENT-TO-EXPECTED-EVENT	DIFUN10D	Converts a LE instance to an EE instance
50	CONVERT-MATRIX-FORM	MONIT10D	Transposes a matrix stored in MIDAS format
51	CONVERT-NUMBER-TO-STRING	MONIT10D	Converts a number (FIXNUM or FLOAT) to a string representation of the same number
52	CREATE-CAUSAL-LINKS	DIFUN10D	Creates EE and LE instances to complete a propagation pathway
53	CREATE-CONSTRAINT-MONITOR-SUMMARY	POPUP10D	Creates the CONSTRAINT-SUMMARY popup window
54	CREATE-DATA-MONITOR-POPUP	POPUP10D	Creates a popup window to display the contents of a DATA-MONITOR instance
55	CREATE-DIAGNOSTIC-SUMMARY	POPUP10D	Creates the DIAGNOSTIC-SUMMARY popup window
56	CREATE-EXPECTED-OR-LATENT-EVENT-POPUP	POPUP10D	Creates a popup window to display the contents of an EE or LE instance
57	CREATE-HYPOTHESIZED-CAUSE-POPUP	POPUP10D	Creates a popup window to display the contents of a HRC instance
58	CREATE-INFERRRED-MALFUNCTION-POPUP	POPUP10D	Creates a popup window to display the contents of an IM instance
59	CREATE-LATENT-EVENT'S	DIFUN10D	Creates LE instances and builds propagation paths
60	CREATE-LINK-POPUP	POPUP10D	Creates a popup window to display the contents of a COMPILED-LINK instance
61	CREATE-MEASURED-OBJECT-POPUP	POPUP10D	Creates a popup window to display the contents of a MEASURED-OBJECT instance
62	CREATE-POTENTIAL-CAUSE-POPUP	POPUP10D	Creates a popup window to display the contents of a PRC instance
63	CREATE-RECORDED-EVENT-POPUP	POPUP10D	Creates a popup window to display the contents of a RE instance

TABLE 5-2: MIDAS Diagnostic Programs (continued)

#	Name	File	Description
64	CREATE-SENSOR-MONITOR-SUMMARY	POPUP10D	Creates the SENSOR-SUMMARY popup window
65	CREATE-SYSTEM-MONITOR-POPUP	POPUP10D	Creates a popup window to display the contents of a SYSTEM-MONITOR instance
66	CREATE-SYSTEM-MONITOR-SUMMARY	POPUP10D	Creates the SYSTEM-SUMMARY popup window
67	CREATE-TEST-POPUP	POPUP10D	Creates a popup window to display the contents of a TEST instance
68	DELETE-ALL-POPUPS	DIFUN10D	Deletes all previously created popup windows
69	DECIPHER-MIDAS-TIME	MONIT10D	Breaks a time stamp in MIDAS format into its components
70	DETERMINE-EVENT-CLASSIFICATION	DIFUN10D	Identifies all SOURCE events associated with an IM
71	DIAGNOSTIC-ACTION-1A	DIFUN10D	Performs diagnostic analysis of an event that is the realization of an EE
72	DIAGNOSTIC-ACTION-1B	DIFUN10D	Performs diagnostic analysis of an event that is the realization of an LE
73	DIAGNOSTIC-ACTION-1C	DIFUN10D	Performs diagnostic analysis of an event that SUPERSEDES another event
74	DIAGNOSTIC-ACTION-2	DIFUN10D	Performs diagnostic analysis of an event that cannot be linked to an existing IM
75	DIAGNOSTIC-ACTION-3	DIFUN10D	Performs diagnostic analysis of an event that is the CONSEQUENCE of an existing IM
76	DIAGNOSTIC-ACTION-4	DIFUN10D	Performs diagnostic analysis of an event that is the SOURCE of an existing IM
77	DIAGNOSTIC-ACTION-5	DIFUN10D	Performs diagnostic analysis of an event that is the CONSEQUENCE of two or more existing IMs
78	DIAGNOSTIC-ACTION-6	DIFUN10D	Performs diagnostic analysis of an event that is the SOURCE of two or more existing IMs
79	DIAGNOSTIC-ACTION-7	DIFUN10D	Performs diagnostic analysis of an event that is the SOURCE of one IM and the CONSEQUENCE of another IM

TABLE 5-2: MIDAS Diagnostic Programs (continued)

#	Name	File	Description
80	DIAGNOSTIC-ACTION-8	DIFUN10D	Performs diagnostic analysis of an event that is the SOURCE of one IM and the CONSEQUENCE of two or more other IMs
81	DIAGNOSTIC-ACTION-9	DIFUN10D	Performs diagnostic analysis of an event that is the SOURCE of two or more IMs and the CONSEQUENCE of another IM
82	DIAGNOSTIC-ACTION-10	DIFUN10D	Performs diagnostic analysis of an event that is the SOURCE of two or more IMs and the CONSEQUENCE of two or more other IMs
83	DIAGNOSTIC-INFERENCE-CONTROL	DIFUN10D	Performs searches and governs execution of all diagnostic actions
84	DRIFT-FILTER	DEMON10D	Computes a drifting mean for a MONITOR instance
85	EVALUATE-ALL-EVIDENCE	DIFUN10D	Evaluates all evidence associated with an IM
86	EVALUATE-EVIDENCE	DIFUN10D	Evaluates evidence contained in the network structure of an IM
87	EVENT-REALIZATION-BOOKKEEPING	DIFUN10D	Performs necessary bookkeeping when a new event is the realization of an EE or LE
88	EXCLUSIVE-EVENT-BOOKKEEPING	DIFUN10D	Performs necessary bookkeeping when a new event is mutually exclusive of another existing event
90	EXPECTED-EVENT-NAME	DIFUN10D	Creates a new sequential name for an EE
91	EXPECTED-EVENT-REALIZATION	DEMON10D	Puts an EE to sleep when it is realized
92	EXPLAIN-EVENT-OCCURRENCE	DEMON10D	Initiates diagnosis when a new event is detected
93	FILL-EVENTS-EXPLAINED	DIFUN10D	Fills the EVENTS-EXPLAINED and the EVENT-NOT-EXPLAINED slots of all events associated with an IM
94	FIND-AND-MAKE-EXPECTED-EVENT	DIFUN10D	Creates an EE based on an event description
95	FIND-AND-MAKE-LATENT-EVENT	DIFUN10D	Creates a LE based on an event description
97	FIND-AND-MAKE-RECORDED-EVENT	DIFUN10D	Creates a RE based on an event description
98	FIND-APPLICABLE-TESTS	DIFUN10D	Finds instances of TEST that are associated with high ranking hypotheses

TABLE 5-2: MIDAS Diagnostic Programs (continued)

#	Name	File	Description
99	FIND-CAUSAL-CHAIN	DIFUN10D	Locates propagation paths through an IM event network
100	FIND-EVENTS-EXPLAINED	DIFUN10D	Finds all events associated with an IM that can be explained by a given event
101	FIND-EVENTS-IN-CAUSAL-CHAIN	DIFUN10D	Lists events in a causal propagation path
102	FIND-MONITOR-EVENTS	DEMON10D	Finds an appropriate PE for a sensor failure
103	FIND-STATE-EVENTS	DEMON10D	Finds an appropriate PE for a state change
104	FIND-TREND-EVENTS	DEMON10D	Finds an appropriate PE for a trend detection
105	FIRST-ORDER-FILTER	DEMON10D	Computes an EWMA for incoming data
106	GENSEA	DIFUN10D	General recursive search used to link new events to existing IMs
107	GENSUM	MONIT10D	Computes the sum of a list of numbers
108	GET-ACTIVE-OBJECT-LIST	DIFUN10D	Lists all active instances of specified type
109	GET-LINK-CONDITIONS	DIFUN10D	Lists all conditions associated with a link
110	GET-MATRIX-VALUE	MONIT10D	Retrieves a specified matrix value
111	GET-OBJECT-LIST	DIFUN10D	Lists all instances of specified type
112	HYPOTHESIZED-ROOT-CAUSE-NAME	DIFUN10D	Creates a sequential name for a new HRC
113	INCREMENT-DESIG	DIFUN10D	Increments a sequential designation
114	INFERRED-MALFUNCTION-NAME	DIFUN10D	Creates a sequential name for a new IM
115	INFORM-SYSTEM-MONITOR	DEMON10D	Updates a SYSTEM-MONITOR when an event is detected
116	INITIALIZE-MIDAS	MIDAS10D	Resets MIDAS to its original status
117	INITIALIZE-MONITORS	MIDAS10D	Resets all MIDAS Monitors
118	INTERROGATE-MONITOR	MONIT10D	Interrogates a DATA-MONITOR for evidence of an EE or LE
119	LATENT-EVENT-NAME	DIFUN10D	Creates a sequential name for a new LE
120	LATENT-EVENT-REALIZATION	DEMON10D	Puts to sleep a LE when it is realized
121	LEARN-MONITOR-SETTINGS	MONIT10D	Sets MONITOR limits based on accumulated data
122	LINK-EVENT-TO-MALFUNCTION	DIFUN10D	Performs bookkeeping to attach a new event to an IM

TABLE 5-2: MIDAS Diagnostic Programs (continued)

#	Name	File	Description
123	LIMIT-TEST	MONIT10D	Computes the number of values in a list above, below, and equal to a given limit
124	LMAX	MONIT10D	Computes the maximum value from a list
125	LMIN	MONIT10D	Computes the minimum value from a list
126	LOAD-MIDAS-FILES	MIDAS10D	MIDAS Diagnostics load utility
127	LOAD-MIDAS-PDM	MIDAS10D	MIDAS Diagnostics load utility
128	LOCATE-LOCAL-CAUSES	DIFUN10D	Finds all LOCAL-CAUSES of an event
129	LOCATE-PRECURSORS	DIFUN10D	Finds all PRECURSORS of an event
130	LOCATE-SUCCESSORS	DIFUN10D	Finds all SUCCESSORS of an event
131	LU-DECOMP-2	MONIT10D	Computes the LU decomposition of a square, symmetric matrix
132	MAKE-DATABASE-INSTANCE	DIFUN10D	Makes an instance of specified type and initializes slot values
133	MAKE-EXPECTED-EVENT	DIFUN10D	Creates an EE instance
134	MAKE-HYPOTHESIZED-ROOT-CAUSE	DIFUN10D	Creates a HRC instance
135	MAKE-INFERRRED-MALFUNCTION	DIFUN10D	Creates an IM instance
136	MAKE-LATENT-EVENT	DIFUN10D	Creates a LE instance
137	MAKE-MATRIX	MONIT10D	Makes a matrix of specified size
138	MAKE-NAME	DIFUN10D	Makes a MIDAS instance name
139	MAKE-POPUP-FORM	POPUP10D	Creates a popup window of type POPUP-FORM
140	MAKE-PREDICTION	MONIT10D	Makes a prediction of based on extrapolation of data
141	MAKE RECORDED-EVENT	DIFUN10D	Creates a RE instance
142	MATCH-EVENT-DESCRIPTION	DIFUN10D	Finds events that match a given description
143	MATINV	MONIT10D	Computes the inverse of a matrix decomposed into upper and lower triangular components
144	MATMULT	MONIT10D	Computes the dot product of two matrices
145	MEAN	MONIT10D	Computes the mean of a list of numbers
146	MEMBERP	DIFUN10D	Determines if a specified element is in a list

TABLE 5-2: MIDAS Diagnostic Programs (continued)

#	Name	File	Description
147	MIDAS-AUTOPilot	MIDAS10D	Controls automatic diagnosis of a series of events or data files
148	MIDAS-COMPILER	MIDAS10D	Controls automatic compilation of diagnostic files
149	MIDAS-ROOM	DIFUN10D	Displays free space remaining in memory
150	MIDAS-TIME	DIFUN10D	Returns clock time as a string
151	MIDAS-TIMER	DIFUN10D	Times the evaluation of a MIDAS function
152	MIDAS-UTIL-CONTROL	MIDAS10D	Controls execution of diagnostic utilities
153	MIDAS-WAIT	MIDAS10D	Puts MIDAS to sleep for a specified time period
154	NTHCAR	DIFUN10D	Returns the nth CAR of a nested list
155	RANGE	MONIT10D	Computes the range of a list of numbers
156	READ-DBASE-DATA	MONIT10D	Controls entry of data from a DBASE file
157	RECORDED-EVENT-NAME	DIFUN10D	Creates a sequential name for a new RE
158	RECORD-EVENT-OCCURRENCE	DEMON10D	Creates a new RE when an event is detected
159	REDUCE-LINKS	DIFUN10D	Produces a minimum set of links from the list provided by GENSEA
160	REMOVE-EVENT	DEMON10D	Archives and deletes an EE or LE instance
161	REMOVE-GMACS-EDITOR	MIDAS10D	Removes the GMACS editor from the GoldWorks environment
162	REMOVE-LOCAL-CAUSES-FROM-HYPOTHESIS	DIFUN10D	Removes the LOCAL-CAUSES of a specified event from the HRCs associated with an IM
163	REMOVE-MALFUNCTION	DIFUN10D	Archives and deletes all instances associated with a specified IM
164	REMOVE-PROCESS-MODEL	MIDAS10D	Removes a process model from memory
165	REMOVE-REPEATED-VALUES	DIFUN10D	Removes multiple occurrences of an element from a list
166	REMOVE-ROOT-CAUSE	DEMON10D	Deletes a HRC instance
167	RESET-MONITOR-THRESHOLDS	MIDAS10D	Allows one or more thresholds to be reset for all DATA-MONITOR instances
168	RUN-TEST	MONIT10D	Determines the longest run of numbers in a list

TABLE 5-2: MIDAS Diagnostic Programs (continued)

#	Name	File	Description
169	SAVE-SCENARIO-RESULTS	MIDAS10D	Saves to disk file all instances created during a diagnostic session
170	SAVE-MONITORS	MIDAS10D	Saves all MONITOR instances to a specified disk file
171	SDEV	MONIT10D	Computes the standard deviation of a list of numbers
172	SELF-TEST	MONIT10D	Tests for repeated values in a list
173	SET-EVALUATION-LEVEL	DIFUN10D	Determines the level of evidence evaluation for an IM
174	SET-GOLDWORKS-ENVIRON	MIDAS10D	Modifies User Interface display parameters
175	SET-LINK-FLAG	DIFUN10D	Checks for a valid link in a GENSEA search
176	SET-MATRIX-VALUE	MONIT10D	Inserts a specified value into a matrix
177	SET-RECURSION-FLAG	DIFUN10D	Tests recursion criteria in a GENSEA search
178	SET-SYSTEM-ANALYSIS-FLAG	DEMON10D	Adds a SYSTEM-MONITOR to the list of monitors requiring analysis
179	SORT-AND-DISPLAY-CAUSES	DIFUN10D	Lists in ranked order all HRCs associated with an IM
180	SORT-BY-FIRST-ELEMENT	DIFUN10D	Sorts a list of lists by the first element in each sublist
181	SORT-BY-LENGTH	DIFUN10D	Sorts a list of lists by the length of each sublist
182	SORT-BY-LIKELIHOOD	DIFUN10D	Ranks a list of HRCs by likelihood
183	SORT-CAUSAL-LINKS	DIFUN10D	General routine for sorting link lists produced by GENSEA search
184	SUBSET	DIFUN10D	Determines if one set is a subset of another
185	SUM	MONIT10D	Computes the sum of a list of numbers
186	TEST-CONDITION	DIFUN10D	Tests for the existence of specified events
187	TEST-INSTANCE	DIFUN10D	Tests if an instance with a specified name exists
188	TEST-LIST-OF-EVENTS	DIFUN10D	Tests if a specified event is one of a list of events
189	TRAIN-MONITORS	MIDAS10D	A utility that reads data files and sets DATA-MONITOR thresholds
190	TRANSFORM-DIAGNOSTIC-CONDITION	DIFUN10D	Translates a link condition into a set of fundamental NOT and ADD conditions

TABLE 5-2: MIDAS Diagnostic Programs (continued)

#	Name	File	Description
191	UNLINK-EVENT-FROM-MALFUNCTION	DIFUN10D	Performs bookkeeping to disassociate an event from an IM
192	UPDATE-ALL-DISPLAYS	DEMON10D	Updates the DIAGNOSTIC-SUMMARY when a change is made to likelihood type or threshold
193	UPDATE-DIAGNOSTIC-MODEL	DIFUN10D	Activates and deactivates links as indicated by existing event conditions
194	UPDATE-FILTERED-MEAS-VECTOR	DEMON10D	Updates the FILTERED-MEAS-VECTOR slot of a DATA-MONITOR
195	UPDATE-GLOBAL-CONDITIONS	DEMON10D	Updates the Event and Diagnosis Status windows
196	UPDATE-LIKELIHOOD	DEMON10D	Updates the likelihood displayed in the DIAGNOSTIC-SUMMARY
197	UPDATE-MEAS-TIME-VECTOR	DEMON10D	Updates the MEAS-TIME-VECTOR slot of a DATA-MONITOR
198	UPDATE-MEASURED-OBJECT	DEMON10D	Updates the slots of an instance of MEASURED-OBJECT when a new event is detected
199	UPDATE-OBJECT	DEMON10D	Updates a MEASURED-OBJECT instance
200	UPDATE-POSTULATION	DEMON10D	Updates the POSTULATION slots of a MEASURED-OBJECT instance
201	UPDATE-RAW-MEAS-VECTOR	DEMON10D	Updates the RAW-MEAS-VECTOR slot of a DATA-MONITOR
202	UPDATE-STATISTICS-VECTORS	DEMON10D	Updates the MEAN, SDEV, and RANGE vectors of a DATA-MONITOR
203	UPDATE-TEST-EVIDENCE	DEMON10D	Updates the diagnosis whenever TEST evidence is received
204	X-TYPE	MONIT10D	Converts a list of number from SINGLE-FLOAT to DOUBLE-FLOAT or vice versa

Loading and compiling are order sensitive operations and must be performed carefully. Interactive utility programs -- contained in file MIDAS10D (described in section 5.3.1.) -- are provided to load and compile the diagnostic code. Use of these utilities is recommended.

5.3.1. MIDAS10D

The MIDAS10D file contains utility programs to manipulate the Diagnostics Component of MIDAS. To start the utilities:

- 1) Load GoldWorks. If GoldWorks has been previously loaded to run the Model Builder or Model Translator, or the GoldWorks compiler has been loaded, it is recommended that the user exit and reload GoldWorks to clear memory,
- 2) Set the default directory to the directory containing the MIDAS10D file. This can be done from Top Level LISP by entering: (cd pathname)⁴,
- 3) From Top Level LISP, enter: (load "MIDAS10D").

The utility programs will be loaded into memory, as indicated by the "Loading Utilities" message. Then the utility control program will run automatically, displaying the MIDAS banner and presenting the user with a list possible utility actions.

There are a total of eleven (11) utility actions, but only actions that are consistent with the current state of the MIDAS environment will be displayed. For example, immediately after loading MIDAS10D, there are five possible actions:

- 1) Load Diagnostic Routines,
- 2) Remove Menu Interface,
- 3) Remove GMACS Editor,
- 4) Compile MIDAS Source Code, and
- 5) None of the Above.

To initiate an action, at the prompt, enter the number that corresponds to the desired action. The user is asked to confirm the selected action before the action is initiated. After an action has been completed, the user has the option of performing additional actions or starting the

⁴ Here, pathname is a GoldWorks pathname. Examples of the GoldWorks pathname convention are "C:\GW\KBS\" for directories and "C:\GW\KBS\FOO.LSP" for files. The default directory can also be changed from GMACS using the Cntl-X C command.

MIDAS User Interface (assuming the diagnostic routines and a process model have been loaded).

All utility actions are executed by the utility control program. The control program can be started any time after loading MIDAS10D by entering:

(midas-util-control)

from Top Level LISP. During a diagnostic session, the utilities can be accessed from the MIDAS User Interface using the *Access Utilities* command option (see **Input**).

Each action is a program or combination of programs that may ask the user for additional information. Descriptions of each utility action follow:

→ **Load Diagnostic Routines**

This action loads all routines associated with the Diagnostics Component of MIDAS and is available only if these routines have not been previously loaded. Typically, this action is the first step in preparing MIDAS for a diagnostic session.

First, the program will prompt the user for the pathname of the directory containing the Diagnostics Component files. Please have this information available prior to performing this action⁵. After a pathname has been entered, the program checks to verify that the necessary MIDAS files reside in the indicated directory. The user is allowed to re-enter the pathname if the directory was incorrectly specified. Once the pathname is verified, file loading begins immediately. The SCREEN TOOLKIT and DBASE interfaces provided with GoldWorks are loaded first. Then the diagnostics files are loaded, as indicated by the "Loading Diagnostics" message. When loading is complete, the diagnostic environment is automatically initialized.

→ **Load Process Model**

This action allows the user to load a process model into memory, and is only available if the diagnostic routines have been loaded. Loading a process model is typically the second step in preparing MIDAS for a diagnostic session. Process models are created

⁵ If MINSTAL.BAT is used to install MIDAS, the diagnostic files will reside in the \\GW\\KBS\\MIDAS\\DIAGNOST\\ directory on the GoldWorks device.

by the Model Builder and Model Translator Components of MIDAS. This utility assumes that a process model has been previously prepared and written to a disk file.

The user will be asked to enter the pathname of the file containing the process model. If the indicated file cannot be found, the user must re-enter the pathname. Otherwise, loading begins immediately, as indicated by the "Loading Model" message.

→ **Change Process Model**

This action allows the user to change the process model currently in memory, and is available only if the diagnostic routines have been loaded and a process model already resides in memory.

The user will be asked to enter the pathname of the new process model file. If the indicated file cannot be found, the user must re-enter the pathname. The process model currently in memory will be removed⁶ before file loading begins, as indicated by the "Loading Model" message.

→ **Remove Menu Interface**

This action allows the user to remove the GoldWorks Menu Interface, recovering approximately 500K RAM. The interface cannot be subsequently reloaded without exiting GoldWorks. This option will not appear if the menu interface has previously been removed.

→ **Remove GMACS Editor**

This action allows the user to remove the GMACS Editor, recovering approximately 420K RAM. GMACS cannot be subsequently reloaded without exiting GoldWorks. This option will not appear if GMACS has previously been removed.

→ **Modify GoldWorks Environment**

This action gives the user some control over the display during and after garbage collection. This option is always available.

The user is first asked if the "GC" symbol should be displayed during garbage collection. If answered NO, there will be no indication when garbage collection is

⁶ When a process model is removed, any diagnostic objects in memory are deleted.

occurring. If answered YES, the "GC" symbol will be displayed in the lower left corner of the screen whenever garbage collection is in progress.

The user will also be asked if memory utilization should be displayed after each garbage collection. If answered YES, a display will show how memory is partitioned and how much free memory remains after each garbage collection. This information will appear in the Text Output Window (see section 5.3.2.2.).

→ **Use MIDAS Autopilot**

The MIDAS Autopilot is a program designed to perform MIDAS diagnostic sessions automatically, saving intermediate and final results from one session before beginning another. This feature allows the user to run several diagnostic sessions unattended (useful for testing and running case studies). Unless the Archive Records to Disk File parameter (see *Set Diagnostics*) is set to YES, any temporary objects created during diagnosis may be lost.

The Autopilot can use either *data file* or *event sequence* input. The Autopilot automatically disables all MIDAS interrupts and menu functions, meaning the user cannot interact with MIDAS when the Autopilot is in use.

The Autopilot queries the user for details of the diagnostic sessions that are to be performed. This interactive segment of the program proceeds with the following questions:

- 1) Please indicate if EVENTS or DATA are to be used as input. >

If a series of event sequences are to be tested (i.e. bypassing the monitors) answer EVENTS. If a series of data files are to be read, answer DATA. The Autopilot can conduct a diagnostic session using either events or data files, but cannot mix the two formats within a single session.

If question (1) is answered EVENTS, the program inquires as to the event sequences that are to be run by asking the following questions:

- 2) Enter the Nth event in sequence M. Enter STOP to indicate the end of this sequence. >

The Autopilot will prompt the user to enter event sequences one event at a time. Events must be in MIDAS description format:

(object new-state new-trend new-status)⁷

Entering STOP will tell the Autopilot that there are no more events in the current sequence.

- 3) Add another sequence? [YES/NO] >

Answer NO when all event sequences have been entered. The Autopilot will begin executing the sequences immediately.

If question (1) is answered DATA, the program will ask for a series of files containing the data to be processed:

- 4) Enter the filename of the Nth data file to read. Enter STOP to indicate no more files. >

The Autopilot will prompt the user to enter file names one at a time. File names must include suffixes and will be checked to insure the indicated file exists⁸. When all files have been entered, type STOP, and diagnosis will begin immediately.

The Autopilot will display messages indicating which scenario is running and what stage of diagnosis is currently being performed. The results of each diagnosis are

⁷ The elements new-state, new-trend, and new-status are the values in the CONSEQUENT-STATE, CONSEQUENT-TREND, and CONSEQUENT-STATUS slots of the corresponding POTENTIAL-EVENT instance. If a slot is empty, NIL should be used in the event description.

⁸ The correct device and directory must be entered prior to starting the Autopilot utility using the *Set Data Entry* command option.

stored in the Archive Directory on the Archive Device (see *Set Diagnostics*) in files titled "SCEN.X" where X is an integer indicating the order of scenario execution (e.g. the first scenario run will be stored in a file titled "SCEN.1"). If the **Interrupt Default Setting** (see *Set Diagnostics*) has been set to SAVE, then the Autopilot will save the current status of all effected IMs whenever an RE is detected; otherwise, only the final diagnosis will be saved. After using the Autopilot, SCEN files should be renamed immediately, since any subsequent use of the Autopilot will write over the files.

→ **Set Monitor Thresholds**

This action gives the user the ability to reset monitor thresholds, save current monitor objects to a disk file, or train monitors automatically from data files. For more detailed information on monitor training, see section 5.5.1.

This utility is structured similarly to the Autopilot and queries the user for details of the actions desired. The user is asked the following questions:

- 1) Delete current monitor thresholds ? [YES/NO] >

If answered YES, a list of monitor thresholds will be presented and the user will be asked to indicate which thresholds to reset. Resetting a threshold deletes the current threshold value but does not replace it. Therefore, if any thresholds are reset, it will be necessary to set new values manually or using the training utility. If no thresholds are to be reset, answer NONE. Answering ALL will result in all thresholds of all monitors being reset. If a single threshold is indicated (by entering the number corresponding to the desired threshold) then the value of that threshold will be reset in all monitor objects.

- 2) Enter the filename of the Nth data file to read. Enter STOP to indicate no more files. >

Data files for monitor training should be entered one at a time and should include the proper suffix. Once MIDAS verifies that the file exists, the user is asked to confirm the selection before the file is added to the training queue. When all files have been entered, type STOP and training will begin immediately.

3) Save monitors to a file ? [YES/NO] >

If answered YES, the user will be prompted for the pathname of a file in which to save the monitor objects.

Monitor training is cumulative. That is, existing thresholds will only be changed if the new thresholds are less restrictive than those already established.

→ **Compile MIDAS Source Code**

This actions allows the user the option of compiling the MIDAS source code (i.e compiling LSP files to FAS files). Compiled code can provide a 20%-25% speed improvement with less frequent garbage collection if the memory partition is structured correctly⁹. Since FAS files are initially provided, this option is useful only when changes are made to the original source code.

Complete compilation of all diagnostic files can take 2 - 20 hours, depending on the platform used. Aborting a compilation in progress using Cntl-Break may result in the destruction of the original source code files and is, therefore, not recommended.

When this option is selected, the GoldWorks compiler will be loaded automatically. Because of the large memory requirement of the compiler, any process model in memory will be removed with the loss of any existing diagnostic objects. The program asks the user several questions before beginning compilation:

1) Please Indicate File to be Compiled. [# or ALL] >

Above this prompt will be a list of all files available for compilation. The first time this prompt is given there will be six (6) entries, each with a number to the right of the file name. To compile a file, enter the number corresponding to the appropriate file name. Because of the sensitivity to the order of compilation, it is recommended that ALL be entered. Answering ALL will compile all files in the

⁹ When running MIDAS from compiled code, the GoldWorks atom-to-cons ratio should be set to 3:2 to optimize performance. This differs from the 6:1 ratio suggested in the GoldWorks documentation.

correct order. Alternately, compilation order problems can be avoided by loading all diagnostic routines into memory using the **Load Diagnostic Routines** utility before compilation.

2) Compile More Files ?

If answered YES, an updated list of available files is displayed and the user is prompted to enter another file number. Note, files previously marked for compilation are removed from the list. If answered NO, the user is prompted for the pathname of the directory in which to place the compiled files.

3) Optimize Code For SPACE Or For SPEED ?

The user has the option of producing fast code or compact code. This is a GoldWorks feature that experience has shown makes little difference.

After these questions have been answered, compilation proceeds automatically. Remember, aborting the compilation after it begins (using Cntl-Break) may destroy the original source code files. When compilation is complete, it is recommended that the user exit and reload GoldWorks to clear the compiler from memory.

→ **Prepare Screen Dump**

This action primes MIDAS for a character screen dump. After performing this action, entering Cntl-Break from the MIDAS User Interface will initiate the screen dump procedure described in the Gold Hill Tech News (Vol. 3 No. 2). This action must be performed once per screen dump.

→ **None of the Above**

This action does nothing and can be used to exit the utilities without changing MIDAS.

5.3.2. Running a Diagnostic Session

The MIDAS User Interface can be started immediately after loading the diagnostic routines and a process model using MIDAS10D. MIDAS can be subsequently restarted from Top Level LISP by entering: (MIDAS).

5.3.2.1. LOGIN

Initially, the user is shown the WELCOME SCREEN which contains the Title Page illustrated in figure 5-2. To progress further, it is necessary to login to MIDAS:

- 1) Click on the *LOGIN* command option under the **LOGIN** menu item in the upper left corner of the screen.
- 2) Enter a valid name/password pair¹⁰ in the appropriate sections of the LOGIN Page shown in figure 5-3.
- 3) Click on the OK box to exit.

If the name/password pair is valid, then the ACTION SCREEN will appear with the message "Welcome To MIDAS!", otherwise, the WELCOME SCREEN will reappear.

5.3.2.2. ACTION SCREEN

The ACTION SCREEN, shown in figure 5-4, is divided into five sections: Menu Items, Event Status Window, Diagnosis Status Window, Popup Window Area, and Text Output Area. In the MIDAS color scheme, permanent windows are blue, popup (temporary) windows are cyan, static text is gray, and active text is white. All interaction with MIDAS is conducted through this screen.

There are five (5) menu items available in the ACTION SCREEN that can be used to access the various MIDAS command options. To select a command option, place the mouse pointer over the appropriate menu item to pull down the option menu. Place the pointer over the desired option and click the mouse button. Detailed descriptions of the menu items and their associated command options are given in section 5.3.2.4.

¹⁰ To change the name/password combinations that MIDAS recognizes at login, it is necessary to modify the CHECK-LOGIN-FILE routine in the DEMON10D file. At the beginning of the routine, the variable *login-pairs* is bound to a list of lists. Each sublist is a name/password pair. The first element is the NAME; the second element is the PASSWORD. NIL can be used as either a NAME or a PASSWORD.

The Event Status Window and Diagnosis Status Window provide an overview of the current system state. The Event Status Window alerts the user to any active, abnormal REs with the warning "*** Abnormal Events Detected ***" and indicates how many abnormal events are currently active. The Diagnosis Status Window alerts the user to any active IMs with the warning "*** Abnormal Condition Detected ***" and indicates how many IMs are currently present.

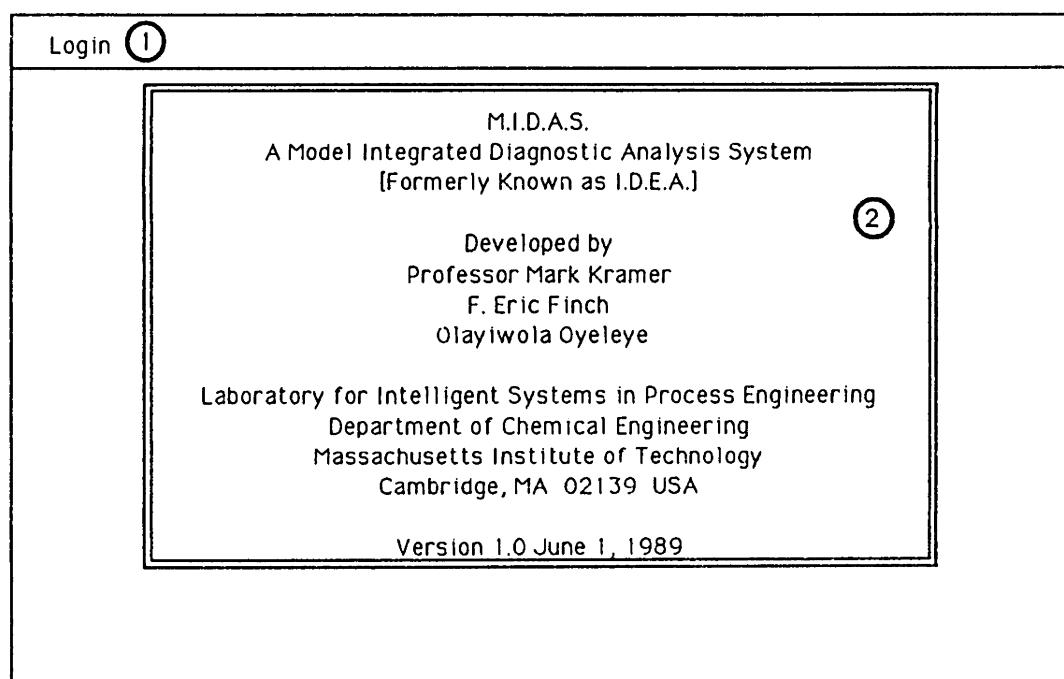
Popup windows (indicated by cyan borders) appear in the space below the Event Status and Diagnosis Status Windows. Once invoked, a popup window must receive an "answer" before it disappears and returns control of MIDAS to the user. Clicking on CANCEL will always delete a popup window without any change to MIDAS. When entering parameters (e.g. using the *LOGIN*, *Set Diagnostics*, or *Set Data Entry* command options) changes are made only if the window is exited by clicking on OK. For popup windows that display slot values of process model or diagnostic objects (see **Database Objects and Diagnostic Objects**) clicking on OK exits the window without changing MIDAS.

At the bottom of the ACTION SCREEN is the Text Output Window. Here, text messages are presented in white on a blue background. This window can hold a maximum of five lines and scrolls from bottom to top. Messages lost due to scrolling cannot be recovered. Messages can also be lost if a popup window overwrites the Text Output Window.

Two useful features of MIDAS are 1) it can display the amount of free memory remaining after every garbage collection, and 2) it times the evaluation of new REs. These features can help the user during diagnosis. If free memory drops below 500K atom space or 100,000 cons, MIDAS is in danger of crashing. The user should immediately free more space using the *Delete All Popups* or *Malfunction Corrected* commands. Error messages indicative of a crash induced by lack of memory are Special Stack Group Overflow, Atom Space Full, or Stack Group Reset. These errors are usually nonrecoverable; therefore, it is advised that if free memory approaches these limits, steps should be taken to recover additional memory¹¹.

¹¹ Three actions can be performed to recover memory during a diagnostic session:

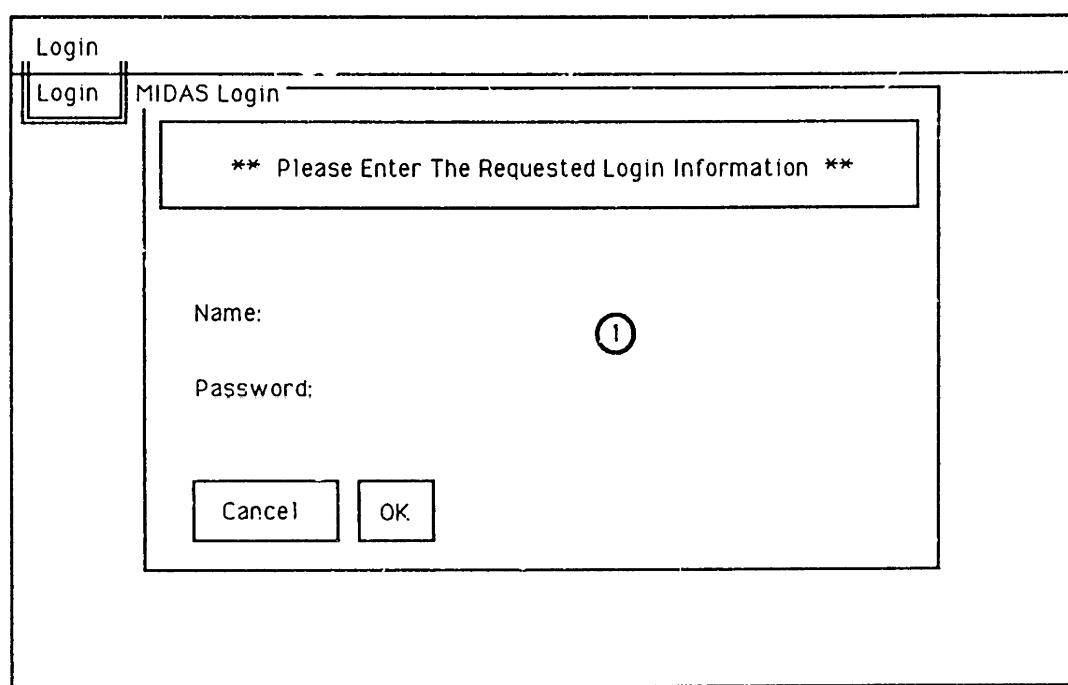
- 1) Remove the GoldWorks Menu Interface,
- 2) Remove the GMACS Editor, or
- 3) Use the *Delete All Popups* command option.



