1

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

**DR: the Diagnostic Remodeler Algorithm for
Automated Model Acquisition through
Fault Knowledge Re-Use**

by

Suhayya Abu-Hakima, B.Eng., M.Eng.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6

September, 1994

© copyright

1994, Suhayya Abu-Hakima

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN
IRREVOCABLE NON-EXCLUSIVE
LICENCE ALLOWING THE NATIONAL
LIBRARY OF CANADA TO
REPRODUCE, LOAN, DISTRIBUTE OR
SELL COPIES OF HIS/HER THESIS BY
ANY MEANS AND IN ANY FORM OR
FORMAT, MAKING THIS THESIS
AVAILABLE TO INTERESTED
PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE
IRREVOCABLE ET NON EXCLUSIVE
PERMETTANT A LA BIBLIOTHEQUE
NATIONALE DU CANADA DE
REPRODUIRE, PRETER, DISTRIBUER
OU VENDRE DES COPIES DE SA
THESE DE QUELQUE MANIERE ET
SOUS QUELQUE FORME QUE CE SOIT
POUR METTRE DES EXEMPLAIRES DE
CETTE THESE A LA DISPOSITION DES
PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP
OF THE COPYRIGHT IN HIS/HER
THESIS. NEITHER THE THESIS NOR
SUBSTANTIAL EXTRACTS FROM IT
MAY BE PRINTED OR OTHERWISE
REPRODUCED WITHOUT HIS/HER
PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE
DU DROIT D'AUTEUR QUI PROTEGE
SA THESE. NI LA THESE NI DES
EXTRAITS SUBSTANTIELS DE CELLE-
CI NE DOIVENT ETRE IMPRIMES OU
AUTREMENT REPRODUITS SANS SON
AUTORISATION.

ISBN 0-315-98436-8

Name Suhayya Abu-Hakima

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

0984

U·M·I

SUBJECT TERM

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
 General 0289
 Ancient 0290
 Linguistics 0291
 Modern 0401
Literature 0294
 Classical 0295
 Comparative 0297
 Medieval 0298
 Modern 0316
 African 0591
 American 0305
 Asian 0352
 Canadian (English) 0355
 Canadian (French) 0593
 English 0311
 Germanic 0312
 Latin American 0315
 Middle Eastern 0313
 Romance 0314
 Slavic and East European

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
 General 0321
 Biblical Studies 0319
 Clergy 0320
 History of 0322
 Philosophy of 0469
Theology

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
 Archaeology 0326
 Cultural 0327
 Physical 0310
Business Administration 0272
 General 0770
 Accounting 0454
 Banking 0338
 Management 0385
Canadian Studies 0501
Economics 0503
 General 0505
 Agricultural 0508
 Commerce-Business 0509
 Finance 0510
 History 0511
 Labor 0358
 Theory 0366
Folklore 0351
Geography 0578
Gerontology 0578
History

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0337
United States 0398
History of Science 0398
Law 0615
Political Science 0616
 General 0617
 International Law and Relations 0814
 Public Administration 0452
Recreation 0626
Social Work 0627
Sociology 0938
 General 0631
 Criminology and Penology 0628
 Demography 0629
 Ethnic and Racial Studies 0630
 Individual and Family Studies 0700
 Industrial and Labor Relations 0344
 Public and Social Welfare 0709
 Social Structure and Development 0999
 Theory and Methods 0453
Transportation 0453
Urban and Regional Planning 0453
Women's Studies

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
 General 0285
 Agronomy 0475
 Animal Culture and Nutrition 0476
 Animal Pathology 0359
 Food Science and Technology 0478
 Forestry and Wildlife 0479
 Plant Culture 0480
 Plant Pathology 0817
 Plant Physiology 0777
 Range Management 0746
Biology 0306
 General 0287
 Anatomy 0308
 Biostatistics 0309
 Botany 0379
 Cell 0329
 Ecology 0353
 Entomology 0369
 Genetics 0793
 Limnology 0410
 Microbiology 0307
 Molecular 0317
 Neuroscience 0416
 Oceanography 0433
 Physiology 0821
 Radiation 0778
 Veterinary Science 0472
 Zoology 0786
Biophysics 0760
 General 0425
 Medical 0996

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
 General 0300
 Audiology 0992
 Chemotherapy 0567
 Dentistry 0350
 Education 0769
 Hospital Management 0758
 Human Development 0982
 Immunology 0347
 Medicine and Surgery 0569
 Mental Health 0570
 Nursing 0380
 Nutrition 0354
 Obstetrics and Gynecology 0381
 Occupational Health and Therapy 0571
 Ophthalmology 0419
 Pathology 0572
 Pharmacology 0382
 Pharmacy 0573
 Physical Therapy 0574
 Public Health 0575
 Radiology 0575
 Recreation

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences 0485
Chemistry 0749
 General 0486
 Agricultural 0487
 Analytical 0488
 Biochemistry 0738
 Biochemistry 0490
 Inorganic 0491
 Nuclear 0494
 Organic 0495
 Pharmaceutical 0754
 Physical 0495
 Polymer 0754
 Radiation 0405
Mathematics 0605
Physics 0986
 General 0606
 Acoustics 0608
 Astronomy and Astrophysics 0748
 Atmospheric Science 0607
 Atomic 0798
 Electronics and Electricity 0759
 Elementary Particles and High Energy 0609
 Fluid and Plasma 0610
 Molecular 0752
 Nuclear 0756
 Optics 0611
 Radiation 0463
 Solid State 0346
Statistics 0984
Applied Sciences 0346
Applied Mechanics 0984
Computer Science

Engineering 0537
 General 0538
 Aerospace 0539
 Agricultural 0540
 Automotive 0541
 Biomedical 0542
 Chemical 0543
 Civil 0544
 Electronics and Electrical 0348
 Heat and Thermodynamics 0545
 Hydraulic 0546
 Industrial 0547
 Marine 0794
 Materials Science 0548
 Mechanical 0743
 Metallurgy 0551
 Mining 0552
 Nuclear 0549
 Packaging 0765
 Petroleum 0554
 Sanitary and Municipal 0790
 System Science 0428
Geotechnology 0796
Operations Research 0795
Plastics Technology 0994
Textile Technology

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

The undersigned hereby recommend to
the Faculty of Graduate Studies
and Research acceptance of the thesis,

**“DR: the Diagnostic Remodeler Algorithm for
Automated Model Acquisition through
Fault Knowledge Re-Use”**

submitted by Suhayya Abu-Hakima, B.Eng., M.Eng.

in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Professor S.A. Mahmoud
Chair, Department of Systems and Computer Engineering

Professor N. Dawes
Thesis Supervisor, Department of Systems and Computer Engineering

Professor F. Oppacher
Thesis Supervisor, School of Computer Science

Professor S. Matwin
External Examiner,
Department of Computer Science, Ottawa University

Abstract

This thesis addresses the problem of automated model acquisition for diagnosis. A new algorithm, the Diagnostic Remodeler has been defined, developed and validated. The algorithm automatically generates black box models that have an explicit representation of component function and input-to-output behaviour. DR re-uses well-structured fault knowledge with a minimum of device dependent background knowledge as input in its first phase, DR-1, and device dependent and device independent background knowledge in its second phase, DR-2, to generate the black box component models of a device.

The algorithm has been successfully tested with an existing real-world aircraft engine fault base from JETA. The DR generated engine subsystem model has been validated by an engine Expert at NRC's Institute for Aerospace Research Propulsion Laboratory. Furthermore, the generated model has been found to identify inconsistencies and gaps, and as a result novel faults missing in the original fault knowledge.