① Login Menu Item

② Title Page

Figure 5-2: MIDAS WELCOME Screen



① Login Page

Figure 5-3: MIDAS LOGIN Page

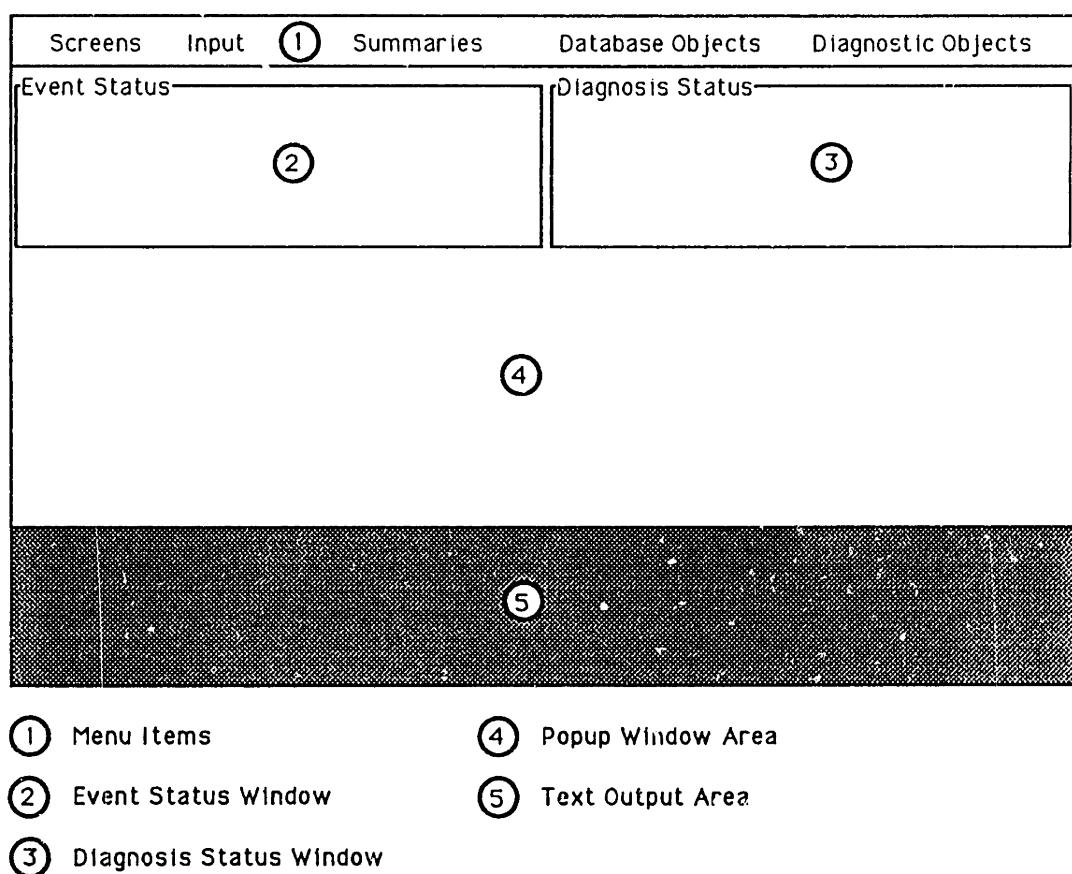


Figure 5-4: MIDAS ACTION Screen

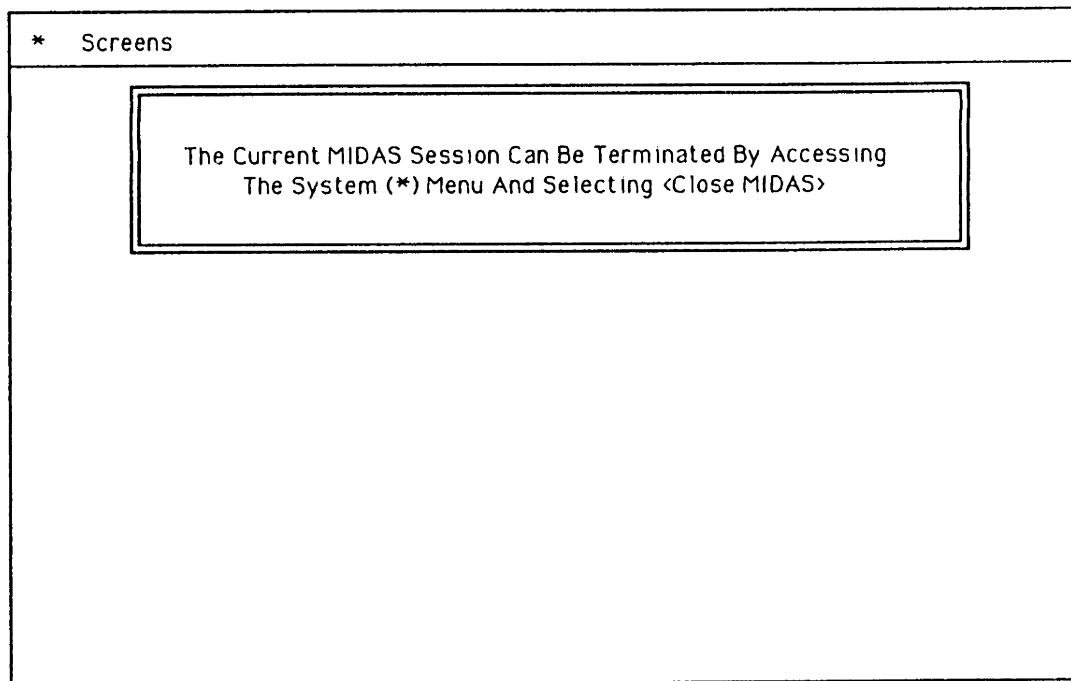


Figure 5-5: MIDAS EXIT Screen

5.3.2.3. Exit Screen

A user can terminate the current diagnostic session only from the EXIT SCREEN illustrated in figure 5-5. The EXIT SCREEN has two menu items: * and Screens. * allows the user to end the diagnostic session by selecting the *Close MIDAS* option; Screens allows the user to move back to the WELCOME or ACTION SCREEN.

5.3.2.4. Commands

MIDAS commands can be accessed via the menu items. To execute a command, place the mouse pointer over the appropriate menu item and click on the desired command option from the pull-down menu. Menu items and command options are described in detail below:

→ **Screens**

Allows movement between various MIDAS screens. Command options are

⇒ *WELCOME-SCREEN*

Brings up the WELCOME SCREEN from which the user must login again.

⇒ *ACTION-SCREEN*

Brings up the ACTION SCREEN. No Effect if entered from ACTION SCREEN.

⇒ *EXIT-SCREEN*

Brings up the EXIT SCREEN, allowing the user to end the diagnostic session.

→ **Input**

The command options under this item allow the user to change MIDAS parameters, input new information, and manage the MIDAS knowledge base. Command options are

⇒ *Initialize MIDAS*

Initializes MIDAS by deleting all objects created during diagnosis and resetting the process states and links to their nominal values, leaving MIDAS in its "bootup" state. This command will bring up a popup window asking the user to confirm the

command before proceeding. All data accumulated by the monitors is lost when this command is executed.

⇒ *Delete All Popups*

Deletes all popup windows stored in memory. This feature does not effect diagnosis, but can be used to recover memory during an extended diagnostic session. This command will bring up a popup window asking the user to confirm the action before it is completed. Periodic use of this command is recommended (use of this option can be considered an additional form of garbage collection).

⇒ *Malfunction Corrected*

This command can be used to remove from active memory all objects associated with a malfunction that has been corrected or has disappeared spontaneously (i.e. a transient or false alarm). Objects removed from memory will be archived on disk if the Archive Records to Disk File option is set to YES (see *Set Diagnostics*).

This command brings up a popup window prompting the user to enter the name of the IM that is to be removed. Enter the IM name in the data field¹² and click the OK box. Before completing this command, MIDAS checks to insure that all abnormal symptoms of the malfunction have disappeared.

⇒ *Access Utilities*

This command allows the user access to the utility control program from the User Interface. All utility actions can be performed normally, and when finished, the User Interface is restarted. Instructions for the various utility actions are provided in section 5.3.1.

⇒ *Set Diagnostics*

This command brings up a popup window that allows the user to change MIDAS diagnostic settings. Diagnostic settings include:

Maximum Precursor Search Depth

The maximum number of potential links that will be examined in the attempt to locate precursor events to the current event. This value must be a positive

¹² Clicking the mouse cursor over the data field will bring up a list of all possible IMs.

integer between 0 and 10 (default = 2). The value of this parameter has a strong influence on both the ability of MIDAS to link related events and the time necessary for diagnosis (i.e. large values may significantly increase the time needed for diagnosis).

Maximum Successor Search Depth

The maximum number of potential links that will be examined in the attempt to locate successor events to the current event. This value must be a positive integer between 0 and 10 (default = 2). The value of this parameter has a strong influence on both the ability of MIDAS to link related events and the time necessary for diagnosis (i.e. large values may significantly increase the time needed for diagnosis).

Interrogation Default Setting

Allows a default value to be set for monitor interrogation. The following values are allowed (default = NONE):

- § **T:** A "T" response will be returned for all monitor interrogations regardless of the monitor determination. This setting will result in the creation of the maximum number of EEs and LEs (many of which may be erroneous). This setting should never be used when event sequences are being entered manually (i.e. the monitors are bypassed).
- § **NIL:** A "NIL" response will be returned for all monitor interrogations regardless of the monitor determination. This setting will suppress creation of EEs and LEs.
- § **ASK:** A popup window will appear after each monitor interrogation asking the user to confirm the response determined by the monitor. If the **Display Popup Messages** parameter is set to NO, then this window is not displayed and the monitor response is used.
- § **NONE:** The monitor response will be used without asking the user for confirmation. If monitors are not being used, then NIL is returned.

Interrupt Default Setting

This setting allows the user to control the interrupt feature of MIDAS. The interrupt parameter is important since the command options can be accessed only during an interrupt. Three values are allowed (default = T):

- § **T:** Data entry will be interrupted after every RE detection.
- § **NIL:** Data entry will not be interrupted for any reason.
- § **SAVE:** Same as NIL, except a partial save is conducted after each RE detection. The most recently modified IM is saved to a SCEN file before proceeding (see **MIDAS Autopilot** utility). The Autopilot utility will automatically set this parameter to SAVE.
- § **NONE:** Data entry will be interrupted whenever a new RE is detected that modifies the existing diagnosis.

Level of Evidence Evaluation

The link structures inside an IM cluster can become complex when several events have been detected. When this situation occurs, full evaluation of the network can be prohibitively time consuming. As an alternative to full network evaluation, MIDAS offers two approximate evidence evaluation options.

- § **FULL:** A full evaluation of the evidence network is performed. Although the most rigorous, this setting can significantly delay event processing when the evidence network is large or highly connected.
- § **APPROXIMATE:** Evaluation of the network evidence is bounded at two layers. This setting reduces the time necessary for processing while capturing most of the information contained in the network.
- § **DOWN&DIRTY:** Only links directly associated with a suspected source event are evaluated for evidence. This setting will produce the quickest response.

§ **VARIABLE:** MIDAS will automatically adjust the level of evaluation based on the time elapsed during evaluation of the last RE. If the evaluation time of the last event exceeded the **Maximum Event Processing Time**, then the level of evaluation is decreased. If the evaluation time of the last event was less than the **Minimum Event Processing Time**, then the level of evaluation is increased.

Maximum Event Processing Time (minutes)

This setting determines the maximum evaluation time allowed per event before the level of evidence evaluation is decreased. This setting is only active when the **Level of Evidence Evaluation** is VARIABLE.

Minimum Event Processing Time (minutes)

This setting determines the minimum evaluation time allowed per event before the level of evidence evaluation is increased. This setting is only active when the **Level of Evidence Evaluation** is VARIABLE.

Type of Likelihood Displayed

MIDAS has six choices of likelihood function. The formulas used for calculating each type of likelihood are discussed in section 5.6. Likelihood is used to rank different root cause candidates, the root cause with the greatest likelihood being the prime candidate. The choice of likelihood function may effect the final diagnosis or the recommended tests. Modification of this setting will automatically update the Diagnostic Summary (see Summaries).

Relative Likelihood Display Threshold

This setting determines the minimum value of Relative Likelihood that will be displayed in the Diagnostic Summary (see Summaries). Values must be integers between -50 and 50 (default = -1). This setting is active when the **Type of Likelihood Displayed** parameter is set to

RELATIVE-LIKELIHOOD, or
NORMALIZED-RELATIVE-LIKELIHOOD.

Modification of this setting will automatically update the Diagnostic Summary.

Probabilistic Likelihood Display Threshold

This setting determines the minimum value of Probabilistic Likelihood that will be displayed in the Diagnostic Summary (see **Summaries**). Values must be between 0 and 1 (default = 0). This setting is active when the **Type of Likelihood Displayed** parameter is set to

PROBABILISTIC-LIKELIHOOD,
NORMALIZED-PROBABILISTIC-LIKELIHOOD,
CONDITIONAL-PROBABILITY, or
NORMALIZED-CONDITIONAL-PROBABILITY.

Modification of this setting will automatically update the Diagnostic Summary.

Test Evidence Weight

When calculating Relative Likelihood (section 5.6.1) or Normalized Relative Likelihood (section 5.6.2), this value determines the evidential weight assigned to TESTS. This value must be positive (default = 10).

Recorded Event Evidence Weight

When calculating Relative Likelihood (section 5.6.1) or Normalized Relative Likelihood (section 5.6.2), this value determines the evidential weight assigned to REs. This value must be positive (default = 1).

Expected or Latent Event Evidence Weight

When calculating Relative Likelihood (section 5.6.1) or Normalized Relative Likelihood (section 5.6.2), this value determines the evidential weight assigned to EEs and LEs. This value must be positive (default = 0.1).

Display Popup Messages?

MIDAS will occasionally display messages as popup windows (e.g. whenever a RE is detected, the result of an interrogation must be confirmed, or an error has occurred). The user can disable this feature by giving this parameter a NO value (default = YES). It should be noted that if this feature is disabled, the user may not be alerted to problems that arise during diagnosis. This setting is set to NO by the Autopilot.

Display Window Messages?

MIDAS will display messages in the Text Output Window at the bottom of the ACTION SCREEN. These messages include: data vectors sent to monitors, detected events, memory remaining (after garbage collection), and the time spent processing a RE. The user may disable this feature by giving this setting a NO value (default = YES). This setting is set to NO by the Autopilot.

Archive Records to Disk File?

During a diagnostic session, MIDAS may delete objects from memory. Such purges can be triggered by the diagnostic reasoning (certain hypotheses may be retracted as more information becomes available) or by a user request (see *Malfunction Corrected*). If this setting is YES (default = NO), then a permanent record of all deleted objects is written to a disk file in the **Archive Directory** on the **Archive Device**. Otherwise, all record of deleted objects is lost. This setting is set to YES by the Autopilot.

Archive Device

This setting determines the device on which archive files (including SCEN files) are written (default = C:).

Archive Directory

This setting determines the directory in which archive files (including SCEN files) are written (default = \\GW\\KBS\\MIDAS\\ARCHIVE). Note that directory pathnames must be entered with double backslashes even though only a single backslash is displayed. This directory must be created prior to running MIDAS¹³.

Learning Monitors

If **Learning Mode** is PARTIAL (see *Set Data Entry*), then all monitors listed here will learn threshold values from data. The learning feature of MIDAS is discussed in detail in section 5.5.1. Any number of monitors can be selected to learn thresholds. This setting is set automatically when using the **Set Monitor Thresholds** utility.

¹³ MINSTAL.BAT automatically creates the default Archive Directory:
"\\GW\\KBS\\MIDAS\\ARCHIVE\\".

Drift Compensating Monitors

All monitors listed here will undergo drift compensation. The drift compensation feature of MIDAS is discussed in detail in section 5.5.3. Any number of monitors can be set for drift compensation.

Modify Monitor

This setting allows monitors to be modified by the user during a diagnostic session. Entering a monitor name will bring up a popup window displaying current slot values for the designated monitor. Changes can be made to most slot values listed. This feature is useful for tuning monitors during a diagnostic session.

⇒ Set Data Entry

This command brings up a popup window that allows the user to change the current data entry settings. Data entry settings include:

Device

Device holding the current data file (default = C:).

Directory

Directory containing the current data file. Note, double backslashes must be entered even though only a single backslash is displayed.
(default = \\GW\\KBS\\MIDAS\\DATA).

Data File

Name of the current data file. When entering file name, a DBF suffix should be included in the file name to indicate a DBASE file.

Current Record

Record at which to begin reading data (default = 1).

End Record

Record at which data entry will stop (set by **Entry Mode**).

Entry Mode

One of three possible modes (default = AUTOMATIC):

- § **SINGLE:** A single record is read before stopping.
- § **GROUP:** A group of N records is read before stopping, where N is the contents of the **Group Size** setting.
- § **AUTOMATIC:** All records in the data file are read before stopping.

Group Size

Defines the number of records to be read before stopping. Must have a value if **Entry Mode** is set to GROUP.

Learning Mode

If the current data file contains data obtained during normal (fault-free) operation, the monitors can be set to learn appropriate test limits using this setting (see section 5.5.1). Three modes are possible (default = NONE):

- § **NONE:** No monitors are set to learn thresholds (i.e. all are set to detect abnormal events).
- § **PARTIAL:** Some monitors may be set to learn thresholds (see *Set Diagnostics*).
- § **TOTAL:** All monitors are set to learn thresholds.

⇒ *Enter Data*

Brings up a popup window asking the user to confirm that data entry should begin using the current settings (to change current settings see *Set Data Entry*).

⇒ *Enter New Events*

This command brings up a popup window that allows the user to enter an event into the MIDAS knowledge base manually, bypassing the monitors. Clicking on the desired event will create a new RE object and start diagnosis as if the event had

been detected by the monitors. The monitors, however, are not updated; therefore, it is recommended that users enter events manually or via monitor detection (but not both) during a single diagnostic session. When entering events manually, the **Interrogation Default Setting** should be set to NIL.

→ **Summaries**

Allows the user to view various summaries of the current diagnosis via popup windows. Command options are

⇒ *Sensor Summary*

Brings up a popup window containing a summary of the current status, state, and trend and a forecast of the future state for all SENSOR MONITORS.

⇒ *Constraint Summary*

Brings up a popup window containing a summary of the current status, state, and trend and a forecast of the future state for all CONSTRAINT MONITORS.

⇒ *System Summary*

Brings up a popup window containing a summary of the current status and state and a forecast of the future state for all SYSTEM MONITORS.

⇒ *Diagnostic Summary*

Brings up a popup window containing a summary of

IM Classifications

The classifications of all IMs are displayed. Malfunctions are classified as

- PERSISTENT if primary malfunction symptoms are persistent,
- SPURIOUS if primary malfunction symptoms have disappeared without secondary symptoms (often indicative of a false alarm),
- OSCILLATORY if primary malfunction symptoms alternately appear and disappear,

- ONGOING-TRANSIENT if primary malfunction symptoms have disappeared leaving only secondary symptoms,
- COMPLETED-TRANSIENT if both primary and secondary symptoms have disappeared,
- CORRECTED if all symptoms have disappeared and the operator has removed the malfunction from active memory.

IM Causes

A list of root cause candidates for each IM is displayed in ranked order. The top root cause has highest likelihood with likelihood decreasing for root causes listed progressively lower in the list. The likelihood value or interval is listed to the right of the root cause name.

The type of likelihood calculation used for ranking is determined by the **Type of Likelihood Displayed** parameter and can be changed from the User Interface (see *Set Diagnostics*). Only those root causes with likelihood greater than the **Display Threshold** (see *Set Diagnostics*) are displayed.

RE Associations

All active REs are indicated with their associated IM.

Active Tests

Any TESTS that are currently active are indicated.

→ **Database Objects**

Allows the user to view information on various objects contained in the process model. To exit the popup window, click on the OK box at the lower left corner of the window. Command options are

⇒ *View Variables*

Brings up a popup window containing information on the slot values of MEASURED VARIABLE objects in the process model database.

- ⇒ *View Constraints*
Brings up a popup window containing information on the slot values of MEASURED CONSTRAINT objects in the process model database.
 - ⇒ *View Systems*
Brings up a popup window containing information on the slot values of MEASURED SYSTEM objects in the process model database.
 - ⇒ *View Root Causes*
Brings up a popup window containing information on the slot values of POTENTIAL ROOT CAUSE objects in the process model database.
 - ⇒ *View Links*
Brings up a popup window containing information on the slot values of CAUSAL or FUNCTIONAL LINK objects in the process model database.
 - ⇒ *View Tests*
Brings up a popup window containing information on the slot values of TEST objects in the process model database.
 - ⇒ *View Monitors*
Brings up a popup window containing information on the slot values of MONITOR objects in the process model database.
- **Diagnostic Objects**
Allows the user to view information on various objects created by the diagnostic reasoning. To exit the popup window, click on the OK box at the lower left corner of the window. Command options are
- ⇒ *View Recorded Events*
Brings up a popup window containing information on the slot values of currently active RE objects.

⇒ *View Expected Events*

Brings up a popup window containing information on the slot values of currently active EE objects.

⇒ *View Latent Events*

Brings up a popup window containing information on the slot values of currently active LE objects.

⇒ *View Hypoth(esized) Causes*

Brings up a popup window containing information on the slot values of currently active HRC objects.

⇒ *View Malfunctions*

Brings up a popup window containing information on the slot values of currently active IM objects.

5.4. Creating Data Files for MIDAS

MIDAS can read data from files in DBASE III format. A sample data file -- TESTDATA.DBF -- is provided to illustrate the appropriate file structure.

Rather than enter data into DBASE files directly, a program -- CONVERT.PRG -- is supplied to automatically convert an ASCII format file into DBASE format. This program is run from within DBASE. Data must be written into the ASCII file in a specific format. Data records, each occupying a single line, must be written in chronological order with the oldest record at the top of the file. Records will consist of the following nine (9) data fields in order from left to right:

Monitor ID

This field must contain the ID of the monitor to which the data is to be sent (i.e. the value in the ID slot of the appropriate monitor object). The field is 32 characters wide with the monitor ID left justified.

Year

The year of measurement represented as a field of four (4) numbers (e.g. 1989).

Month

The month of measurement represented as a field of two (2) numbers (e.g. October = 10). For months with numerical designation less than 10, the preferred format includes a zero (e.g. February = 02), although a leading blank space can also be used.

Day

The day of measurement represented as field of two (2) numbers (e.g. 14). Again, for days with number designation less than 10 a leading zero is preferred.

Hours

The measurement hour represented in twenty four hour format as a field of two (2) numbers. Numbers less than 10 can have either a leading zero or a leading blank space.

Minutes

The measurement minute represented as a field of two (2) numbers. Numbers less than 10 can have either a leading zero or a leading blank space.

Seconds

The measurement second represented as a field of two (2) numbers. Fractional seconds are not allowed; thus, one second is the smallest time increment accepted by MIDAS. Numbers less than 10 can have either a leading zero or a leading blank space.

Measurement Value

A field fourteen (14) characters wide containing a real number with four decimal places. The contents of this field represent the numerical measurement sent to the monitor. The entry in this field must be right justified (i.e. the decimal place is always in position 10 in the field).

Measurement Units

An eight (8) character field containing a shorthand units designation for the measurement. This information is displayed when data recorded are entered but is not used in diagnostic reasoning. For example, "BTU/HR" could be used to show units for a heat flux. Units should be left justified in the field.

To use the conversion program:

- 1) Copy the ASCII file into the same directory as DBASE, CONVERT.PRG, and TESTDATA.DBF,
- 2) Make a copy of TESTDATA.DBF called RAWDATA.DBF (be careful to save the original TESTDATA.DBF file),
- 3) Rename the ASCII file (or make a copy) as "COMB.DAT",
- 4) Start DBASE, and at the "." prompt enter: DO CONVERT,
- 5) Exit DBASE.

RAWDATA.DBF will contain a MIDAS interpretable version of the ASCII file. It is recommended that the user rename RAWDATA.DBF at this time (retaining the DBF suffix). The original ASCII file, now named COMB.DAT, is unchanged by the procedure.

5.5. Monitors

The Monitors are a set of database objects and associated LISP code that converts numerical process data into the qualitative event descriptions used by the MIDAS diagnostic reasoning system. The basic mechanism used for conversion is based on statistical process control techniques (Shewhart control charts and exponentially weighted moving averages).

Because MIDAS is event driven, it is crucial that the monitors work well. If the monitors cannot detect an event, MIDAS cannot diagnose the cause. If the monitors detect too many events (i.e. false alarms), then diagnostic performance will be degraded. In most cases, the monitors will require "tuning". No specific procedure or rules for tuning are recommended here. A detailed examination of monitor tuning procedures is beyond the scope of this work and not central to MIDAS. Instead, the user is left to experiment to determine appropriate tuning parameters for a specific process under study. The goals of the tuning process should be to

- Minimize false event detection,
- Maximize true event detection,
- Determine appropriate prediction horizons (based on process time scales).

Of course, the first two goals are in conflict and a tradeoff must be established. Tuning is accomplished by making adjustments to monitor parameters (governing how data is accumulated), monitor thresholds, and the significance of the statistical tests used for detection. Using the **Set Monitor Thresholds** utility will create reasonable thresholds, assuming the data used for training is free of abnormal disturbances. Other monitor parameter must be set manually.

To provide the maximum degree of flexibility, the monitors implemented in MIDAS incorporate several special features, including: multiple decision criteria that can be easily modified to change the overall significance of the tests, the ability to learn preliminary decision thresholds from normal data, the ability to compensate for drift in selected process values or residuals, and the ability to forecast future values. These features are discussed in more detail below.

5.5.1. Learning Monitor Thresholds

Before the monitors can detect abnormal events, they must acquire information on allowable operating limits (thresholds). Because a process will experience some normal variability and raw measurements will contain noise, it is necessary to determine these limits statistically. Typically, a control limit will be set three (3) standard deviations and a warning limit will be set two (2) standard deviations from the nominal mean value in either direction.

Determining these limits can be a tedious task, especially if there are a large number of measured variables or constraints. To assist the user in determining limits, MIDAS monitors can be instructed to learn appropriate limits by reading one or more files containing data obtained during normal operation (see **Set Monitor Thresholds** utility). The limits obtained provide a starting point but may need fine adjustment. Limits determined by this procedure will be sensitive to the number of measurements in the file and to any non-random variations contained in the data. Both effects can be minimized by using a large data file to learn monitor limits. This procedure assumes that random errors are normally distributed.

MIDAS has three learning modes: TOTAL, PARTIAL, and NONE. Learning mode is set using the *Set Data Entry* command option. If learning mode is NONE, then no monitors learn limits from data. This is the appropriate setting for doing diagnosis. If learn mode is PARTIAL, then only those monitors listed as **Learning Monitors** (see *Set Diagnostics*) learn from data¹⁴. If learn mode is TOTAL, then all monitors learn from data (Note: the **Learning Monitors** parameter need not be filled when mode is TOTAL). When using the **Set Monitor Thresholds** utility, learning mode is automatically set to TOTAL (i.e. all monitors will learn from data).

When a monitor is set to learn limits, it cannot detect abnormal events, and in fact, may set limits incapable of recognizing certain abnormal events if abnormal events are included in the training data. Therefore, it is important that monitors learn thresholds only from normal data. Monitors limits can be reset by repeated use of the learn feature. Each application of the learn feature is cumulative. That is, if two data files are used to learn thresholds, the thresholds ultimately derived will be the maximum (for upper thresholds) and minimum (for lower thresholds) derived from the two files. Because extreme thresholds are retained, it may be necessary to reset the thresholds (see **Set Monitor Thresholds**) if they become too wide.

The routine LEARN-MONITOR-SETTINGS, contained in the MONIT10D file, guides the learning procedure and can be modified with some knowledge of LISP.

5.5.2. Monitor Test Criteria

The statistical tests used by MIDAS monitors can be modified by making simple changes in the source code (if MIDAS is to be run from compiled code, the source code must subsequently be recompiled). Some knowledge of LISP is required to make these modifications.

The MIDAS source code in question is in the MONIT10D file. The statistical tests are performed by the routine ANALYZE-DATA using four basic test functions: SELF-TEST, LIMIT-TEST, RUN-TEST, and TREND-TEST. Each of these functions accepts a list of numbers and performs an analysis. For example, LIMIT-TEST determines how many list

¹⁴ PARTIAL learning should not be used when data contains abnormal events.

elements are greater than, less than and equal to a given limit value. More detailed descriptions are included with the in-code documentation.

The statistical tests in ANALYZE-DATA are divided into three groups:

Status Evaluation

These tests are designed to determine if the data being received is reliable. Gross sensor failures should be detected by these tests before state or trend events are created.

1) Repeated Values Test

Tests for pairs of identical data means.

2) Max-Min Test

Tests for data means outside the feasible range as determined by the minimum and maximum values.

3) Variance Test 1 (optional)

Checks for standard deviations outside the SDEV control limit.

4) Variance Test 2 (optional)

Checks for standard deviations outside the SDEV warning limit.

5) Range Test

Checks the range of means for unusually large value changes.

State Evaluation

These tests check for a shift in the mean value.

1) Control Limit Test

Checks for means outside the MEAN control limit.

2) Warning Limit Test

Checks for means outside the MEAN warning limit.

Trend Evaluation

This test checks for a continuing trend in the process data.

1) Trend Test

Tests means for a run of increasing or decreasing values.

Each test has the form: (> (test function) N) where test function is an expression containing SELF-TEST, LIMIT-TEST, RUN-TEST, or TREND-TEST. The number N determines the significance of the test. To modify a test, the entire test can be deleted (i.e. commented out) or N can be changed to increase or decrease the test significance. Higher values of N will generally increase the test significance.

5.5.3. Drift Compensation

In certain situations, the nominal value of a measured variable or constraint will drift over time even if no malfunction is present. For example, constraints that involve integration of noisy data may, under normal conditions, experience a random walk or the ambient environment may slowly vary with diurnal and seasonal cycles.

Drift compensation in MIDAS is accomplished using a low-pass first order filter (i.e. an exponentially weighted moving average with small constant). If no abnormal events have been detected by the monitor, the drift filter value is assumed to represent the normal variable or constraint value. By comparing the new nominal value with the old value stored by the monitor, appropriate changes can be made to all monitor limits. It is assumed that drift of the nominal value does not affect the noise characteristics of the measurement.

If a monitor is set to compensate for drift, malfunctions that involve gradual deviation from normal, such as heat exchanger fouling, may not be detected. Caution must be exercised when using the drift compensation option. Whenever a variable may be influenced by a gradual failure, drift compensation is not recommended. Drift compensation will not, however, adversely affect the ability of the monitor to detect malfunctions that result in sudden deviation from normal operating conditions.

Drift compensation can be used to eliminate certain potential malfunctions from the process model. For example, if the cooling water utility demand of a process is affected by day/night

temperature variations, external temperature variations must be modeled as a possible "malfunction", monitor thresholds must be set sufficiently wide to mask the variations, or multiple sets of monitors must be used to explain flowrate changes. However, if the mean demand is allowed to drift over time, slow variations will result in moving monitor thresholds rather than abnormal events, and temperature variation can be eliminated as a potential malfunction.

The routine that performs drift compensation is COMPENSATE-FOR-DRIFT and resides in the MONIT10D file. The constant used by the drift filter can be modified using the **Modify Monitor** parameter (see *Set Diagnostics*). It is recommended that this value remain small. Drift compensation will be performed only for those monitors listed in the **Drift Compensating Monitors** list (see *Set Diagnostics*).

5.6. Hypothesis Likelihood Formulas

MIDAS can calculate the likelihood of a HRC using any of six possible formulas described below. Likelihood is used to rank HRCs, with the most likely generally appearing first whenever a list is presented. The choice of likelihood formula may effect ranking order, and therefore, the ultimate diagnosis.

5.6.1. Relative Likelihood

Relative Likelihood is computed by summing weighting values for all evidence associated with a HRC. Evidence supporting the HRC is added to the sum; evidence opposing the HRC is subtracted from the sum as shown in equation 5-1.

$$RL(H_i) = \sum_j^{SE} w_j - \sum_j^{OE} w_j \quad (5-1)$$

Here, $RL(H_i)$ is the Relative Likelihood of hypothesis H_i , OE represents the subset of all evidence (i.e. events) opposing H_i , SE represents the subset of all evidence supporting H_i ,

and w_j are the evidential weight values assigned to event j. Note, $OE \cup SE = AE^{15}$, where AE is the set of all evidence associated with an IM (i.e. every event associated with an IM will be either supporting or opposing evidence for every HRC associated with the IM).

The weighting value given each type of evidence (i.e. TESTS, REs, EEs, and LEs) can be adjusted from the User Interface (see *Set Diagnostics*). $RL(H_i)$ can be positive or negative. Positive values indicate a hypothesis for which supporting evidence outweighs opposing evidence. Negative values indicate opposing evidence outweighs supporting evidence. When Relative Likelihood is selected, only those HRCs with likelihood greater than the **Relative Likelihood Display Threshold** (see *Set Diagnostics*) will be displayed in the Diagnostic Summary.

5.6.2. Normalized Relative Likelihood

Normalized Relative Likelihood is a variation on Relative Likelihood that replaces a single summation of all evidence with two weighted evidence averages in the form of an evidential interval as shown in equation 5-2.

$$NRL(H_i) = [s^{NRL}(H_i), p^{NRL}(H_i)] \quad (5-2)$$

Here, $s^{NRL}(H_i)$ is the *supportability* of hypothesis H_i and $p^{NRL}(H_i)$ is the *plausibility* of hypothesis H_i . Supportability, calculated by equation 5-3, provides a lower bound on likelihood. Plausibility, calculated using equation 5-4, provides an upper bound. These intervals should exhibit the properties of the evidential intervals described by Shafer (1976).

If Normalized Relative Likelihood is selected in the **Type of Likelihood Displayed** (see *Set Diagnostics*), then evidential intervals will be displayed in the Diagnostic Summary, but ranking will be determined by the Relative Likelihood function.

¹⁵ This relationship is valid only when **Level of Evidence Evaluation** (see *Set Diagnostics*) is set to FULL.

$$s^{\text{NRL}}(H_i) = \left[\frac{\sum_{j}^{\text{SE}} w_j}{\sum_{j}^{\text{AE}} w_j} \right] \quad (5-3)$$

$$p^{\text{NRL}}(H_i) = 1 - \left[\frac{\sum_{j}^{\text{OE}} w_j}{\sum_{j}^{\text{AE}} w_j} \right] \quad (5-4)$$

5.6.3. Probabilistic Likelihood

Probabilistic Likelihood is calculated by taking the product of the probabilities of events. There are two probabilities associated with each event: the probability of accurate detection (β_i), and the probability of false detection (α_i). In principle, these probabilities can be computed for each event from knowledge of the statistical tests used for detection. However, in the current version of MIDAS, monitors lack this capability, and default probabilities are assigned based on event type. REs and TESTS are assigned $\beta_i = 0.95$ and $\alpha_i = 0.05$. EEs and LEs are assigned $\beta_i = 0.60$ and $\alpha_i = 0.40$. Probabilistic Likelihood is computed by the formula in equation 5-5.

$$PL(H_i) = \prod_j^{\text{SE}} \beta_j \cdot \prod_j^{\text{OE}} \alpha_j \quad (5-5)$$

Here, $PL(H_i)$ is the Probabilistic Likelihood of hypothesis H_i ($0 \leq PL(H_i) \leq 1$). When Probabilistic Likelihood is selected as the **Type of Likelihood Displayed** (see *Set Diagnostics*), only those hypotheses with Probabilistic Likelihood greater than the **Probabilistic Likelihood Display Threshold** will appear in the Diagnostic Summary.

5.6.4. Normalized Probabilistic Likelihood

Normalized Probabilistic Likelihood is an evidential interval consisting of a lower likelihood bound (supportability) and an upper likelihood bound (plausibility) given by the following expressions.

$$NPL(H_i) = [s^{NPL}(H_i) , p^{NPL}(H_i)] \quad (5-6)$$

$$s^{NPL}(H_i) = PL(H_i) \quad (5-7)$$

$$p^{NPL}(H_i) = 1 - PL(\sim H_i) \quad (5-8)$$

$$PL(\sim H_i) = \prod_j^{SE} \alpha_j \cdot \prod_j^{OE} \beta_j \quad (5-9)$$

$NPL(H_i)$ is the Normalized Probabilistic Likelihood, $s^{NPL}(H_i)$ is the supportability, and $p^{NPL}(H_i)$ is the plausibility of hypothesis H_i . $PL(\sim H_i)$ is the "unlikelihood" function for hypothesis H_i , expressing the likelihood that hypothesis H_i is not the true malfunction.

5.6.5. Conditional Probability

Conditional Probability is a likelihood formula based on Probabilistic Likelihood that incorporates knowledge of the prior probability of a malfunction. The Conditional Probability of a hypothesis H_i is computed by the equations 5-10 and 5-11.

$PP(H_i)$ is the *relative prior probability* and $NPP(H_i)$ is the *normalized prior probability* of hypothesis H_i ($0 \leq NPP(H_i) \leq 1.0$). $PP(H_i)$ is stored in the PRIOR-PROBABILITY slot of the PRC associated with H_i and must be provided during model construction. $NPP(H_i)$ assumes that one of the malfunction candidates is present in the system (i.e. excludes the no malfunction hypothesis).

$$CP(H_i) = \left[\frac{PL(H_i) \cdot NPP(H_i)}{\sum_i^N PL(H_i) \cdot NPP(H_i)} \right] \quad (5-10)$$

$$NPP(H_i) = \left[\frac{PP(H_i)}{\sum_i^N PP(H_i)} \right] \quad (5-11)$$

5.6.6. Normalized Conditional Probability

Normalized Conditional Probability produces an evidential interval of the same form as the Normalized Relative Likelihood and Normalized Probabilistic Likelihood. Equations 5-12 thru 5-16 are used to compute Normalized Conditional Probability.

$$NCP(H_i) = [s^{NCP}(H_i), p^{NCP}(H_i)] \quad (5-12)$$

$$s^{NCP}(H_i) = CP(H_i) \quad (5-13)$$

$$p^{NCP}(H_i) = CP(\sim H_i) \quad (5-14)$$

$$CP(\sim H_i) = \left[\frac{PL(\sim H_i) \cdot NPP(\sim H_i)}{\sum_i^N PL(\sim H_i) \cdot NPP(\sim H_i)} \right] \quad (5-15)$$

$$NPP(\sim H_i) = 1 - NPP(H_i) \quad (5-16)$$

Here, $NCP(H_i)$ is the Normalized Conditional Probability, $s^{NPL}(H_i)$ is the supportability, and $p^{NPL}(H_i)$ is the plausibility of hypothesis H_i . $CP(\sim H_i)$ and $NPP(\sim H_i)$ are the corresponding "unlikelihood" functions for $CP(H_i)$ and $NPP(H_i)$, respectively.

6.0. MIDAS Case Study

This section presents the results of an extensive case study performed to test MIDAS. The goal of the study was to determine performance statistics for Diagnostics Component of MIDAS and verify the process model produced by the Model Builder and Model Translator Components.

The case study examines MIDAS diagnostic performance in a simulated process environment. A simple chemical process was devised and a dynamic simulation of the process was developed. The process was designed to incorporate features challenging to MIDAS. For example, the process contains multiple feedback loops and nonlinear relations. Every effort was made to make the simulation realistic and representative of a real process.

The issues involved in scaling MIDAS to larger processes are discussed in section 7.1. The case study, though limited in scale, incorporates most of the model features that can be expected in real processes. Additional studies of MIDAS performance in both real and simulated processes are planned or currently underway.

6.1. Case Study Information

In the following sections, general background information on the process, process model, simulation program, and simulated fault cases is presented. This information defines the scope of the case study.

6.1.1. Process

The process used in the study is the Jacketed CSTR Process illustrated in figure 6-1. This process consists of twenty one (21) "units" (including feed and effluent streams). The units consist of

- Ten (10) sensors,
- Three (3) PI controllers,
- Two (2) control valves,
- One (1) centrifugal pump,

- One (1) jacketed reactor vessel,
- Two (2) feed streams, and
- Two (2) effluent streams.

Temperature control is accomplished by a cascade control system using measurements of reactor temperature and cooling water flowrate to adjust cooling water flow. Fluid level in the reactor is controlled by varying outlet flow. In addition to the indicated sensors, it is assumed that all controller output signals are available. The temperature controller output signal is the cooling water flow setpoint. Other controller setpoints are not measured directly.

The primary reaction ($A \rightarrow B$) is catalyzed, exothermic, and governed by a first order Arrhenius rate expression shown in equation (6-1).

$$\text{Rate}_{A \rightarrow B} = C_A [\alpha_1 e^{-(\beta_1/RT_T)}] \quad (6-1)$$

The side reaction ($A \rightarrow C$) is endothermic and first order as shown in equation (6-2). Under normal operating conditions, the side reaction is slow, accounting for only 0.23% of total reaction.

$$\text{Rate}_{A \rightarrow C} = C_A [\alpha_2 e^{-(\beta_2/RT_T)}] \quad (6-2)$$

In these equations, C_A and T_T are the reactant concentration and temperature in the reactor vessel and R is the gas constant. Both reactions are irreversible.

The nominal residence time of fluid in the reactor is twelve (12) minutes. The nominal cooling load is seventy five thousand six hundred (75600.0) KJ per minute. Other nominal process variable values are presented in Table 6-2.

6.1.2. Process Model

The process model was constructed by linking SDG unit models for the twenty one (21) units in the Jacketed CSTR Process and running the Model Builder and Model Translator programs developed by Oyeleye (1989). The SDG unit models for this process were also developed by Oyeleye and are included in the MIDAS Unit Model Library. The complete ESDG model for

the process contains one hundred nine (109) nodes, two hundred sixteen (216) arcs, and one hundred nine (109) root causes.

Creation of the process model required approximately six (6) hours of human interaction to input the process topology and resolve certain ambiguities in the qualitative models. The Model Builder and Model Translator required approximately eight (8) hours of computer time to create the basic process model¹.

After creation of the basic process model, constraint equations and tests were added to the model manually. Four (4) constraint equations are determinable given the available instrumentation: an overall reactor material balance, a reactor chemical species balance , a cooling water pressure drop equation, and a reactor effluent pressure drop equation. The constraints are formulated as shown in equations 6-3, 6-4, 6-5, and 6-6.

Inventory Constraint:

$$(L - L_{t=0}) \cdot A - \int_0^t (F_{in} - F_{out}) \cdot dt = R1 \quad (6-3)$$

Mol Balance Constraint:

$$\begin{aligned} & (C_A + C_B + C_C^{nominal}) \cdot L \cdot A - (C_A^{t=0} + C_B^{t=0} + C_C^{t=0}) \cdot L_{t=0} \cdot A \\ & - \int_0^t F_{in} C_A^{feed} \cdot dt + \int_0^t F_{out} (C_A + C_B + C_C^{nominal}) \cdot dt = R2 \end{aligned} \quad (6-4)$$

¹ The Model Builder and Translator were run in GoldWorks on a PC compatible 386 (20MHz) computer. With an improved Linker Interface and a faster implementation (either recoded in a faster language or moved to a faster platform), model creation can be made sufficiently fast to respond to process modifications.