The algorithm has been applied for a second, very different domain, namely the generation of a full device model for a coffee maker. The DR software used for the engine application, was re-used to demonstrate its generality for the coffee application. In addition, encoded generalized component descriptions, in the form of device independent background knowledge descriptors were re-used across the two application domains, namely, the engine and coffee domains. The coffee maker device model was verified by comparing it to the expected hand-coded model.

Acknowledgments

I am indebted to many people who have inspired, and motivated me to complete this work. Firstly, I am grateful to many, specifically Jack Brahan, Martin Brooks and Alex Mayman, at the Institute for Information Technology (IIT) of the National Research Council (NRC) of Canada, for providing me with the opportunity, and the encouragement to complete this work.

I am also grateful to many colleagues at the Knowledge Systems Lab (KSL) at IIT, for comments, and very helpful criticism, in lab presentations on this work. I would like to specifically thank Mike Halasz and Sieu Phan who were the principal authors of JETA, they are easy to work with and know. I would like to acknowledge the excellent, sometimes heated, and friendly discussions I have had with my colleagues, Peter Turney and Peter Clark, on machine learning and knowledge re-use. I would also like to thank Innes Ferguson for his continued friendship, and for his, Martin's, Mike's and Jeff's, excellent comments on the first draft of the thesis.

I am grateful to my friend Jennifer Campbell, previously the IIT branch librarian and currently at NRC's main branch of the Canadian Institute for Scientific and Technical Information (CISTI), for her encouragement and tireless efforts in seeking vast references to support my research. I would also like to acknowledge her assistant Monique Day, also from CISTI, for her hard work in processing my hundreds of reference requests. Without their efforts the annotated bibliography would not have been as extensive, or as useful as it has proven to be.

I would like to thank Jeff Bird of NRC's Institute for Aerospace Research (IAR) Propulsion Lab, who leant me his expertise in acquiring the background knowledge for the jet engine component model, and who validated my results. Jeff has always been a supporter of advancing methods for engine diagnosis.

I would like to thank my two supervisors Nick Dawes and Franz Oppacher, who I have known for many years, and who have supported my fairly independent style of research, and provided me with good direction, when I needed it.

I would also like to acknowledge the researchers in the artificial intelligence model-based diagnosis community, for providing me with the inspiration to take up the challenge to demonstrate that well constructed fault-based systems, such as JETA are equal in providing a rich knowledge representation, that could be re-used as part of the solution for automatically acquiring models for complex devices.

I am also indebted to my family for inspiration like none other. I would like to thank my mother Dr. Aida S. Arif (Archaeologist), and father Dr. Ahmad Mustafa Abu-Hakima (Historian), for their love and dedication to excellent research and their continuous demonstration of the rewards of hard work.

Finally, this Ph.D. thesis, as any other, is dedicated to my husband Kenneth Edward Grigg and my son Tayseer Caelum Grigg, who truly deserve to receive it jointly with me as recognition for their unwavering love, and support throughout the journey to its completion.

Table of Contents

1.0	Introduction	1
1.1	Definitions	4
1.2	Thesis Objective	7
1.3	Contributions.....	11
1.4	Thesis Structure.....	13
2.0	Relevant Artificial Intelligence Techniques.....	14
2.1	Fault-Based Reasoning.....	15
2.1.1	Noteworthy Papers.....	18
2.1.2	Summary	19
2.2	Model-Based Reasoning	20
2.3.1	Noteworthy Papers.....	26
2.3.2	Summary	30
2.3	Case-Based Reasoning	30
2.3.1	Noteworthy Papers.....	36
2.3.2	Summary	40
2.4	Machine Learning for Knowledge Acquisition.....	41
2.4.1	Noteworthy Papers.....	46
2.4.2	Summary	50
2.5	Integrated Diagnostic Techniques	51
2.5.1	Noteworthy Papers.....	54
2.5.2	Summary	67
2.6	Literature Review Summary	68

3.0	Model Acquisition: Diagnostic Remodeler Design	73
3.1	Hypothesis of Diagnostic Remodeler (DR) Algorithm.....	73
3.1.1	Utility of DR Algorithm	73
3.2	Novelty of DR Algorithm	74
3.3	DR Algorithm.....	75
3.3.1	Background.....	75
3.3.2	Method	77
3.3.3	DR Objective Statement	79
3.3.4	DR Algorithm Steps.....	80
3.3.5	DR Assumptions	83
3.4	DR-1 Input: Fault Based Knowledge	83
3.5	DR-1 Outputs	85
3.6	DR-2 Input: Background Knowledge	86
3.6.1	Device Dependent Background Knowledge.....	86
3.6.2	Device Independent Background Knowledge	87
3.7	DR-2 Outputs	89
3.8	DR Empirical Evaluation: Fault Knowledge Input to DR-1	90
3.8.1	JETA	90
3.8.1.1	How a Jet Engine Works	91
3.8.1.2	Reasoning Strategies in JETA	93
3.8.1.3	Frame-Based Knowledge Representation of JETA	94
3.8.1.4	DR-1 Input: JETA's Fault-Based Knowledge.....	95
3.8.1.5	DR-2 Input: Engine Device Dependent Background Knowledge.....	101
3.8.1.6	DR-2 Input: Engine Device Independent Background Knowledge	103
3.8.2	Coffee Maker	104

3.8.2.1	Frame-Based Knowledge Representation of Coffee Maker Faults	105
3.8.2.2	DR-1 Input: Coffee Maker Fault-Based Knowledge.....	106
3.8.2.3	DR-2 Input: Coffee Maker Device Dependent Background Knowledge	108
3.8.2.4	DR-2 Input: Coffee Maker Device Independent Background Knowledge ...	108
3.9	DR Algorithm Evaluation	110
3.9.1	Validation of Engine application.....	111
4.0	Implementation of Diagnostic Remodeler Algorithm.....	114
4.1	High Level Design	114
4.2	DR Algorithm Structure.....	116
4.3	Get Components Module	117
4.4	Get Component Connections Module	119
4.5	Get Components Parameters Module.....	123
4.6	Device Dependent Background Knowledge	127
4.6.1	Engine Device Dependent Background Knowledge	128
4.6.2	Coffee Maker Device Dependent Background Knowledge.....	132
4.7	Translate 1 Module: Component Connections Meaningful Text	133
4.8	Translate 2 Module: Component Parameters Meaningful Text	135
4.9	Device Independent Background Knowledge.....	137
4.9.1	Engine Device Independent Background Knowledge	138
4.9.2	Coffee Maker Device Independent Background Knowledge	144
4.10	Match Module	146

5.0	Results of Diagnostic Remodeler (DR) Implementation	150
5.1	General Results and Testing of DR Assumptions	150
5.2	Impact of Background Knowledge in relation to Fault Knowledge	154
5.3	DR-1 Output.....	159
5.3.1	Engine Application	159
5.3.1.1	Extracted Components	159
5.3.1.2	Inferred Component-to-Component Connections	160
5.3.1.3	Extracted Component Parameters	161
5.3.2	Coffee Maker Application	163
5.3.2.1	Extracted Components	164
5.3.2.2	Inferred Component-to-Component Connections	164
5.3.2.3	Extracted Component Parameters	166
5.4	DR-2 Output.....	169
5.4.1	Engine Application Resultant Model.....	169
5.4.1.1	Meaningful Text: Component Connection Relations.....	170
5.4.1.2	Meaningful Text: Component Parameters.....	173
5.4.1.3	Component Behaviour Models.....	174
5.4.2	Coffee Maker Resultant Model.....	182
5.4.2.1	Meaningful Text: Component Connection Relations.....	182
5.4.2.2	Meaningful Text: Component Parameters.....	184
5.4.2.3	Component Behaviour Models.....	185
5.5	Gaps, Inconsistencies and Novel Faults	188
5.5.1	Engine Application	189
5.5.2	Coffee Maker Application	194