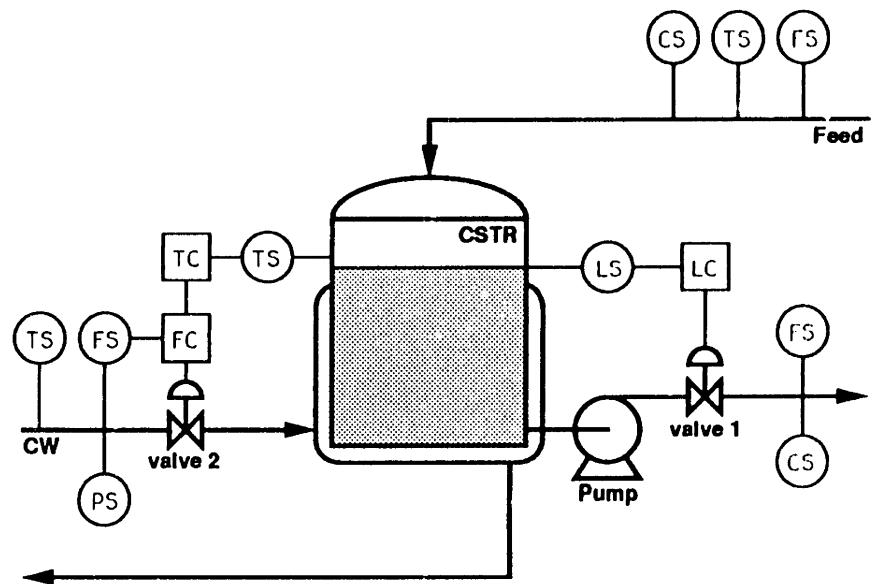


Figure 6-1: Jacketed CSTR Process

Cooling Water Pressure Drop Constraint:

$$F_{cw} - \frac{P_{cw}^{0.5}}{R_{jacket}^0 + 5.0 \exp(0.0545 \cdot C_F)} = R3 \quad (6-5)$$

Effluent Pressure Drop Constraint:

$$F_{out} - \frac{(\rho L g + \Delta P_{pump}^0)^{0.5}}{R_{effluent}^0 + 5.0 \exp(0.0545 \cdot C_L)} = R4 \quad (6-6)$$

In these equations, R1-R4 are the constraint residuals (nominally zero). F_{in} and F_{out} are the reactor feed flowrate and outlet flowrate, respectively. C_A , C_B , and C_C are the reactor concentrations of chemical species A, B, and C. C_L and C_F are the output signals from the level and cooling water flow controllers, respectively. L is the fluid level in the reactor and A is the area cross section of the reactor (a constant). R^0 is the nominal resistance in the indicated flow path.

Other than the addition of constraints and tests, the process model was not modified after creation by the Model Translator². The temptation to "fine tune" various aspects of the model by resolving ambiguities, removing redundant information, or breaking potential causal loops was resisted in order for the case study results to reflect performance on a "raw" process model. In any real application, fine tuning the process model using knowledge of specific process characteristics is desirable. The results of this case study, therefore, represent a minimum level of performance of MIDAS.

Monitor thresholds were set using fault-free data generated by the simulation program using the **Set Monitor Thresholds** utility. No additional tuning of the monitors was performed.

² Some of the objects were renamed because the names generated by the Model Translator were unnecessarily complex or convoluted. However, no changes were made to the structure of the process model or the conditions attached to links.

The performance of the automatic threshold training program can be judged by the number of monitor errors committed during the case study. The optional variance check of the monitors was disabled for this case study.

The process model consisted of

- One hundred forty four (144) POTENTIAL-EVENT instances,
- One hundred nine (109) POTENTIAL-ROOT-CAUSE instances,
- Three hundred thirty four (334) COMPILED-LINK instances,
- Eighteen (18) MEASURED-OBJECT instances,
- Eighteen (18) DATA-MONITOR instances, and
- Twenty (20) TEST instances,

All PRCs had equal prior probability (set to 1.0). Approximately 30% of CL objects were activatable.

The MIDAS settings used for the case study were as follows:

Maximum Precursor Search Depth	-	2
Maximum Successor Search Depth	-	2
Interrogation Default Setting	-	NONE
Interrupt Default Setting	-	SAVE
Level of Evidence Evaluation	-	VARIABLE
Maximum Event Processing Time	-	10
Minimum Event Processing Time	-	2
Type of Likelihood Displayed	-	CONDITIONAL-PROBABILITY
Probabilistic Likelihood Display Threshold	-	0.0

None of the monitors were set to use drift compensation.

For this process, the event model can be seen to be approximately 50% larger than the ESDG model from which it is derived. This growth factor is a function of the fraction of measured nodes in the ESDG model. The larger the fraction of measured nodes, the larger the growth factor; the smaller the fraction of measured node, the smaller the growth factor. In fact, it is possible for an event model to be smaller than an ESDG model if less than 10% of ESDG nodes are measured. Unfortunately, when such a small fraction of nodes are measured, the

quality of diagnosis is likely to be poor. The increase in model size and complexity that normally accompanies conversion to the event model format can be viewed as the cost of creating representing dynamic relationships explicitly rather than implicitly, as in the ESDG.

The average connectivity of the process model is 9 links/node³. The connectivity distribution of nodes is illustrated in figure 6-2. This figure shows the number of nodes with five or fewer (0-5) links, five to ten (5-10) links, ten to fifteen (10-15) links, and so on.

High average connectivity can be a warning that the process model will not perform well. Besides increasing the time necessary for search and evidence evaluation, high connectivity robs the model of the event sequence information needed to produce a superior diagnosis. In a maximally connected model (i.e. every node connected to every other node), there is no sequence information -- any event can cause or be caused by any other event, all events must be considered SOURCE events, and multiple IM clusters are impossible.

In the Jacketed CSTR process model, the nodes associated with reactor temperature are highly connected, since almost any process change will affect reactor temperature. Nodes associated with cooling water setpoint are highly connected because any disturbance effecting reactor temperature can be transferred to cooling water setpoint while leaving reactor temperature in a seemingly normal state. Other highly connected nodes in the model are associated with cooling water flow controller signal, level controller signal, produce concentration of A and B, and product flowrate. Consequently, these events may produce complex IM clusters with large candidate sets when classified as SOURCE events.

An average connectivity of ten (10) to fifteen (15) can be expected for many processes. An average of twenty five (25) to thirty (30) may drastically reduce the quality of diagnosis and increase the time needed to complete the inference cycle.

³ This average connectivity is twice the gross average connectivity (4.6 links/node) computed using the formula:

$$\text{Gross Ave. Connectivity} = (2 \times (\# \text{ of Links})) / (\# \text{ of PEs})$$

This difference indicates a skewed connectivity distribution.

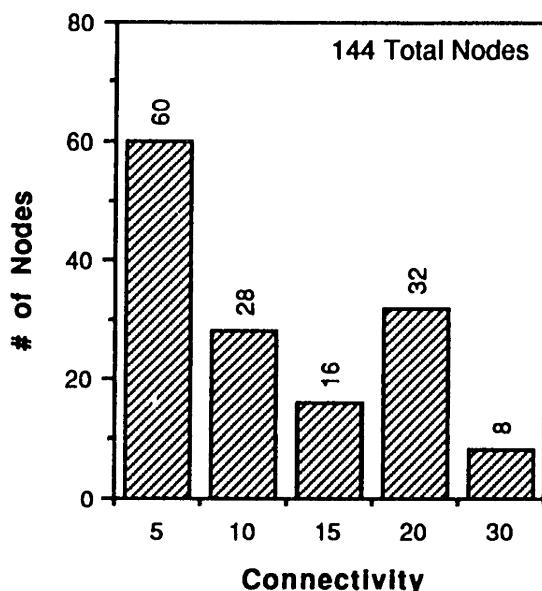


Figure 6-2: Node Connectivity Distribution

6.1.3. Simulation

The simulation for the Jacketed CSTR Process is written in FORTRAN77 and runs in batch mode to produce files containing simulated sensor data. Constraint residuals are computed by the simulation program and treated identically to other measurements.

The simulation uses dynamic models of system interactions to reproduce the time dependent behaviors exhibited by real processes, including controller overshoot and underdamped oscillatory behavior. Random errors are added to measurement values to simulate sensor noise. Noisy values are used in all controller calculations.

Sensor and constraint values are written to data files in random order so measurements are not always read by MIDAS in the same order. Values are written at different frequencies to simulate longer sampling times for certain sensors. Temperatures and controller signals, for

example, are written to the data files at three (3) times the frequency of concentration measurements.

Tuning of the simulated controllers (all using a PI control algorithm) was accomplished largely by trial and error to achieve reasonable control response to step changes in input variables. Tuning was not changed during the case study.

The simulation program can simulate one hundred six (106) different fault modes, including bias or fixed failure of sensors and controllers. The fault trajectory and ultimate extent can be specified, in effect, making the set of possible fault cases infinite. The trajectory is determined by entering an exponential time constant. All faults approach their ultimate extent using a decaying exponential function. For example, with a time constant (τ) of 0.01, it takes seventy (70) simulated minutes for a fault to reach 50% of its ultimate extent. With a time constant of 1.0, it takes forty two (42) simulated seconds to reach the same extent, and only 0.42 simulated seconds with a τ of 100.0. A delay period can be specified before the fault enters the process. Up to three separate faults can be simulated during a single program run⁴. Each fault can have a different delay period.

Table 6-1 presents the failure modes in the process model and indicates whether the failure mode is included in the simulation. Table 6-2 describes the nominal values of fault variables and presents the range of ultimate extents used to generate fault cases. Appendix 2 gives more details on the simulation, including a listing of the FORTRAN code.

⁴ A second or third "fault" can actually be a variable or parameter returning to its nominal value. This feature is used to simulate transients.

TABLE 6-1: Unit Failure Modes for Jacketed CSTR Process

Mode	Unit	Failure Mode	Simulated	Comments ¹
1	Reactor Feed	Source Concentration A High	Yes	
2	Reactor Feed	Source Concentration A Low	Yes	
3	Reactor Feed	Source Pressure High	Yes	increased flow
4	Reactor Feed	Source Pressure Low	Yes	decreased flow
5	Reactor Feed	Source Temperature High	Yes	
6	Reactor Feed	Source Temperature Low	Yes	
7	CSTR	Inlet Blockage	No	
8	CSTR	Outlet Blockage	Yes	
9	CSTR	Leak to Environment	No	18
10	CSTR	Loss of Mixing	No	
11	CSTR	Catalyst 1 Deactivation	Yes	
12	CSTR	Catalyst 2 Deactivation	Yes	
13	CSTR	Fire	Yes	
14	CSTR	Loss of Insulation	Yes	
15	Pump	Blockage	No	8
16	Pump	Fire	No	unobservable
17	Pump	Loss of Insulation	No	unobservable
18	Pump	Leak to Environment	Yes	
19	Pump	Cavitation	No	22
20	Pump	Overheating	No	unobservable
21	Pump	Motor Speed High	Yes	increased ΔP
22	Pump	Motor Speed Low	Yes	decreased ΔP
23	Pump	Motor Failed	No	22
24	Pump	Shaft Broken	No	22
25	Valve 1	Blockage	No	8
26	Valve 1	Fire	No	unobservable
27	Valve 1	Loss of Insulation	No	unobservable
28	Valve 1	Leak to Environment	No	18
29	Valve 1	Stuck Open	Yes	
30	Valve 1	Stuck Closed	Yes	
31	Reactor Effluent	Sink Pressure High	Yes	
32	Reactor Effluent	Sink Pressure Low	Yes	
33	Jacket Feed	Source Pressure High	Yes	

TABLE 6-1: Unit Failure Modes for Jacketed CSTR Process (continued)

Mode	Unit	Failure Mode	Simulated	Comments ¹
34	Jacket Feed	Source Pressure Low	Yes	
35	Jacket Feed	Source Temperature High	Yes	
36	Jacket Feed	Source Temperature Low	Yes	
37	Valve 2	Blockage	Yes	jacket blockage
38	Valve 2	Fire	No	
39	Valve 2	Loss of Insulation	No	
40	Valve 2	Leak to Environment	No	45
41	Valve 2	Stuck Open	Yes	
42	Valve 2	Stuck Closed	Yes	
43	CSTR Jacket	Fouling	Yes	
44	CSTR Jacket	Leak to Reactor	Yes	
45	CSTR Jacket	Leak to Environment	Yes	
46	Jacket Effluent	Sink Pressure High	Yes	
47	Jacket Effluent	Sink Pressure Low	Yes	
48	Level Controller	Failed High	Yes	
49	Level Controller	Failed Low	Yes	
50	Level Controller	Biased High	Yes	
51	Level Controller	Biased Low	Yes	
52	Level Controller	Setpoint High	Yes	
53	Level Controller	Setpoint Low	Yes	
54	Temperature Controller	Failed High	Yes	
55	Temperature Controller	Failed Low	Yes	
56	Temperature Controller	Biased High	Yes	
57	Temperature Controller	Biased Low	Yes	
58	Temperature Controller	Setpoint High	Yes	
59	Temperature Controller	Setpoint Low	Yes	
60	CW Flow Controller	Failed High	Yes	
61	CW Flow Controller	Failed Low	Yes	
62	CW Flow Controller	Biased High	Yes	
63	CW Flow Controller	Biased Low	Yes	
64	CW Flow Controller	Setpoint High	No	54
65	CW Flow Controller	Setpoint Low	No	55
66	Feed Flow Sensor	Failed High	Yes	

TABLE 6-1: Unit Failure Modes for Jacketed CSTR Process (continued)

Mode	Unit	Failure Mode	Simulated	Comments ¹
67	Feed Flow Sensor	Failed Low	Yes	
68	Feed Flow Sensor	Biased High	Yes	
69	Feed Flow Sensor	Biased Low	Yes	
70	Feed Temperature Sensor	Failed High	Yes	
71	Feed Temperature Sensor	Failed Low	Yes	
72	Feed Temperature Sensor	Biased High	Yes	
73	Feed Temperature Sensor	Biased Low	Yes	
74	Feed Concentration Sensor	Failed High	Yes	
75	Feed Concentration Sensor	Failed Low	Yes	
76	Feed Concentration Sensor	Biased High	Yes	
77	Feed Concentration Sensor	Biased Low	Yes	
78	Reactor Temperature Sensor	Failed High	Yes	
79	Reactor Temperature Sensor	Failed Low	Yes	
80	Reactor Temperature Sensor	Biased High	Yes	
81	Reactor Temperature Sensor	Biased Low	Yes	
82	Reactor Level Sensor	Failed High	Yes	
83	Reactor Level Sensor	Failed Low	Yes	
84	Reactor Level Sensor	Biased High	Yes	
85	Reactor Level Sensor	Biased Low	Yes	
86	Product Flow Sensor	Failed High	Yes	
87	Product Flow Sensor	Failed Low	Yes	
88	Product Flow Sensor	Biased High	Yes	
89	Product Flow Sensor	Biased Low	Yes	
90	Concentration A Sensor	Failed High	Yes	
91	Concentration A Sensor	Failed Low	Yes	
92	Concentration A Sensor	Biased High	Yes	
93	Concentration A Sensor	Biased Low	Yes	
94	Concentration B Sensor	Failed High	Yes	
95	Concentration B Sensor	Failed Low	Yes	
96	Concentration B Sensor	Biased High	Yes	
97	Concentration B Sensor	Biased Low	Yes	
98	CW Flow Sensor	Failed High	Yes	
99	CW Flow Sensor	Failed Low	Yes	

TABLE 6-1: Unit Failure Modes for Jacketed CSTR Process (continued)

Unit	Failure Mode	Simulated	Comments ¹
100	CW Flow Sensor	Biased High	Yes
101	CW Flow Sensor	Biased Low	Yes
102	CW Temperature Sensor	Failed High	Yes
103	CW Temperature Sensor	Failed Low	Yes
104	CW Temperature Sensor	Biased High	Yes
105	CW Temperature Sensor	Biased Low	Yes
106	CW Pressure Sensor	Failed High	Yes
107	CW Pressure Sensor	Failed Low	Yes
108	CW Pressure Sensor	Biased High	Yes
109	CW Pressure Sensor	Biased Low	Yes

- ¹ Three types of comments: 1) If a fault mode cannot be detected using the given instrumentation, this fact will be indicated by the comment "unobservable",
 2) If a fault mode cannot be simulated, the failure number of a qualitatively identical simulated fault mode is given,
 3) If a fault mode cannot be simulated as stated, the simulated effect is stated.

TABLE 6-2: Simulated Failure Mode Effects on Jacketed CSTR Process

Mode	Variable Effected	Nominal Value	Failure Range	Units
1	Reactor Feed Concentration	20.0	[20.0 , 30.0]	Kmol / m ³
2	Reactor Feed Concentration	20.0	[20.0 , 0.0]	Kmol / m ³
3	Reactor Feed Flowrate	0.25	[0.25 , 0.35]	m ³ / min
4	Reactor Feed Flowrate	0.25	[0.25 , 0.0]	m ³ / min
5	Reactor Feed Temperature	30.0	[30.0 , 50.0]	C
6	Reactor Feed Temperature	30.0	[30.0 , 10.0]	C
8	Outlet Resistance	18.0	[100.0 , 600.0]	min Kg ^{0.5} /m ⁴
11	Primary Activation Energy	25000.0	[25000.0 , 30000.0]	KJ / Kmol
12	Secondary Activation Energy	45000.0	[45000.0 , 54000.0]	KJ / Kmol
13	External Heat Flux	0.0	[0.0 , 10000.0]	KJ / min
14	External Heat Flux	0.0	[0.0 , -10000.0]	KJ / min
18	Pump Leak Resistance	1000000.0	[90000.0 , 1000.0]	min Kg ^{0.5} /m ⁴
21	Pump ΔP	48000.0	[48000.0 , 60000.0]	Kg /m ²
22	Pump ΔP	48000.0	[48000.0 , 0.0]	Kg /m ²
29	Valve 1 Position	74.7	[74.7 , 0.0]	% Closed
30	Valve 1 Position	74.7	[74.7 , 100.0]	% Closed
31	Reactor Effluent Pressure	0.0	[0.0 , 10332.0]	Kg /m ²
32	Reactor Effluent Pressure	0.0	[0.0 , -10332.0]	Kg /m ²
33	CW Source Pressure	56250.0	[56250.0 , 80000.0]	Kg /m ²
34	CW Source Pressure	56250.0	[56250.0 , 0.0]	Kg /m ²
35	CW Source Temperature	20.0	[20.0 , 40.0]	C
36	CW Source Temperature	20.0	[20.0 , 0.0]	C
37	Jacket Resistance	0.0	[0.0 , 150.0]	min Kg ^{0.5} /m ⁴
41	Valve 2 Position	59.3	[59.3 , 0.0]	% Closed
42	Valve 2 Position	59.3	[59.3 , 100.0]	% Closed
43	Heat Transfer Coefficient	1901.0	[1901.0 , 1600]	KJ / min C
44	Internal Leak Resistance	1000000.0	[20000.0 , 5000.0]	min Kg ^{0.5} /m ⁴
45	External Leak Resistance	1000000.0	[20000.0 , 1000.0]	min Kg ^{0.5} /m ⁴
46	Jacket Effluent Pressure	0.0	[0.0 , 10332.0]	Kg /m ²
47	Jacket Effluent Pressure	0.0	[0.0 , -10332.0]	Kg /m ²
48	LC Output	74.7	[74.7 , 0.0]	% Closed
49	LC Output	74.7	[74.7 , 100.0]	% Closed
50	LC Output	0.0	[0.0 , 10.0]	% Closed

TABLE 6-2: Simulated Failure Mode Effects on Jacketed CSTR Process (continued)

Mode	Variable Effected	Nominal Value	Failure Range	Units
51	LC Output	0.0	[0.0 , -10.0]	% Closed
52	LC Setpoint	2.0	[2.0 , 2.5]	m
53	LC Setpoint	2.0	[2.0 , 1.5]	m
54	TC Output	0.9	[0.9 , 1.6]	m ³ / min
55	TC Output	0.9	[0.9 , 0.0]	m ³ / min
56	TC Output	0.0	[0.0 , 0.2]	m ³ / min
57	TC Output	0.0	[0.0 , -0.2]	m ³ / min
58	TC Setpoint	80.0	[80.0 , 90.0]	C
59	TC Setpoint	80.0	[80.0 , 70.0]	C
60	FC Output	59.3	[59.3 , 100.0]	% Closed
61	FC Output	59.3	[59.3 , 0.0]	% Closed
62	FC Output	0.0	[0.0 , 10.0]	% Closed
63	FC Output	0.0	[0.0 , -10.0]	% Closed
66	Feed Flow Measurement	0.25	[0.25 , 0.35]	m ³ / min
67	Feed Flow Measurement	0.25	[0.25 , 0.0]	m ³ / min
68	Feed Flow Measurement	0.0	[0.0 , 0.1]	m ³ / min
69	Feed Flow Measurement	0.0	[0.0 , -0.1]	m ³ / min
70	Feed Temp. Measurement	30.0	[30.0 , 50.0]	C
71	Feed Temp. Measurement	30.0	[30.0 , 10.0]	C
72	Feed Temp. Measurement	0.0	[0.0 , 7.5]	C
73	Feed Temp. Measurement	0.0	[0.0 , -7.5]	C
74	Feed Conc. Measurement	20.0	[20.0 , 30.0]	Kmol / m ³
75	Feed Conc. Measurement	20.0	[20.0 , 0.0]	Kmol / m ³
76	Feed Conc. Measurement	0.0	[0.0 , 2.0]	Kmol / m ³
77	Feed Conc. Measurement	0.0	[0.0 , -2.0]	Kmol / m ³
78	Reactor Temp. Measurement	80.0	[80.0 , 130.0]	C
79	Reactor Temp. Measurement	80.0	[80.0 , 0.0]	C
80	Reactor Temp. Measurement	0.0	[0.0 , 10.0]	C
81	Reactor Temp. Measurement	0.0	[0.0 , -10.0]	C
82	Reactor Level Measurement	2.0	[2.0 , 2.75]	m
83	Reactor Level Measurement	2.0	[2.0 , 1.2]	m
84	Reactor Level Measurement	0.0	[0.0 , 0.5]	m
85	Reactor Level Measurement	0.0	[0.0 , -0.5]	m

TABLE 6-2: Simulated Failure Mode Effects on Jacketed CSTR Process (continued)

Mode	Variable Effected	Nominal Value	Failure Range	Units
86	Product Flow Sensor	0.25	[0.25 , 0.35]	m^3 / min
87	Product Flow Sensor	0.25	[0.25 , 0.0]	m^3 / min
88	Product Flow Sensor	0.0	[0.0 , 0.1]	m^3 / min
89	Product Flow Sensor	0.0	[0.0 , -0.1]	m^3 / min
90	Reactor Conc. A Measurement	2.85	[2.85 , 30.0]	Kmol / m^3
91	Reactor Conc. A Measurement	2.85	[2.85 , 0.0]	Kmol / m^3
92	Reactor Conc. A Measurement	0.0	[0.0 , 0.71]	Kmol / m^3
93	Reactor Conc. A Measurement	0.0	[0.0 , -0.71]	Kmol / m^3
94	Reactor Conc. B Measurement	17.11	[17.11 , 30.0]	Kmol / m^3
95	Reactor Conc. B Measurement	17.11	[17.11 , 0.0]	Kmol / m^3
96	Reactor Conc. B Measurement	0.0	[0.0 , 4.25]	Kmol / m^3
97	Reactor Conc. B Measurement	0.0	[0.0 , -4.25]	Kmol / m^3
98	CW Flow Measurement	0.9	[0.9 , 2.0]	m^3 / min
99	CW Flow Measurement	0.9	[0.9 , 0.0]	m^3 / min
100	CW Flow Measurement	0.0	[0.0 , 0.2]	m^3 / min
101	CW Flow Measurement	0.0	[0.0 , -0.2]	m^3 / min
102	CW Temp. Measurement	20.0	[20.0 , 40.0]	C
103	CW Temp. Measurement	20.0	[20.0 , 0.0]	C
104	CW Temp. Measurement	0.0	[0.0 , 5.0]	C
105	CW Temp. Measurement	0.0	[0.0 , -5.0]	C
106	CW Pressure Measurement	56250.0	[56250.0 , 800000.0]	Kg / m^2
107	CW Pressure Measurement	56250.0	[56250.0 , 0.0]	Kg / m^2
108	CW Pressure Measurement	0.0	[0.0 , 14000.0]	Kg / m^2
109	CW Pressure Measurement	0.0	[0.0 , -14000.0]	Kg / m^2

Note: Quoted sensor failure ranges are for in-range failure.

6.1.4. Simulated Faults

One hundred (100) faults were chosen at random (with replacement) from the set of one hundred nine (109) PRCs and were assigned random extents and trajectories. These faults were the basis of the case study. The faults selected are presented in Table 6-3.

TABLE 6-3: Randomly Generated Faults for Case Study

#	Mode	Extent (% of range)	Trajectory Time Constant	Description
1	8	95	0.1	CSTR_OUTLET_BLOCKAGE
2	21	99	1.0	PUMP_MOTOR_SPEED_HIGH
3	39	45	10.0	¹ VALVE_2 LOSS_OF_INSULATION
4	26	46	10.0	² VALVE_1_FIRE
5	6	42	0.1	REACTOR_FEED_TEMPERATURE_LOW
6	37	32	10.0	VALVE_2_BLOCKAGE
7	11	12	10.0	CSTR_CATALYST_1_DEACTIVATION
8	91	70	100.0	³ CONC_A_SENSOR FAILED_LOW
9	39	40	100.0	¹ VALVE_2 LOSS_OF_INSULATION
10	102	79	1.0	³ CW_TEMP_SENSOR FAILED_HIGH
11	87	34	0.1	³ PRODUCT_FLOW_SENSOR FAILED_LOW
12	105	55	1.0	CW_TEMP_SENSOR_BIASED_LOW
13	12	70	0.01	⁵ CSTR_CATALYST_2_DEACTIVATION
14	35	25	10.0	JACKET_FEED_SOURCE_TEMP_LOW
15	72	55	0.01	FEED_TEMP_SENSOR_BIASED_HIGH
16	90	91	10.0	³ CONC_A_SENSOR FAILED_HIGH
17	91	52	0.1	⁴ CONC_A_SENSOR FAILED_LOW
18	83	52	100.0	³ REACTOR_LEVEL_SENSOR FAILED_LOW
19	102	2	0.01	⁴ CW_TEMP_SENSOR FAILED_HIGH
20	8	84	1.0	CSTR_OUTLET_BLOCKAGE
21	14	88	1.0	CSTR_LOSS_OF_INSULATION
22	108	10	10.0	CW_PRESSURE_SENSOR_BIASED_HIGH
23	2	66	0.01	REACTOR_FEED_CONC_A_LOW
24	26	35	1.0	² VALVE_1_FIRE
25	15	71	1.0	CSTR_OUTLET_BLOCKAGE
26	86	50	1.0	⁴ PRODUCT_FLOW_SENSOR FAILED_HIGH
27	79	92	1.0	⁴ REACTOR_TEMP_SENSOR FAILED_LOW
28	87	57	100.0	³ PRODUCT_FLOW_SENSOR FAILED_LOW
29	29	33	100.0	VALVE_1_STUCK_OPEN
30	100	88	100.0	CW_FLOW_SENSOR_BIASED_HIGH
31	73	9	10.0	⁵ FEED_TEMP_SENSOR_BIASED_LOW
32	2	86	0.01	REACTOR_FEED_CONC_A_LOW

TABLE 6-3: Randomly Generated Faults for Case Study (continued)

#	Mode	Extent (% of range)	Trajectory Time Constant	Description
33	61	100	0.01	CW_FLOW_CONTROLLER_FAILED_LOW
34	1	89	0.1	REACTOR_FEED_CONC_A_HIGH
35	8	24	0.01	CSTR_OUTLET_BLOCKAGE
36	49	22	100.0	LEVEL_CONTROLLER_FAILED_LOW
37	79	55	10.0	⁴ REACTOR_TEMP_SENSOR FAILED LOW
38	21	56	0.01	PUMP_HIGH_MOTOR_SPEED
39	88	84	1.0	PRODUCT_FLOW_SENSOR_BIASED_HIGH
40	86	54	0.01	³ PRODUCT_FLOW_SENSOR FAILED LOW
41	85	84	10.0	REACTOR_LEVEL_SENSOR_BIASED_LOW
42	89	15	0.01	PRODUCT_FLOW_SENSOR_BIASED_LOW
43	15	26	100.0	CSTR_OUTLET_BLOCKAGE
44	13	4	0.01	⁵ CSTR_FIRE
45	19	30	0.1	PUMP_LOW_MOTOR_SPEED
46	7	24	10.0	¹ CSTR_INLET_BLOCKAGE
47	41	73	100.0	VALVE_2_STUCK_OPEN
48	70	69	0.1	⁴ FEED_TEMP_SENSOR FAILED HIGH
49	51	99	100.0	LEVEL_CONTROLLER_BIASED_HIGH
50	59	53	0.01	TEMP_CONTROLLER_SETPOINT_LOW
51	6	45	0.1	REACTOR_FEED_SOURCE_TEMP_LOW
52	43	35	0.1	CSTR_JACKET_FOULING
53	67	27	0.1	⁴ FEED_FLOW_SENSOR FAILED LOW
54	41	99	1.0	VALVE_2_STUCK_OPEN
55	69	49	1.0	FEED_FLOW_SENSOR_BIASED_LOW
56	99	92	0.01	³ CW_FLOW_SENSOR FAILED LOW
57	9	98	1.0	PUMP_LEAK_TO_ENVIRONMENT
58	12	5	0.1	CSTR_CATALYST_1_DEACTIVATION
59	43	99	10.0	CSTR_JACKET_FOULING
60	109	7	0.1	CW_PRESSURE_SENSOR_BIASED_LOW
61	11	60	0.01	CSTR_CATALYST_1_DEACTIVATION
62	6	27	0.01	REACTOR_FEED_TEMP_LOW
63	37	73	0.01	VALVE_2_BLOCKAGE
64	18	42	0.1	PUMP_LEAK_TO_ENVIRONMENT

TABLE 6-3: Randomly Generated Faults for Case Study (continued)

#	Mode	Extent (% of range)	Trajectory Time Constant	Description
65	99	89	0.1	⁴ CW_FLOW_SENSOR_FAILED_LOW
66	96	91	100.0	CONC_B_SENSOR_BIASED_HIGH
67	35	74	1.0	CSTR_JACKET_FEED_TEMP_HIGH
68	66	24	0.1	⁴ FEED_FLOW_SENSOR_FAILED_HIGH
69	57	8	100.0	⁵ TEMP_CONTROLLER_BIAS_LOW
70	109	0	0.1	⁶ CW_PRESSURE_SENSOR_BIAS_LOW
71	13	12	0.1	⁵ CSTR_FIRE
72	49	76	0.01	LEVEL_CONTROLLER_FAILED_LOW
73	92	7	10.0	⁵ CONC_A_SENSOR_BIASED_HIGH
74	78	94	100.0	³ REACTOR_TEMP_SENSOR_FAILED_HIGH
75	38	39	0.1	¹ VALVE_2_FIRE
76	57	44	10.0	TEMP_CONTROLLER_BIAS_LOW
77	20	96	1.0	² PUMP_OVERHEATING
78	32	81	1.0	EFFLUENT_SINK_PRESSURE_LOW
79	104	99	0.01	CW_TEMP_SENSOR_BIASED_HIGH
80	109	91	1.0	CW_PRESSURE_SENSOR_BIASED_LOW
81	3	5	1.0	REACTOR_FEED_PRESSURE_HIGH
82	73	0	0.1	⁶ FEED_TEMP_SENSOR_BIASED_LOW
83	95	75	0.1	CONC_B_SENSOR_FAILED_LOW
84	36	10	0.01	JACKET_FEED_SOURCE_TEMP_LOW
85	84	53	0.01	REACTOR_LEVEL_SENSOR_BIASED_HIGH
86	83	53	0.1	⁶ REACTOR_LEVEL_SENSOR_FAILED_LOW
87	29	37	10.0	VALVE_1_STUCK_OPEN
88	71	38	0.1	FEED_TEMP_SENSOR_FAILED_LOW
89	51	7	1.0	⁵ LEVEL_CONTROLLER_BIASED_LOW
90	38	48	100.0	¹ VALVE_2_FIRE
91	26	68	1.0	² VALVE_1_FIRE
92	101	67	1.0	CW_FLOW_SENSOR_BIASED_LOW
93	102	0	0.01	⁶ CONC_A_SENSOR_BIASED_LOW
94	85	63	0.1	REACTOR_LEVEL_SENSOR_BIASED_LOW
95	11	5	10.0	CSTR_CATALYST_1_DEACTIVATION
96	42	58	0.1	VALVE_2_STUCK_CLOSED

TABLE 6-3: Randomly Generated Faults for Case Study (continued)

#	Mode	Extent (% of range)	Trajectory Time Constant	Description
97	39	35	1.0	¹ VALVE_2_LOSS_OF_INSULATION
98	46	25	0.1	⁵ JACKET_EFFLUENT_PRESSURE_HIGH
99	98	27	0.1	⁴ CW_FLOW_SENSOR_FAILED_HIGH
100	27	11	0.1	² VALVE_1_LOSS_OF_INSULATION

Notes: ¹ Fault not simulated.

² Fault unobservable with given instrumentation.

³ In-range sensor failure.

⁴ Out-of-range sensor failure.

⁵ No events detected

⁶ Unusable

These randomly generated faults had the following characteristics:

- Six (6) faults could not be simulated by the simulation program,
- Five (5) faults were unobservable with the given instrumentation,
- Eight (8) faults had extents too small to produce events with the given monitor thresholds,
- Five (5) faults were omitted for other reasons (e.g. a zero extent or an extent that caused problems within the simulator),
- Seventy six (76) faults were used for the case study.

Of the seventy six (76) usable faults, thirty four (34) were sensor failures. These sensor failures were distributed as follows:

- Ten (10) were in-range sensor failure,
- Eleven (11) were out-of-range sensor failure,
- Thirteen (13) were sensor bias, and
- Fourteen (14) were failure or bias of controlled variable sensors.

All fault episodes lasted ninety (90) simulated minutes with the fault introduced after fifteen (15) simulated minutes of normal operation.

6.2. Case Study Results

A total of four hundred eighty six (486) events were detected and diagnosed. Thirty one (31) cases exhibited compensatory response of at least one measured variable. Sixteen (16) cases exhibited inverse response of at least one measured variable. There were twenty three (23) major monitor errors -- false alarms or failure to detect latent or expected events. These monitor errors produced second IM clusters. In almost every case, there was at least one minor monitor error -- out-of-order events or erroneous latent or expected events.

An accurate diagnosis lists the true fault as a member of the ranked fault candidate set. This definition is unchanged from previous sections. Because MIDAS ranks faults by the likelihood of a candidate being the true fault, however, resolution is computed differently from systems that do not rank faults. In MIDAS, resolution is defined as the number of fault candidates with likelihood rank greater than or equal to the rank of the true fault⁵. This definition is functionally identical to the previous definition. Resolution represents the maximum number of fault candidates that must be considered (i.e. tested) before identifying the true fault. In an unranked set, fault candidates can be examined in any order and the true fault could be the last examined. In a ranked set, fault candidates are examined in order of decreasing likelihood.

It is possible for several faults to have equal likelihood rankings. Rankings are based on the relative amount of evidence (events) supporting and opposing a candidate. Thus, in IM clusters with only a single event, all fault candidates must have equal ranking. In IM clusters with more than one event, candidates will separate into several ranking *tiers*. Within a given tier all candidates have equal ranking. For example, there may be a 1st tier containing several highest ranked candidates, a 2nd tier containing several lower ranked candidates, a 3rd tier with still lower ranked candidates, and so on.

In a perfect diagnosis, the true fault would be alone in the 1st tier (i.e. a resolution of 1). However, a perfect diagnosis is not always attainable. For example, several faults may be indistinguishable given the available instrumentation. In these cases, the best attainable diagnosis consists of the true fault and all indistinguishable faults in the 1st tier.

⁵ The true fault is included in this measure, so if there are no other faults ranked greater than or equal to the true fault, the resolution is one (1).

The basic results of the study, averaged over all cases, can be summarized as follows:

- MIDAS produced an accurate final diagnosis in 100% all cases,
- MIDAS ranked the true fault in the 1st tier in 82% of all cases,
- MIDAS ranked the true fault in the 2nd tier in 8% of all cases,
- MIDAS had an average final resolution of 4.2 (i.e. a performance rating of $\Phi = 0.97$).

These results are based on the final diagnosis of each case, after all events had been detected. A more detailed breakdown of the case study results is presented in the following sections.

6.2.1. Event Distribution

One case produced thirty five (35) event detections (a result of oscillatory behavior). Other cases produced only a single event. Figure 6-3 shows the distribution of events per case. The mean was 6.4 events per case; the median was 5 events per case.

A typical pattern was to detect several events (i.e. three to six) soon after the fault entered the process. Depending of the case, a second group of events might be detected approximately thirty (30) simulated minutes after the first group. In a small number of cases, a third group of events might be detected after another thirty (30) simulated minutes, near the end of the simulation.

The second and third groups would contain events marking compensatory or inverse response. Cases with small extent and gradual trajectory (i.e. $\tau \leq 0.1$) showed less inverse response than cases involving large, sudden process disturbances.

Cases with twelve (12) or more events represented such a small sample of cases that reliable conclusions cannot be drawn for MIDAS performance on these cases. In subsequent analysis, results are presented only up to eleven (11) events.

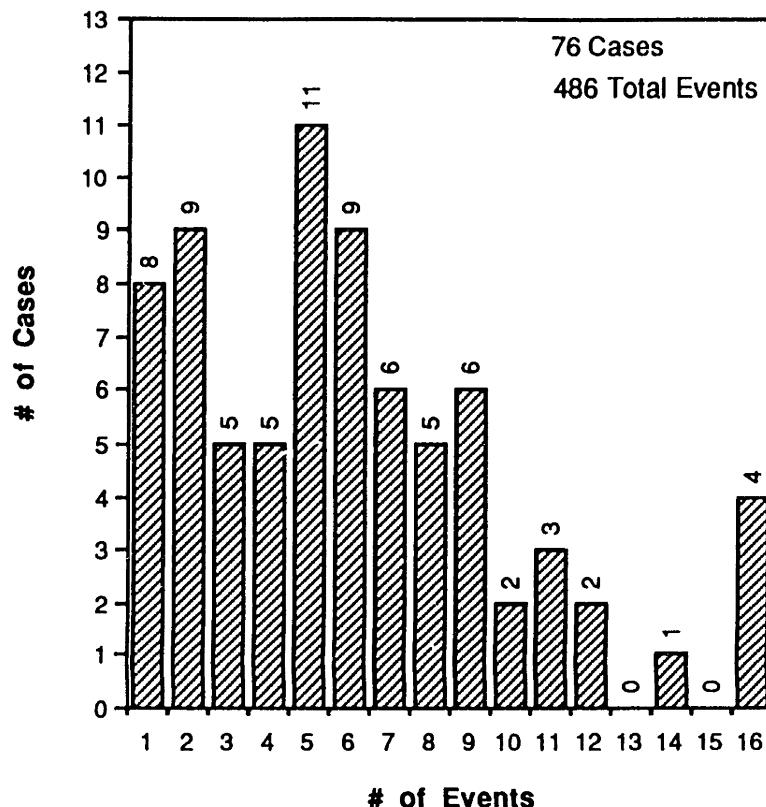


Figure 6-3: Distribution of Events

6.2.2. Evaluation Times

The speed of MIDAS is measured by determining how long it takes to complete the inference cycle for an event. The time elapsed per event showed great variation from case to case, however, when all cases are considered, consistent patterns arose. All cases were run on a PC compatible 386 computer with 20MHz clock and 10MB RAM. Evaluation times can be expected to vary on other platforms.

Figure 6-4 presents the distribution of event evaluation times. Only four hundred fifty one (451) events are considered in this analysis. Events representing the second or greater cycle of an oscillatory response are omitted for consistency. Adding these events would lower the overall average of one hundred five (105) seconds per event, since purely oscillatory events can be processed by MIDAS in under sixty (60) seconds. In one case, for example, product

concentration oscillating between the NORMAL and LOW state produced thirteen (13) events with an average elapsed time of fifty eight (58) seconds per event. It will be shown that these oscillatory events are not representative of other second or third group events.

Figure 6-5 illustrates the average time elapsed per event as a function of the number of events observed. Two distinct regions are identified. The first event detected takes an average of forty two (42) seconds to evaluate. Subsequent events takes, on average, twenty two (22) seconds longer to evaluate than the immediately preceding event. After detection of seven events, this pattern changes. The eighth event takes approximately twice as long to evaluate as the seventh, and subsequent evaluation times increase at the rate of sixty three (63) seconds per event. This increase is the result of a jump in cluster complexity as causal loops close in the IM. The problem of causal loops is discussed in section 6.3.

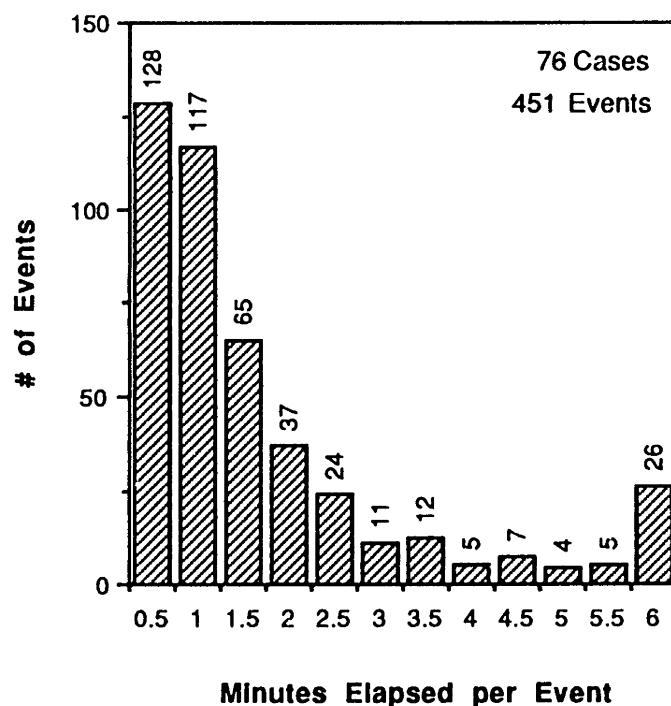


Figure 6-4: Evaluation Time Distribution

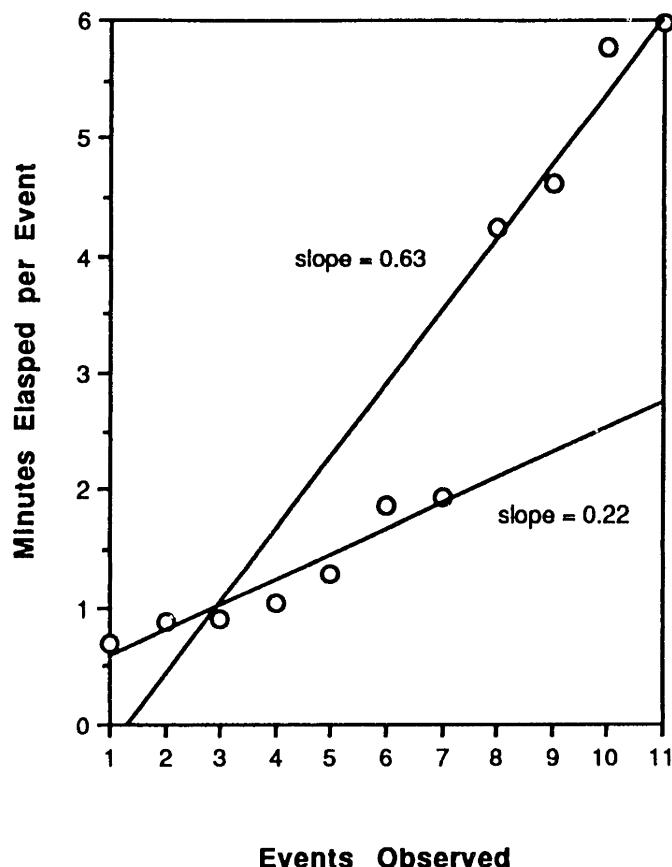


Figure 6-5: Time Elapsed per Event

6.2.3. Tier Performance

Table 6-4 presents the accuracy, tier of true fault, and performance of MIDAS as a function of the number of events observed. Figure 6-6 illustrates this data graphically by plotting the percent of cases in which the true fault was ranked in the 1st or 2nd tier.

These results show MIDAS to be an extremely effective diagnosis system, achieving near perfect accuracy and ranking the true fault in the 1st or 2nd tier in over ninety (90) percent of cases with seven or fewer events observed. To compare these results with other systems,

consider the performance (Φ) parameter. MIDAS can consistently eliminate from the diagnosis over 90% of all possible root causes.

TABLE 6-4: Overall Diagnostic Performance of MIDAS†

Events Observed	Accuracy (% of Cases)	Ranked in Top Tier (% of Cases)	Ranked in 1st or 2nd Tier (% of Cases)	Average Performance (Φ)
1	93	89	97	0.88
2	100	90	97	0.94
3	100	82	100	0.95
4	100	91	98	0.95
5	100	88	98	0.96
6	100	77	92	0.96
7	97	76	90	0.95
8	100	67	80	0.94
9	100	63	84	0.93
10	100	38	76	0.92
11	100	36	63	0.92

† Statistics compiled based on a sample of 76 cases.

While the percent of cases in which the true fault is ranked in the 1st or 2nd tier decreases for large numbers of observed events, the performance does not experience nearly so drastic a decline. For cases with large numbers of observed events, performance is a more reliable indicator, because as events are observed, more tiers are possible. It is possible for the true fault to slip to lower tiers while the resolution of the diagnosis remains constant. The process of tier formation is discussed further in section 6.3.2.

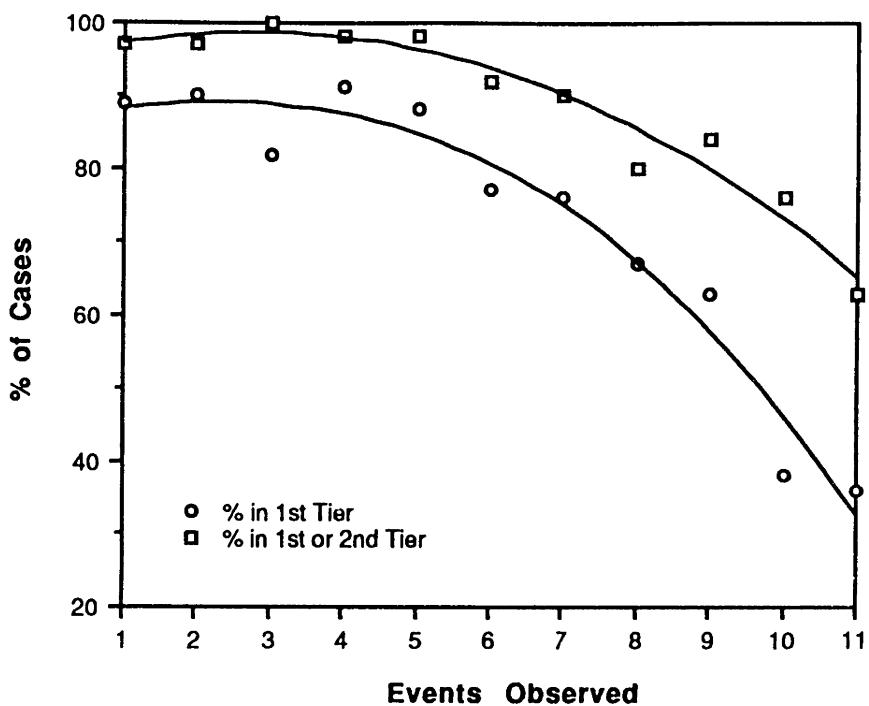


Figure 6-6: Tier of True Fault

Figure 6-7 illustrates the relationship between tier rankings by plotting the log of the average *normalized ranking* (NR)⁶ of the true fault when it is not in the 1st tier. From this figure, it can be seen that when the true fault is placed in the second tier, its ranking is typically within an order-of-magnitude of the top ranked candidate. Since all candidates are ranked using the CONDITIONAL-PROBABILITY ranking formula which creates ranking between zero and one, true faults ranked in the second tier have absolute rankings close to the top ranked candidate. The similarity between the rankings in the true fault in the first and second tiers means that MIDAS can still produce a near perfect diagnosis with the true fault in the second tier. In most results, the first and second tiers are lumped to generate performance statistics.

⁶ The normalized ranking is calculated by dividing the rank of the true fault by the rank of candidates in the 1st tier.

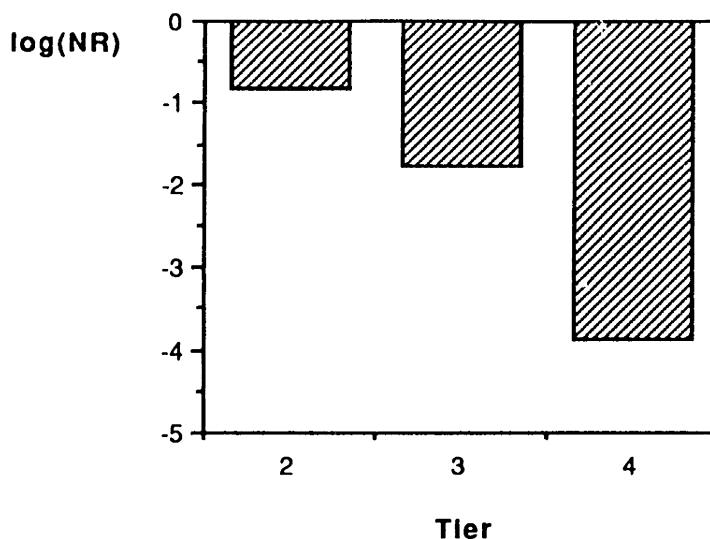


Figure 6-7: Average Normalized Rank of True Fault

6.2.4. Resolution

Figure 6-8 shows the resolution distribution for all cases. The mean resolution is four (4) faults, and the median resolution is two (2) faults. This discrepancy is the result of a small number of cases with poorer resolution. Not considered in figure 6-8 is the number of indistinguishable faults. In most cases with resolution of four or less, MIDAS achieved the maximum resolution possible within the limitations of on-line instrumentation.

Figure 6-9 illustrates resolution as a function of the number of events observed. Three distinct regions can be identified in this figure. The best resolution was produced by cases with four to seven (4-7) events. Average resolution within this range was nearly flat, indicating that after four (4) events little diagnostic information is supplied by subsequent event detections. Cases with three (3) or fewer events had less resolution. With so few events, MIDAS has too little information to produce better resolution. In this range, each additional event detection adds significantly to the diagnostic resolution as indicated by the steep decreasing slope of the resolution curve. Cases with eight (8) or more events produced a slight rise in average resolution. This decrease in resolution was the result of causal loops forming in the IM cluster and is discussed further in section 6.3.2.

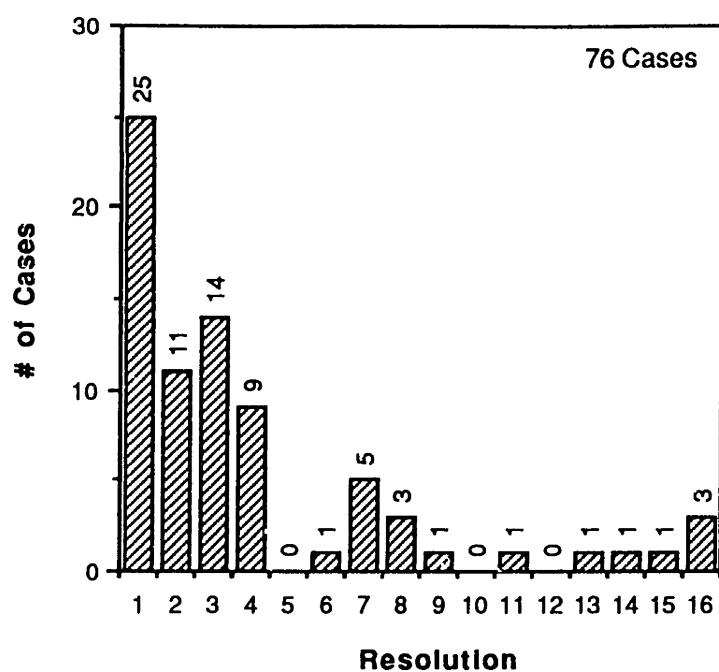


Figure 6-8: Distribution of Resolution

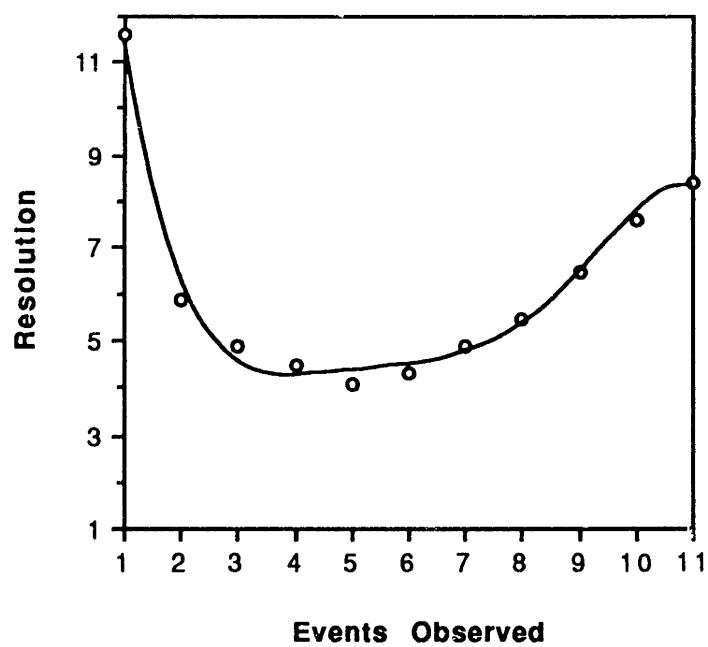


Figure 6-9: Average Resolution

6.2.5. Diagnosis Examples

In this section, statistics are temporarily abandoned in favor of detailed examination of several diagnosis examples. These examples were not part of the random case study, but rather, were chosen to demonstrate specific features of MIDAS, including the diagnosis of transients and multiple faults.

Example 1: Jacket Leak to the Reactor

In this example, a leak develops that allows water to enter the reactor from the cooling jacket. At its ultimate extent, the leak represents approximately 10% of the reactor feed flowrate. The fault begins fifteen (15) minutes into the simulation and takes almost five (5) minutes to reach its full extent. Table 6-5 shows the progression of the diagnosis.

TABLE 6-5: Diagnosis of Jacket Leak to Reactor

Event Description	Detection Time (minutes)	Tier of True Fault	Rank of True Fault	Resolution	Φ
Inventory Constraint High	25.5	1	0.143	7	0.94
Level Controller Signal High	26.0	1	0.442	2	0.99
Product Flowrate High	30.0	1	0.473	2	0.98
Cooling Water Flowrate Low	31.5		(creates second cluster)	0.94 ¹	
Reactor Level High	32.0	1	0.333	3	0.98
Cooling Water Flow Controller Low	32.0		(linked to second cluster)	0.94 ¹	
Product Concentration L Low	33.0	1	0.333	3	0.98
*Cooling Water Setpoint Low	35.0	1	0.333	3	0.98
Reactor Level Normal	52.5	1	0.487	2	0.99

* Event causes second IM cluster to coalesce with primary IM cluster.

¹ When two clusters exist, all candidates of the second cluster are considered to be ranked higher than the true fault in calculating performance (Φ).

- Event 1: The first event detected is a violation of the reactor inventory constraint. A deviation in the positive direction indicates excess fluid entering the reactor and would seemingly lead to immediate diagnosis of a jacket leak to the reactor, but a total of seven equally viable fault candidates are created. Because the constraint is calculated using three separate sensor measurements, the event is also consistent with a fixed failure or bias in any of these sensors.
- Event 2: The second event detected is a high level controller signal. This event is still consistent with a high bias of the reactor level sensor⁷, so this candidate remains in the 1st tier with jacket leak. the other five candidates are not supported by this event and drop to the 2nd tier with rankings of 0.023.
- Event 3: The third event is a high product flowrate, a downstream consequence of the high level controller signal. Both jacket leak and level sensor bias remain in the 1st tier, but the additional evidence provided by this event raises their rankings slightly while lowering the ranking of other candidates.
- Event 4: When a low cooling water flowrate is detected as the fourth event, it cannot be linked to the existing IM cluster. This is an out-of-order event that MIDAS fails to correctly interpret. The process model links the events Cooling Water Setpoint Low, Cooling Water Flow Controller Signal Low, and Cooling Water Flowrate Low in the sequence illustrated in figure 6-10. With this fault, direct cooling and dilution of the reactant tend to lower the reactor temperature. The temperature controller responds by lowering the cooling water setpoint which eventually leads to a low cooling water flowrate. Here, the reactor temperature appears normal because the control system is compensating effectively. The cooling water flowrate event is detected before cooling water setpoint and cooling water flow controller signal events because of suboptimal thresholds in the monitors. MIDAS was unable to anticipate the correct event order because the **Maximum Precursor Search Depth** was set to two (2) links for all case study faults. The closest abnormal event is more than three (3) links from Cooling Water Flowrate Low. MIDAS will have to rely on its correction features to coalesce the IM clusters when the intervening events are detected.

⁷ The event is not consistent with a high fixed failure of the reactor level sensor because a high level event has not been detected.

- Event 5: The fifth event is the detection of a high reactor level. With this event, the diagnosis marginally degrades as high fixed failure of the level sensor is pulled from the lower tiers and placed in the 1st tier with level sensor high bias and jacket leak. All 1st tier candidates have an absolute ranking of 0.333. Since the sum of all ranks is equal to unity, the ranks of the four (4) events in the lower tiers cannot add to more than 0.001.
- Event 6: The event sequence illustrated in figure 6-10 is being detected in reverse order. The new event -- Cooling Water Controller Signal Low -- is linked to Cooling Water Flowrate Low, but Cooling Water Setpoint Low fails the interrogation test (a monitor error) and two IM clusters remain.
- Event 7: The detection of low product concentration is a downstream consequence of previous events and does not change the primary diagnosis. Although the diagnosis does not improve, MIDAS performs an alarm filtering function by relating this event to the previously postulated malfunction.
- Event 8: Cooling Water Setpoint Low is detected at thirty five (35) minutes into the simulation. The second IM cluster is absorbed by the primary cluster five (5) minutes after its creation. MIDAS successfully corrects the earlier error. The primary diagnosis is unchanged by this procedure. Despite the fact that MIDAS temporarily postulated two malfunctions, the true fault was always in the 1st tier of the primary IM cluster.
- Event 9: When the reactor level returns to normal, fixed failure of the level sensor is removed from the 1st tier to produce a final diagnosis consisting of two top ranked fault candidates: jacket leak to reactor and level sensor high bias.

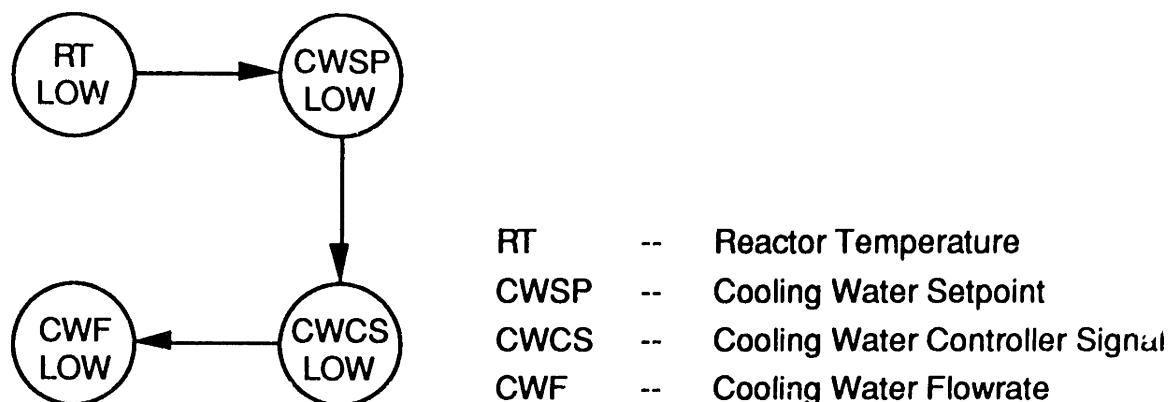


Figure 6-10: Causal Model of Temperature Control

Example 2: Reactor Feed Flowrate Low (transient)

In this example, the reactor feed flowrate begins dropping after fifteen (15) minutes, and reaches a low value of 0.218 cubic meters per minute (nominal value is 0.25 m³/minute) before returning to normal twenty five (25) minutes into the simulation. Table 6-6 shows the progression of the diagnosis.

Event 1: A low reactor feed flowrate is the first event detected. There are four (4) fault candidates: feed flowrate sensor low bias or failure, CSTR inlet blockage, or low upstream source pressure (the true fault).

Event 2: Detection of a low reactor level drops sensor bias and failure to the 2nd tier. With the available instrumentation, CSTR inlet blockage and low upstream pressure are indistinguishable; therefore, this represents the best possible diagnosis.

Event 3: Low product flowrate is a downstream consequence of low level, but an intervening undetected event -- level controller signal low -- separates the new event from the existing cluster. MIDAS interrogates the level controller signal and creates an EXPECTED-EVENT for Level Controller Signal Low to complete the link.

- Event 4: Reactor temperature low is a direct downstream consequence of previous events and does not change the diagnosis.
- Event 5: The fifth event is the realization of the EXPECTED-EVENT created to link the third event to the IM cluster. The new RE replaces the EE in the IM cluster but does not provide additional information.

TABLE 6-6: Diagnosis of Feed Flowrate Transient

Event Description	Detection Time (minutes)	Tier of True Fault	Rank of True Fault	Resolution	Φ
Feed Flowrate Low	19.5	1	0.250	4	0.97
Reactor Level Low	22.5	1	0.475	2	0.99
Product Flowrate Low	22.5	1	0.498	2	0.99
Reactor Temperature Low	23.0	1	0.500	2	0.99
Level Controller Signal Low	23.0	1	0.500	2	0.99
Product Concentration A Low	25.5	2	0.474	2	0.99
Cooling Water Flowrate Low	26.0	1	0.500	2	0.99
Cooling Water Setpoint Low	26.0	1	0.500	2	0.99
CW Flow Controller Signal Low	32.0	1	0.500	2	0.99
Reactor Temperature High	33.0	1	0.500	2	0.99
Product Concentration A Normal	36.0	1	0.500	2	0.99
Product Concentration A Low	43.5	1	0.500	2	0.99
Feed Flowrate Normal	43.5	1	0.500	2	0.99
Reactor Level High	45.0	1	0.500	2	0.99
Product Concentration A Normal	45.0	1	0.500	2	0.99
Cooling Water Flowrate Normal	49.5	1	0.500	2	0.99
Product Flowrate Normal	49.5	1	0.500	2	0.99
Cooling Water Setpoint Normal	59.0	1	0.500	2	0.99
CW Flow Controller Signal Normal	59.0	1	0.500	2	0.99
Reactor Temperature Normal	65.0	1	0.500	2	0.99
Level Controller Signal Normal	65.0	1	0.500	2	0.99
Reactor Level Normal	67.5	1	0.500	2	0.99

- Event 6: Detection of low reactant concentration temporarily knocks the true fault out of the first tier. Now CSTR inlet blockage is the sole member of the 1st tier ranked at 0.526. Low upstream source pressure is the sole member of the second tier ranked at 0.474. Although the true fault is in the 2nd tier, its absolute ranking is virtually identical to the 1st tier. Such temporary turnabouts can arise as evidence is evaluated in an increasingly complex IM cluster.
- Event 7: As in the jacket leak example, the first event detected in the temperature control loop is cooling water flowrate low. Unlike the previous example, however, MIDAS interrogates cooling water setpoint and cooling water controller signal and creates two EEs to correctly link the new event to the existing IM cluster.
- Event 8: Detection of cooling water setpoint is the realization of the EE created in the last inference cycle. Note the order of detection is different from the previous example, demonstrating the variability of event detection order.
- Event 9: Detection of cooling water controller signal is the realization of the second EE created as a result of event 7.
- Event 10: High reactor temperature is detected as the feed flowrate returns to 0.25 m³/minute and the cooling water flowrate is still low from the earlier disturbance. This is an example of an *apparent inverse response*⁸. The diagnosis is unaffected.
- Event 11: Reactor concentration of A returns to normal (compensatory response).
- Event 12: Reactor concentration of A is detected low (apparent inverse response).
- Event 13: Feed flowrate is certified as having returned to normal after an eighteen minute delay. The IM is reclassified as an ONGOING-TRANSIENT, to indicate that the source of the malfunction has disappeared but the disturbance is still propagating through the process. The diagnosis is unchanged.
- Event 14: Reactor level is detected high (apparent inverse response).

⁸ Apparent inverse response is inverse response resulting from poor monitor thresholds, badly tuned controllers, multiple faults, or transients. It is distinguished from true inverse response which is an inherent process behavior.

- Event 15: Reactor concentration of A returns to normal (compensatory response).
- Event 16: Cooling water flowrate returns to normal (compensatory response).
- Event 17: Product flowrate returns to normal (compensatory response).
- Event 18: Cooling water setpoint returns to normal (compensatory response).
- Event 19: Cooling water flow controller signal returns to normal (compensatory response).
- Event 20: Reactor temperature returns to normal (compensatory response).
- Event 21: Level controller signal returns to normal (compensatory response).

- Event 22: When reactor level returns to normal, all events have returned to normal and the IM is classified as a COMPLETED-TRANSIENT. The disturbance is over and the process has resumed normal operation. All events and fault candidates associated with the IM cluster are archived to a disk file for later review. MIDAS is ready for to diagnose any new disturbances entering the process. No reset is required.

Example 3: Pressure Surge Induced Failure of the Cooling Water Temperature Sensor

In this example, a transient surge in cooling water source pressure lasting ten (10) minutes destroys the temperature probe in the jacket feed pipe. This is a case of multiple faults, specifically an induced sensor failure. The diagnosis is presented in Table 6-7.

- Event 1: The pressure surge is detected immediately. The fault candidates are cooling water pressure sensor high bias, high failure, or high cooling water source pressure.

- Event 2: The signal from the cooling water temperature sensor goes out-of-range and the sensor is flagged as having failed. This event creates a second IM cluster, correctly diagnosing the two separate faults effecting the process. The two fault candidates of the second cluster are cooling water flow sensor failed high and cooling water flow sensor failed low⁹.

⁹ This is a minor semantic problem in the process model. The fault -- cooling water sensor failed -- is represented by two separate fault modes representing each possible failure direction.

Event 3: Detection of a low cooling water controller signal is confirmation that a real disturbance is underway and not a dual sensor failure. High cooling water source pressure becomes the sole candidate in the 1st tier of the first IM cluster with a ranking of 0.905.

TABLE 6-7: Diagnosis of Induced Sensor Failure (Cooling Water Temperature Sensor)

Event Description	Detection Time (minutes)	Tier of True Fault	Rank of True Fault	Resolution	Φ
Cooling Water Pressure High	16.5	1	0.333	3	0.98
*CW Temperature Sensor Failed	17.0	1	0.500	2	0.99
CW Flow Controller Signal Low	20.0	1	0.905	1	1.00
CW Pressure Sensor Failed	22.5	1	0.487	2	0.99
CW Pressure Sensor OK	40.5	1	0.487	2	0.99
Cooling Water Pressure Normal	46.5	1	0.487	2	0.99
CW Flow Controller Signal Normal	53.0	1	0.487	2	0.99

* Event creates a second IM cluster.

Note: Two clusters are represented in this table. The first cluster is associated with the transient high pressure malfunction. The second cluster is associated with the sensor failure. The second cluster consists of only the second event.

Event 4: The sudden pressure surge violates a range check and the monitor flags the sensor as having failed -- another example of monitor error. Pressure sensor failure is lifted to the 1st tier of IM cluster 1. The true fault is retained in the 1st tier. No events associated with the cooling water pressure sensor will be detected as long as the sensor is flagged as failed.

Event 5: With the pressure surge over, the monitor finds the pressure sensor OK.

Event 6: Cooling water pressure is verified as normal.

Event 7: The cooling water flow controller signal returns to normal. The first IM cluster is classified as a COMPLETED-TRANSIENT and archived. The second IM cluster is still a PERSISTENT failure and remains in the active memory awaiting correction.

6.3. Case Study Analysis

This section presents an analysis and evaluation of MIDAS. The case study indicates that MIDAS successfully solves the problems that spurred its creation: out-of-order events, multiple independent failures, and basic dynamics including control system compensation and transients. An outgrowth of this success is that a new set of problems have been identified. A major goal of this section is to outline the issues that need to be addressed by the next generation of diagnostic systems. Adapting MIDAS (or devising new methods) to solve these problems can form the basis for additional research projects in general diagnosis automation.

Although beyond the scope of the current case study, a comparison of MIDAS with human diagnosticians would provide an interesting relative performance statistic. To date, MIDAS has only been tested ad hoc against the diagnostic skills of its human creators. In these comparisons, MIDAS was consistently able to produce larger sets of viable candidates than its human competitors, and was often able to surprise its designers with unexpected (but correct) event interpretations. Because MIDAS lacks a mechanism to learn by example, however, it would repeat mistakes that a human diagnostician would learn to recognize.

6.3.1. On-Time Performance

A disappointing characteristic, systematically demonstrated by MIDAS, was the lengthy time required for event evaluation. An average evaluation time of one hundred five (105) seconds is simply too slow for many real-time applications. More disturbing is the apparent unbounded growth of the average evaluation time as IM clusters grow.

On average, evaluation time per event was linear within both defined regions. Within a single case, the time necessary to evaluate an event fluctuated considerably. The fastest event took only eight (8) seconds, but it was not uncommon for events to require six hundred (600) seconds to evaluate.

The only solution to this problem is take measures to increase the speed of MIDAS. There are six (6) possible solutions, listed below in rough order of increasing effort:

- Run MIDAS on a faster platform,
- Run the Monitors outside MIDAS (as a parallel process),
- Code MIDAS more efficiently (retaining the same fundamental steps),
- Revise MIDAS algorithms for greater efficiency,
- Remove MIDAS from GoldWorks, ..
- Code MIDAS is a raster programming language.

To create a system with acceptable real-time performance, a combination of two or more of these actions will be necessary.

More efficient code might include a different algorithm for evidence evaluation. Tests of MIDAS indicate that over 80% of the computational resources required by the inference cycle are used to evaluate evidence to rank candidates. If a single source of the poor on-time performance of MIDAS is to be identified, it is the routine EVALUATE-EVIDENCE. If this routine were not run in every cycle, the inference cycle would require a nearly constant twenty (20) seconds per event.

EVALUATE-EVIDENCE is slow because it does a complete, depth-first search of all causal paths within an IM cluster starting at each SOURCE event. When causal loops result in multiple source events and complex IM clusters, on-time performance suffers. Possible improvements to the evaluation algorithm are discussed in section 7.3.

6.3.2. Diagnostic Performance

Diagnostic resolution was near the theoretical maximum for most cases. In over 90% of cases, MIDAS listed the true fault as either its first or second choice, and even when the true fault was not in the top two tiers, resolution was still good.

Comparing the results in table 6-5 with those in table A3-4, it can be seen that MIDAS produces a more stable and higher resolution diagnosis than SLD for the jacket leak to reactor fault. Although this is only one example, the characteristics of greater stability and higher

resolution are expected to be exhibited consistently by MIDAS. A complete comparison of MIDAS with other diagnostic systems, such as DIEX and SLD, awaits further research.

MIDAS has two advantages over single model methods such as DIEX and SLD:

- MIDAS ranks candidates by relative likelihood, and
- MIDAS can use sequences instead of snapshots.

The first trait results in a tremendous increase in observed resolution. Assuming MIDAS and SLD create candidate sets of equal size, for example, MIDAS would have to place the true fault in the lowest tier for resolution to equal that of the unranked SLD candidate set. The second trait allows MIDAS to interpret entire new classes of process behaviors, such as transients.

Another MIDAS strength is its robustness to event detection order and detection errors. Each of the three examples discussed in section 6.2.5 include monitor errors or out-of-order events. In the first example, an out-of-order event caused temporary formation of a second IM cluster. MIDAS corrected this mistake when the missing events were detected without ever sacrificing the accuracy or resolution of the primary cluster diagnosis. In the second example, the same problem is observed, but MIDAS was able to anticipate the correct event order and create expected events. Moreover, when apparent inverse response was detected in the second example, MIDAS was able to maintain the resolution of the diagnosis. In the third example, a monitor falsely flagged a sensor as having temporarily failed. In all examples, accuracy and resolution were maintained despite these problems. A survey of the random case study faults indicates that almost every fault episode contained at least one example of a monitor error or an out-of-order event. Because of robustness to these situations, MIDAS is an extremely stable diagnostic system.

One result of the case study that requires some analysis is the apparent drop in the number of true faults ranked in the 1st or 2nd tier when more than seven (7) events have been observed, shown in table 6-4 and figure 6-6. In general, the number of tiers must be less than or equal to the number of events observed (including expected and latent events) as shown in equation 6-7.

$$\{\text{number of tiers}\} \leq \{\text{number of events in the IM cluster}\} \quad (6-7)$$

A hypothetical example of tier formation is illustrated in figure 6-11. After observing N events, there are seven (7) fault candidates in the 1st tier. Assuming the true fault is represented by HRC-4, the resolution of the diagnosis is seven (7). At $N+1$ events, the 1st tier splits and four candidates, including the true fault, are dropped to the 2nd tier. The resolution of the diagnosis, however, remains constant at seven (7) faults. At $N+2$ events, both the 1st and 2nd tier split to form four tiers. The true fault drops to the 3rd tier, but the resolution of the diagnosis improves to six (6) faults. This tier splitting phenomenon partially explains the results of Table 6-4 and figure 6-6. As more events are observed, there are simply more tiers in which the true fault can reside with equal resolution.

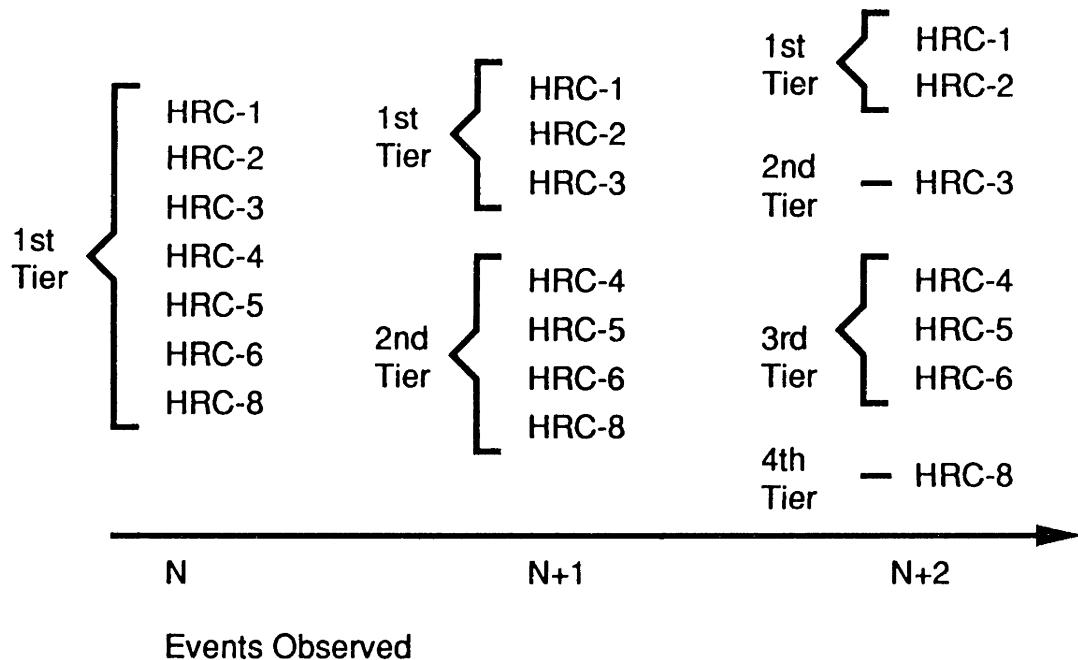


Figure 6-11: Tier Formation

Despite its superior overall performance, in approximately 5% of cases, MIDAS produced poor fault resolution. Accuracy was never lost, but the true fault was dropped far below the top ranked candidate. In one case, for example, the true fault ended the episode in the 16th

tier¹⁰. The cases in which MIDAS produced poor resolution were sudden catalyst deactivation (slow catalyst deactivation was diagnosed better), CSTR jacket fouling, and temperature controller bias.

These cases were similar in that they involved a large number of events (more than eight). It was also observed that MIDAS would achieve good resolution (i.e. rank the true fault in the 1st or 2nd tier) early in the fault episode and would lose resolution only after five or more events had been observed. The apparent rise in average resolution after eight events -- illustrated in figure 6-9 -- marks the influence of these cases on overall performance.

Three sources of this poor performance have been identified:

High Model Connectivity

All faults producing poor resolution had reactor temperature (high or low), cooling water setpoint (high or low), or other highly connected events as primary symptoms. Moreover, these faults did not have other primary symptoms that were not highly connected.

There are two effects at work in these cases:

- 1) Reactor temperature and cooling water setpoint events are the primary symptoms of a large number of faults, so a large candidate set is produced when these events are classified as the source of an IM cluster, and
- 2) The high connectivity of the events reduces the ability to discriminate source events from consequence events.

Both effects make diagnosis of these faults difficult. Large candidate sets mean a greater probability of the true fault dropping to a lower tier as events are detected. Higher connectivity means a complex IM cluster with no unique source event.

Apparent Inverse and Oscillatory Response

Inverse response modeled in the ESDG and event models is meant to represent true inverse response. Control systems are designed to produce compensatory response and in the ESDG

¹⁰ It should be noted that even in the 16th tier, the resolution was still superior to an unranked candidate set.

model are not modeled as sources of inverse response. Apparent inverse and oscillatory response can be detected, however, whenever a PI controller compensates a disturbance.

The critical factors leading to apparent inverse or oscillatory response are the speed of onset of the disturbance, the magnitude of the disturbance, the tuning parameters of the controller, and the detection thresholds of the monitors. Apparent inverse and oscillatory response will occur only when a controller exhibits underdamped behavior.

In figure 6-12, underdamped compensation of a disturbance, starting at time = 10, is shown. Included are detection thresholds indicating when the controlled variable is high or low. A monitor observing this variable will indicate an event whenever the variable value crosses a detection threshold. The situation illustrated in figure 6-12 results in two events as the value crosses the upper detection thresholds and then crosses back to the normal operating regime. This is the expected qualitative behavior of a control system.

If the disturbance were slightly larger, the detection thresholds slightly tighter, or the controller tuned differently, the situation illustrated in figure 6-13 can arise. Although the fundamental system behavior has not changed, four events are detected as the variable first appears high, returns to normal, appears low, and returns to normal again -- an example of apparent inverse response. Figure 6-14 shows how this situation can quickly turn into apparent oscillatory response if thresholds are tightened slightly more.

Since faults create disturbances of differing magnitude, monitors thresholds are established that produce the optimal tradeoff between sensitivity to small magnitude disturbances and insensitivity to false alarms and process dynamics such as underdamped behavior. The tradeoff virtually assures that apparent inverse response will occasionally be observed. Moreover, the underdamped behavior of controlled variables can produce similar behavior in other process variables.

Apparent inverse response can also result in cases involving multiple faults or transients. In these cases, the process attains a new steady-state in response to the initial disturbance and is disturbed again when the original fault disappears or another fault enters the process.

Because apparent inverse response is not included in the underlying models of process behavior, MIDAS is unpredictable in its treatment. Sometimes it handles the situation nicely, as in the feed flowrate transient discussed in section 6.2.5. Sometimes it becomes confused

and loses resolution. In general, MIDAS will handle apparent inverse response well if a single strong source event exists for the IM cluster and will handle it poorly if no strong source can be identified. The problem of apparent inverse and oscillatory response must be treated at the detection stage. A possible solution to this problem is discussed in section 7.6.