6.0	Concluding Remarks.....	197
6.1	Addressing the Thesis Objective Statement.....	199
6.2	Usability of DR Output.....	205
6.3	Thesis Contributions	209
6.4	Issues Addressed by DR.....	211
6.5	Future Work.....	213
7.0	References.....	217
	Appendix A: Engine Application	235
	A.1 Graphical View of JETA Network in Browser.....	235
	A.2 Correspondence with Engine Domain Expert.....	236
	A.3 Engine Test Run Annotated by Domain Expert.....	237
	Appendix B: Coffee Maker Application	238
	B.1 Coffee Maker Device Knowledge Base.....	238
	B.2 DR Coffee Maker Test Run.....	243

1.0 Introduction

Motivation

Developers of fault-based diagnostic systems diligently acquire knowledge from maintenance manuals and human experts, and structure it in a knowledge base so that technicians may better troubleshoot and repair complex devices. In some cases, several years are spent in designing, implementing, and testing such systems. The most difficult problem these developers face is in the acquisition, representation, and organization of the fault knowledge. The resultant knowledge base is often well-structured, and becomes a valuable assistant when integrated into a troubleshooting tool for maintenance personnel.

Artificial Intelligence (AI) researchers in the model-based diagnosis (MBD) community dismiss fault-based diagnostic (FBD) systems far too easily. Many MBD authors¹ incorrectly assume that fault-based knowledge is still represented and organized as a flat file of *if-then* production rules as it was in the days of MYCIN [Scott et al. 77; Clancey 86]. MYCIN was the earliest well-known fault-based diagnostic system and it was implemented in the 1970's. Much work has evolved and improved on MYCIN's main diagnostic themes. Today's FBD systems recognize the need for developing systems that explicitly separate the control from the data in reasoning. This separation aids in tractable reasoning and in searching for diagnoses in finite time [Goel et al. 87]. This separation is also essential in justifying system behaviour and in generating good explanations [Abu-Haki 88; Abu-Hakima and Oppacher 90]. Currently developed FBD systems address

1. For specific examples, see MBD annotated papers in Appendix A.2, including, [Azmoodeh 91], [Hamscher and Struss 90], and [Lee et al. 90].

complex real-world problems, are highly structured, and thus, their knowledge bases are very efficiently searched.

Despite these FBD strengths, MBD researchers repeatedly criticize fault-based systems as limited, and inadequate for troubleshooting novel faults. What is often forgotten by the MBD supporters, are the number of implemented FBD systems that are in successful daily use, as exemplified by the literature review (chapter 2.0). Furthermore, MBD has been shown to be computationally expensive and intractable for complex devices. MBD systems are additionally limited by the accuracy of their models. If a model does not correctly propagate the behaviours of a device, it will not be able to adequately diagnose the device, let alone discover novel faults.

The work in this thesis is intended to form a bridge between the FBD and MBD communities. Although the two communities are striving towards the same goal, mainly the efficient and accurate diagnosis of devices, they have not closely examined or taken advantage of their commonly shared problems. They share problems in knowledge acquisition for diagnosis, be it fault or model knowledge. The researchers in the two camps need to address common approaches for structuring, reasoning about, explaining and re-using knowledge. MBD researchers have started to make use of fault hierarchies for commonly observed problems so that they may reduce the computational complexity of their approach (specifically, [Bizzari et al. 90; Dewberry and Carnes 90; Hoffman et al. 91]). The complexity problem in MBD has sidetracked researchers from attempting to develop real-world diagnostic systems for complex devices. Instead, many MBD researchers have only attacked toyworld problems such as two-gate adder circuits. Others however, have started to work on hybrid diagnostic systems and are integrating multiple AI paradigms in

an attempt to address the common problems in diagnosing complex systems. Complex applications in MBD include several for space systems by NASA teams (examples include, [Hofmann et. al 91; Guo 91]).

One problem in bridging FBD and MBD, is the problem of relating, and possibly re-using device fault knowledge as model knowledge. To address this problem, the Diagnostic Remodeler (DR) algorithm, the subject of this dissertation, has been formulated, implemented, and tested. DR illustrates that well-structured fault knowledge can be mapped and re-used, as model knowledge. DR addresses the re-use of existing complex device fault knowledge in conjunction with background knowledge for the generation of black box component models of a device. These component models represent device structure, behaviour, and function. DR thus maintains two views for the diagnostic knowledge of a single device or subsystem, a fault-based view, and a model-based one. These two views help illustrate that fault and model knowledge for the same device are related, and given one view, the second can be extracted. DR illustrates that the model view can be extracted from the fault view in conjunction with background knowledge. Other MBD researchers have shown that the fault view can be extracted by compiling diagnoses based on the model view [Bizzari et al. 90; Meisner et al. 90; Pearce 88]. Compilation can form a hybrid MBD and FBD system that produces diagnoses in finite time [Meisner et al. 90].

DR has been applied to two problems. The first application is to the Jet Engine Troubleshooting Assistant (JETA) fault knowledge [Halasz et al. 92; Abu-Hakima et al. 93b]. JETA was selected since it satisfied the requirement that an existing, and well-structured knowledge base for a complex device be used. JETA also addresses a real-world problem, the troubleshooting of a jet engine. The re-use of JETA's fault knowledge to gen-

erate a black box component model of engine subsystems, specifically, the main fuel control system, has demonstrated that DR can address a significant problem. This is supported by the validation of the model by a domain expert².

The second application of DR is to model a full device for a simpler application, a coffee maker. The coffee maker fault knowledge was formulated to demonstrate that DR could be re-used for another very different domain. In addition, it was possible to relax DR's assumptions, investigate their simplification, and show how DR can work with a minimal set of constraints for other, well-structured fault knowledge. Furthermore, the coffee maker device is one that can be more easily understood by other AI researchers, and thus, its model could be more easily evaluated by persons who are not domain experts.

To properly introduce the objectives, methodology, and contributions of the DR algorithm, it is worthwhile to review the exact definitions used in its description.

1.1 Definitions

Well-structured device fault knowledge used by the DR algorithm is defined as a set of diagnostic *frames* representing nodes that are organized in a *directed network*.

- A *frame* is a data structure commonly used in knowledge-based systems, to group a collection of fields (known as frame *slots*). Frames generally have a *name slot* that uniquely identifies them in a knowledge base, and a number of other slots whose values directly affect problem solving³.
- For the DR algorithm, a frame is assumed to have *slots* that uniquely identify:
 - the frame (or node) name, and

2. Refer to chapter 3.0, section 3.9.1 for a detailed description of the validation of DR's engine application.

3. Refer to Figure 3.3 in section 3.4 (chapter 3.0), for an example of a complex frame with 14 slots for the engine application.

- explicit inheritance knowledge (encoded as parental, sibling and/or child slots, depending on the fault knowledge application).
- A *directed network* (of which a hierarchy is a special case), organizes the frames (or nodes), of a diagnostic application according to specific relations. In the case of a hierarchy, parent-child links relate one level of the hierarchy to the next. In some applications, frame slots may encode additional implicit or explicit relations, that lead to alternate faults based on specific slot value combinations in a node. For example, slots may explicitly lead to siblings of the current node or frame. Such relations between nodes are less structured and result in directed networks rather than hierarchies. For the DR algorithm, two aspects of a directed network are used:
 - a starting point, which could be any node in the network (in the case of a diagnostic hierarchy, this could be an identifiable root node), and
 - identifiable terminal nodes which have no children.

A complex directed network is defined as one that has more than 40 nodes, and a simple one, is one that has 30 nodes⁴.

Black-Box Component Model

A *black-box component model* is shown in Figure 1.1. Such a model has an identifiable set of input parameters ($i_1, i_2, \dots, i_n$) and output parameters ($o_1, o_2, \dots, o_n$). In certain domains, such as complex control applications (e.g. process control in manufacturing, aircraft engine control signalling, etc.), some of the output parameter values are used to control the input parameter values. These outputs are identified as feedback signals ($f_1, f_2, \dots, f_n$), true in component operational modes ($m_1, m_2, \dots, m_n$). The feedback signal parameters are a function of the outputs and inputs of the component in a particular operational mode.⁵ Thus, feedback signal f_a in mode a is, $f_a = \text{relation}_a(i_a, o_a, m_a)$, where for mode a , input _{a} /output _{a} can be a combination of different inputs and outputs.

4. The engine application has a directed network of approximately 200 nodes, and the coffee maker application has a hierarchy of approximately 30 nodes. Refer to chapter 3.0, section 3.4 for the discussion of the fault knowledge input to DR.

5. The black box component modelling is based on that traditionally followed for control [Treager 79].

Feedback parameters are often identified as control variables, since their values result in increasing and/or decreasing changes, (denoted by plus or minus, +/-, signs) in the output and/or input, (I/O) parameters. Thus, a black box component model has:

- inputs, those with a +/- sign are defined as input control variables, and denote a respective increase/decrease in value that would affect component operation in the corresponding control mode,
- outputs, those with a +/- sign are defined as main output control variables, and denote a respective increase/decrease in value that would affect component operation in the corresponding control mode,
- behavioural relations, which are defined as proportional, inversely proportional, or piecewise linear:
 - an *input* control variable which has the *same change in value*, denoted by its sign (either a +/-), as that of a main *output* variable is defined to imply a *proportional I/O behaviour relation* for its component,
 - an *input* control variable whose *value changes in an opposite direction* with its respective main *output* variable, is defined to exhibit an *inversely proportional I/O behaviour relation* for its component, and
 - a component whose input and output variables exhibit a *combination* of proportional and inversely proportional *behaviour relations* is defined to have an overall *piecewise linear behaviour relation*.

Figure 1.1. Black Box Component Model

The black box component models output by DR have structure, behaviour and function as defined in the model-based reasoning literature [Abu-Hanna and Benjamins 90; Sticklen et al. 88]:

- structure is defined as the physical properties of a device such as its components, and any interconnections,

- behaviour is defined the inputs and outputs and any behavioural relations between them for specific modes of operation exhibited by a component (can be interpreted as states in the MBD literature), and
- function is defined as the goal or purpose of a component within a device⁶.

Background Knowledge

Background knowledge used as input into DR is of two types:

- The first type of knowledge is device dependent background knowledge. This knowledge is in the form of a component-dependent description. It provides glossary knowledge, mapping the fault-based encoded name of the component to a meaningful symbolic name. It also represents any component-specific modes of operation, and their respective input/output variables, with a plus/minus (+/-) sign indicating a direction for any changes in I/O parameter values (see chapter 3.0 - sections 3.6, 3.8 - and chapter 5.0 - section 5.2, for specifics).
- The second type of background knowledge is device independent background knowledge. This knowledge is in a form of a re-usable, generic description, for a component type, that gets instantiated according to the extracted component types that model a device. These generic component descriptions are designed to be placed in a design, or modelling library, so that they may be re-used for different devices. Their description includes a function (the purpose or goal of the component, e.g. to pump, to control, to act as a source, to filter, etc.), inputs, outputs, regulation inputs, and a behavioural relation, describing how the inputs change with respect to the outputs for particular modes of operation (proportional, inverse proportional or piecewise linear), (see chapter 3.0 - sections 3.6, 3.8 - and chapter 5.0 - section 5.2, for details).

1.2 Thesis Objective

The objective of the thesis is to develop and test an algorithm to demonstrate that automated model acquisition from fault-based knowledge is practical. To this end, the Diagnostic Remodeler algorithm has been developed. The objective of the DR algorithm is to discover and refine a component behavioural model with explicit function. The

6. Overall device function is not extracted or described by DR since it was not needed.

objective is to generate a behaviour model from fault knowledge which may be suitable, but its use not necessary in this dissertation, for diagnosis.

In the most general sense, the algorithm must identify the components of the device, and generate links between those components from fault-based knowledge, as well as generate hypotheses for the behaviour and function of the components. Another objective of the DR algorithm is to generate a component model that can be used to identify novel faults in the fault-based knowledge. This is interpreted to represent a validation of fault knowledge since DR can be shown to uncover gaps and inconsistencies in fault knowledge.

Method

The hypothesis of the thesis is that knowledge of faults can be re-used to extract black box component model knowledge. In this sense, the thesis addresses the problem of automated model acquisition for diagnosis. A new algorithm, the Diagnostic Remodeler (DR) algorithm, is introduced and its implementation results presented [Abu-Hakima 93a; Abu-Hakima 94a; Abu-Hakima 94b]. The DR algorithm for the automated generation of models that represent component structure, behaviour, and function, is defined, implemented, and tested. The DR algorithm uses fault-based knowledge and two types of background knowledge to generate component models of a device. The two types of background knowledge used as input are device independent and device dependent knowledge.

In the first phase of the DR algorithm, DR-1, fault-based knowledge from a typical, well-structured troubleshooting system, is used to extract component-connection rela-

tions. In the second phase, DR-2, the extracted components are matched against background knowledge to derive specific behaviours and function. In addition, all gaps and inconsistencies in the original fault knowledge component-connection links are tagged. These are shown to uncover novel faults not encoded in the original fault knowledge.

Two types of diagnostic strategies are used in this thesis. The first is the traditional fault-based strategy which uses a directed network of faults (a generalization of a diagnostic hierarchy), to represent possible problems with a physical system. This directed network or diagnostic hierarchy, is constructed on the basis of heuristics from experienced technicians, and maintenance manuals written to troubleshoot the physical device. A classical diagnostic hierarchy is shown in Figure 1.2⁷. The general fault nodes are refined successively to each subproblem level until the leaf problem nodes, which represent maintenance actions on the physical components of the device, are reached. The hierarchy is navigated by a reasoning algorithm, which implicitly encodes some of the heuristics of the expert (for example, reason about the most common faults that are easiest to check, first). The knowledge base, the reasoning algorithm, and the user interface then form the diagnostic system used by the maintenance technician. Occasionally, the dynamic trace of a diagnostic session is kept as an example in a case base for future reference.

7. A fault hierarchy for the engine application is shown in Appendix B.1.

Figure 1. 2Diagnostic Hierarchy

The second diagnostic strategy this work relies on is a model-based one. The model represents the actual physical components of a device, connections between them, and the behavioural states the components can exhibit. Thus, a physical device may be modelled on the basis of its structure and/or its behaviour. A structural model could be component, process, or constraint oriented. It is dependent on what behaviour one wishes to diagnose. For example, an electronic circuit could be modelled as a set of components (resistors, diodes, transistors, etc.) with input/output characteristics mapped from the component connections (wires). A process and object model could represent a physical process such as that used in an oil refining operation. A third type of structural model is a constraint-oriented model whose set of possible faults correspond to some subset of model input/output variables that relate the individual and aggregate behaviours of the physical components. A behavioural model represents a set of constraints, or relations, among input/output variables, and internal parameters. Behavioural models could be temporal, discrete, continuous, dynamic, etc. For the thesis, a combined behavioural model with an explicit representation of function is used. One of the most difficult problems in model-

based diagnosis is the generation of complex models. This is generally achieved by a human expert. This thesis addresses the problem of automated model acquisition from fault-based knowledge.