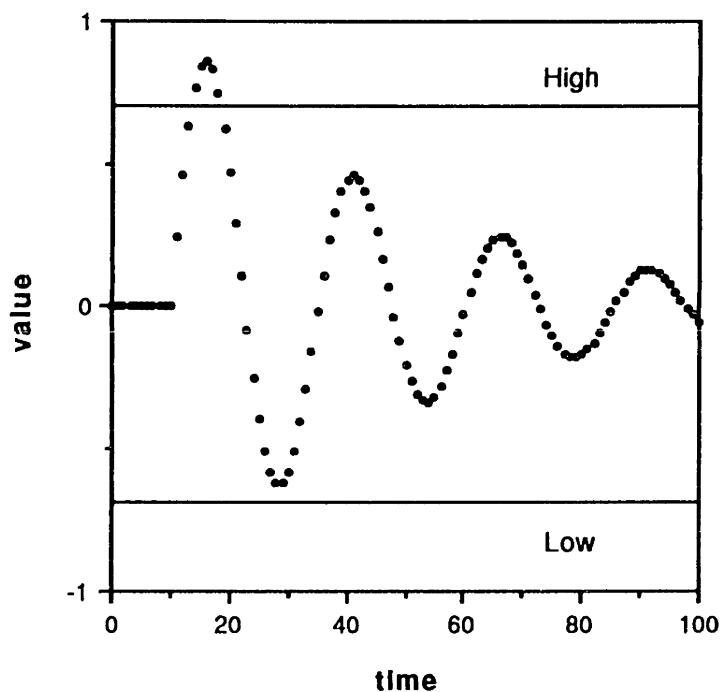


Figure 6-12: Compensatory Response

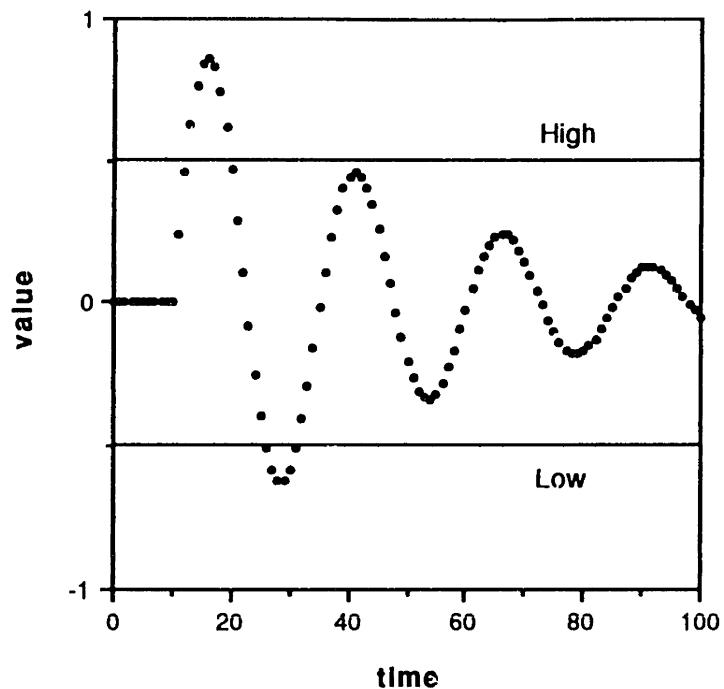


Figure 6-13: Apparent Inverse Response

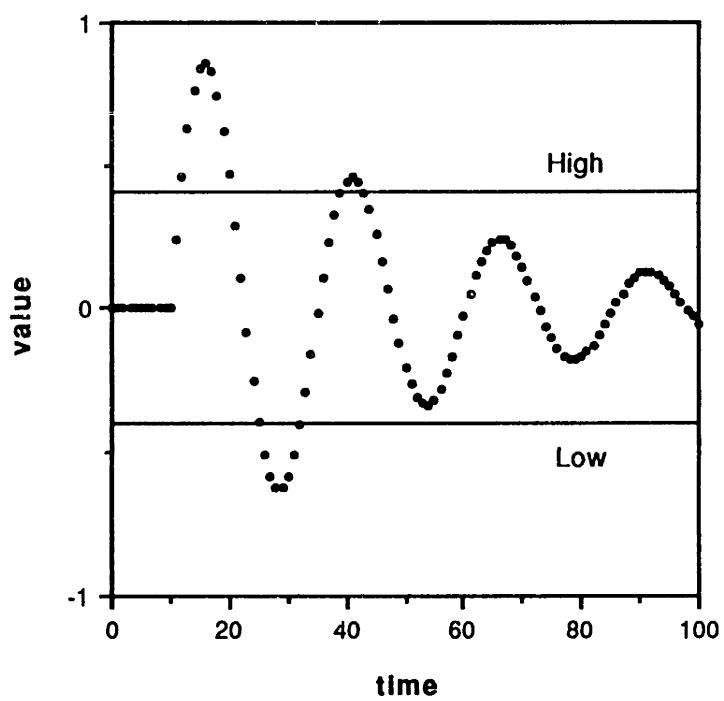


Figure 6-14: Apparent Oscillatory Response

Causal Loops

Identifying a single strong source event is impossible when a causal loop exists in an IM cluster. The Jacketed CSTR Process contains such a causal loop related to the temperature control system, as shown in figure 6-15.

This loop contains eight events and can only be active if apparent inverse response of the temperature controller is detected. It is no coincidence that in the cases MIDAS produced poor resolution, the diagnosis degraded after approximately eight events, involved apparent inverse response, and reactor temperature and cooling water setpoint were the primary symptoms. The root of the problem in those cases is this causal loop.

Since all events in a causal loop are considered as possible sources, the highest ranked candidate will be the one that has the most primary symptoms among the possible source events. At this point, ambiguities in the process model can produce erroneous results.

For example, consider the fault CSTR inlet blockage. Qualitatively, this fault can reduce reactor temperature by cutting the flow of cool feed or increase the reactor temperature by increasing the residence time. The ESDG model did not have the quantitative information necessary to resolve this ambiguity, and therefore, carried it into the process model by making both reactor temperature high and reactor temperature low primary symptoms of CSTR inlet blockage. In fact, CSTR inlet blockage is a local cause of every event in the causal loop shown in figure 6-15. Because jacket fouling can cause only a decrease in reactor temperature and has only four primary symptoms, CSTR inlet blockage will always be ranked greater than jacket fouling when this loop is detected.

In the final analysis, the faults which MIDAS had the most difficulty diagnosing were simply difficult to diagnose -- by man or machine. Having been fooled by a fault once, however, a human will learn to watch for apparent inverse and oscillatory response, causal loops, and high connectivity. MIDAS will not learn automatically and must be taught to recognize these same situations.

The avenues available make MIDAS robust to these cases are improved event detection schemes, changes in the process model, or modification to the diagnostic reasoning algorithm. For example, the causal loop in figure 6-15 could have been averted by widening the thresholds of the appropriate monitors. Alternately, the ambiguities of the model, such as the

effect of CSTR inlet blockage on reactor temperature, could have been resolved during model creation. A third solution is to model all controlled variables as possible inverse response variables. The performance of MIDAS on all these difficult cases could have been improved significantly by tuning either the monitors and model.

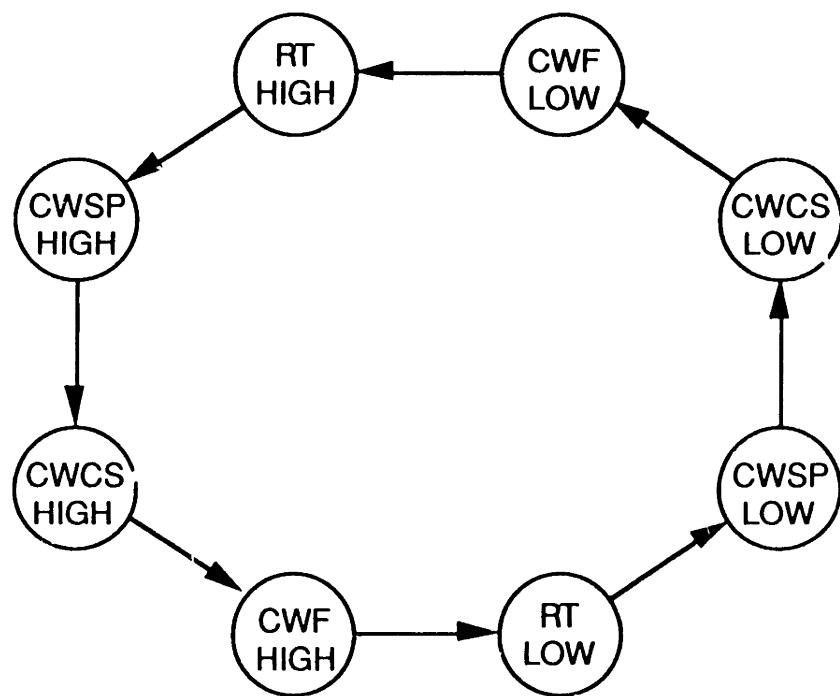


Figure 6-15: Temperature Control Causal Loop

7.0. Ideas and Discussion

This section summarizes the major concepts responsible for MIDAS and proposes avenues for improving MIDAS in the future. The objective is to make recommendations for continued research on MIDAS and automated diagnostic systems in general.

The key concepts responsible for the successes of MIDAS are integration, generality, and abstraction. Integration assumes that using knowledge from two or more sources is preferable to using knowledge from a single source. In MIDAS, integration refers to the consolidation of multiple models of process behavior, including causal digraphs, constraint equations, and system models. Experience bears out faith in this assumption. Knowledge source integration can provide substantially greater gains in diagnostic resolution than incremental improvements in any single knowledge source.

Generality results from the fact that the basic diagnostic and modeling techniques used in MIDAS are not specific to any particular system or class of systems. Indeed, the high level of modeling flexibility allowed in MIDAS can create difficult choices between equivalent process models. Diagnostic systems that lack generality, such as shallow expert systems, must be completely rebuilt for every new application or process modification.

The enabling force behind both the integration of knowledge sources and general diagnostic algorithms is the concept of abstraction. When viewed at an appropriate level of abstraction, previously disparate models and diagnosis strategies can be combined. The search for abstraction is ultimately the key to solving the diagnosis problem and other problems in artificial intelligence.

7.1. Scaling MIDAS to Real Plants

If MIDAS were used in a real process application, it can be assumed that the process model required will be larger than those used in development and testing. Therefore, some comments on the expected scaling properties of the MIDAS reasoning algorithm are warranted.

So long as memory limitations are not violated, MIDAS should experience only minor reductions in search performance with large models. MIDAS is always focused on a small

portion of the process model within several links of the latest event. The overall size of the model effects only the lowest level database searches. Bounding of the search algorithm can be controlled using the **Maximum Precursor Search Depth** and **Maximum Successor Search Depth** settings.

The critical factor in determining search performance is not the overall size of the process model but the connectivity of the model. The model connectivity is the number of links initiating and terminating at the average node. Connectivity determines the breadth component of the search. Since the depth component is bounded, breadth is the only component free to grow.

Connectivity is a weak function of overall model size, being primarily dependent on other factors, including the degree of process interaction, the types of units involved, and modeling style. A small, highly interactive process consisting of units exhibiting integrating behavior (e.g. tanks or integrating controllers) can have a much higher connectivity than a large process with little interaction and few controllers. The modeling style can also have a strong influence on connectivity. The style incorporated in the Model Translator Component of MIDAS produces process models with a moderate average connectivity that may reduce performance in large models. Reducing the average connectivity of models created by the Model Translator is a possible subject for continued research.

As shown in section 6.0, on-time performance will deteriorate as IM clusters grow. Because large models can give rise to large IM clusters, this may be a problem in extended malfunction episodes. To some degree, MIDAS can control this problem by adjusting the **Level of Evidence Evaluation** discussed in section 5.0. Other possible solutions are explored in section 7.3.

As a matter of modeling efficiency, it is desirable to decompose large models into smaller models with minimal interaction. For extremely large models, the results of such a decomposition would be several smaller process models, each diagnosed by a separate MIDAS or swapped in and out of memory as required.

7.2. Implementation Alternatives

There is a motivation for removing MIDAS from GoldWorks. Being a complete expert system development environment, GoldWorks contains many extraneous features. Rules, assertions, attempts, and sponsors are all unnecessary for MIDAS, yet these features slow overall performance. On the fastest PC platforms available, MIDAS is unacceptably slow for most real-time applications. While GoldWorks is not entirely responsible for the speed problem, a significant computational effort is currently wasted on redundant or superfluous activities. Using MIDAS, with GoldWorks, on a more powerful platform may improve performance, but remains a suboptimal solution.

The majority of MIDAS is implemented in Common LISP and could be ported directly to more powerful computer running LISP. A superior solution would be to recode MIDAS in an inherently faster programming language. Recoding MIDAS would provide the opportunity to consolidate and boost the efficiency of many functions and routines that were coded more for ease of modification and debugging than for elegance.

Separating MIDAS from GoldWorks will not be an entirely painless task. MIDAS makes extensive use of the GoldWorks Frame Database and Screen Interface. If another OOP environment is used, GoldWorks database calls can be replaced with an appropriate substitute function; otherwise, a more elaborate data storage and retrieval structure must be developed.

The MIDAS User Interface relies heavily on predefined objects supplied with GoldWorks and would be lost if MIDAS were separated from GoldWorks. At no time, however, was the User Interface intended to be the final operator interface of a practical diagnostic system. The User Interface is designed solely as a developer's tool, and a new interface can be considered a necessary step in the evolution of the system. Interface development should not be viewed as a strong deterrent from severing MIDAS from the GoldWorks environment. This research has not dealt with the issues surrounding the design of an operator interface.

Portions of MIDAS have already been transferred to G2¹ implemented on a VAXStation 3500. This new system -- dubbed EDDIE (Event Driven Diagnostic Inference Engine) -- implements the MIDAS diagnostic reasoning algorithm as a set of generic rules, essentially proving the contention of section 2.3.5 that modern rule-based systems can be used as an

¹ Gensym Corporation, Cambridge, MA.

alternative to conventional programming languages. The advantages of EDDIE and G2 are the real-time operating system, data acquisition and networking facilities, and superior speed and graphic capabilities. A real-time operating system, such as that provided by G2, will be necessary for significant improvements to be made in the MIDAS reasoning algorithm.

7.3. Evidence Evaluation

In section 6.3, the evidence evaluation phase of the inference cycle was implicated as a major source of poor on-time performance. This problem arises because evaluation in its current implementation performs a complete network search of an IM cluster at each inference cycle. The time required for evaluation (t_{eval}) can be approximated by equation 7-1:

$$t_{eval} = S \cdot C^K \cdot t_{link} \quad (7-1)$$

where S is the number of source events in the cluster, C is the average node connectivity, t_{link} is the time required to evaluate one causal link², and K is the average length of a causal path in the cluster. As a cluster grows, K increases subject to the constraint in equation 7-2:

$$0 \leq K \leq (N-1) \quad (7-2)$$

where N is the number of events in the cluster. From these equations, the origin of the precipitous drop in on-time performance for large IM clusters is apparent.

MIDAS combats this problem with the **Level of Evidence Evaluation** setting. This parameter can set an upper bound on K , as shown in Table 7-1. This solution is not optimal since much of the evidence of the causal network can be lost when an artificial bound is placed on K . The maximum number of evidence tiers is reduced and faults with multiple primary symptoms may be ranked unjustifiably high.

² The time required per link (t_{link}) is a function of the number of fault candidates associated with the cluster and will increase as the candidate set grows.

TABLE 7-1: Bounding of Evidence Evaluation

Level of Evidence Evaluation	Maximum Evaluation Depth (κ)
FULL	10
APPROXIMATE	1
DOWN&DIRTY	0
VARIABLE	1 - 10*

* Adjusts automatically based on evaluation time of last inference cycle.

A superior solution is to develop an alternate algorithm for evidence evaluation. It should be possible to develop a procedure that updates evidence incrementally, adding a new event to existing evidence without re-evaluating the entire network in every inference cycle. It might still be necessary to evaluate the entire network when a latent source or a causal loop develops, but it is wasteful of computer resources to evaluate the full network when CONSEQUENCE events are detected.

7.4. Augmented Causal Models

Some of the most promising avenues for advancement of the basic MIDAS program are related to improvements in the event models used for modeling process behaviors. Two types of improvement are possible: adding modeling paradigms or adding information to the existing paradigms. In the following section several ideas for improving causal models are presented.

7.4.1. Full Event Model Paradigm

The event models used by the current MIDAS system do not fulfill the potential of the event model paradigm. In the current models, only three types of event are used extensively:

- qualitative state changes (variables or constraint residuals),
- gross sensor failures, and
- results of off-line tests.

Types of events that could be added to existing models, include:

- qualitative state changes (systems),
- changes in trends, and
- operator actions.

The prospects for adding each of these event types to MIDAS are discussed below:

Trends

Trends can be modeled analogously to variables. Most of the structure necessary to handle trend events already exists within MIDAS. Qualitative trend modeling, however, has not been studied as extensively as qualitative state modeling. Cheung and Stephanopoulos (1989a) (1989b) have begun to redress this imbalance. When trend modeling is better understood, trend events can compliment existing state events.

Operator Actions

Operator actions do not contain diagnostic information, but may change aspects of the process that will require corresponding changes in the process model. Process changeovers, switching controllers to manual, taking units (including sensors) off-line for maintenance, and initiation of startup or shutdown procedures are all actions that fundamentally change certain aspects of the process model. Rather than develop separate process models for each situation, one could consider nesting several different process models in one process model superstructure using the existing activatable link mechanism. Detection of an operator action event could be used as a trigger to activate the links of one model and deactivate the links of another. This would allow MIDAS to operate beyond the time at which the first operator action is taken.

Robustness to operator actions should be considered a high priority for a practical diagnostic system. In many situations, it will not be possible to wait passively for several events to occur before taking some action. Yet, an operator may be reluctant to initiate an action if it means losing all his diagnostic aids.

Systems

The system models developed for the SLD methodology provide a starting point for the development of dynamic system models in event graph format. Functional decomposition and the intersystem dependency relationships are powerful concepts, but the diagnostic procedures and rules of SLD are not dynamic and must be discarded.

The key to developing dynamic system models lies in exploring system trajectories in addition to system states. Normal trajectories will be circles or spirals inside the FUNCTIONAL region of δ_c/δ_m space. A typical normal trajectory is illustrated in figure 7-1A. Note that in this figure, δ_c and δ_m are not absolute values and can be positive or negative.

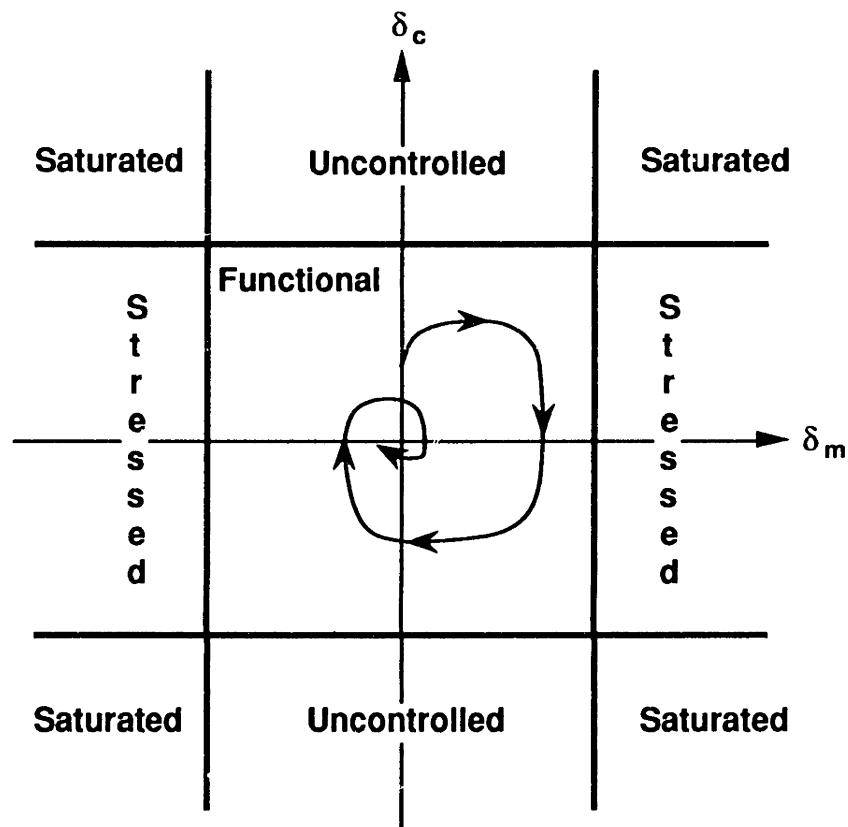


Figure 7-1A: Normal System Trajectory

For the system depicted in the figure, clockwise trajectories indicate a working control system; counterclockwise trajectories indicate a failed control system. All systems will have a characteristic trajectory rotation direction that depends on whether δ_m increases or decreases in response to an increase in δ_c . Trajectories with no rotation or with rotation counter the characteristic direction are symptoms of control failure. The distance from the trajectory to the origin is a measure of the stress on the system.

Using a combination of system states and trajectories, early erroneous states can be recognized. For example, the lack of rotation in trajectory 1, shown in figure 7-1B, indicates a controller failure or a stuck valve. The clockwise rotation of trajectory 2, however, indicates an operating control system despite the UNCONTROLLED state. Trajectory 2 will eventually end in a STRESSED or SATURATED state. Trajectory 3 has counterclockwise rotation. Because clockwise rotation is characteristic for this example, trajectory 3 is indicative of a controller failure.

Because multiple system state changes can be expected for most fault episodes, intrasystem links connecting various system events must be developed. The intravariable links used to model compensatory and inverse variable response can be used as a guide for this task.

7.4.2. Semi-Quantitative Causal Models

One method of enhancing the causal models used in MIDAS is to add certain quantitative information to the existing models. The most useful information would be the *gain* and *delay* associated with disturbance propagation along a causal link.

Faults may influence the gain or delay associated with a link. For example, the delay between two flow or pressure events will be effected by faults that increase or decrease flow resistance. In light of this problem, gains and delays should be expressed in order-of-magnitude form. Using orders-of-magnitude rather than precise numerical values will also reduce the effort required to develop augmented models. Mavrovouniotis and Stephanopoulos (1988) presents techniques for reasoning using order-of-magnitude values.

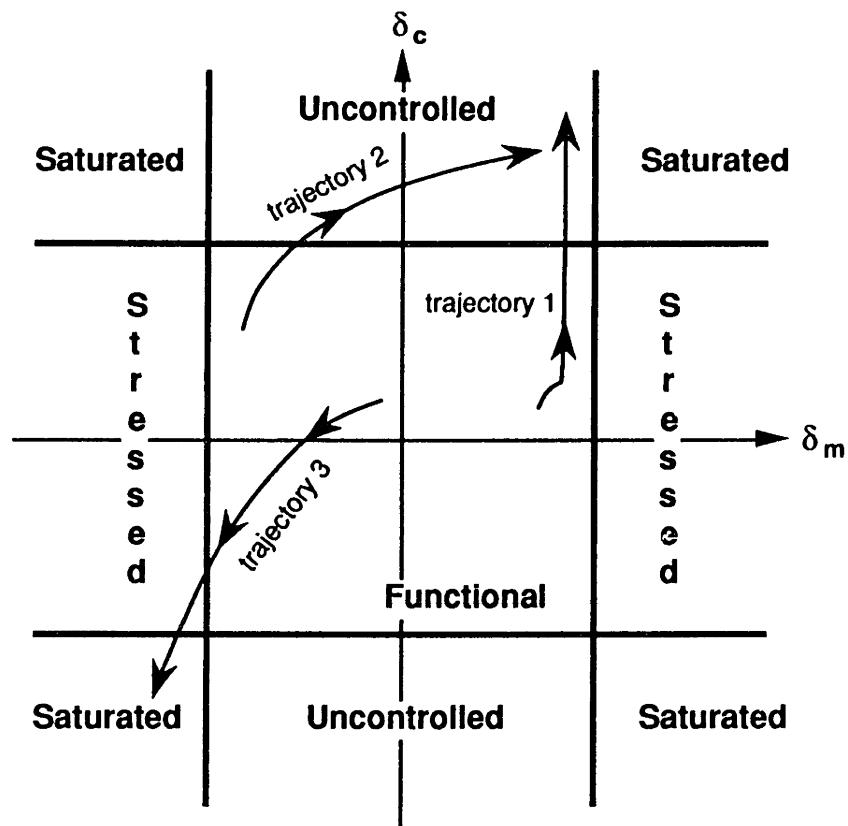


Figure 7-1B: Disturbed System Trajectories

The major use of gain and delay information is reasoning based on non-detection of events. In the current version of MIDAS, causal links can be considered as "can cause" links (CCLs). In effect, a CCL implies that the uplink event can cause the downlink events, but the downlink event is not guaranteed to occur or be detected. Therefore, failure to detect a downstream event cannot be used in diagnostic reasoning.

Using gain and delay information in conjunction with known monitor detection thresholds, a "will cause" link (WCL) can be defined. The WCL would have an attached time period indicating when the downlink event will be detected. Derivation of this "waiting period" is illustrated in figure 7-2.

For this example, the gain along the link is 0.65, the delay is 10 seconds, and the event detection thresholds are identical. Here, variable 1 is associated with the uplink event and

variable 2 is associated with the downlink event. Because of uncertainty in the actual detection threshold, indicated by the banded detection range, the uplink event will be detected fourteen (14) to twenty (20) seconds after the start of the disturbance. The disturbance starts effecting the downstream variable after a ten (10) second delay. Disturbance growth of variable 2 is slower than variable 1 because of the attenuation of the link. Detection of the downlink event will occur thirty two (32) to forty one (41) seconds after the start of the disturbance. To be conservative, the waiting period on a WCL must assume that the uplink event is detected at the earliest possible time and the downlink event detected at the latest possible time. The waiting period for this example is twenty seven (27) seconds. It should be noted that this example assumes the variables have a constant, known growth curve. In general, this may not be true.

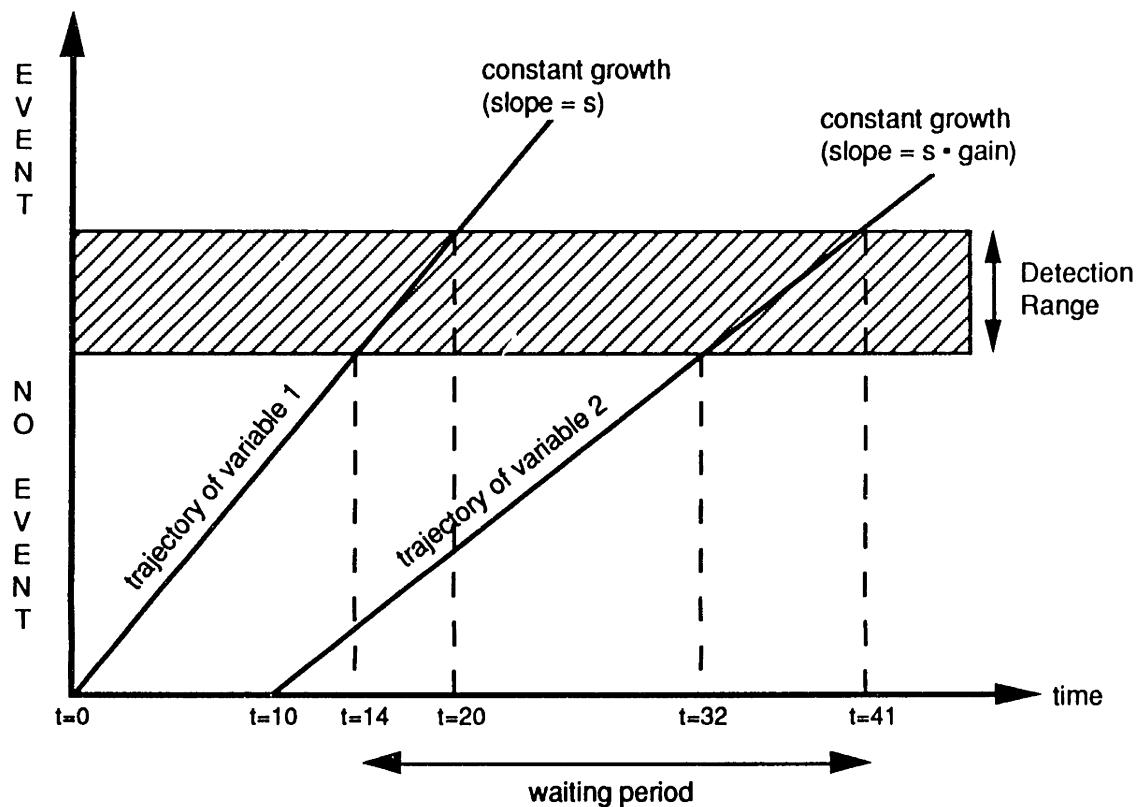


Figure 7-2: Derivation of Waiting Period for a WCL

If the time period expires without observing the downlink event, disturbance propagation along the link must be blocked, indicating the true fault must be a member of the fault set listed in the NOT condition attached to the link³. This type of reasoning would provide MIDAS with a potent means of confirming hypotheses.

The cost associated with adding this type of reasoning to the existing algorithm is the increased modeling effort required to develop link gains and delays. The use of order-of-magnitude values will help reduce this effort. In cases where the gain or delay cannot be determined, a long default delay can be used to convert a WCL to a CCL. A real-time operating system is required for implementation of WCLs.

7.4.3. Induced Failure Models

Another possible enhancement of the basic event model is the addition of links to represent modes of induced failure. The concept is illustrated in figure 7-3.

Figure 7-3 depicts a portion of an event model consisting of five (5) potential root causes and three (3) potential events. PRCs are connected to PEs by LCLs. PEs are connected by PSLs, and PRCs are connected by *induced failure links* (IFLs). In this example, an IFL indicates that the fault represented by PRC-2 can induce the fault represented by PRC-3.

IFLs may provide a mechanism for capturing the confusing behaviors produced when multiple failures interact by postulating a primary failure and one or more induced failures. In figure 7-3, for example, observation of PE-2 and PE-3 can be linked to a previous observation of PE-1 by postulating PRC-2 as a primary failure and PRC-3 as an induced failure. Without the IFL, detection of PE-2 would create a second IM cluster (correctly diagnosing two failures), but detection of PE-3 would be opposing evidence for PRC-2.

To complete the analogy to existing links, IFLs could be activatable and have attached conditions. The link in figure 7-3, for example, might be active only when PE-2 has been observed.

³ This situation is also consistent with the true fault not being listed in any ONLY-IF condition attached to the link.

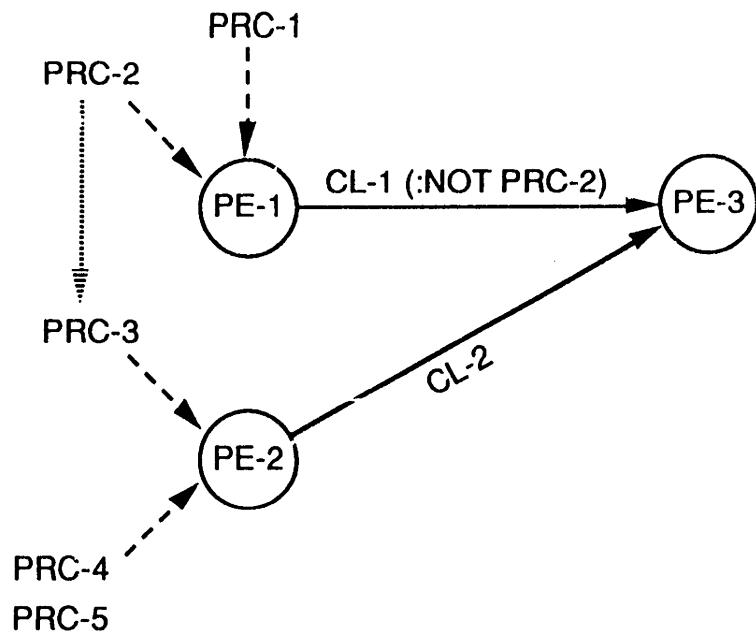


Figure 7-3: Induced Failure Link Example

7.5. Alternate Reasoning Strategies

Another means to improve MIDAS is to augment the existing reasoning strategies. Because MIDAS already completely exploits the information contained in qualitative event models, only two approaches are possible for augmentation:

- 1) Relax the assumptions in the current reasoning strategy, or
- 2) Develop reasoning strategies based on semi-quantitative models.

Both approaches will be discussed in subsequent discussion.

7.5.1. AND Worlds and OR Worlds

MIDAS was designed to explain observed events using the smallest number of inferred malfunctions. If an observed event can be explained by causal propagation from previously

observed events, it is assumed to have occurred due to causal propagation rather than a second malfunction. This assumption puts MIDAS completely within an "OR" world.

The OR world is characterized by an IM cluster diagnosis in which all hypotheses are related by an implicit OR. For example, in figure 7-4, if all three potential events are observed, the OR world diagnosis can be written:

(PRC-1 OR PRC-2)

PE-1 is classified as a SOURCE event and its local causes form the candidate set. PE-2 and PE-3 are classified as C' NSEQUENCE events.

A complete "AND" world diagnosis for the same situation is the following:

- 1) (PRC-1 OR PRC-2)

or
- 2) (PRC-1 OR PRC-2) AND (PRC-3)

or
- 3) (PRC-1 OR PRC-2) AND (PRC-4 OR PRC-5)

or
- 4) (PRC-1 OR PRC-2) AND (PRC-3) AND (PRC-4 OR PRC-5)

The AND world diagnosis formulates all possible multiple failure scenarios. The OR world diagnosis is simply one component of the complete AND world diagnosis. The second component of the diagnosis postulates PRC-1 or PRC-2 as the causes of PE-1 and PRC-3 as the cause of PE-2 and PE-3. The third component postulated PRC-1 or PRC-2 as the cause of PE-1 and PE-2 and PRC-4 or PRC-5 as the cause of PE-3. The fourth component assumes that all events have a different cause.

Fault candidate rankings would be based on the a priori likelihood of the candidate, the evidence supporting and opposing the candidate, and the likelihoods of all AND world diagnosis components in which the candidate appears. Because most AND world diagnosis components involve multiple faults, the likelihood of a component is related to the likelihood of two or more simultaneous faults. For independent faults, this likelihood will be small. For

dependent faults (e.g. induced faults) the likelihood may be close to the likelihood of the OR world diagnosis.

In the current version of MIDAS, AND world hypotheses have been avoided for two reasons:

- 1) Multiple failures are assumed to be relatively rare, and
- 2) For large IM clusters, the number of AND world hypotheses can grow combinatorially.

In certain situations, however, it may be helpful for operators to have the full AND world diagnosis available as an option.

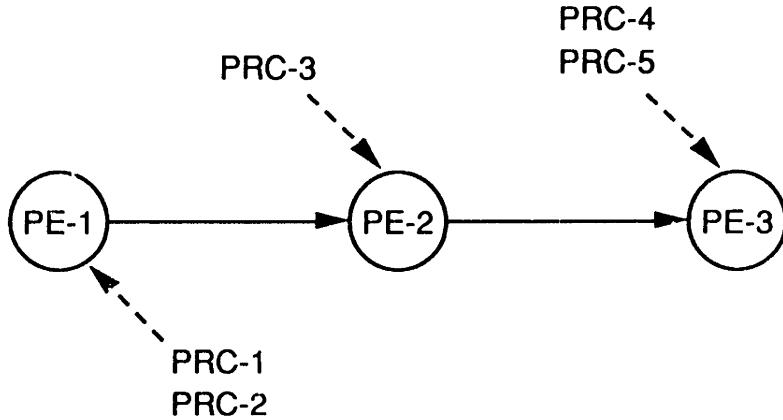


Figure 7-4: AND World Diagnosis Example

7.5.2. Breaking Causal Loops

It was shown in sections 4.0 and 6.0 that causal loops in the process model can seriously degrade the diagnostic resolution of MIDAS. Here, a technique for breaking causal loops is proposed. This technique requires knowledge of link gains and delays, discussed in section 7.4.2.

Consider the event model presented in figure 7-5. Assuming all three events are observed, it is impossible to determine a unique source event in the current MIDAS implementation. One

of the causal links is erroneous, but given the uncertainties of event detection, it cannot be isolated given the information in current event models. Using gain and delay information, however, causal links could be checked for consistency.

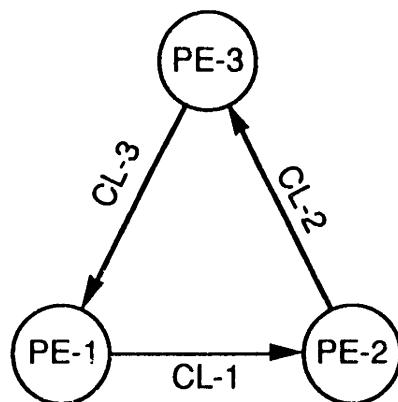


Figure 7-5: Causal Loop

For example, if CL-3 has a gain of 0.6 and a delay of 10 seconds, if the detection threshold of the PE-1 monitor is approximately twice as sensitive as the PE-3 monitor, and if PE-3 is observed more than 25 seconds after detection of PE-1, then the causal loop in figure 7-5 can be broken at CL-3. PE-1 can be classified as the sole source event⁴, and diagnostic resolution maintained.

The logic behind this determination can be explained with the help of figure 7-6. Figure 7-6A plots the deviation value of the variable that gives rise to the detection of PE-3. The variable begins deviating from its normal value at $t = 0$. Three possible growth curves for the disturbance are plotted, corresponding to an increasing, constant, and decreasing rate of disturbance growth. It is assumed that the true value of the variable lies inside the region defined by these curves.

The detection threshold of the PE-3 monitor is shown as a banded region to indicate

⁴ PE-1 will be the sole source event assuming PE-1 does not share any local causes with PE-2 or PE-3.

uncertainty in the exact detection threshold. Below the banded region, the monitor will never detect PE-3. Above the region the monitor will always detect PE-3. The time points t_1 and t_2 define the period in which detection of PE-3 will occur.

The hypothesis to be tested is whether detection of PE-3 is consistent with PE-3 being a SOURCE event with propagation of the disturbance along CL-3. The alternate hypothesis is that PE-1 is the SOURCE event and the disturbance has propagated to PE-3 along CL-1 and CL-2.

Figure 7-6B shows similar curves for the variable that gives rise to PE-1. Because the hypothesis being tested is whether PE-3 could be responsible for PE-1, the variable in figure 7-6B begins deviating at $t = 10$ seconds⁵ -- the delay associated with link CL-3. The gain of link CL-3 is 0.6, so all growth curves in figure 7-6B are flatter than those in figure 7-6A. However, the greater sensitivity of the PE-1 monitor means the detection range is lower. The time points t_3 and t_4 define the detection region of PE-1 under the current hypothesis. If detection of PE-3 occurs more than $(t_2 - t_3) = 25$ seconds after detection of PE-1, the hypothesis that PE-3 caused PE-1 cannot be true. PE-3 must be a CONSEQUENCE event.

When a causal loop is closed, all links in the loop must undergo this analysis. If any link fails the procedure, the causal loop can be broken at that link. Because of the uncertainties involved, it is possible that all links will pass the procedure, in which case, the causal loop remains intact and all events must be considered possible sources.

This technique requires a willingness to make certain assumptions pertaining to disturbance growth rates. If the actual variable value falls outside the region defined by the growth curves, the technique may fail, possibly breaking a causal loop at the wrong link and condemning MIDAS to an inaccurate diagnosis.

Engineering judgement must be exercised in devising growth curves. The only completely safe curves, guaranteed for all failure scenarios, are infinite growth (as an upper bound) and zero growth (as a lower bound). Unfortunately, these curves are useless in breaking causal loops.

⁵ Here, it is assumed the delay is known exactly. In actuality, delay is an uncertain parameter.