1.3 Contributions

Five main contributions have resulted from this Ph.D. thesis:

- The definition, implementation and testing of the general Diagnostic Remodeler algorithm, which can automatically acquire a component behaviour model with explicit function for a device, or device subsystem, has been achieved.
- The Diagnostic Remodeler algorithm results (the black box component models) can be used to validate, and uncover novel faults missing from the original fault knowledge. DR has been successfully applied for modelling a complex real-world system, namely the Main Fuel System, for the J85-CAN-15 gas turbine engine that powers the CF-5 jet aircraft, used to train pilots of the Canadian Forces. The input to the first phase (DR-1) of the algorithm for the engine model is the Jet Engine Troubleshooting Assistant (JETA) fault knowledge. The input to the second phase of the algorithm, DR-2, is device dependent and device independent background knowledge acquired and encoded with the help of an aircraft engine Expert, Jeff Bird, at NRC's Propulsion Laboratory. The background knowledge includes explicit representation of inter-component feedback, and multiple component inputs and outputs common in control problems but not traditionally addressed in acquiring or using models in the AI community. In addition, the Main Fuel System model output by DR has been validated by the engine Expert. The generated model has been used to uncover novel faults, identified as gaps and inconsistencies in JETA's fault knowledge. In addition, the model illustrates that JETA is missing a layer of knowledge, specific to control component faults. Hence, the DR generated model has uncovered several novel faults not encoded in JETA (for discussion of DR's engine application gaps see chapter 5, section 5.5).

- The Diagnostic Remodeler software has been successfully re-used for a second very different problem, the automatic model acquisition of a full device component model from fault knowledge, for a coffee maker. The model again uncovers some gaps and inconsistencies in the fault knowledge. The coffee maker example demonstrates the generality of the algorithm and its potential wide applicability across various domain applications.
- In implementing the DR algorithm's device independent knowledge, it was possible to formulate new generalized component descriptions, which were successfully re-used with minor modifications between the engine and the coffee maker domains. These generic descriptions present a novel manner to describe component function (what a component is designed to do), inputs, outputs, primary control variables and resultant behaviours. Behaviours are specifically represented as proportional, inverse proportional or piecewise linear, input-to-output relations, as dictated by the respective I/O variables whose values change according to the mode the component operates in. These modes categorize component behaviour relations in device operations. The generic component descriptions can form the basis for CAD (computer aided design) libraries so that component models may be re-used for device design or diagnosis.
- The fifth contribution is the thorough review of techniques relevant to diagnosis in Artificial Intelligence [Abu-Hakima 94c]. This review was completed to confirm that the DR algorithm concepts are unique, and have not been addressed in the related AI literature. The review is supported by an extensive annotated bibliography [Abu-Hakima 94c], intended as a guide for the reader to significant and related work. The review is subdivided into eight sets, with the first five, the most relevant, analysed and summarized in chapter 2.0. They are:
 - i. fault-based diagnostic techniques
 - ii. model-based diagnostic techniques
 - iii. case-based reasoning techniques relevant to diagnosis
 - iv. machine learning for knowledge acquisition relevant to diagnosis
 - v. integrated diagnostic techniques

1.4 Thesis Structure

In the remainder of the thesis, the literature review is presented in chapter 2.0, the Diagnostic Remodeler algorithm design is described in chapter 3.0. The implementation of the DR algorithm's individual software modules are described in chapter 4.0. The results of applying the DR algorithm to JETA's complex jet engine fault knowledge to model the Main Fuel System is described in chapter 5.0. In addition, the results of applying DR to coffee maker fault knowledge to generate a full device component model is described in chapter 5.0. The achievements of the thesis and directions for future work are discussed in the concluding chapter 6.0. Thesis references directly mentioned in the chapters are listed in chapter 7.0.

Appendix A contains information relevant to the engine application. Appendix A.1 has a graphical representation of JETA's fault knowledge. Appendix A.2 documents the correspondence between the engine expert and the author, and includes generated background knowledge descriptions. Appendix A.3 contains the raw DR output models validated for the engine application. Appendix B holds information relevant to the coffee maker application. Specifically, Appendix B.1 contains the coffee maker fault knowledge base used as a second application to test DR. Appendix B.2 contains the test results for modelling the full coffee maker device using its respective fault knowledge.

2.0 Relevant Artificial Intelligence Techniques

Chapter 2.0 provides an analysis and summary tables for the literature reviewed and annotated in [Abu-Hakima 94c]. The review is intended as a guide to the approaches, applications and issues of using AI techniques to research and develop diagnostic systems, with special emphasis on knowledge acquisition. The review was subdivided into eight sets of publications, the summaries of the first five sets of papers are directly relevant to diagnosis and make up sections 2.1 to 2.5 respectively. The following form the eight sets:

i. Fault-Based Diagnostic Techniques

These papers are representative of reasoning on the basis of common faults and troubleshooting to isolate a problem and suggest a subsequent repair. The knowledge in these systems is mostly based on rules of thumb (heuristics) of experienced technicians and repair manuals. Some knowledge is manually interpreted sensor data which the technician is asked to incorporate. The knowledge is often represented as rules or frames in decision networks or troubleshooting trees.

ii. Model-Based Diagnostic Techniques

This set reviews papers that describe reasoning on the basis of quantitative or qualitative device models to diagnose failures. Quantitative models include simulations and numerical models. Qualitative models include structural, behavioural and functional models.

iii. Case-Based Reasoning Techniques

Case-Based reasoning examines past cases and uses the results of past case solutions for problem solving. This technique, although applicable to diagnosis, has thus far had limited use.

iv. Machine Learning for Knowledge Acquisition

Machine learning is a key technique for knowledge acquisition. Machine learning includes empirical and analytic learning. Empirical learning focuses on learning for classification (including learning rules from data for performing shallow diagnoses). Analytic learning addresses learning for problem solving tasks. Such tasks include planning, design, natural language understanding, control and execution.

v. Integrated Diagnostic Techniques

These papers describe work that integrate AI techniques for diagnosis. The integrated diagnostic techniques combine:

- data analysis and interpretation for diagnosing faults including machine learning,
- reasoning based on common faults and troubleshooting to isolate the problem,
- reasoning on the basis of numerical or behavioral models to diagnose failures,
- examining past case solutions and using the results to diagnose new faults.

vi. Knowledge-Base Management

This set addresses the problem of knowledge base creation and management. This is a key problem in the development and implementation of any knowledge-based application. The annotated bibliography for these papers is in [Abu-Hakima 94c].

vii. User Interface

Some papers describing user interface approaches for diagnostic systems are reviewed in [Abu-Hakima 94c]. The user interface is a key aspect of any software project including diagnostic applications as it provides the front end for the target user.

viii. Overviews

Surveys and overviews relevant to diagnosis are reviewed in [Abu-Hakima 94c].

The papers in the first five sections of the review are most relevant to the Diagnostic Remodeler algorithm. For each of the five sets of related work the key papers are discussed and a tabular summary highlights the novelty of the approaches. In chapter 3.0 the uniqueness of the Diagnostic Remodeler algorithm is related to the published literature and the novelty of the algorithm is described.

2.1 Fault-Based Reasoning¹

Fault-based reasoning (FBR) is used in many diagnostic systems. FBR knowledge is largely based on maintenance manuals and interactions with an expert to capture heuristic knowledge about the maintenance and repair of a device or process. The knowledge in these systems is often represented as rules or frames which are organized into troubleshooting hierarchies such as that shown in Figure 2.1. At the top level of the hierarchy is the general knowledge representing a problem with the device. This general problem is refined systematically until the leaf nodes of the hierarchy which represent physical repairs or adjustments to the device are reached. Once these repairs are completed by a

1. Note that summaries for fault-based reasoning are in annotated literature review of [Abu-Hakima 94c].

technician, some diagnostic systems re-test for the fault(s) by backtracking through the hierarchy to confirm that any faults have been cleared.

Figure 2.1. Diagnostic Hierarchy

FBR systems have evolved considerably since the development of MYCIN [Scott et al. 77]. MYCIN was developed to provide advice treatment for microbial infections. MYCIN inspired several follow-on systems such as NEOMYCIN which had an expanded knowledge base and EMYCIN, a shell based on MYCIN [Clancey 86]. Clancey also used EMYCIN to develop GUIDON, an intelligent tutoring program for diagnosis [Clancey 86]. The MYCIN programs started with rules and evolved into meta-rules in NEOMYCIN which provided some structure to an otherwise flat knowledge base. These programs also advocated the use of certainty factors which are probabilities attached to the rules and combined to provide a measure of certainty or uncertainty to a diagnosis. The MYCIN approach remains a very widely used approach in FBR systems as exemplified by the literature review of [Abu-Hakima 94c].

Chandrasekaran is one of the earlier advocates for more structured organization and reasoning in FBR. He argues for the use of structured hierarchies such as that in MDX

that could be generalized for diagnosis [Chandrasekaran et al. 79]. MDX is a medical FBR system that diagnoses cholestasis, a form of liver disease [Chandrasekaran and Mittal 83]. Chandrasekaran also argues for the use of local uncertainties in the nodes of the hierarchies that could be combined in evaluating the evidence for final sets of diagnoses [Chandrasekaran et al. 82]. One of the interesting results discussed in another paper is that the hierarchical activation in diagnostic hierarchies is more computationally efficient than rule-based pattern matching [Goel et al. 87]. This is one of the few results in AI that evaluates computational complexity for testing the efficiency of one approach over another. Chandrasekaran introduced the argument for generic tasks [Chandrasekaran 86]. His landmark paper on the use of hypothesis-based structured reasoning to solve problems across various application tasks in AI is still widely referenced. In it, he proposes generic tasks which share a common problem solving methodology be it, diagnosis, design, planning or any other task.

In RATIONALE, many of the ideas advocated by Chandrasekaran for structuring FBR systems and handling uncertainty are applied [Abu-Hakima 88]. This is a good approach for explicitly representing knowledge so that it may be explained [Abu-Hakima and Oppacher 90]. RATIONALE diagnoses faults with Xerox workstations. RATIONALE generates dynamic and static template-based explanations that include why, how and what-if responses. Explanation remains a major objective of FBR systems and most systems have why and how explanation, but do not necessarily generate hypothetical (what-if) ones.

In the Jet Engine Troubleshooting Assistant (JETA), knowledge is structured as frames in a diagnostic network that is hierarchical in nature (see Appendix A.1 for a

graphical view), in a manner similar to what Chandrasekaran advocates [Halasz et al. 92]. What distinguishes JETA from many other FBR systems is that the troubleshooting methodology of the technician is implicitly represented in the diagnostic hierarchy. JETA is applied to a complex real-world application, the diagnosis of the J85-CAN-15 engine which powers the CF-5 jet aircraft used to train pilots in the Canadian Forces. It was developed at the National Research Council of Canada. JETA will be described in Chapter 3.0 as it is used as one target application for the Diagnostic Remodeler algorithm.

2.1.1 Noteworthy Papers

Table 1 provides a summary of the noteworthy papers discussed further below in the annotated literature of the approximately 50 FBR papers reviewed in [Abu-Hakima 94c]. FBR is the most mature of the AI techniques used for diagnosis. Applications range from ones in medicine to space. The key issues that remain in FBR are uncertainty handling and structured knowledge acquisition. Some developers prefer the use of mean time between failures (MTBF) data from the manufacturer in combination with fault histories to attach probabilities to represented knowledge [Goker and Kuru 90]. Other developers argue for the use of belief networks which have been successful in a range of applications [Henrion et al. 91].

Sensor data is often used as input to a diagnostic system with a monitoring component. Hughes has discussed 100 telemetry parameters that monitor spacecraft and advise ground control crew on potential problems [Hughes 89]. Simulator data is also used to generate or check FBR knowledge [Wagner 91]. Machine Learning is often used to interpret and find patterns in data so that it may be used in FBR as described in section 2.4.

Explanation-based learning, often referred to as speed-up learning, has been used by some developers of diagnostic systems to refine and shorten reasoning paths in the knowledge base as described in sections 2.4 and 2.5.

Table 1. Fault-Based Reasoning (FBR) noteworthy papers

Who	System	FBR novelty	Application
[Attias and Sveitis 90] McDonnell Douglas St. Louis, Missouri USA	INSIDE	none	Aircraft Inertial Navigation Diagnosis
[Bramucci et al. 89] A&C Rome Italy	SADE	none	Liver Disease Diagnosis
[Ebrahimi and Perry 88] Abacus and Rockeddyne USA	FICX	Correlates failure modes, English text using keywords and performance sensor data.	High Pressure Turbo Pump of Space Shuttle Main Engine (SSME)
[Goker and Kuru 90] Borgazici U. Istanbul Turkey	---	Uses MTBF, parts data and fault histo- ries in calculating uncertainty.	Turkish Airlines 310 Airbus Diagno- sis
[Hughes 89] NASA Goddard Greenbelt, Maryland USA	CLEAR	Monitors 100 param- eters in real time for diagnosis.	Troubleshooting the Cosmic Background Explorer (COBE)
[McBryan et al. 90] McDonnell Douglas St. Louis, Missouri USA	---	Uses geometric pat- terns to track and predict target trajec- tory.	Tracking Targets on Cockpit Radar
[Wagner 91] General Dynamics Fort Worth, Texas USA	---	Controls faults advi- sor developed through trial and error with simulator.	Diagnosing C130 Aircraft
[White 91] U. of Trondheim Trondheim Norway	VIBEX	Simulation used to analyze vibration data and FBR tree used for troubleshooting.	Diagnosis of Turbomachinery

2.1.2 Summary

Two problems with FBR that are often cited are the difficulty in maintaining the knowledge base and in dealing with novel faults. In response to these two key problems, researchers are developing other approaches such as model-based and case-based reasoning described in sections 2.2 and 2.3 respectively. These two approaches are computation-

ally expensive and in an effort to make them more efficient are combined with FBR. The integrated approach to diagnosis is described in section 2.5. Despite its shortcomings FBR has proven to be a very effective means of developing diagnostic tools. There have been many successful real-world applications based on FBR as exemplified by the wide range of applications in the annotated literature review.

2.2 Model-Based Reasoning²

Model-based reasoning (MBR) for diagnosis concentrates on reasoning about the expected and correct functioning of a device as illustrated in figure 2.2. A device is modelled based on its components and their expected behaviour [Hamscher and Struss 90]. Such models range from quantitative to qualitative ones. All models attempt to approximate device behaviour as accurately as possible, or as required by the application. Once a device model is stable then a device's observed behaviour can be predicted from that model. If a discrepancy in behaviour is detected, then possible candidates that represent assumed component faults can be generated. These candidates are generated based on assumptions that describe correct model behaviour. Sequential diagnosis is used to choose observations, augment a prediction for the candidate faults and update the list of candidates until a dominant candidate is found.