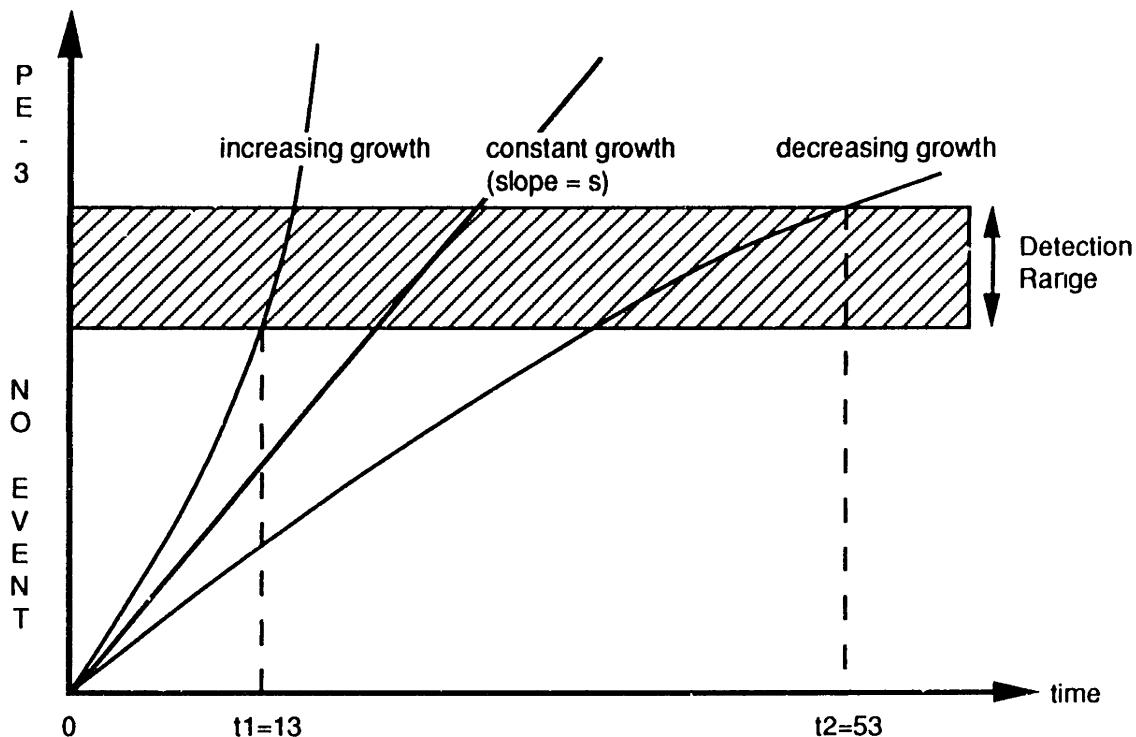


Figure 7-6A: Detection Range of PE-3

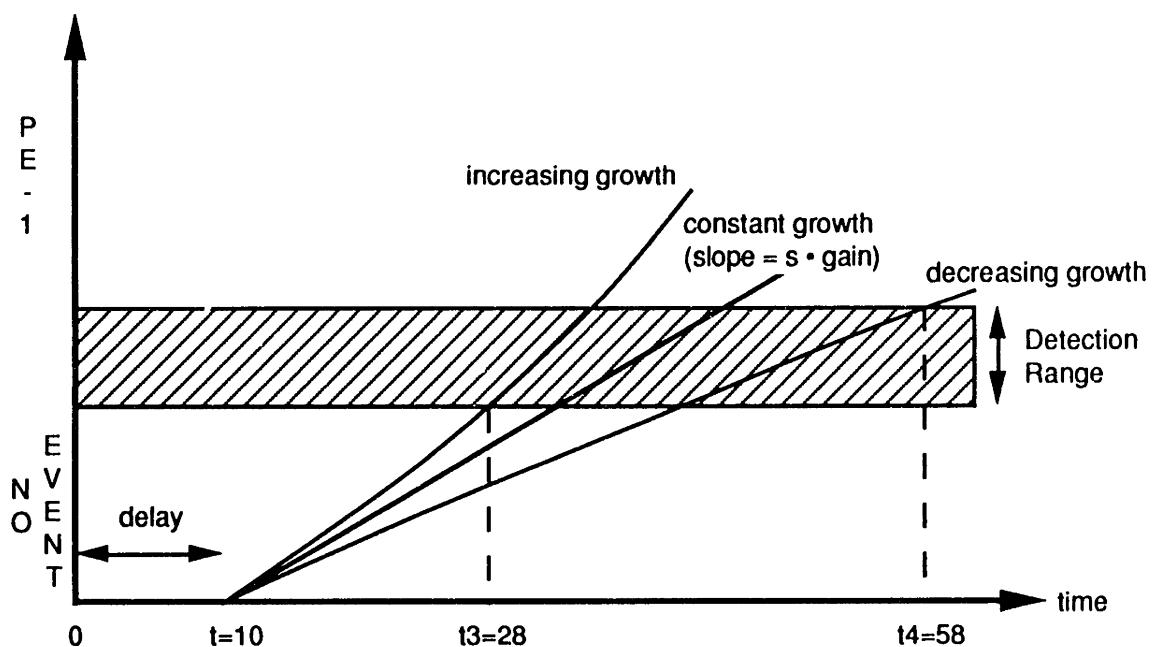


Figure 7-6B: Detection Range of PE-1

7.6. Improved Detection

Although MIDAS was designed to be robust to detection errors such as out-of-order events, some of the current problems encountered by MIDAS are best solved by improving the detection scheme rather than changing the model or algorithm. Specifically, the problem of apparent inverse and oscillatory response can be best addressed by more robust detection schemes.

In the current detection scheme, the value of a monitored variable is used for detection. This is equivalent to forming a constraint equation for the variable, as shown in equation 7-3,

$$X(t) - X^{ss} = R(t) \quad (7-3)$$

where $X(t)$ is the time dependent value of the variable and X^{ss} is the nominal steady-state value. $R(t)$ is the residual of the constraint (nominally zero). $X(t)$ is assumed to be measured every dt seconds.

Apparent inverse response results when the detection scheme picks up on short lived dynamics that do not represent either the initial or ultimate response. One technique to make the detection scheme less sensitive to these dynamics is to monitor the integral of the variable rather than the point value. Equation 7-4 presents a constraint that is qualitatively identical to equation 7-3, but should be robust to apparent inverse and oscillatory response produced by underdamped control response.

$$\int_0^t (X(t) - X^{ss}) \cdot dt = R(t) \quad (7-4)$$

Because the amplitude of the various peaks in the underdamped oscillation decay in magnitude, equation 7-4 should theoretically always indicate a HIGH state (if the initial deviation is in the positive direction) or a LOW state (if the initial deviation is in the negative direction). If true, then the detection scheme always indicates the initial response. MIDAS would have no trouble diagnosing this situation.

In practice, the constraint in equation 7-4 may oscillate between a HIGH and a NORMAL (or a LOW and a NORMAL) indication, because as $X(t)$ reaches a trough, the integral will

approach zero and may dip below the detection threshold. If this situation occurs, MIDAS will still have no problem with diagnosis, since the oscillation is between the initial and ultimate response.

7.7. Explanation and Advice

Currently, explaining how MIDAS reaches its conclusions requires a sound knowledge of the underlying model and algorithm. Conceptually, there is no reason why MIDAS could not develop coherent explanations of its diagnoses, automatically. The raw material for an explanation facility is already in place. The causal network of the IM cluster and the evidence slots of the HRCs provide all the information necessary. The task of forming an intelligible explanation requires only that this information be presented in a "user friendly" format when requested. Investigation of the best format for providing this information to the operator is beyond the scope of the current project.

Formulating appropriate actions for operators to take in response to a malfunction is also beyond the scope of this project. However, providing operators with "canned" advice is a relatively simple problem. Based on the current set of fault candidates and their rankings, MIDAS could look up the best response from a existing database of operating procedures. If none of the candidates are particularly hazardous and insufficient events have been detected to make a conclusive diagnosis, MIDAS might recommend waiting for more events to be observed. Otherwise, advice might consist of a list of actions to take or actions not to take. When an operator initiates an action, it would be logged as an operator action event, activating and deactivating appropriate process model links (see section 7.4.1).

8.0. References

- [1] Aldanondo, M. and T. Sheridan, "Failure Detection, Location and Discrimination Between Plant Component and Sensor Causation," *IEEE Trans. on Systems, Man, and Cybernetics*, submitted, (1986).
- [2] Andow, P.K., "Difficulties in Fault-Tree Synthesis for Process Plant," *IEEE Trans. on Reliability*, **R-29** (1), 2 (1980).
- [3] Andow, P.K., "Fault Diagnosis using Intelligent Knowledge Based Systems," *Chem. Eng. Res. Des.*, **63**, 368 (1985).
- [4] Arkes, H.R. and A.R. Harkness, "Effect of Making a Diagnosis on Subsequent Recognition of Symptoms," *J. Exp. Psych.*, **6** (5), 568 (1980).
- [5] Asbjornsen, O.A., "Control and Operability of Process Plants," *Comp. Chem. Eng.*, **13** (4/5), 351 (1989).
- [6] Berenblut, B.J. and H.B. Whitehouse, "A Method for Monitoring Process Plant Based on a Decision Table Analysis," *Chem. Eng.*, **318**, 175 (1977).
- [7] Buchanan, B.G. and E.H. Shortliffe, Rule-Based Expert Systems, Addison-Wesley, Reading, MA (1984).
- [8] Bylander, T., S. Mittal and B. Chandrasekaran, "CSRL: A Language for Expert Systems for Diagnosis," Proc. IJCAI, Los Angeles, CA (1983).
- [9] Chan L.K., K.P. Hapuarachchi and B.D. Macpherson, "Robustness of X and R Charts," *IEEE Trans. on Reliability*, **37** (1), 117 (1988).
- [10] Charniak, E. and D. McDermott, Introduction to Artificial Intelligence, Addison-Wesley, Reading, MA (1985).
- [11] Cheung, J. and G. Stephanopoulos, "Representation of Process Trends Part I. A Formal Representation Framework," MIT Laboratory for Intelligent Systems in Process Engineering, Report LISPE-89-052 (1989a).
- [12] Cheung, J. and G. Stephanopoulos, "Representation of Process Trends Part II. The Problem of Scale and Qualitative Scaling," MIT Laboratory for Intelligent Systems in Process Engineering, Report LISPE-89-059 (1989b).
- [13] Chung, D.T. and M. Modarres, "A Method of Fault Diagnosis: Presentation of a Deep Knowledge System," *Comp. Chem. Eng.*, submitted, (1986).
- [14] Clancey, W.J., "The Epistemology of a Rule-Based Expert System -- A Framework for Explanation," *Artificial Intelligence*, **20**, 215 (1983).
- [15] Corsberg, D., "Alarm Filtering: Practical Control Room Upgrade Using Expert Systems Concepts," *Intech*, **34** (4), 39 (1987).
- [16] Corsberg, D.R. and D. Wilkie, "An Object-Oriented Alarm-Filtering System," Power Plant Dynamics, Control, and Testing Symposium, Knoxville, TN (1986).

- [17] Cowan, J.D. and D.H. Sharp, "Neural Nets and Artificial Intelligence," *Daedalus*, **117** (1), 85 (1988).
- [18] Crowe, C.M., "Recursive Identification of Gross Errors in Linear Data Reconciliation," *AICHE J.*, **34** (4), 541 (1988).
- [19] Davis, R., "Diagnostic Reasoning Based on Structure and Behavior," in Bobrow D.G. (Ed.), Qualitative Reasoning About Physical Systems, MIT Press, Cambridge, MA, 347 (1984).
- [20] De Kleer, J., "An Assumption-Based TMS," *Artificial Intelligence*, **28** (2), 127 (1986a).
- [21] De Kleer, J., "Extending The ATMS," *Artificial Intelligence*, **28** (2), 163 (1986b).
- [22] De Kleer, J., "Problem Solving With The ATMS," *Artificial Intelligence*, **28** (2), 197 (1986c).
- [23] De Kleer, J. and J.S. Brown, "Theories of Causal Ordering," *Artificial Intelligence*, **29**, 33 (1986).
- [24] De Kleer, J. and J.S. Brown, "Qualitative Physics Based on Confluences," in Bobrow D.G. (Ed.), Qualitative Reasoning About Physical Systems, MIT Press, Cambridge, MA, 7 (1984).
- [25] Dhillon, B.S. and S.N. Rayapat, "Chemical System Reliability: A Review," *IEEE Trans. on Reliability*, **37** (2), 199 (1988).
- [26] Di Sessa, A.A., "Phenomenology and the Evolution of Intuition," in Gentner D. and Stevens A. (Eds.), Mental Models, Lawrence Erlbaum, Hillsdale, NJ, 15 (1983).
- [27] Duncan, K.D., "Fault Diagnosis Training for Advanced Continuous Process Installations," in Rasmussen J., Duncan K. and Leplat J. (Eds.), New Technology and Human Error, Wiley, New York, 209 (1987).
- [28] Duncan, K.D., "Training for Fault Diagnosis in Industrial Process Plant," in Rouse W.B. and Rasmussen J. (Eds.), Human Detection and Diagnosis of System Failures, Plenum, New York (1981).
- [29] Dhurjati, P.S., D.E. Lamb and D.L. Chester, "Experience in the Development of an Expert System for Fault Diagnosis in a Commercial Scale Chemical Process," in Reklaitis, G.V. and Spriggs, H.D. (Eds.), Proc. 1st Intl. Conf. on the Foundations of Computer-Aided Process Operations, Elsevier, Amsterdam (1987).
- [30] Duda, R.O. and P.E. Hart, Pattern Classification and Scene Analysis, Wiley, New York (1973).
- [31] Fikes, R. and T. Kehler, "The Role of Frame-Based Representation in Reasoning," *Comm. ACM*, **28**, 904 (1985).
- [32] Finch, F.E. and M.A. Kramer, "The Handling of Dynamics, Multiple Faults, and Out-of-Order Alarms in the MIDAS Diagnosis System," paper 37e, AIChE Spring National Meeting, Houston, TX (1989).

- [33] Finch, F.E. and M.A. Kramer, "Narrowing Diagnostic Focus using Functional Decomposition," *AICHE J.*, **34** (1), 25 (1988).
- [34] Finch, F.E. and M.A. Kramer, "Expert System Methodology for Process Malfunction Diagnosis," Proc. 10th IFAC World Congress on Automatic Control, Munich, FRG (1987).
- [35] Fink, P.K. and J.C. Lusth, "Expert Systems and Diagnostic Expertise in the Mechanical and Electrical Domains," *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-**17**, 340 (1987).
- [36] Forbus, K.D., "Qualitative Process Theory," in Bobrow D.G. (Ed.), Qualitative Reasoning About Physical Systems, MIT Press, Cambridge, MA, 85 (1984).
- [37] Fusillo, R.H. and G.J. Powers, "A Synthesis Method for Chemical Plant Operating Procedures," *Comp. Chem. Eng.*, **11** (4), 369 (1987).
- [38] Gertler, J., "Failure Detection and Isolation in Complex Process Plants -- A Survey," IFAC Microcomputer Application in Process Control, Istanbul, Turkey (1986).
- [39] Gettys, C.F. and S.D. Fisher, "Hypothesis Plausibility and Hypothesis Generation," *Org. Beh. and Human Perf.*, **24**, 93 (1979).
- [40] Griffin, C.D., D.V. Croson and J.J. Feeley, "Kalman Filtering Applied to a Reagent Feed System," *Chem. Eng. Prog.*, **84** (10), 45 (1988).
- [41] Harmon, P., "Inventory and Analysis of Existing Expert Systems," *Expert System Strategies*, **2** (8), (1986).
- [42] Hayes, P.J., "The Naive Physics Manifesto," in Michie D. (Ed.), Expert Systems in the Microelectronic Age, Edinburgh Univ. Press, Edinburgh (1979).
- [43] Henneman, R.L. and W.B. Rouse, "On Measuring the Complexity of Monitoring and Controlling Large-Scale Systems," *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-**14** (1), 99 (1984).
- [44] Hillis, W.D., The Connection Machine, MIT Press, Cambridge, MA (1985).
- [45] Himmelblau, D.M., Fault Detection and Diagnosis in Chemical and Petrochemical Processes, Elsevier, Amsterdam (1978).
- [46] Hoskins, J.C. and D.M. Himmelblau, "Artificial Neural Network Models of Knowledge Representation in Chemical Engineering," *Comp. and Chem. Eng.*, **12**, 881 (1988).
- [47] Iordache, C., R.S.H. Mah and A.C. Tamhane, "Performance Studies of the Measurement Test for Detection of Gross Errors in Process Data," *AICHE J.*, **31**, 1187 (1985).
- [48] Iri, M., K. Aoki, E. O'Shima and H. Matsuyama, "An Algorithm for Diagnosis of System Failures in the Chemical Process," *Comp. and Chem. Eng.*, **3**, 489 (1979).

- [49] Isermann, R., "Process Fault Detection Based on Modeling and Estimation Methods -- A Survey," *Automatica*, **20**, 387 (1984).
- [50] Iwasaki, Y. and H.A. Simon, "Causality in Device Behavior," *Artificial Intelligence*, **29**, 3 (1986).
- [51] Kalman, R.E., "A New Approach to Linear Filtering and Prediction Problems," *J. Basic Engr.*, **82d**, 35 (1960).
- [52] Karpman, G. and B. Dubuisson, "Complexity Index for System Diagnosability," *IEEE Trans. on Systems, Man, and Cybernetics*, **SMC-15** (2), 294 (1985).
- [53] Kragt, H. and J. Bonten, "Evaluation of a Conventional Process-Alarm System in a Fertilizer Plant," *IEEE Trans. on Systems, Man, and Cybernetics*, **SMC-13** (4), 586 (1983).
- [54] Kramer, M.A. and F.E. Finch, "Fault Diagnosis of Chemical Processes," in Tzafestas, S.G. (Ed.), Knowledge-Based Systems Diagnosis, Supervision and Control, Plenum, New York (1989).
- [55] Kramer, M.A. and J. Leonard, "Improvement of the Backpropagation Algorithm for Training Neural Networks," *Comp. and Chem. Eng.*, submitted, (1989).
- [56] Kramer, M.A., "Automated Diagnosis of Malfunctions Based on Object Oriented Programming," *J. Loss Prev. Process Ind.*, **1**, 226 (1988).
- [57] Kramer, M.A., "Expert Systems for Process Fault Diagnosis: A General Framework," in Reklaitis, G.V. and Spriggs, H.D. (Eds.), Proc. 1st Intl. Conf. on the Foundations of Computer-Aided Process Operations, Elsevier, Amsterdam (1987).
- [58] Kramer, M.A., "Malfunction Diagnosis using Quantitative Models with Non-Boolean Reasoning in Expert Systems," *AICHE J.*, **33**, 130 (1987b).
- [59] Kramer, M.A., "Integration of Heuristic and Model-Based Inference in Chemical Process Fault Diagnosis," IFAC Workshop on Fault Detection and Safety in Chemical Plants, Kyoto, Japan (1986).
- [60] Kuipers, B., "Qualitative Simulation as Causal Explanation," *IEEE Trans. on Systems, Man, and Cybernetics*, **SMC-17**, 432 (1987).
- [61] Kuipers, B., "Qualitative Simulation," *Artificial Intelligence*, **29**, 289 (1986).
- [62] Kuipers, B., "Commonsense Reasoning About Causality: Deriving Behavior from Structure," in Bobrow D.G. (Ed.), Qualitative Reasoning About Physical Systems, MIT Press, Cambridge, MA, 169 (1984).
- [63] Kumamoto, H. and E.J. Henley, "Automated Fault Tree Synthesis by Disturbance Analysis," *Ind. Eng. Chem. Fundam.*, **25**, 233 (1986).
- [64] Lapp, S.A. and G.J. Powers, "Computer-aided Synthesis of Fault Trees," *IEEE Trans. on Reliability*, **R-26**, 2 (1977).

- [65] Lees, F.P., "Process Computer Alarm and Disturbance Analysis: Review of the State of the Art," *Comp. and Chem. Eng.*, **7** (6), 669 (1983).
- [66] Lees, F.P., "Research on the Process Operator," in Edwards E. and Lees F.P. (Eds.), The Human Operator in Process Control, Taylor and Francis, London, 386 (1974).
- [67] Liu, K. and J. Gertler, "A Supervisory (Expert) Adaptive Control Scheme," Proc. 10th IFAC World Congress on Automatic Control, Munich, FRG (1987).
- [68] Long, A.B., "Computerized Operator Decision Aids," *Nucl. Safety*, **25**, 512 (1984).
- [69] MacGregor, J.F., "On-Line Statistical Process Control," *Chem. Eng. Prog.*, **84** (10), 21 (1988).
- [70] Mah, R.S.H. and A.C. Tamhane, "Detection of Gross Errors in Process Data," *AICHE J.*, **28** (5), 828 (1982).
- [71] Mah, R.S., G.M. Stanley and D.M. Downing, "Reconciliation and Rectification of Process Flow and Inventory Data," *Ind. Eng. Chem. Process Des. Dev.*, **15** (1), 175 (1976).
- [72] Mavrovouniotis, M. and G. Stephanopoulos, "Formal Order-of-Magnitude Reasoning in Process Engineering," MIT Laboratory for Intelligent Systems in Process Engineering, Report LISPE-88-028 (1988).
- [73] McClelland, J.L. and D.E. Rumelhart (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition Vol. 1-3, MIT Press, Cambridge, MA (1986).
- [74] Mehle, T., C.F. Gettys, C. Manning, S. Baca and S. Fisher, "The Availability Explanation of Excessive Plausibility Assessments," *Acta Psych.*, **49**, 127 (1981).
- [75] Milne, R., "Strategies for Diagnosis," *IEEE Trans. on Systems, Man, and Cybernetics*, **SMC-17**, 333 (1987).
- [76] Milne, R., "Fault Diagnosis Through Responsibility," Proc. IJCAI, Los Angeles, CA (1985).
- [77] Modarres, M. and T. Cadman, "A Method of Alarm System Analysis for Process Plants," *Comp. and Chem. Eng.*, submitted, (1986).
- [78] Moore, R.L., L.B. Hawkinson, C.G. Knickerbocker and L.M. Churchman, "A Real-Time Expert System for Process Control," Proc. First Conference on Artificial Intelligence Applications (IEEE), Denver, CO, 529 (1984).
- [79] Narasimhan, S., C.S. Kao and R.S.H. Mah, "Detecting Changes of Steady State Using the Mathematical Theory of Evidence," *AICHE J.*, **33** (11), 1930 (1987).
- [80] Newquist, H.P., "True Stories: The Reality of AI Applications," *AI Expert*, **2** (2), 63 (1987).

- [81] O'Shima, E., "Computer Aided Plant Operation," *Comp. and Chem. Eng.*, **7**, 311 (1983).
- [82] Oyeleye, O.O., Sc.D. Thesis: Qualitative Modeling of Continuous Chemical Processes and Applications to Fault Diagnosis, Massachusetts Institute of Technology (1989).
- [83] Oyeleye, O.O. and M.A. Kramer, "Qualitative Simulation of Chemical Process Systems: Steady-State Analysis," *AIChE J.*, **34** (9), 1441 (1988).
- [84] Palowitch, B.L., Sc.D. Thesis: Fault Diagnosis of Process Plants using Causal Models, Massachusetts Institute of Technology (1987).
- [85] Park, S.W. and D.M. Himmelblau, "Structural Design for Systems Fault Diagnosis," *Comp. Chem. Eng.*, **11** (6), 713 (1987).
- [86] Pau, L.F., Failure Diagnosis and Performance Monitoring, Marcel Dekker, New York (1981).
- [87] Pau, L.F., "Survey of Expert Systems for Fault Detection, Test Generation, and Maintenance," *Expert Systems*, **3** (2), 100 (1986).
- [88] Pitblado, R.M. and I.A. Lake, "Guidelines for the Application of Hazard Warning," *Chem. Eng. Res. Des.*, **65**, 334 (1987).
- [89] Rasmussen, J., "The Role of Hierarchical Knowledge Representation in Decisionmaking and System Management," *IEEE Trans. on Systems, Man, and Cybernetics*, **SMC-15**, 234 (1985).
- [90] Rasmussen, J., "Skills, Rules, and Knowledge; Signals, Signs, and Symbols; and Other Distinctions in Human Performance Models," *IEEE Trans. on Systems, Man, and Cybernetics*, **SMC-13** (3), 257 (1983).
- [91] Rasmussen, J., "Models of Mental Strategies in Process Plant Diagnosis," in Rouse W.B. and Rasmussen J. (Eds.), Human Detection and Diagnosis of System Failures, Plenum, New York (1981).
- [92] Rich, S.H. and V. Venkatasubramanian, "Model-Based Reasoning in Diagnostic Expert Systems for Chemical Process Plants," *Comp. and Chem. Eng.*, **11**, 111 (1987).
- [93] Romagnoli, J.A. and G. Stephanopoulos, "Rectification of Process Measurements in the Presence of Gross Errors," *Chem. Eng. Sci.*, **36** (11), 1849 (1981).
- [94] Rosenberg, J., R.S.H. Mah and C. Iordache, "Evaluation of Schemes for Detecting and Identifying Gross Errors in Process Data," *Ind. Eng. Chem. Res.*, **26**, 555 (1987).
- [95] Rosenof, H., R. Moore, G. Stanley and R. Smith, "A Tutorial on Real-Time Expert Systems," Proc. Conference on Expert Systems Applications for the Electric Power Industry (EPRI), Orlando, FL (1989).
- [96] Rouse, W.B., "Models of Human Problem Solving: Detection, Diagnosis, and Compensation for System Failures," *Automatica*, **19** (6), 613 (1983).

- [97] Rouse, W.B., "Experimental Studies and Mathematical Models of Human Problem Solving Performance in Fault Diagnosis Tasks," in Rouse W.B. and Rasmussen J. (Eds.), Human Detection and Diagnosis of System Failures, Plenum, New York, 119-216 (1981).
- [98] Rouse, W.B. and R.M. Hunt, "Human Problem Solving in Fault Diagnosis Tasks," in Rouse W.B. (Ed.), Advances in Man-Machine Systems Research, JAI Press, Greenwich, CT (1984).
- [99] Rowan, D.A., "AI Enhances On-Line Fault Diagnosis," *InTech*, **35** (5), 52 (1988).
- [100] Rowan, D.A., "Chemical Plant Fault Diagnosis using Expert Systems Technology: A Case Study," IFAC Workshop on Fault Detection and Safety in Chemical Plants, Kyoto, Japan (1986).
- [101] Sachs, P.A. and A.M. Paterson, "ESCORT - An Expert System for Complex Operations in Real Time," *Expert Systems*, **3** (1), 22 (1986).
- [102] Scarl, E.A., J.R. Jamieson and C.I. Delaune, "Diagnosis and Sensor Validation through Knowledge of Structure and Function," *IEEE Trans. on Systems, Man, and Cybernetics*, **SMC-17** (3), 360 (1987).
- [103] Shafaghi, A., P.K. Andow and F.P. Lees, "Fault Tree Synthesis Based on Control Loop Structure," *Chem. Eng. Res. Des.*, **62**, 101 (1984).
- [104] Shafer, G., A Mathematical Theory of Evidence, Princeton Univ. Press, Princeton, NJ (1976).
- [105] Sheridan T.B., "Understanding Human Error and Aiding Human Diagnostic Behavior in Nuclear Power Plants," in Rouse W.B. and Rasmussen J. (Eds.), Human Detection and Diagnosis of System Failures, Plenum, New York (1981).
- [106] Shewhart, W.A., Economic Control of Quality in Manufactured Product, Van Nostrand, New York (1931).
- [107] Shiozaki, J., H. Matsuyama, E. O'Shima and M. Iri, "An Improved Algorithm for Diagnosis of System Failures in the Chemical Process," *Comp. and Chem. Eng.*, **9**, 285 (1985).
- [108] Shum, S.K., J.F. Davis, W.F. Punch and B. Chandrasekaran, "An Expert System Approach to Malfunction Diagnosis in Chemical Plants," *Comp. and Chem. Eng.*, **12**, 27 (1988)
- [109] Stanley, G.M. and R.S.H. Mah, "Estimation of Flows and Temperatures in Process Networks," *AICHE J.*, **23** (5), 642 (1977).
- [110] Stefik, M. and D.G. Bobrow, "Object-Oriented Programming: Themes and Variations," *The AI Magazine*, **6** (4), 40 (1986).
- [111] Stephanopoulos, G., J. Johnston, T. Kriticos, R. Lakshmanan, M. Mavrovouniotis and C. Siletti, "DESIGN-KIT: An Object-Oriented Environment for Process Engineering," *Comp. and Chem. Eng.*, **11**, 655 (1987).

- [112] Swets, J.A., "Measuring the Accuracy of Diagnostic Systems," *Science*, **240**, 1285 (1988).
- [113] Tamhane, A.C. and R.S.H. Mah, "Data Reconciliation and Gross Error Detection in Chemical Process Networks," *Technometrics*, **27** (4), 409 (1985)
- [114] Tversky, A. and Kahneman, D., "Judgement under Uncertainty: Heuristics and Biases," *Science*, **185**, 1124 (1974).
- [115] Ulerich, N.H. and G.J. Powers, "On-Line Hazard Aversion and Fault Diagnosis in Chemical Processes: The Digraph + Fault Tree Method," *IEEE Trans. on Reliability*, **37**, 171 (1988).
- [116] Umeda, T., T. Kuriyama, E. O'Shima and H. Matsuyama, "A Graphical Approach to Cause and Effect Analysis of Chemical Processing Systems," *Chem. Eng. Sci.*, **35**, 2379 (1980).
- [117] Washio, T., M. Kitamura and K. Sugiyama, "Development of Failure Diagnosis Method Based on Transient Information of Nuclear Power Plant," *J. Nucl. Sci. Tech.*, **24**, 452 (1987).
- [118] Watanabe, K. and D.M. Himmelblau, "Incipient Fault Diagnosis of Nonlinear Processes with Multiple Causes of Faults," *Chem. Eng. Sci.*, **39**, 491 (1984).
- [119] Watanabe, K. and D.M. Himmelblau, "Fault Diagnosis in Nonlinear Chemical Process: Part I. Theory," *AIChE J.*, **29** (2), 243 (1983a).
- [120] Watanabe, K. and D.M. Himmelblau, "Fault Diagnosis in Nonlinear Chemical Process: Part II. Application to a Chemical Reactor," *AIChE J.*, **29** (2), 250 (1983b).
- [121] Williams, M.D., J.D. Hollan and A.L. Stevens, "Human Reasoning About a Simple Physical System," in Gentner D. and Stevens A. (Eds.), Mental Models, Lawrence Erlbaum, Hillsdale, NJ, 131 (1983).
- [122] Winston, P.H. and B.K.P. Horn, Lisp, Addison-Wesley, Reading, MA (1984).
- [123] Yoon, E.S., Ph.D. Thesis: Process Failure Diagnostics using the Symptom Sub-Tree Model, Massachusetts Institute of Technology (1982).
- [124] Yoon, W.C. and J.M. Hammer, "Deep Reasoning Fault Diagnosis: An Aid and a Model," *IEEE Trans. on Systems, Man, and Cybernetics*, **18** (4), 659 (1988).
- [125] Yufik, Y.M. and T.B. Sheridan, "Hybrid Knowledge-Based Decision Aid for Operators of Large Scale Systems," *Large Scale Systems*, **10**, 133 (1986).
- [126] Zadeh, L.A., "Fuzzy Sets," *Inf. Control*, **8**, 338 (1965).

A1. APPENDIX: Process Model Examples

This appendix presents examples of the objects found in a process model formatted for use in MIDAS. All object types are shown, including the screen interface objects required by the User Interface.

A1.1. POTENTIAL-EVENT Object

POTENTIAL-EVENT objects must be created for every anticipated event. For most monitored process variables (or constraint residuals), eight PEs are required to account for all possible transitions between the three (3) qualitative states and the two (2) qualitative statuses:

- 1) NORMAL → HIGH
- 2) NORMAL → LOW
- 3) LOW → HIGH
- 4) HIGH → LOW
- 5) HIGH → NORMAL
- 6) LOW → NORMAL
- 7) SENSOR FAILED
- 8) SENSOR OK

Event transitions from LOW → HIGH and HIGH → LOW are included because the monitors used by MIDAS can miss the intervening NORMAL state if it is not prolonged. Another means to solve this problem would be to modify the inference cycle to create two events (e.g. LOW → NORMAL → HIGH) whenever a dual transition (e.g. LOW → HIGH) is detected.

When filling the EXCLUSIVE-EVENTS slot, all events involving state change are mutually exclusive as are all events involving status change. State change PEs will have five (5) exclusive events and status change PEs will have one (1) exclusive event.

If an event does not entail a change in state, trend, or status, the corresponding PRIOR-STATE, PRIOR-TREND, PRIOR-STATUS, CONSEQUENT-STATE, CONSEQUENT-TREND, and CONSEQUENT-STATUS slots are left empty or filled with

NIL. Alternately, compound events comprised of two or more significant changes can be defined.

POTENTIAL-EVENT Example for the event NORMAL → HIGH:

```
(DEFINE-INSTANCE EV-1
  (:print-name "EV-1"
   :doc-string ""
   :is POTENTIAL-EVENT)
  (ID EV-1)
  (TYPE VARIABLE)
  (OBJECT MV-REACTOR_TEMPERATURE)
  (PRIOR-STATE NORMAL)
  (PRIOR-STATUS NIL)
  (PRIOR-TREND NIL)
  (CONSEQUENT-STATE HIGH)
  (CONSEQUENT-STATUS NIL)
  (CONSEQUENT-TREND NIL)
  (EXCLUSIVE-EVENTS EV-2 EV-3 EV-4 EV-5 EV-6)
  (ACTIVE NO)
  )
```

A1.2. COMPILED-LINK Object

A COMPILED-LINK object connecting two variable or constraint residual nodes is of the subclass: COMPILED-CAUSAL-LINK. This differentiates this link from one used to represent a dependency between system nodes (i.e. a COMPILED-FUNCTIONAL-LINK).

COMPILED-LINK Example:

```
(DEFINE-INSTANCE CAUSAL-LINK-17
  (:print-name "CAUSAL-LINK-17"
   :doc-string ""
   :is COMPILED-CAUSAL-LINK)
  (ACTIVE NO)
  (DEACTIVATE-IF
    ((MV-CW_FLOW_CONTROLLER_SIGNAL NORMAL))
    ((MV-CW_FLOW_CONTROLLER_SIGNAL HIGH)))
  (ACTIVATE-IF
    ((MV-CW_FLOW_CONTROLLER_SIGNAL LOW)))
  (NOT-CONDITIONS
    (:NOT (PRC-CW_FLOW_SENSOR_FAILED_HIGH
          PRC-CW_FLOW_SENSOR_BIASED_HIGH)))
```

```

PRC-VALVE_2_STUCK_OPEN
PRC-VALVE_2_LEAK_TO_ENVIRONMENT
PRC-CSTR_JACKET_LEAK_TO_ENVIRONMENT
PRC-JACKET_EFFLUENT_SINK_PRESSURE_LOW
PRC-CW_FLOW_CONTROLLER_FAILED_LOW
PRC-CW_FLOW_CONTROLLER_BIASED_HIGH
PRC-CW_FLOW_CONTROLLER_FAILED_HIGH)))
(ID CAUSAL-LINK-17)
(SOURCE-NODE MV-COOLING_WATER_FLOWRATE)
(SOURCE-STATE LOW)
(RESULT-NODE MV-CW_FLOW_CONTROLLER_SIGNAL)
(RESULT-STATE NORMAL)
)

```

A1.3. MEASURED-OBJECT Object

The example of the class MEASURED-OBJECT shown below belongs to the subclass MEASURED-VARIABLES. Other subclasses include: MEASURED-CONSTRAINTS and MEASURED-SYSTEMS.

MEASURED-OBJECT Example:

```

(DEFINE-INSTANCE MV-PRODUCT_FLOWRATE
  (:print-name "MV-PRODUCT_FLOWRATE"
   :doc-string "OUTPUT SIGNAL in SENSOR: PRODUCT_FLOWRATE_SENSOR"
   :is MEASURED-VARIABLES)
  (LOCAL-CAUSES PRC-PRODUCT_FLOW_SENSOR FAILED_LOW
    PRC-PRODUCT_FLOW_SENSOR_BIASED_LOW
    PRC-PRODUCT_FLOW_SENSOR FAILED_HIGH
    PRC-PRODUCT_FLOW_SENSOR_BIASED_HIGH
    PRC-VALVE_1_STUCK_OPEN PRC-VALVE_1_STUCK_CLOSED
    PRC-VALVE_1_LEAK_TO_ENVIRONMENT PRC-VALVE_1_BLOCKAGE
    PRC-REACTOR_EFFLUENT_SINK_PRESSURE_LOW
    PRC-REACTOR_EFFLUENT_SINK_PRESSURE_HIGH
    PRC-PUMP_SHAFT_BROKEN PRC-PUMP_MOTOR FAILED
    PRC-PUMP_MOTOR_SPEED_LOW
    PRC-PUMP_MOTOR_SPEED_HIGH
    PRC-PUMP_CAVITATION PRC-PUMP LEAK_TO_ENVIRONMENT
    PRC-PUMP_BLOCKAGE PRC-CSTR_JACKET LEAK_TO_REACTOR
    PRC-CSTR_OUTLET_BLOCKAGE
    PRC-CSTR LEAK_TO_ENVIRONMENT
    PRC-CSTR_INLET_BLOCKAGE)
  (EVENT-SET EV-41 EV-42 EV-43 EV-44 EV-45 EV-46 EV-47 EV-48)
  (ID MV-PRODUCT_FLOWRATE)
  (TYPE VARIABLE))

```

```
(SUCCESSOR-LINKS CAUSAL-LINK-39 CAUSAL-LINK-40 CAUSAL-LINK-41
  CAUSAL-LINK-42 CAUSAL-LINK-43 CAUSAL-LINK-44
  CAUSAL-LINK-45 CAUSAL-LINK-46 CAUSAL-LINK-47
  CAUSAL-LINK-48 CAUSAL-LINK-49 CAUSAL-LINK-50
  CAUSAL-LINK-51 CAUSAL-LINK-52 CAUSAL-LINK-53
  CAUSAL-LINK-54 CAUSAL-LINK-55 CAUSAL-LINK-56)
(PRECURSOR-LINKS CAUSAL-LINK-179 CAUSAL-LINK-180 CAUSAL-LINK-181
  CAUSAL-LINK-182 CAUSAL-LINK-183 CAUSAL-LINK-255
  CAUSAL-LINK-256 CAUSAL-LINK-257 CAUSAL-LINK-258
  CAUSAL-LINK-259 CAUSAL-LINK-322))
```

A1.4. POTENTIAL-ROOT-CAUSE Object

Although in this example, the state change events in the PRIMARY-DEVIATION slot use an abbreviated form of the MIDAS description format, a status change event would need to use the full format (e.g. (MV-VALVE_1 NIL NIL FAILED)).

POTENTIAL-ROOT-CAUSE Example:

```
(DEFINE-INSTANCE PRC-VALVE_1_STUCK_CLOSED
  (:print-name "PRC-VALVE_1_STUCK_CLOSED"
   :doc-string "VALVE_STUCK_CLOSED in CONTROL_VALVE: VALVE_1"
   :is POTENTIAL-ROOT-CAUSE)
  (ID PRC-VALVE_1_STUCK_CLOSED)
  (PRIMARY-DEVIATION
    ((MV-PRODUCT_FLOWRATE LOW)
     (MV-COOLING_WATER_SETPOINT HIGH)
     (MV-COOLING_WATER_SETPOINT LOW)
     (MV-CONCENTRATION_A LOW)
     (MV-CONCENTRATION_B HIGH)
     (MV-REACTOR_TEMPERATURE HIGH)
     (MV-REACTOR_TEMPERATURE LOW)
     (MV-REACTOR_LEVEL HIGH)
     (MC-EFFL_PRESSURE_DROP LOW)
     (MC-MOL_BALANCE LOW)))
  (APPLICABLE-TESTS T-VALVE_1_RESPONSE_TEST_POSITIVE
    T-VALVE_1_RESPONSE_TEST_NEGATIVE)
  (PRIOR-PROBABILITY 1.0))
```

A1.5. TEST Object

Inference using test events is based entirely on the contents of the various CONDITIONS slots of the TEST instance. In the example below, an ONLY-IF condition has been attached to the TEST to indicate that if the PUMP tests positive for leak, only a PUMP_LEAK could be the root cause. The complimentary TEST instance:

T-PUMP_LEAK_INSPECTION_NEGATIVE

would have a NOT condition of the form

(:NOT (PRC-PUMP_LEAK_TO_ENVIRONMENT))

to indicate that this root cause can be eliminated from the HRC set if visual inspection fails to reveal a leak.

Note that like other events, TEST instances always have a mutually exclusive TEST instance.

TEST Object Example:

```
(DEFINE-INSTANCE T-PUMP_LEAK_INSPECTION_POSITIVE
  (:print-name "T-PUMP_LEAK_INSPECTION_POSITIVE"
   :doc-string ""
   :is TEST)
  (ID T-PUMP_LEAK_INSPECTION_POSITIVE)
  (EXCLUSIVE-TESTS T-PUMP_LEAK_INSPECTION_NEGATIVE)
  (ONLY-IF-CONDITIONS (:ONLY-IF (PRC-PUMP_LEAK_TO_ENVIRONMENT))))
```

A1.6. DATA-MONITOR Object

DATA-MONITOR objects are divided into two subclasses: SENSOR-MONITOR and CONSTRAINT-MONITOR. This distinction is purely for bookkeeping purposes. The two subclasses are identical.