In MBR there are many approaches and levels of detail for models. They range from causal black box component models represented as semantic networks, with links specifying the relations between them, to full blown numerical simulations for complex systems and processes, that have taken decades to perfect. Widman and Loparo provide a

2. Note that references for model-based reasoning are in the annotated literature review [Abu-Hakima 94c].

survey of AI and simulation and highlight their relation to model-based reasoning [Widman and Loparo 89]. The authors point out that simulation can benefit from deep reasoning and hypothesis verification, while knowledge-based systems can make simulation packages easier to use and enable them to provide more flexible models. The authors point out that AI and simulation:

- both represent knowledge and expertise about a device and its behaviour,
- both need to represent and reason about uncertainty,
- both use logical decisions to control program flow, and that
- one can equate reasoning from first principles in deep causal models to simulation.

Figure 2.2 Model-Based Reasoning for Diagnosis

Generating models is a key problem in MBR. Some researchers generate causal models, others generate models with structure and behaviour while others generate functional models for devices. Some of the representative modelling approaches are highlighted in this summary.

Sykes gives six key steps for developing a qualitative model (QM) [Sykes et al. 88]:

- 1. define and bound a physical system by identifying its components, their allowable states, input/output attributes, eligible values for attributes, transfer functions and component interconnections,**
- 2. build a QM using a network where each node represents the specific behaviour of a component in the physical system,**
- 3. run the QM to predict system behaviour under normal/abnormal conditions and tabulate results in a symptom/malfunction matrix,**
- 4. decompose the symptom/malfunction matrix into symptom value/attribute pairs,**
- 5. generate an optimal search procedure that takes into account costs of testing the attribute values and component reliability, and**
- 6. implement a diagnostic system as a query/response to the optimal search procedure.**

Davis was one of the earlier proponents of MBR [Davis 84]. He describes a theory to exploit reasoning on the basis of device structure and behaviour. He defines paths of causal interpretation. He also describes constraint suspension used to identify which components are responsible for which faults. He argues that we need to balance complexity versus model completeness in diagnosis. Thus, we need to enumerate and layer categories of failure. He also advocates the use of the relaxation of assumptions to bridge faults in the topology itself. This is one of the landmark papers in the area of model-based reasoning. Quite a bit of work has followed Davis' examples and theories.

De Kleer and Williams published a key paper on MBR for diagnosis describing GDE, the General Diagnostic Engine [de Kleer and Williams 87]. GDE infers behaviour from device structure and functionality. It is applied to digital circuits and makes use of an ATMS (Assumption-Based Truth Maintenance System). This work forms the cornerstone of ATMS-based MBR systems. It was followed by many papers that criticized the approach as not computationally practical in diagnosing faults with large complex systems. Some of the papers criticizing GDE propose the use of hierarchical fault-based rea-

soning to reduce the computational complexity of de Kleer and Williams' approach.

Struss lists four shortcomings of de Kleer's GDE [Struss 89]:

- an inability to deal with a large number of components,
- no temporal reasoning capabilities,
- missing fault knowledge, and
- uncertain measurement handling.

He then describes how GDE+ handles: simple dynamic aspects, multiple tests, hierarchical knowledge and unreliable observations. He enhances GDE to include hierarchical device subsystem reasoning, temporal reasoning, the unknown fault assumption to address intermittent faults and uncertain measurements. GDE+ is a migration back to heuristic or empirical diagnoses using fault-based reasoning. Struss points out that neither GDE nor GDE+ address: changing device structures, complex temporal behaviour (feedback), uncertainty or the use of qualitative models in reasoning. Struss and Dressler describe GDE+'s ability to exploit contradictions between the assumed correct behaviour of components and observations [Struss and Dressler 89]. They analyze whether faultiness of components would explain observations. The paper argues for the representation of a fault view for each component. It points out that a fault and a healthy view (state) for a component cannot be true in the same time instant (consistent belief rule). It also gives the 'no good inference rule' where the node and its converse, which represents a fault, cannot be true at the same instant. The GDE ATMS is then modified to reason with the fault as well as the no-fault behaviour of a device. Their work gives excellent insight into combining model and fault-based diagnosis to deal with GDE's shortcomings.

A noteworthy algorithm, IDA, the Incremental Diagnostic Algorithm (implemented in Prolog), makes use of constraint logic programming on a weak fault model to

bound the computational complexity of MBR [Mozetic 92]. The algorithm is applied to a simple adder circuit and it is not clear how well it would scale up to a complex problem such as an aircraft engine.

MBR authors have argued about the definition of device functionality versus behaviour. Sticklen describes modelling a device's functionality [Sticklen et al. 88]. This is achieved by:

- decomposing the device into sub-devices,
- stating abstractly the functions, goals and purpose of the device, and
- representing the manner of achieving the device functions, goals and purpose.

This functional modelling can be likened to distributing the causal knowledge of the device. In naive physics, changes in the small are used to derive changes in the large. This is achieved by moving up or down a hierarchy representing the process, or device being modelled. In a causal network, changes are woven into a comprehensive net. A detailed device representation strategy for device components and functions and their inter-relations is provided by the authors. They argue that instead of trying to derive device and component behaviour (using naive physics) they are trying to make use of behaviour in device modelling.

Malin and Liefken give a good definition of functionality [Malin and Liefken 91]. They argue that functionality is the set of goals the device is intended/designed to achieve. Thus, to model functionality one needs to model the structure, behaviour, goals and the timing of the device. Structure is device components and their interrelations. Device behaviour is what the device does. The approach has been applied to the space shuttle remote manipulator system (RMS) as a demonstration for how it is applicable to complex

problems. At the AAAI-94 workshop on device function, function was defined as more abstract than behaviour but generally followed Sticklen's definition and it does not closely relate function to behaviour as Malin and Liefken do [Kumar 94].

Abu-Hanna and his colleagues describe a model typology and a model ontology [Abu-Hanna et al. 92b]. As illustrated in figure 2.3 the typology is a heterarchy of model types which has at the root the model which is specialized to a case (specific device model) or a domain (representing a family of devices). The case is then refined to a kernel or abstract type. The kernel is then refined to device structure, behaviour or topology. The abstract type is refined to function and state. The domain of devices type is refined to device and universal which are further refined to kernel, abstract and use. In this manner, the family of devices can essentially use the knowledge of the individual devices. The model ontology is what defines a model: its components, functions and connections. This is represented as a semantic network. The approach combines aspects of traditional model-based and fault-based reasoning and attempts to look at the many views that knowledge of a device can represent based on the eventual use of that knowledge. The approach is demonstrated on a simple car ignition problem. The scalability of the approach (to the diagnosis of an aircraft engine for example) is questionable. However viewing knowledge in multiple contexts as such, is new and quite related to the ideas of the Diagnostic Remodeler (DR). The DR algorithm re-uses fault base knowledge in a new context, namely the construction of a model rather than simply for fault based reasoning as was the original intention of acquiring the knowledge.

Figure 2.3 Model Typology [Abu-Hanna et al. 92b]

2.3.1 Noteworthy Papers

Table 2 provides a summary of noteworthy papers, discussed further below, found in the literature that advance the techniques and application of MBR. Azmoodeh describes the GMS (Generic Maintenance System) work at British Telecom [Azmoodeh 91]. He argues that MBR is needed to adapt maintenance to new networks or changes in network designs. GMS uses a simulation to generate predicted network behaviour and is used to diagnose integrated broadband communication networks. It has a knowledge base of the generalized structure and behaviour of network components. Its model is represented as a semantic network with constraints representing structure and behaviour applied to the generalized descriptions of objects (components). Azmoodeh gives a good description of how the constraints are represented and how to apply them to network objects.