DATA-MONITOR Example:

```
(DEFINE-INSTANCE FEED_FLOWRATE_SENSOR
  (:print-name "FEED_FLOWRATE_SENSOR"
   :doc-string ""
   :is SENSOR-MONITOR)
  (STATE-EVENTS EV-81 EV-82 EV-83 EV-84 EV-85 EV-86)
  (MONITOR-EVENTS EV-87 EV-88)
  (PREDICTION-VALID NO)
  (PREDICTED-STATUS OK)
  (PREDICTED-TREND STEADY)
  (PREDICTED-STATE NORMAL)
  (PERCEIVED-TREND STEADY)
  (PERCEIVED-STATE NORMAL)
  (PERCEIVED-MONITOR-STATUS OK)
  (RANGE-LCL 0.0)
  (RANGE-NOMINAL 3.38884F-03)
  (RANGE-UCL 0.3535)
  (SDEV-LCL 0.0)
  (SDEV-LWL 0.0)
  (SDEV-NOMINAL 2.39627F-03)
  (SDEV-UWL 999.9)
  (SDEV-UCL 999.9)
  (MEAN-LCL 2.41308F-01)
  (MEAN-LWL 2.43677F-01)
  (MEAN-NOMINAL 2.50195F-01)
  (MEAN-UWL 2.55682F-01)
  (MEAN-UCL 2.57984F-01)
  (POLYNOMIAL-ORDER 3)
  (DRIFT-CONSTANT 0.1)
  (FILTER-CONSTANT 0.9)
  (FILTERED-SAMPLE-SIZE 8)
  (RAW-SAMPLE-SIZE 2)
  (DRIFT-HORIZON 8)
  (ANALYSIS-HORIZON 6)
  (PREDICTION-HORIZON 2)
  (MAXIMUM-VALUE 0.35)
  (MINIMUM-VALUE 0.0)
  (ID FEED_FLOWRATE_SENSOR)
  (ASSOCIATION MV-FEED_FLOWRATE)
  (MONITOR-TYPE SENSOR-MONITOR)
  )
```

A1.7. User Interface Objects

The User Interface objects are created at the time of process model construction. In essence, these objects tell the User Interface the names (IDs) of the objects in the process model. Creation of these objects is not mandatory, although substantial portions of the User Interface will not work if they are not created.

All User Interface objects are of the class POPUP-CHOOSE provided in the SCREEN TOOLKIT of GoldWorks. All are virtually identical excepts for the values in the INSTRUCTIONS and CONTENTS slots. The INSTRUCTIONS slot is filled with a banner indicating the class of object listed. The CONTENTS slot contains a list of all objects of the specified class. Note that the ANSWER slot contains a demon (ACTIVATE-POPUP or ACTIVATE-CHOSEN-EVENT) that is not reliably saved in GoldWorks version 1.1 and may need to be added manually using the editor.

The CONTENTS slot is filled by a list of lists where sublists follow a specific format:

("TEXT DESCRIPTION OF INSTANCE" <INSTANCE ID>)

See CHOOSE-TEST and CHOOSE-NEXT-EVENT below for examples of this format. The first sublist must always be

(" CANCEL " :CANCEL)

or it will be impossible to cancel the indicated action.

A total of eight (8) User Interface objects must be created:

- 1) CHOOSE-TEST
- 2) CHOOSE-VARIABLE
- 3) CHOOSE-CONSTRAINT
- 4) CHOOSE-SYSTEM
- 5) CHOOSE-MONITOR
- 6) CHOOSE-POTENTIAL-CAUSE
- 7) CHOOSE-LINK
- 8) CHOOSE-NEXT-EVENT

Examples of these objects are presented below.

Interface Object Examples:

```
(MAKE-INSTANCE 'CHOOSE-TEST
  :print-name "CHOOSE-TEST"
  :doc-string ""
  :is 'POPUP-CHOOSE
  :slots
  '(
    (INSTRUCTIONS :VALUE
      (" ** Select Test ** "))
    (BORDER-COLOR :VALUE :CYAN)
    (ANSWER :WHEN-MODIFIED (ACTIVATE-POPUP))
    (TOP :VALUE 8)
    (LEFT :VALUE 32)
    (CENTER :VALUE :NO-CENTERING)
    (CONTENTS :VALUE
      ((" CANCEL " :CANCEL)
       (" PUMP LEAK INSPECTION TRUE " T-PUMP LEAK INSPECTION TRUE)
       (" PUMP LEAK INSPECTION FALSE " T-PUMP LEAK INSPECTION FALSE)
       .
       .
       .
       (" CSTR LEAK INSPECTION TRUE " T-CSTR LEAK INSPECTION TRUE)
       (" CSTR LEAK INSPECTION FALSE " T-CSTR LEAK INSPECTION FALSE)
     ))
    (FORCE-CHOICE :VALUE :YES)))
  ++++++
```

```
(MAKE-INSTANCE 'CHOOSE-VARIABLE
  :print-name "CHOOSE-VARIABLE"
  :doc-string ""
  :is 'POPUP-CHOOSE
  :slots
  '(
    (INSTRUCTIONS :VALUE
      (" ** Select Variable ** "))
    (BORDER-COLOR :VALUE :CYAN)
    (ANSWER :WHEN-MODIFIED (ACTIVATE-POPUP))
    (TOP :VALUE 8)
    (LEFT :VALUE 50)
    (CENTER :VALUE :X)
    (CONTENTS :VALUE
      ( (" CANCEL " :CANCEL)))
    (FORCE-CHOICE :VALUE :YES)))
```

```
+++++
(MAKE-INSTANCE 'CHOOSE-CONSTRAINT
  :print-name "CHOOSE-CONSTRAINT"
  :doc-string ""
  :is 'POPUP-CHOOSE
  :slots
  '(
    (INSTRUCTIONS :VALUE
      (" ** Select Constraint ** "))
    (BORDER-COLOR :VALUE :CYAN)
    (ANSWER :WHEN-MODIFIED (ACTIVATE-POPUP))
    (TOP :VALUE 8)
    (LEFT :VALUE 50)
    (CENTER :VALUE :X)
    (CONTENTS :VALUE
      ( (" CANCEL " :CANCEL)))
    (FORCE-CHOICE :VALUE :YES)))
+++++
(MAKE-INSTANCE 'CHOOSE-SYSTEM
  :print-name "CHOOSE-SYSTEM"
  :doc-string ""
  :is 'POPUP-CHOOSE
  :slots
  '(
    (INSTRUCTIONS :VALUE
      (" ** Select System ** "))
    (BORDER-COLOR :VALUE :CYAN)
    (ANSWER :WHEN-MODIFIED (ACTIVATE-POPUP))
    (TOP :VALUE 8)
    (LEFT :VALUE 50)
    (CENTER :VALUE :X)
    (CONTENTS :VALUE
      ( (" CANCEL " :CANCEL)))
    (FORCE-CHOICE :VALUE :YES)))
+++++
(MAKE-INSTANCE 'CHOOSE-MONITOR
  :print-name "CHOOSE-MONITOR"
  :doc-string ""
  :is 'POPUP-CHOOSE
  :slots
  '(
    (INSTRUCTIONS :VALUE
      (" ** Select Monitor ** "))
    (BORDER-COLOR :VALUE :CYAN))
```

```
(ANSWER :WHEN-MODIFIED (ACTIVATE-POPUP))
(TOP :VALUE 8)
(LEFT :VALUE 50)
(CENTER :VALUE :X)
(CONTENTS :VALUE
  (" CANCEL " :CANCEL)))
(FORCE-CHOICE :VALUE :YES)))
```

+++++

```
(MAKE-INSTANCE 'CHOOSE-POTENTIAL-CAUSE
  :print-name "CHOOSE-POTENTIAL-CAUSE"
  :doc-string ""
  :is 'POPUP-CHOOSE
  :slots
  '(
    (INSTRUCTIONS :VALUE
      (" ** Select Cause ** "))
    (BORDER-COLOR :VALUE :CYAN)
    (ANSWER :WHEN-MODIFIED (ACTIVATE-POPUP))
    (TOP :VALUE 8)
    (LEFT :VALUE 50)
    (CENTER :VALUE :X)
    (CONTENTS :VALUE
      (" CANCEL " :CANCEL)))
    (FORCE-CHOICE :VALUE :YES)))
```

+++++

```
(MAKE-INSTANCE 'CHOOSE-LINK
  :print-name "CHOOSE-LINK"
  :doc-string ""
  :is 'POPUP-CHOOSE
  :slots
  '(
    (INSTRUCTIONS :VALUE
      (" ** Select Link ** "))
    (BORDER-COLOR :VALUE :CYAN)
    (ANSWER :WHEN-MODIFIED (ACTIVATE-POPUP))
    (TOP :VALUE 8)
    (LEFT :VALUE 50)
    (CENTER :VALUE :X)
    (CONTENTS :VALUE
      (" CANCEL " :CANCEL)))
    (FORCE-CHOICE :VALUE :YES)))
```

```
+++++
(MAKE-INSTANCE 'CHOOSE-NEXT-EVENT
  :print-name "CHOOSE-NEXT-EVENT"
  :doc-string ""
  :is 'POPUP-CHOOSE
  :slots
  '(
    (INSTRUCTIONS :VALUE
      ("      ** Enter Next Event **      "))
    (BORDER-COLOR :VALUE :CYAN)
    (ANSWER :WHEN-MODIFIED (ACTIVATE-CHOSEN-EVENT))
    (TOP :VALUE 8)
    (LEFT :VALUE 32)
    (CENTER :VALUE :NO-CENTERING)
    (CONTENTS :VALUE
      ( ("          CANCEL          " :CANCEL)
        (" REACTOR_TEMPERATURE      : NORMAL --> HIGH  " EV-1)
        (" REACTOR_TEMPERATURE      : NORMAL --> LOW   " EV-2)
        .
        .
        ("MASS BALANCE VIOLATED    : NORMAL --> HIGH  " EV-113)
        ("MASS BALANCE VIOLATED    : NORMAL --> LOW   " EV-113)
      )))
    (FORCE-CHOICE :VALUE :YES)))
```

A2. APPENDIX 2: Jacketed CSTR Simulation

This appendix provides detailed information on the dynamic simulator used to generate data for the MIDAS Case Study.

Table A2-1 provides the actual extents and random seeds used to generate case data. These values could be used in conjunction with the program listing to create exact duplicates of the data files used in the case study. Note that the Jacketed CSTR Simulation program makes three external function calls to GGNML, GGUBS, and SIMPLOT. GGNML and GGUBS are IMSL random number generator routines. SIMPLOT is a custom plotting routine. The SIMPLOT call can be removed with no effect on the performance of the simulator.

In the simulation, fluid flow is computed using the formula

$$\text{FLOW} = \left[\frac{\Delta P}{R} \right]^{0.5} \quad (\text{A2-1})$$

where ΔP is the pressure differential and R is the pipe resistance. Complex pipe networks can be decomposed into serial and parallel flow paths, shown in figure A2-1. The formulas used to compute the combined resistance of multiple paths are presented in equation (A2-2) and equation (A2-3). Implicit in the use of these equations is the assumption that negligible delay exists in all flow paths.

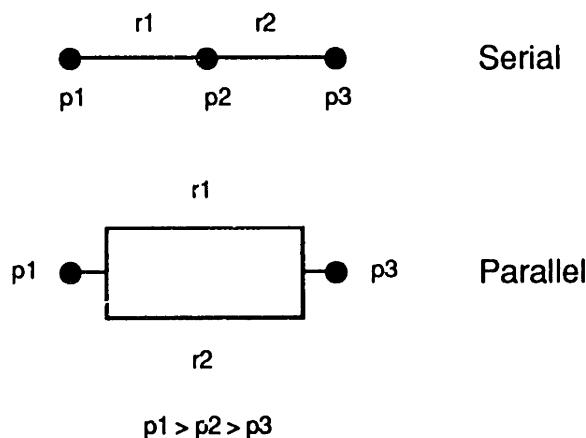


Figure A2-1: Serial and Parallel Flow Paths

TABLE A2-1: Values used for Case Study Fault Simulations

Case	Extent	Seed	Comments	Case	Extent	Seed	Comments
1	575.0	54748411	Shutdown @ 46.80 min.	33	-15.0	874626	
2	59880.0	44714974		34	54720.0	63612381	
3	21.6	7581471		35	0.084	381921	
4	48.0	95758183		36	0.304	2828317	
5	25600.0	3237137		37	-0.425	6453417	
6	0.86	4714831		38	-0.015	888121	
7	35.8	12383121		39	230.0	347182	
8	0.165	38219391		40	400.0	377317	
9	-2.75	8484841		41	33600.0	112931	
10	51300.0	4656811		42	43.0	72477812	
11	15.0	56741831		43	70.0	554161	
12	7.5	8748182		44	-10.0	67384282	
13	27.5	6416261		45	74.7	979384	
14	-10.0	33372731		46	21.0	3456123	
15	1.58	3832195		47	1795.0	774721	
16	200.0	718684		48	-0.1	1118231	
17	520.0	1113821	Shutdown @ 39.62 min.	49	99.0	46467712	
18	-8800.0	546748		50	-0.049	73482221	
19	1400.0	7617322		51	0.072	882381	
20	6.8	4747128	Shutdown @ 55.52 min.	52	1000.0	3383122	Shutdown @ 37.20 min.
21	455.0	762382		53	25250.0	4546723	
22	0.5	287371		54	1600.0	6754721	
23	-10.0	8736312		55	-980.0	3379391	
24	0.11	5454521		56	28000.0	9989283	
25	83.0	2617231	Shutdown @ 25.42 min.	57	24.6	2873181	
26	0.176	7517273		58	110.0	56382923	
27	-0.009	45231721		59	52620.0	87899232	
28	2.8	1389392		60	-0.1	25123712	
29	0.0	28859841		61	3.8	782347	
30	28.9	92838775		62	34.8	3672131	
31	220.0	5657722		63	200.0	55436723	
32	58.3	388489	Shutdown @ 27.54 min.	64	-0.016	878872	

TABLE A2-1: Values used for Case Study Fault Simulations (continued)

Case	Extent	Seed	Comments	Case	Extent	Seed	Comments
65	1200.0	4732312		80	-0.70	9547372	
66	18.0	6656732	Shutdown @ 24.24 min.	81	-0.134	17854723	
67	0.05	884251523		82	-0.315	77230001	
68	127.0	9948281		83	25250.0	4937582	
69	-0.088	11125361		84	83.0	6523428	
70	-8369.0	6773472		85	2583.0	7346827	
71	5.0	4723492		86	4.0	46532809	
72	-1274.0	542732		jlr	10000.0	1.0	777324672
73	0.255	16371991		fft	0.20 (15) 0.25 (25)	0.1 100.0	36707181
74	4.28	7732451		isf	80000.0 (15) 56250.0 (25) -200.0 (16)	100.0 1.0 100.0	65734882
75	18.0	821731					
76	0.265	701237935					
77	-1.0	6527190	Shutdown @ 15.04 min.				
78	84.0	28108273	Shutdown @ 25.18 min.				
79	-40.0	873915					

Note: Case numbers do not correspond to fault numbers in Table 6-3.
 Faults 3, 4, 9, 24, 46, 70, 75, 77, 82, 90, 91, 97, and 100 were not simulated.

$$R_{\text{serial}} = r_1 + r_2 \quad (\text{A2-2})$$

$$R_{\text{parallel}} = \frac{1}{(r_1^{-1} + r_2^{-1})} \quad (\text{A2-3})$$

Figure A2-2 diagrammatically defines the variables used in the simulation program. Note that all pipe resistances are modeled at point resistances (indicated by darkened valve symbols).

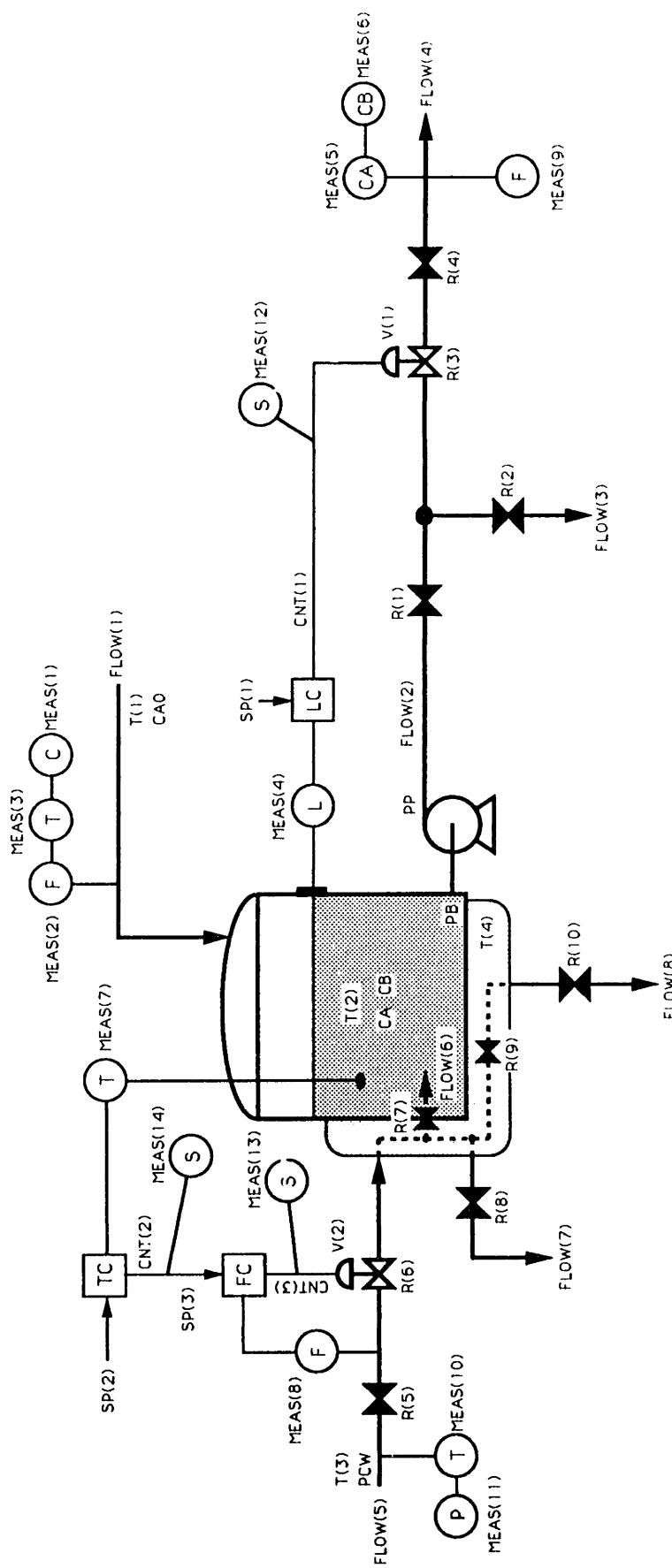


Figure A2-2: Model Diagram for Jacketed CSTR Process

Jacketed CSTR Simulation Program:

```

C23456789      ** JACKETED CSTR DYNAMIC SIMULATION PROGRAM **

C
C      F. ERIC FINCH
C      DEPARTMENT OF CHEMICAL ENGINEERING
C      MASSACHUSETTS INSTITUTE OF TECHNOLOGY
C      CAMBRIDGE, MA 02139
C
C      REVISED 5/89
C
C      ** VARIABLE DEFINITIONS **
C
C      ALPHA1      - PRIMARY RATE CONSTANT PHE-EXPONENTIAL FACTOR (1/MIN)
C      BETA1       - PRIMARY ACTIVATION ENERGY (KJ/KMOL)
C      ALPHA2      - SECONDARY RATE CONSTANT PRE-EXPONENTIAL FACTOR (1/MIN)
C      BETA2       - SECONDARY ACTIVATION ENERGY (KJ/KMOL)
C      B(3,3)      - ARRAY OF CONTROLLER CONSTANTS FOR PID CONTROL
C                      (IF B(I,2)=0 & B(I,3)=0 THEN P CONTROL,
C                      IF B(I,3)=0 THEN PI CONTROL)
C      CA0         - FEED CONCENTRATION (A) (KMOL/M3)
C      CA          - REACTOR CONCENTRATION (A) (KMOL/M3)
C      CB          - REACTOR CONCENTRATION (B) (KMOL/M3)
C      CC          - REACTOR CONCENTRATION (C) (KMOL/M3)
C      CNT(3)     - CONTROLLER OUTPUT VECTOR
C      CP,CPOLD   - HEAT CAPACITY OF REACTOR CONTENTS (KJ/KG C)
C      CP1         - FEED HEAT CAPACITY (KJ/KG C)
C      CP2         - COOLING WATER HEAT CAPACITY (KJ/KG C)
C      CWPD        - COOLING WATER PRESSURE DROP CONSTRAINT RESIDUAL (M3/MIN)
C      DATAFILE    - FILE NAME FOR OUTPUT
C      DAY         - DAY STAMP
C      DELAY(3)    - DELAY UNTIL FAULT INITIATION (MIN)
C      DERROR(3)   - VECTOR OF DIFFERENCES BETWEEN ERROR AT
C                      SUCCESSIVE TIME STEPS
C      DEXT(3)     - FAULT EXTENT DIFFERENCE AT CURRENT TIME STEP
C      DEXT0(3)    - DIFFERENCE BETWEEN FINAL EXTENT AND INITIAL
C                      EXTENT OF VARIABLE AFFECTED BY FAULT
C      DSEED       - SEED FOR GAUSSIAN RANDOM NUMBER GENERATOR
C      DT          - TIME INCREMENT OF MAIN LOOP (MIN)
C      ERROR(3)    - VECTOR OF CONTROLLER ERRORS
C      EXTENT0(3)  - ULTIMATE FAULT EXTENT (CONTEXT DEPENDENT!!)
C      EPD         - EFFLUENT PRESSURE DROP CONSTRAINT RESIDUAL (M3/MIN)
C      F(3)         - FAULT CODES (CAN HAVE UP TO 3 SIMULTANEOUS FAULTS)
C      FF          - ADJUSTABLE PARAMETER
C      FINTEG     - INTEGRAL OF FLOW RESIDUAL (M3)
C      FLOW(8)     - VECTOR OF PROCESS FLOWRATES (M3/MIN)
C      FMODE(2)    - CHARACTER ARRAY OF SENSOR FAILURE MODES
C      FTTYPE(19)  - CHARACTER ARRAY OF FAULT TYPES
C      HOUR        - HOUR STAMP
C      ISEED       - SEED FOR RANDOM NUMBER GENERATOR
C      LEVEL       - CSTR LEVEL (M)
C      M           - NUMBER OF FAULTS TO BE SIMULATED
C      MASSBAL    - RESIDUAL OF INVENTORY CONSTRAINT (M3)

```

C MEAS(18,2) - ARRAY OF PROCESS MEASUREMENTS; COL 2 IS
 C MOST RECENT, COL1 IS EWMA HISTORY
 C MINUTE - MINUTE STAMP
 C MOLBAL - MOL BALANCE CONSTRAINT RESIDUAL (KMOL)
 C MOLIN - CUMULATIVE MOL INFLUX (KMOL)
 C MOLOUT - CUMULATIVE MOL OUTFLUX (KMOL)
 C MON - MONTH STAMP
 C N - TOTAL NUMBER OF SENSORS AND CONSTRAINTS
 C NORMVAL(18) - ARRAY OF NORMAL STEADY STATE PROCESS MEASUREMENTS
 C PB,PBCOMP - PRESSURE AT TANK OUTLET (KG/M²)
 C PCW - COOLING WATER SUPPLY PRESSURE (KG/M²)
 C PP,PP0 - PUMP DIFFERENTIAL PRESSURE (KG/M²)
 C QEXT - EXTERNAL HEAT SOURCE(SINK) TO CSTR (KJ/MIN)
 C QJAC - HEAT TRANSFERED IN HEAT EXCHANGER (KJ/MIN)
 C QREAC1 - PRIMARY HEAT OF REACTION (KJ/KMOL)
 C QREAC2 - SECONDARY HEAT OF REACTION (KJ/K.MOL)
 C R(10) - VECTOR OF FLOW RESISTANCES (MIN KG^{0.5}/M⁴)
 C RAND(3) - RANDOM NUMBER VECTOR
 C RCOMP(10) - COMPUTED VALUES FOR FLOW RESISTANCE
 C BASED ON SENSOR MEASUREMENT DATA
 C REG1(3),
 C REG2(3) - STORAGE REGISTERS FOR PAST CONTROLLER ERRORS
 C RHO,RHOLD - DENSITY OF REACTOR CONTENTS (KG/M³)
 C RHO1 - DENSITY OF FEED (KG/M³)
 C RHO2 - DENSITY OF COOLING WATER (KG/M³)
 C R0(10) - NOMINAL VALUES FOR FLOW RESISTANCE (MIN KG^{0.5}/M⁴)
 C RRATE1 - RATE OF PRIMARY REACTION IN CSTR (KMOL/M³ MIN)
 C RRATE2 - RATE OF SECONDARY REACTION IN CSTR (KMOL/M³ MIN)
 C RC - OVERALL RESISTANCE FOR CW STREAM (MIN KG^{0.5}/M⁴)
 C RE - OVERALL RESISTANCE FOR EFFLUENT STREAM (MIN KG^{0.5}/M⁴)
 C S0,S1,S2,S3,
 C S4,S5,S6 - PROGRAM CONTROL PARAMETERS
 C SDEV(18) - STANDARD DEVIATION OF RANDOM SENSOR NOISE
 C SEC - SECOND STAMP
 C SENSORS(18) - CHARACTER ARRAY OF SENSOR NAMES
 C SP(3) - VECTOR OF CONTROLLER SETPOINTS
 C T(4) - VECTOR OF PROCESS TEMPERATURES (C)
 C TC(3) - EXPONENTIAL TIME CONSTANT FOR FAULT EXTENT
 C TAREA - FLOOR AREA OF CSTR (M²)
 C TEMP - A TEMPORARY STORAGE REGISTER
 C TH - TIME HORIZON OF SIMULATION (MIN)
 C THETA - EWMA (EXP. FILTER) PARAMETER
 C TIME - SIMULATION TIME (MIN)
 C TUNE - CONTROLLER TUNING LOGICAL CONTROL
 C TUNEI,TUNEJ - CONTROLLER TUNING ARRAY ELEMENTS
 C UA - HEAT EXCHANGER CONSTANT (KJ/MIN C)
 C UNITS(18) - CHARACTER ARRAY OF SENSOR UNITS
 C V(2) - VECTOR OF CONTROL VALVE POSITIONS [0,100]
 C VOL,VOLD - CSTR LIQUID VOLUME (M³)
 C YEAR - YEAR STAMP
 C ZCOUNT - COUNTER FOR PRINTING OUTPUT
 C ZLIM - PRINT INTERVAL (MIN)
 C
 C ** BEGIN PROGRAM **

```

C
C  DECLARE AND DIMENSION ALL VARIABLES
C
C  REAL VARIABLES (ALL DOUBLE PRECISION)
C
      REAL *8  ALPHA1,ALPHA2,BETA1,BETA2,B(3,3),CA0,CA,CB,CC,CNT(3),
1      CP,CPOLD,CP1,CP2,CWPD,DELAY(3),DERROR(3),
1      DEXT(3),DEXT0(3),DSEED,DT,EPD,
1      ERROR(3),EXTENT0(3),FF,FINTEG,FLOW(8),JEP,
1      LEVEL,MASSBAL,MEAS(18,2),MOLBAL,MOLIN,MOLOUT,
1      NORMVAL(18),PB,PBCOMP,
1      PCW,PP,PP0,QEXT,QJAC,QREAC1,QREAC2,R(10),RAND(3),
1      RCOMP(10),REG1(3),REG2(3),REP,RHO,RHOLD,RHO1,RHO2,
1      R0(10),RRATE1,RRATE2,RC,RE,RV(18),SP(3),SDEV(18),T(4),
1      TAREA,TC(3),TEMP,TH,THETA,TIME,UA,V(2),VOLD,ZLIM
C
C  INTEGER VARIABLES
C
      INTEGER *2  DAY,F(3),HOUR,MINUTE,M,MON,RO(18,2),SAMP(3,23),S0,
1      S1(3),S2(3),S3,S4,S5,S6,S7,S8,SEC,SS,TUNE,TUNEI,
1      TUNEJ
      INTEGER *4  N,YEAR,ZCOUNT
      INTEGER *4  ISEED
C
C  CHARACTER VARIABLES
C
      CHARACTER *8  UNITS(18)
      CHARACTER *12  FMODE(2)
      CHARACTER *32  SENSORS(18),DATAFILE
      CHARACTER *40  FTYPE(23)
C
      5  CONTINUE
C
C  INITIALIZE CHARACTER DATA
C
C
C  INITIALIZE FAULT NAMES
C
      DATA FTYPE /NO FAULT
1      'BLOCKAGE AT TANK OUTLET
1      'BLOCKAGE IN JACKET
1      'JACKET LEAK TO ENVIRONMENT
1      'JACKET LEAK TO TANK
1      'LEAK FROM PUMP
1      'LOSS OF PUMP PRESSURE
1      'JACKET EXCHANGE SURFACE FOULING
1      'EXTERNAL HEAT SOURCE (SINK)
1      'PRIMARY REACTION ACTIVATION ENERGY
1      'SECONDARY REACTION ACTIVATION ENERGY
1      'ABNORMAL FEED FLOWRATE
1      'ABNORMAL FEED TEMPERATURE
1      'ABNORMAL FEED CONCENTRATION
1      'ABNORMAL COOLING WATER TEMPERATURE
1      'ABNORMAL COOLING WATER PRESSURE

```

```

1      'ABNORMAL JACKET EFFLUENT PRESSURE
1      'ABNORMAL REACTOR EFFLUENT PRESSURE
1      'ABNORMAL LEVEL CONTROLLER SETPOINT
1      'ABNORMAL TEMPERATURE CONTROLLER SETPOINT
1      'CONTROL VALVE (CV-1) STUCK
1      'CONTROL VALVE (CV-2) STUCK
1      'SENSOR FAULT      '/

C
C  INITIALIZE SENSOR NAMES
C
DATA  SENSORS  /'FEED_CONCENTRATION_SENSOR
1          'FEED_FLOWRATE_SENSOR
1          'FEED_TEMPERATURE_SENSOR
1          'REACTOR_LEVEL_SENSOR
1          'CONCENTRATION_A_SENSOR
1          'CONCENTRATION_B_SENSOR
1          'REACTOR_TEMPERATURE_SENSOR
1          'COOLING_WATER_FLOWRATE_SENSOR
1          'PRODUCT_FLOWRATE_SENSOR
1          'COOLING_WATER_TEMPERATURE_SENSOR
1          'COOLING_WATER_PRESSURE_SENSOR
1          'LEVEL_CONTROLLER_OUTPUT_SIGNAL
1          'CW_FLOW_CONTROLLER_OUTPUT_SIGNAL
1          'COOLING_WATER_SETPOINT
1          'INVENTORY_CONSTRAINT
1          'CW_PRESSURE_DROP_CONSTRAINT
1          'EFFL_PRESSURE_DROP_CONSTRAINT
1          'MOL_BALANCE_CONSTRAINT      '/

C
C  INITIALIZE SENSOR UNITS
C
DATA  UNITS  /'KMOL/M3
1          'M3/MIN
1          'C
1          'M
1          'KMOL/M3
1          'KMOL/M3
1          'C
1          'M3/MIN
1          'M3/MIN
1          'C
1          'KG/M2
1          '% OPEN
1          '% OPEN
1          'M3/MIN
1          'M3
1          'M3/MIN
1          'M3/MIN
1          'KMOL      '/

C
C  INITIALIZE SENSOR FAULT MODE NAMES
C
DATA FMODE /'FIXED BIAS ','FIXED VALUE '
C

```



```

CB      =    17.114
CC      =    0.0226
CP      =    4.20
CP1     =    4.20
CP2     =    4.20
DT      =    0.02
FF      =    0.10
FINTEG  =    0.0
JEP     =    0.0
MOLIN   =    0.0
MOLOUT  =    0.0
N       =    18
PB      =  2000.0
PCW     = 56250.0
PP      = 48000.0
PPO     = 48000.0
QREAC1  = 30000.0
QREAC2  = -10000.0
QEXT    =    0.0
REP     =    0.0
RHO     = 1000.0
RHO1    = 1000.0
RHO2    = 1000.0
S0      =    0
S3      =    0
S4      =    0
S7      =    0
S8      =    0
TAREA   =    1.5
TIME    =    0.0
UA      = 1901.0
VOL     =    3.0

C
C PLOT QUERY
C
C      WRITE (*,800)
C      WRITE (*,735)
C      READ (*,*) S6
C      IF (S6 .EQ. 1) GOTO 550
C
C      OPEN OUTPUT FILE FOR RAW DATA DUMP (MIDAS FORMAT)
C
C      WRITE (*,700)
C      READ (*,710) DATAFILE
C      IF (DATAFILE .EQ. ".") THEN
C          DATAFILE='DUMP.RAW'
C      ENDIF
C
C      OPEN (UNIT=14,FILE=DATAFILE)
C
C      ENTER DATE STAMP DATA
C
C      WRITE (*,720)
C      READ (*,730) MON,DAY,YEAR

```

```

MON=6
DAY=1
YEAR=1989
C
C CONTROLLER TUNING (OPTIONAL)
C
600 CONTINUE
  WRITE (*,740)
  READ (*,*) TUNE
  IF (TUNE .EQ. 0) GOTO 650
610  WRITE (*,745)
  READ (*,*) TUNEI
  IF ((TUNEI .GT. 4) .OR. (TUNEI .LT. 1)) GOTO 610
620  WRITE (*,750)
  READ (*,*) TUNEJ
  IF ((TUNEJ .GT. 3) .OR. (TUNEJ .LT. 1)) GOTO 620
  WRITE (*,760) B(TUNEI,TUNEJ)
  READ (*,*) B(TUNEI,TUNEJ)
  GOTO 600
650 CONTINUE
C
C ENTER EXPONENTIAL FILTER CONSTANT (1=NO EWMA)
C
660  WRITE (*,770)
  READ (*,*) THETA
  IF ((THETA .LE. 0.0) .OR. (THETA .GT. 1.0)) GOTO 660
C
C ENTER PRINT FORMAT
C
  WRITE (*,772)
  READ (*,*) S7
  WRITE (*,773)
  READ (*,*) S8
C
C COMPUTE SEED FOR RANDOM NUMBER GENERATOR
C
  WRITE (*,790)
  READ (*,*) ISEED
  DSEED=DBLE(ISEED)
C
C ENTER FAULT INFORMATION
C
20  CONTINUE
  DO 30 I=1,23
    WRITE(*,810) I,FTYPE(I)
30  CONTINUE
C
35  S0=S0 + 1
  WRITE (*,820)
  READ (*,*) F(S0)
  DO 37 I=1,(S0-1)
  IF (F(I) .EQ. F(S0)) THEN
    WRITE (*,822)
    WRITE (*,823)

```

```

        READ (*,*) S6
        IF (S6 .EQ. 1) GOTO 37
        S0=S0 - 1
        GOTO 40
37    ENDIF
        IF (F(S0) .EQ. 1) GOTO 40
        IF (F(S0) .EQ. 23) THEN
            WRITE (*,821) (I,SENSORS(I),I=1,14)
            WRITE (*,825)
            READ (*,*) S1(S0)
            WRITE (*,830)
            READ (*,*) S2(S0)
            WRITE (*,*) 'NOMINAL VALUE IS : ',MEAS(S1(S0),1)
        ELSE
            IF (F(S0) .EQ. 2) WRITE (*,835) R(1)
            IF (F(S0) .EQ. 3) WRITE (*,835) R(9)
            IF (F(S0) .EQ. 4) WRITE (*,835) R(8)
            IF (F(S0) .EQ. 5) WRITE (*,835) R(7)
            IF (F(S0) .EQ. 6) WRITE (*,835) R(2)
            IF (F(S0) .EQ. 7) WRITE (*,835) PP
            IF (F(S0) .EQ. 8) WRITE (*,835) UA
            IF (F(S0) .EQ. 9) WRITE (*,835) QEXT
            IF (F(S0) .EQ. 10) WRITE (*,835) BETA1
            IF (F(S0) .EQ. 11) WRITE (*,835) BETA2
            IF (F(S0) .EQ. 12) WRITE (*,835) FLOW(1)
            IF (F(S0) .EQ. 13) WRITE (*,835) T(1)
            IF (F(S0) .EQ. 14) WRITE (*,835) CA0
            IF (F(S0) .EQ. 15) WRITE (*,835) T(3)
            IF (F(S0) .EQ. 16) WRITE (*,835) PCW
            IF (F(S0) .EQ. 17) WRITE (*,835) JEP
            IF (F(S0) .EQ. 18) WRITE (*,835) REP
            IF (F(S0) .EQ. 19) WRITE (*,835) SP(1)
            IF (F(S0) .EQ. 20) WRITE (*,835) SP(2)
            IF (F(S0) .EQ. 21) WRITE (*,835) 100.0 - V(1)
            IF (F(S0) .EQ. 22) WRITE (*,835) 100.0 - V(2)
        ENDIF
        WRITE (*,840)
        READ (*,*) EXTENT0(S0)
        WRITE (*,850)
        READ (*,*) DELAY(S0)
C      IF (F(S0) .GE. 20) GOTO 40
        WRITE (*,860)
        READ (*,*) TC(S0)
40    IF (S0 .GE. 3) GOTO 42
        WRITE (*,865)
        READ (*,*) S6
        IF (S6 .EQ. 1) GOTO 20
42    M=S0
        WRITE (*,870)
        READ (*,*) TH
        WRITE (*,873)
        READ (*,*) ZLIM
        IF (ZLIM .EQ. 0.0) THEN
            ZLIM=TH

```

```

        GOTO 45
      ENDIF
      WRITE (*,875)
      READ (*,*) ZLIM
45   ZLIM=ZLIM/DT
      ZCOUNT=IDNINT(ZLIM)
      DO 47 S0=1,M
      IF (F(S0) .EQ. 23) THEN
        WRITE (*,880) FTYPE(F(S0)),SENSORS(S1(S0)),FMODE(S2(S0)+1),
1      EXTENT0(S0),DELAY(S0)
      ELSE
        WRITE (*,890) FTYPE(F(S0)),EXTENT0(S0),DELAY(S0),TC(S0)
      ENDIF
47   CONTINUE
      WRITE (*,895) THETA
      WRITE (*,950)
C
C   TOP OF MAIN ITERATION LOOP
C
50   CONTINUE
C
C   INTRODUCE FAULT
C
      DO 60 S0=1,M
      IF ((F(S0) .EQ. 1).OR.(TIME .LT. DELAY(S0))) GOTO 60
C
C   CALCULATE EXTENT DIFFERENTIAL
C
      IF (DEXT0(S0) .EQ. 0.0) THEN
        IF (F(S0) .EQ. 2) DEXT0(S0)=EXTENT0(S0) - R(1)
        IF (F(S0) .EQ. 3) DEXT0(S0)=EXTENT0(S0) - R(9)
        IF (F(S0) .EQ. 4) DEXT0(S0)=EXTENT0(S0) - R(8)
        IF (F(S0) .EQ. 5) DEXT0(S0)=EXTENT0(S0) - R(7)
        IF (F(S0) .EQ. 6) DEXT0(S0)=EXTENT0(S0) - R(2)
        IF (F(S0) .EQ. 7) DEXT0(S0)=EXTENT0(S0) - PP
        IF (F(S0) .EQ. 8) DEXT0(S0)=EXTENT0(S0) - UA
        IF (F(S0) .EQ. 9) DEXT0(S0)=EXTENT0(S0) - QEXT
        IF (F(S0) .EQ. 10) DEXT0(S0)=EXTENT0(S0) - BETA1
        IF (F(S0) .EQ. 11) DEXT0(S0)=EXTENT0(S0) - BETA2
        IF (F(S0) .EQ. 12) DEXT0(S0)=EXTENT0(S0) - FLOW(1)
        IF (F(S0) .EQ. 13) DEXT0(S0)=EXTENT0(S0) - T(1)
        IF (F(S0) .EQ. 14) DEXT0(S0)=EXTENT0(S0) - CA0
        IF (F(S0) .EQ. 15) DEXT0(S0)=EXTENT0(S0) - T(3)
        IF (F(S0) .EQ. 16) DEXT0(S0)=EXTENT0(S0) - PCW
        IF (F(S0) .EQ. 17) DEXT0(S0)=EXTENT0(S0) - JEP
        IF (F(S0) .EQ. 18) DEXT0(S0)=EXTENT0(S0) - REP
        IF (F(S0) .EQ. 19) DEXT0(S0)=EXTENT0(S0) - SP(1)
        IF (F(S0) .EQ. 20) DEXT0(S0)=EXTENT0(S0) - SP(2)
      ENDIF
C
C   CALCULATE FAULT EXTENT EXPONENTIAL GROWTH FUNCTION
C
      DEXT(S0)=EXP(TC(S0)*(DELAY(S0)-TIME))
C

```

```

C   CALCULATE FAULT EXTENT
C
IF (F(S0) .EQ. 2) R(1)=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 3) R(9)=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 4) R(8)=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 5) R(7)=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 6) R(2)=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 7) PP=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 8) UA=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 9) QEXT=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 10) BETA1=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 11) BETA2=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 12) FLOW(1)=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 13) T(1)=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 14) CA0=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 15) T(3)=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 16) PCW=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 17) JEP=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 18) REP=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 19) SP(1)=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)
IF (F(S0) .EQ. 20) SP(2)=EXTENT0(S0) - DEXT0(S0)*DEXT(S0)

C
60  CONTINUE
C
C   CALCULATE CONTROLLER OUTPUTS
C
100 CONTINUE
C
C   LEVEL CONTROLLER
C
REG2(1)=REG1(1)
REG1(1)=DERROR(1)
DERROR(1)=ERROR(1) - (SP(1) - MEAS(4,1))
ERROR(1)=ERROR(1) - DERROR(1)
CNT(1)=MIN(100.0,MAX(0.0,CNT(1)-B(1,1)*DERROR(1)
1  +B(1,2)*(0.5*DERROR(1)+ERROR(1))*DT
2  +B(1,3)*(2.0*REG1(1)-0.5*REG2(1)-1.5*DERROR(1))/DT))

C
C   REACTOR TEMPERATURE CONTROLLER
C
REG2(2)=REG1(2)
REG1(2)=DERROR(2)
DERROR(2)=ERROR(2) - (SP(2) - MEAS(7,1))
ERROR(2)=ERROR(2) - DERROR(2)
CNT(2)=MAX(0.0,CNT(2)-B(2,1)*DERROR(2)
1  +B(2,2)*(0.5*DERROR(2)+ERROR(2))*DT
2  +B(2,3)*(2.0*REG1(2)-0.5*REG2(2)-1.5*DERROR(2))/DT)

C
C   COOLING WATER FLOW CONTROLLER
C
REG2(3)=REG1(3)
REG1(3)=DERROR(3)
DERROR(3)=ERROR(3) - (SP(3) - MEAS(8,1))
ERROR(3)=ERROR(3) - DERROR(3)

```

```

      CNT(3)=MIN(100.0,MAX(0.0,CNT(3)-B(3,1)*DERROR(3)
1   +B(3,2)*(0.5*DERROR(3)+ERROR(3))*DT
2   +B(3,3)*(2.0*REG1(3)-0.5*REG2(3)-1.5*DERROR(3))/DT))
C
C   EVALUATE SAFETY SYSTEMS
C
IF ((MEAS(4,1) .GE. 2.75).OR.(MEAS(7,1) .GE. 130.0)) THEN
  IF (S3 .EQ. 1) GOTO 150
  DO 110 S0=1,M
    IF (F(S0) .EQ. 12) THEN
      EXTENT0(S0)=0.0
      DEXT0(S0)=0.0
110  ENDIF
    FLOW(1)=0.0
    S3=1
    WRITE (*,970) TIME
  ELSEIF (MEAS(4,1) .LE. 1.2) THEN
    IF (S4 .EQ. 1) GOTO 150
    DO 120 S0=1,M
      IF (F(S0) .EQ. 7) THEN
        EXTENT0(S0)=0.0
        DEXT0(S0)=0.0
120  ENDIF
    PP=0.0
    S4=1
    WRITE (*,980) TIME
  ENDIF
C
C   CALCULATE VALVE POSITIONS
C
150 DO 160 S0=1,M
  IF ((F(S0) .EQ. 21) .AND. (TIME .GE. DELAY(S0))) THEN
    V(1)=100.0 - EXTENT0(S0)*(1.0-DEXT(S0))
  ELSE
    V(1)=MIN(100.0,MAX(0.0,100.0-MEAS(12,2)))
  ENDIF
  IF ((F(S0) .EQ. 22) .AND. (TIME .GE. DELAY(S0))) THEN
    V(2)=100.0 - EXTENT0(S0)*(1.0-DEXT(S0))
  ELSE
    V(2)=MIN(100.0,MAX(0.0,100.0-MEAS(13,2)))
  ENDIF
160 CONTINUE
C
C   CALCULATE FLOWRATES
C
  R(3)=5.0 * EXP(0.0545*V(1))
  R(6)=5.0 * EXP(0.0545*V(2))
C
  RE=(1/((1/R(2))+(1/(R(3)+R(4)))))+R(1)
  RC=(1/((1/R(7))+(1/R(8))+(1/(R(9)+R(10)))))+R(5)+R(6)
C
C   NOTE: THESE FORMULAS WILL NOT WORK WELL IF BOTH AN
C   ABNORMAL EFFLUENT PRESSURE AND A LEAK ARE SIMULATED
C

```

```

FLOW(2)=(1/RE)*(PP+PB-REP)**0.5
FLOW(3)=(1/R(2))*((PP+PB-REP)-(FLOW(2)*R(1))**2.0)**0.5
FLOW(4)=FLOW(2)-FLOW(3)
C
C   FLOW(5)=(1/RC)*(PCW-JEP)**0.5
C   FLOW(6)=(1/R(7))*(PCW-JEP-(FLOW(5)*(R(5)+R(6)))**2.0)**0.5
C   FLOW(7)=(1/R(8))*(PCW-JEP-(FLOW(5)*(R(5)+R(6)))**2.0)**0.5
C   FLOW(8)=FLOW(5)-FLOW(6)-FLOW(7)
C
C   CALCULATE JACKET TEMPERATURE
C
C   T(4)=((U.1*T(2)+RHO2*CP2*FLOW(8)*T(3))/(UA+RHO2*CP2*FLOW(8)))
C
C   CALCULATE HEAT FLUX
C
C   QJAC=UA*(T(2)-T(4))
C
C   CALCULATE CSTR VARIABLES
C
C   LEVEL/VOLUME/DENSITY/HEAT CAPACITY
C
220  VOLD=VOL
      RHOLD=RHO
      CPOLD=CP
C
C   VOL=VOLD+(FLOW(1)+FLOW(6)-FLOW(2))*DT
C   RHO=(1/VOL)*(VOLD*RHO)+  

1  (1/VOL)*(DT*(FLOW(1)*RHO1+FLOW(6)*RHO2-FLOW(2)*RHO))
C
C   CP=(1/VOL)*(VOLD*CP)+  

1  (1/VOL)*(DT*(FLOW(1)*CP1+FLOW(6)*CP2-FLOW(2)*CP))
C
C   LEVEL=VOL/TAREA
IF (LEVEL .LE. 0.0) THEN
    WRITE (*,*) 'FAILURE DUE TO LOW LEVEL'
    STOP
ENDIF
PB=RHO*LEVEL
C
C   CONCENTRATIONS
C
C   RRATE1=ALPHA1*CA*EXP(-BETA1/(8.314*(273.15+T(2))))
C   RRATE2=ALPHA2*CA*EXP(-BETA2/(8.314*(273.15+T(2))))
C
C   CA=(1/VOL)*(VOLD*CA)+  

1  (1/VOL)*(FLOW(1)*CA0-FLOW(2)*CA-RRATE1*VOLD-RRATE2*VOLD)*DT
C
C   CB=(1/VOL)*(VOLD*CB)+  

1  (1/VOL)*(RRATE1*VOLD-FLOW(2)*CB)*DT
C
C   CC=(1/VOL)*(VOLD*CC)+  

1  (1/VOL)*(RRATE2*VOLD-FLOW(2)*CC)*DT
C
C   TEMPERATURE

```

```

C
T(2)=(1/(VOL*RHO*CP))*(VOLD*RHOLD*CPOLD*T(2))+  

1 (1/(VOL*RHO*CP))*(((QREAC1*RRATE1+QREAC2*RRATE2)*VOLD)*DT)+  

1 (1/(VOL*RHO*CP))*((QEXT-QJAC)*DT)+  

1 (1/(VOL*RHO*CP))*(FLOW(1)*RHO1*CP1*T(1)*DT)+  

1 (1/(VOL*RHO*CP))*(FLOW(6)*RHO2*CP2*T(4)*DT)-  

1 (1/(VOL*RHO*CP))*(FLOW(2)*RHOLD*CPOLD*T(2)*DT)
C
C MEASURE SENSED VARIABLES
C
MEAS(1,2)=CA0
MEAS(2,2)=FLOW(1)
MEAS(3,2)=T(1)
MEAS(4,2)=LEVEL
MEAS(5,2)=CA
MEAS(6,2)=CB
MEAS(7,2)=T(2)
MEAS(8,2)=FLOW(5)
MEAS(9,2)=FLOW(4)
MEAS(10,2)=T(3)
MEAS(11,2)=PCW
MEAS(12,2)=100.0 - CNT(1)
MEAS(13,2)=100.0 - CNT(3)
MEAS(14,2)=CNT(2)
C
C MODIFY SENSOR READINGS
C
C CALL GGNML(A,B,C)
C
C GGNML IS A GAUSIAN RANDOM NUMBER GENERATOR PRODUCING A VECTOR C
C OF NORMAL (0,1) RANDOM NUMBERS OF DIMENSION B. THE SEED (A) MUST
C BE DOUBLE PRECISION.
C
DO 250 I=1,14
    CALL GGNML(DSEED,3,RAND)
DO 250 S0=1,M
    IF ((F(S0) .EQ. 23) .AND. (TIME .GE. DELAY(S0))) THEN
        IF ((I .EQ. S1(S0)) .AND. (S2(S0) .EQ. 0)) THEN
            MEAS(I,2)=MEAS(I,2)+RAND(S0)*SDEV(I)+  

1             EXTENT0(S0)*(1.0-DEXT(S0))
        ELSEIF ((I .EQ. S1(S0)) .AND. (S2(S0) .EQ. 1)) THEN
            MEAS(I,2)=EXTENT0(S0)*(1.0-DEXT(S0))
        ELSE
            MEAS(I,2)=MEAS(I,2)+RAND(S0)*SDEV(I)
        ENDIF
    ELSE
        MEAS(I,2)=MEAS(I,2)+RAND(S0)*SDEV(I)
    ENDIF
250 CONTINUE
C
C CALCULATE EXPONENTIAL WEIGHTED MOVING AVERAGE FOR USE IN CONTROLLERS
C
DO 260 I=1,14
    MEAS(I,1)=THETA*MEAS(I,2) + (1.0 - THETA)*MEAS(I,1)

```

```

260  CONTINUE
C
C   EVALUATE QUANTITATIVE CONSTRAINTS:
C
C   INVENTORY, COOLING WATER PRESSURE DROP, EFFLUENT PRESSURE DROP
C
C   INVENTORY CONSTRAINT
C
    FINTEG=FINTEG+(MEAS(2,2)-MEAS(9,2))*DT
    MASSBAL=TAREA*MEAS(4,2)-FINTEG-3.00
C
C   EFFLUENT FLOW CONSTRAINT
C
    PBCOMP=RHO1*MEAS(4,2)
    RCOMP(3)=5.0*EXP(0.0545*(100.0-MEAS(12,2)))
    EPD=MEAS(9,2)-
1   ((1/(RCOMP(3)+R0(1)+R0(4)))*(PBCOMP+PP0)**0.5)
C
C   COOLING WATER FLOW CONSTRAINT
C
    RCOMP(6)=5.0*EXP(0.0545*(100.0-MEAS(13,2)))
    CWPD=MEAS(8,2)-
1   ((1/(RCOMP(6)+R0(5)+R0(9)+R0(10)))*MEAS(11,2)**0.5)
C
C   MOL BALANCE CONSTRAINT
C
    MOLIN=MOLIN+MEAS(1,2)*MEAS(2,2)*DT
    MOLOUT=MOLOUT+(MEAS(5,2)+MEAS(6,2)+0.0226)*MEAS(9,2)*DT
    MOLBAL=(MEAS(5,2)+MEAS(6,2)+0.0226)*TAREA*MEAS(4,2)-60.0-
1   MOLIN+MOLOUT
C
C   DETERMINE VALUES OF CONSTRAINTS FROM SENSORS
C
    MEAS(15,2)=MASSBAL
    MEAS(16,2)=CWPD
    MEAS(17,2)=EPD
    MEAS(18,2)=MOLBAL
C
C   CONVERT TIME TO HOURS, MINUTES, AND SECONDS
C
    SEC=IDNINT(DFRAC(TIME)*60)
    TEMP=(TIME-DFRAC(TIME))/60
    HOUR=IDNINT(TEMP-DFRAC(TEMP))
    MINUTE=IDNINT((TIME-DFRAC(TIME))-(TEMP-DFRAC(TEMP))*60)
C
    IF (SEC .EQ. 60) THEN
      SEC=0
      MINUTE=MINUTE + 1
    ENDIF
C
C   PRINT UPDATED STATUS
C
    IF((ZCOUNT .EQ. IDNINT(ZLIM/3.0)).AND.(S8 .EQ. 1)) THEN
      SS=3

```

```

ELSEIF ((ZCOUNT .EQ. IDNINT(ZLIM/1.5)).AND.(S8 .EQ. 1)) THEN
SS=3
ELSEIF ((ZCOUNT .EQ. IDNINT(ZLIM/2.0)).AND.(S8 .EQ. 1)) THEN
SS=2
ELSEIF (ZCOUNT .EQ. IDNINT(ZLIM)) THEN
ZCOUNT=0
SS=1
ELSE
GOTO 500
ENDIF
C
C PRINT TO RAW DATA OUTPUT FILE
C
C LIST SENSOR READINGS IN RANDOM ORDER
C
C CALL GGUBS(A,B,C)
C
C GGUBS IS A UNIFORM RANDOM NUMBER GENERATOR PRODUCING A VECTOR C
C OF UNIFORM (0,1) RANDOM NUMBERS OF DIMENSION B. THE SEED (A) MUST
C BE DOUBLE PRECISION.
C
DO 261 J=1,18
RO(J,1)=SAMP(SS,J)
RO(J,2)=SAMP(SS,J)
261 CONTINUE
IF (S7 .EQ. 0) GOTO 266
CALL GGUBS(DSEED,18,RV)
DO 265 J=1,SAMP(SS,19)
K=IDNINT(RV(J)*18.0+0.5)
262 IF (RO(K,1) .NE. 0) THEN
RO(J,2)=RO(K,1)
RO(K,1)=0
ELSE
K=K+1
IF (K .GT. 18) K=1
GOTO 262
ENDIF
265 CONTINUE
C
C WRITE SENSOR READING TO FILE
C
266 DO 270 J=1,SAMP(SS,19)
WRITE (14,999) SENSORS(RO(J,2)),YEAR,MON,DAY,HOUR,MINUTE,SEC,
1 MEAS(RO(J,2),2),UNITS(RO(J,2))
270 CONTINUE
C
C CHECK FOR TERMINATION AND ITERATE
C
C NOTE: SIMPLOT IS A SUBPROGRAM USED CREATING DATA FILES FOR PPLOT AND
C CAN BE REMOVED WITHOUT OTHER MODIFICATIONS.
C
500 TIME=TIME + DT
ZCOUNT=ZCOUNT + 1
SP(3)=MEAS(14,2)

```

```

IF (TIME .GT. (TH + DT)) THEN
    WRITE (*,960)
    CLOSE (UNIT=14)
    WRITE (*,990)
    READ (*,*) S5
    IF (S5 .EQ. 1) GOTO 5
    WRITE (*,995)
    READ (*,*) S6
    IF (S6 .EQ. 1) THEN
        CONTINUE
550    CALL SIMPLOT(DATAFILE,N,NORMVAL,SDEV,SENSORS,UNITS)
        ENDIF
    STOP
    ENDIF
C
C   END OF MAIN LOOP
C   ITERATE FOR NEXT TIME STEP
C
        GOTO 50
C
C   FORMAT STATEMENTS
C
700  FORMAT (/5X,'ENTER OUTPUT FILE NAME [DEFAULT=DUMP.RAW]')
710  FORMAT (A32)
720  FORMAT (/5X,'ENTER DATE STAMP [MM/DD/YYYY] ')
730  FORMAT (I2,1X,I2,1X,I4)
735  FORMAT (/5X,'PLOT RESULTS OF PREVIOUS RUN? [0=NO , 1=YES] ')
740  FORMAT (/5X,'OVERRIDE CONTROLLER TUNING ? [0=NO 1=YES] ')
745  FORMAT (/5X,'INDICATE CONTROLLER:',3X,'1=LEVEL CONTROLLER',
           1  /,28X,'2=TEMPERATURE CONTROLLER',
           1  /,28X,'3=RECYCLE FLOW CONTROLLER',
           1  /,28X,'4=COOLING WATER FLOW CONTROLLER ')
750  FORMAT (/5X,'ENTER PARAMETER TO CHANGE:',3X,'1=GAIN',
           1  /,34X,'2=INTEGRAL'
           1  /,34X,'3=DERIVATIVE ')
760  FORMAT (/5X,'ENTER NEW VALUE [CURRENT VALUE IS ',F10.4,'] ')
770  FORMAT (/5X,'ENTER FILTER CONSTANT [0.0 - 1.0] ')
772  FORMAT (/5X,'PRINT OUTPUT IN RANDOM ORDER? [0=NO , 1=YES] ')
773  FORMAT (/5X,'VARIABLE OUTPUT FREQUENCY? [0=NO , 1=YES] ')
775  FORMAT (5X,F4.2)
790  FORMAT (/5X,'ENTER INTEGER SEED FOR RANDOM NUMBER GENERATOR ')
800  FORMAT (//5X,'*** JACKETED CSTR DYNAMIC SIMULATION ***',/)
810  FORMAT (5X,I2,') ',A40)
820  FORMAT (5X,'ENTER DESIRED FAULT TYPE ')
821  FORMAT (//,15X,'PROCESS SENSORS ARE :',//,15(5X,I4,' - ',A,/,)
           1  ,//)
822  FORMAT (/5X,'INDICATED FAULT HAS ALREADY BEEN SELECTED! ')
823  FORMAT (/5X,'CONTINUE? [0=NO , 1=YES] ')
825  FORMAT (/5X,'ENTER NUMBER OF FAILED SENSOR ')
830  FORMAT (/5X,'ENTER TYPE OF SENSOR FAILURE: 0=FIXED BIAS',
           1  /,35X,'1=FIXED VALUE ')
835  FORMAT (/5X,'NOMINAL VALUE IS: ',3X,F14.4)
840  FORMAT (/5X,'ENTER EXTENT OF FAULT (IN APPROPRIATE UNITS) ')
850  FORMAT (/5X,'ENTER FAULT DELAY [min] ')

```

```
860 FORMAT (/,5X,'ENTER TIME CONSTANT [(min)-1]: ')
865 FORMAT (/,5X,'ADD ANOTHER FAULT? [0=NO , 1=YES] ')
870 FORMAT (/,5X,'ENTER TIME HORIZON FOR SIMULATION [min] ')
873 FORMAT (/,5X,'PRINT INTERMEDIATE RESULTS? [1=YES/0=NO] ')
875 FORMAT (/,5X,'ENTER PRINT INTERVAL [min] ')
880 FORMAT (//,10X,'PROCESS CONDITION IS -- ',A40,
1   /,28X,'IN -- ',A32,
2   //,10X,'SENSOR FAULT IS TYPE -- ',A12,
3   /,10X,'FAULT EXTENT IS    -- ',F8.2,
4   /,10X,'FAULT DELAY  IS   -- ',F8.2,' (MIN)')
890 FORMAT (//,10X,'PROCESS CONDITION IS -- ',A40,
1   //,10X,'FAULT EXTENT IS    -- ',F8.2,
2   /,10X,'FAULT DELAY  IS   -- ',F8.2,' (MIN)',
3   /,10X,'TIME CONSTANT IS   -- ',F8.2)
895 FORMAT (/,10X,'FILTER CONSTANT IS   -- ',F8.2)
930 FORMAT (/)
940 FORMAT (5X,A36,' IS ',A16)
950 FORMAT (/,5X,'RUNNING . . . . ',/)
960 FORMAT (/,5X,'END OF RUN',/)
970 FORMAT (//,5X,'***** EMERGENCY SHUTDOWN INITIATED AT ',F6.2,
1   ' MIN ',//)
980 FORMAT (//,5X,'***** LOW LEVEL FORCES PUMP SHUTDOWN AT ',F6.2,
1   ' MIN ',//)
990 FORMAT (/,5X,'PERFORM ANOTHER RUN? [0=NO , 1=YES] ')
995 FORMAT (/,5X,'PLOT RESULTS? [0=NO , 1=YES] ')
999 FORMAT (A32,I4,I2,I2,I2,I2,F14.4,A7)
C
END
```

A3. APPENDIX 3: SLD Example

This appendix demonstrates the use of the SLD methodology, described in section 3.1.2, applied to diagnosis of the Jacketed CSTR Process introduced in section 6.0. The Jacketed CSTR Process is illustrated in figure 6-1.

A3.1. Functional Decomposition

The process can be decomposed into nine (9) systems. There are three (3) control systems and six (6) passive systems. Two (2) control systems form a cascade configuration. Five (5) passive systems are external and one is internal. Table A3-1 summarizes system definitions and designations.

TABLE A3-1: Jacketed CSTR Process System Definitions

Designation	System Description	System Type
S1A	Reactor Temperature Control	Control (primary cascade)
S1B	Cooling Water Flow Control	Control (secondary cascade)
S2	Reactor Level Control	Control
S3	Chemical Reaction	Passive (internal)
S4A	Cooling Water Temperature	Passive (external)
S4B	Cooling Water Pressure	Passive (external)
S5A	Reactant Feed Pressure	Passive (external)
S5B	Reactant Feed Temperature	Passive (external)
S5C	Reactant Feed Concentration	Passive (external)

Unit functions are defined and apportioned to systems in Table A3-2. Functions are listed by unit and function class (PRC, CNT, SEN, or CSEN). In this process, only one unit function is shared. The fluid containment function of the CSTR Jacket is a function of the Reactor Temperature Control system, Cooling Water Flow Control system, Reactor Level Control system, and Chemical Reaction system. In this example, the CSTR Jacket is treated as a separate unit.

TABLE A3-2: Jacketed CSTR Process System Components

Unit	Unit Functions	Class	System(s)
CSTR	Fluid Containment	PRC	S2
	Fluid Conductance	PRC	S2
	Reaction	PRC	S3
	Insulation	PRC	S1A
CSTR Jacket	Fluid Containment	PRC	S1A,S1B,S2,S3
	Fluid Conductance	PRC	S1B
	Heat Transfer	PRC	S1A
	Insulation	PRC	S1A
Pump	Fluid Containment	PRC	S2
	Fluid Conductance	PRC	S2
	Pressurization	PRC	S2
Valve 1	Fluid Containment	PRC	S2
	Fluid Conductance	PRC	S2
	Flow Regulation	CNT	S2
Valve 2	Fluid Containment	PRC	S1B
	Fluid Conductance	PRC	S1B
	Flow Regulation	CNT	S1B
	Insulation	PRC	S1A
Feed Pipe	Fluid Containment	PRC	S5A
	Fluid Conductance	PRC	S5A
	Insulation	PRC	S5B
	Source Concentration	PRC	S5C
	Source Temperature	PRC	S5B
	Source Pressure	PRC	S5A
Effluent Pipe	Fluid Containment	PRC	S2
	Fluid Conductance	PRC	S2

TABLE A3-2: Jacketed CSTR Process System Components (continued)

Unit	Unit Functions	Class	System(s)
CW Draw Pipe	Fluid Containment	PRC	S4B
	Fluid Conductance	PRC	S4B
	Insulation	PRC	S4A
	Source Temperature	PRC	S4A
	Source Pressure	PRC	S4B
CW Drain Pipe	Fluid Containment	PRC	S1B
	Fluid Conductance	PRC	S1B
TC	Computation	CNT	S1A
FC	Computation	CNT	S1B
LC	Computation	CNT	S2
CS (Feed)	Measurement	SEN	S5C
TS (Feed)	Measurement	SEN	S5B
FS (Feed)	Measurement	SEN	S5A
FS (Effluent)	Measurement	SEN	S2
CS (Effluent)	Measurement	SEN	S3
LS	Measurement	CSEN	S2
TS (Reactor)	Measurement	CSEN	S1A
FS (CW Draw)	Measurement	CSEN	S1B
TS (CW Draw)	Measurement	SEN	S4A
PS (CW Draw)	Measurement	SEN	S4B
SS (TC)	Measurement	SEN	S1A
SS (FC)	Measurement	SEN	S1B
SS (LC)	Measurement	SEN	---

The SLD method requires measurement of all controlled and manipulated variables. For this process, the requirement is exceeded. Either the level controller signal or the reactor effluent flowrate could be used as the manipulated variable of the Level Control system. Effluent flowrate is used in this example.

In the cascade control system, the primary Reactor Temperature Control system uses reactor temperature as a controlled variable and the Cooling Water Flow Control system setpoint (i.e. the temperature controller output signal) as a manipulated variable. The Cooling Water Flow Control system uses cooling water flowrate as a controlled variable and the cooling water flow controller output signal as a manipulated variable. For most control systems, the controller output signal, the valve (or control element) position, or the variable directly regulated by the valve (or control element) can be used as a manipulated variable. For the Cooling Water Flow Control system, the variable directly regulated by the valve is the controlled variable, so another alternative is chosen.

Table A3-3 shows the relationship between unit function failures and MIDAS faults as defined in the case study presented in section 6.0. As in the case study, one hundred nine (109) faults are modeled.

A3.2. Process Graph

The process graph for the Jacketed CSTR Process is presented in figure A3-1. External passive systems reside on the periphery of the graph. The Chemical Reaction, Reactor Temperature Control, and Cooling Water Flow Control systems are highly interactive as indicated by the two-way dependencies.

The process graph can be derived from an ESDG model or an event model of the process. Because there are far fewer systems than variables, however, the process graph can often be developed manually with only a modest effort. Where the ESDG model of the Jacketed CSTR Process requires one hundred nine (109) variables and two hundred sixteen (216) arcs (the event model requires one hundred forty four (144) events and three hundred thirty four (334) causal links), the process graph has only nine (9) systems connected by thirteen (13) dependency arcs. When the relative simplicity of the SLD models are considered, the quality of diagnosis that can be achieved with the method is impressive.

TABLE A3-3: Jacketed CSTR Process Malfunction Modes

Unit Type	Failed Function	Equivalent MIDAS failure(s)
CSTR	Fluid Containment	Leak
	Fluid Conductance	Inlet Blockage, Outlet Blockage
	Reaction	Catalyst Deactivation, Loss of Mixing
	Insulation	Fire, Loss of Insulation
CSTR Jacket	Fluid Containment	Leak to Reactor, Leak to Environment
	Fluid Conductance	Blockage
	Heat Transfer	Fouling
	Insulation	Fire, Loss of Insulation
Pump	Fluid Containment	Leak
	Fluid Conductance	Blockage
	Pressurization	Cavitation, Motor Speed High (Low)
	Insulation	Fire, Loss of Insulation, Overheating
Valve	Fluid Containment	Leak
	Fluid Conductance	Blockage
	Flow Regulation	Stuck Open (Closed)
	Insulation	Fire, Loss of Insulation
Draw Pipe	Fluid Containment	Leak
	Fluid Conductance	Blockage
	Insulation	Fire, Loss of Insulation
	Source Concentration	Source Concentration High (Low)
	Source Temperature	Source Temperature High (Low)
	Source Pressure	Source Pressure High (Low)
Pipe	Fluid Containment	Leak
	Fluid Conductance	Blockage
	Insulation	Fire, Loss of Insulation
Controller	Computation	Failed High (Low) Biased High (Low) Setpoint High (Low)
Sensor	Measurement	Failed High (Low) Biased High (Low)

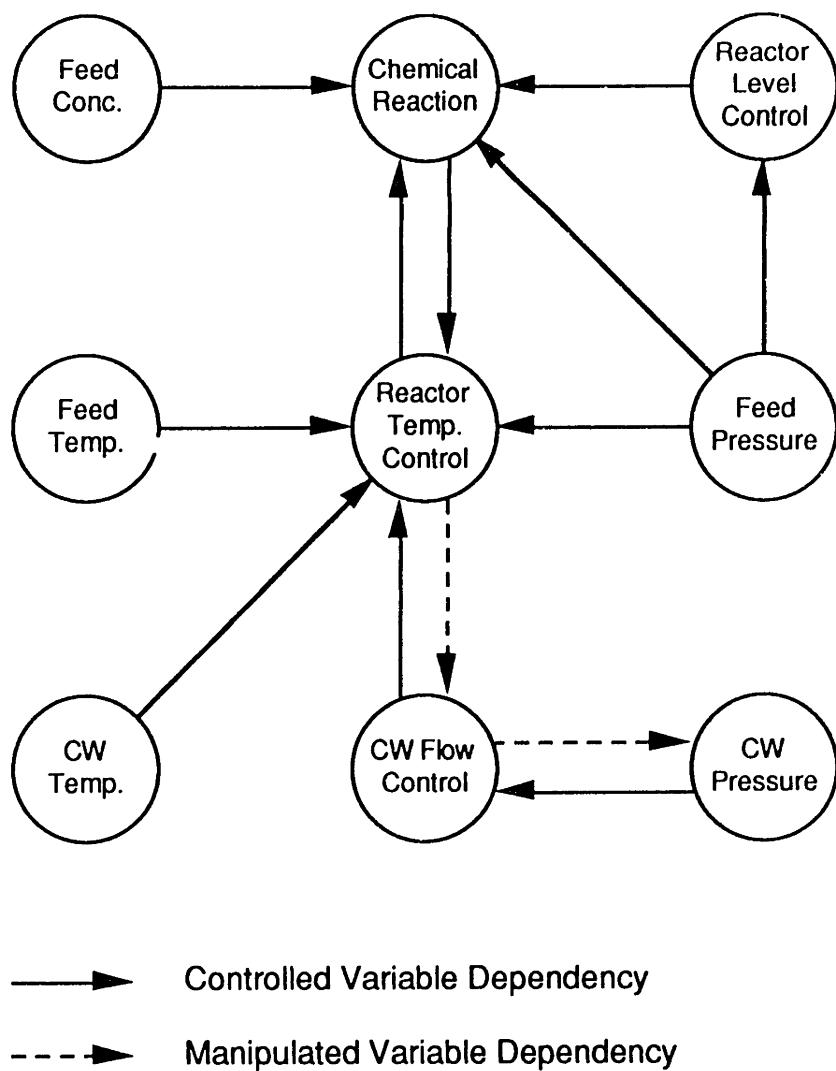


Figure A3-1: Process Graph for Jacketed CSTR Process

A3.3. Diagnosis Example

The example used for demonstration is the Jacket Leak to Reactor fault simulated as part of the MIDAS case study and discussed as Example 1 of section 6.2.5. Table A3-4 presents the diagnosis. Figure A3-2 shows the evolution of the dysfunctional subgraphs as system states change. Note that functional systems are not displayed.

TABLE A3-4: SLD Diagnosis of Jacket Leak to Reactor (see Table 6-5)

MIDAS Event	SLD Event	SLD Resolution	SLD Performance
Inventory Constraint High	-----	-----	-----
Level Controller Signal High	-----	-----	-----
Product Flowrate High	S2 Stressed	31	0.72
Cooling Water Flowrate Low	S1B Uncontrolled	37*	0.67
Reactor Level High	S2 Saturated	33*	0.70
Cooling Water Flow Controller Low	S1B Saturated	2	0.99
Product Concentration B Low	S3 Saturated	2	0.99
Cooling Water Setpoint Low	S1A Stressed	27	0.76
Reactor Level Normal	S2 Stressed	12	0.90

* In these cases, the diagnostic rules concluded there was no single fault explanation.

The inventory constraint and level controller signal are not used in the system models, therefore, the first system event corresponds to the detection of high product flowrate.

Event 1: Product flowrate is the manipulated variable of the Level Control system (S2). Detection of high product flowrate indicates the Level Control system is STRESSED. System S2 can be considered a definite source system since it interacts with only functional systems. Application of the diagnostic rules produces a diagnosis of S2-PRC, S2-SEN, and S2-CNT, implying that any of the unit functions of S2 could have failed. There are thirty one (31) faults effecting unit functions of S2. Jacket Leak to Reactor is included in the set, so diagnosis is accurate. Because candidates are not ranked, the resolution of the diagnosis is the size of the candidate set.

Event 2: Detection of low cooling water flowrate indicates the Cooling Water Control system is in an UNCONTROLLED state. No direct connection exists between the Cooling Water Flow Control and Level Control systems, so two malfunctioning subgraphs -- each with a definite source system -- are present. In most instances, multiple definite source systems are desirable because the diagnoses of each source

can be intersected to produce a final diagnosis. In this case, however, the UNCONTROLLED state of the Cooling Water Flow Control system is an erroneous early response. The diagnosis of the first (S2) subgraph is unchanged and the diagnosis of the new (S1B) subgraph is S1B-CNT. The intersection of these sets is empty. There is no recourse except to assume multiple faults and combine the fault sets to produce an overall set of thirty seven (37) candidates.

MIDAS also had difficulties interpreting this event. Unlike MIDAS, however, SLD has no analog to EXPECTED-EVENTS or LATENT-EVENTS to make it robust to out-of-order events. The interpretation problems of MIDAS stemmed from suboptimal monitor threshold rather than a fundamental limitation of the methodology.

- Event 3: The Level Control system is SATURATED when reactor level is detected high. The diagnosis of the S2 subgraph becomes S2-PRC, S2-CSEN, and S2-CNT, representing twenty seven (27) fault candidates. The Cooling Water Flow Control system is still in the UNCONTROLLED state resulting in an overall diagnosis of thirty three (33) candidates.
- Event 4: With the detection of a low cooling water flow controller signal, the Cooling Water Control system attains its ultimate SATURATED response. The new diagnosis of the S1B subgraph is S1B-PRC, S1B-CSEN, S1B-CNT, S1A-CSEN, S1A-CNT¹ -- a total of twenty three (23) candidates. Intersection of the S2 subgraph diagnosis with the S1B subgraph diagnosis results in a combined diagnosis of $S2\text{-PRC} \cap S1B\text{-PRC}$. The only shared function is CSTR Jacket Fluid Containment, corresponding to two (2) faults -- Jacket Leak to Reactor or Jacket Leak to Environment.
- Event 5: Detection of low product concentration results in a SATURATED Chemical Reaction system. This malfunctioning system joins the S2 dysfunctional subgraph. S2 is still a definite source system, and S3 is a possible source system. The diagnosis of the S2 subgraph does not change.

¹ The diagnostic rules do account for certain types of latent failures. In this example, S1A-CSEN is included in the diagnosis to account for possible latent bias of the reactor temperature sensor.

- Event 6: The detection of low cooling water setpoint means the Reactor Temperature Control system is STRESSED. Because, S1A interacts with both S3 and S1B, the two dysfunctional subgraph are bridged². Now only S2 can be considered a definite source system -- S1B is only a possible source system. Without the intersection effect, all twenty seven (27) fault candidates associated with S2 must be postulated.
- Event 7: When reactor level returns to normal, the Reactor Level Control system returns to the STRESSED state. S2 remains a definite source system, but the diagnostic rules for interaction with S3 indicate a diagnosis of S2-PRC \cap S3-PRC, S2-CSEN, and S2-CNT. The final diagnosis contains twelve (12) candidates, including the true fault: Jacket Leak to Reactor.

A3.4. Analysis

The results of the diagnosis example in the preceding section point to several characteristics of SLD discussed in section 3.0. A summary is presented below:

Poor Initial Response

SLD rules are based on the ultimate response of control systems. When initial response is not the same as the ultimate response, the rules perform poorly. This problem was observed during the period the Cooling Water Flow Control system was UNCONTROLLED. This problem could be mitigated using system trajectories in addition to system states.

Stability

SLD is less stable than the dynamic event models used in MIDAS. SLD does not incorporate the robustness features needed to be stable to variations in event detection order. The use of snapshots (i.e. no memory) also makes SLD an inherently less stable method.

² In MIDAS, this event collapsed the two separate IM clusters.

Compensatory Response

SLD provides its best diagnoses when systems exhibit compensatory response and are in the STRESSED state. This is not surprising, since the system models are designed to capture the compensating behavior of goal-directed systems.

Modeling Ease

Compared to the other qualitative modeling and diagnosis methods discussed in previous sections, SLD is an extremely easy method to use. The system models are an order of magnitude smaller than ESDG or event models. The diagnostic rules are reasonably simple, involving only the union or intersection of faults candidate sets.

Abstraction

The higher level of abstraction involved in the SLD methodology is evident in the lack of deviation directions and the use of lumped models. The advantages of this abstraction are the relatively simple models and diagnostic procedures and the capture of compensating system behavior. The disadvantage is that SLD more suited for initial diagnosis aimed at narrowing the scope of subsequent diagnosis. By itself, SLD does not contain enough detail to isolate a failure beyond the system level.

Rather than compete with MIDAS as an alternative diagnosis methodology, incorporation of SLD into the MIDAS framework will improve both methods.

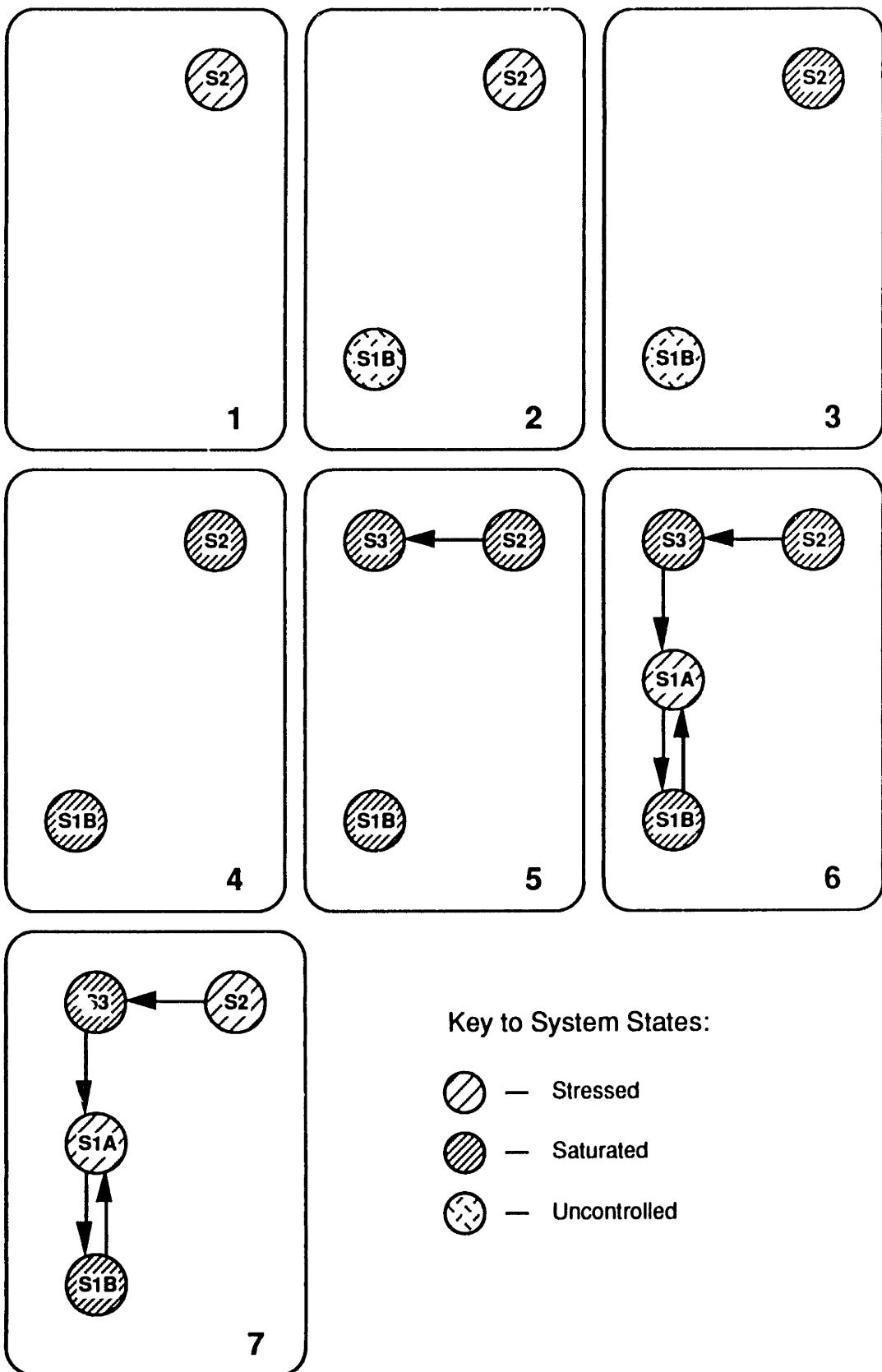


Figure A3-2: Malfunctional Subgraphs for Diagnosis of Jacket Leak