Bonarini and Sassaroli organize knowledge across three dimensions: level of abstraction, uncertainty and epistemological type (teleology, process, function and struc-

ture) [Bonarini and Sassaroli 92]. Their work attempts to address the problems of uncertainty and incomplete knowledge in MBR. They have implemented OMISSYS (Opportunistic Multimodel-based diagnosis SYStem). The human modeler and diagnostician give subjective confidence in each knowledge element. The knowledge is represented using possibly qualitative relations among physical variables and possibly approximate (fuzzy) values. OMISSYS navigates opportunistically through different models to reach a diagnostic goal. The navigation is guided by diagnostic meta-knowledge which ranks all possible choices while trying to satisfy the final goal state. OMISSYS has been applied to the diagnosis of an Industrial Plant. The possible moves through the knowledge are:

- simulation moves (e.g. simulate a particular fault),
- shift focus moves (e.g. expand current focus (specialize), try next model, generalize from current focus),
- ask data moves (measure value on device through human operator), and
- use heuristics.

OMISSYS is a good example of an opportunistic MBR tool.

Horak describes a tool, FDI (Failure Detection and Isolation), designed to detect structured and unstructured failures in a dynamic system [Horak 90]. Structured failures are diagnosed based on actual versus simulation device measures for various subsystems. Unstructured failures are detected by partitioning the system into subsystems with one assumed to be faulty and the others not. Measured data is then compared to that from a simulator until a faulty subsystem can be isolated. It has been applied to F-16 flight control and turbofan engine simulations.

Table 2. Model-Based Reasoning (MBR) noteworthy papers

Who	System	MBR novelty	Application
[Azmoodeh 91] British Telecom UK	GMS	Represent model as semantic net with constraints that represent structure & behaviour applied to generic object descriptions.	Broadband Communication Networks
[Bonarini and Sassaroli 92] Politecnico di Milano Milan Italy	OMISSYS	Navigates through several models opportunistically. Moves include: simulate, shift focus & get data.	Industrial Plant Diagnosis
[Horak 90] Allied-Signa Aerospace Co. Columbia, Maryland USA	FDI	Detects structured faults based on simulation Vs. actual data measured. Detects unstructured faults by assuming 1 faulty component and propagating behaviour. Isolated component data compared to simulation to id fault.	F-16 control system simulation
[Leitch and Stefanini 89] Heriot-Watt U Edinburgh, Scotland and CISE Technologie, Milan Italy	Toolkit	5 layers of knowledge: strategic, tactical, theological, functional & object More advanced than Abu-Hanna's work	power plant thermal cycle diagnosis, data interpretation & spacecraft control, cement manufacturing plant start-up & control
[MacIsaac and Muir 91] GasTops Ltd. Ottawa, Ontario Canada	---	Model aircraft engines using performance parameters while generating fault libraries to typify behaviour.	F-18 Aircraft

The implementation of FDI emphasizes the importance of a 'divide and conquer' approach in model-based diagnosis (already inherent in fault-based diagnosis). The only drawback in this system is the assumption that very accurate simulations can be implemented for complex systems. Accurate complex simulations require a high investment in model development by experts in the field. Thus, Horak's approach is useful for a limited set of MBR problems which have accurate expensive simulations to rely on.

Toolkit has five layers of knowledge [Leitch and Stefanini 89]. The layers of knowledge are: 1) strategic, 2) tactical, 3) teleological (raison d'être), 4) functional (generic functions) and 5) object (where objects are represented). Toolkit has been applied to three very different problems: power plant thermal cycle diagnosis, data interpretation and control of spacecraft, and cement manufacturing plant start-up and on-line monitoring. Toolkit has various task application primitives which it combines to solve problems at hand. This work is very related to, but much more advanced than Abu-Hanna's approach of model topologies and ontologies. It is more advanced as it has already been applied to three different complex problems. Toolkit's knowledge representation includes: a component-based language, an event-graph language and a fuzzy rule-based language. The paper includes a table which summarizes the facilities of the toolkit (e.g. diagnosis, control, simulation, monitoring, interpretation, prediction, identification, etc.). This is excellent work in a complex real-world application and is quite novel in the practical use of MBR.

A traditional, but very valid approach, for the use of performance models for the diagnosis of aircraft engines, has been developed by GasTops Ltd. [MacIsaac and Muir 91]. This approach to performance analysis consists of, converting the generalized stage, (e.g. compressor, combustor, turbine, etc.) characteristics of engine components, to those specific to the engine being modelled. This is achieved by using design point data and/or engine test data. Once the stage characteristics are generated, they are combined to produce overall component maps, and subsequently the entire engine model. Faults are simulated by incorporating a particular fault in the data and propagating its effect to show the effect on the engine. The authors regard this model to be an essential tool of performance monitoring systems. Once the model is validated, it is used to systematically generate a

library of fault signatures. The fault signatures are used, in an on-line mode, to characterize engine behaviour so that a cause of failure may be identified.

2.3.2 Summary

There are many active areas in MBR research for diagnosis, as it is less mature as an AI technique than fault-based reasoning. One of the issues that is heatedly pursued, and has been discussed above, is that of the definition of device structure and behaviour, versus device functionality. Handling uncertainty and bounding complexity in model-based reasoning, are two other areas of important MBR research. Representing and explaining causality is another open area in MBR. Combining MBR with fault-based reasoning to bound computational complexity is another active area with researchers such as Struss. A relatively new and promising area in MBR is exemplified by work from Abu-Hanna, his colleagues, Leitch, and Stefanini, who are all exploring the use of classes of models, to address various application tasks such as diagnosis, scheduling and control.

An area where no work is evident is in automated model acquisition from fault knowledge. Conversely MBR researchers are actively pursuing the generation of FBR knowledge from models used in MBR.

2.3 Case-Based Reasoning³

Case-based reasoning (CBR) is the ability to reason on the basis of past problem solutions. Key issues in achieving this include, indexing cases and representing features as well as adapting cases to new problems. The ability to repair a case that has failed in pro-

3. Note that all case-based reasoning references are in the annotated literature review of [Abu-Hakima 94c].

viding a solution and learning in CBR are also issues. Research and new implementations of CBR systems that include CBR shells, have enjoyed an explosion of activity since 1990. As this summary illustrates, CBR has been applied successfully to many problems including negotiation, planning, design and cooking. It has had some application in diagnosis. Fault-based reasoning can be viewed as a form of generalized case-based reasoning, with FBR being much more successful at focussing its search on relevant branches of a diagnostic hierarchy.

At the American Association of Artificial Intelligence (AAAI) conference of 1992 two of the key players of CBR were invited to share their views on the field. Roger Schank who is considered the founder of CBR gave a very simple definition: "case-based reasoning equals reminding". Chris Hammond echoed this by giving an even simpler definition: "case-based reasoning is remembering". Hammond emphasized that solutions, errors and repairs (optimizations on previous solutions and new results) should be recalled in knowledge-based systems to simplify the task of finding a solution.

Schank and Slade describe why CBR is the answer to the problems of rule-based reasoners (expert systems) in [Schank and Slade 91]. They point out that CBR allows a system to learn from experience and build up an episodic memory, much like that of a human. They argue that explanation-based learning (EBL) is far superior to induction in adapting the case knowledge for new problems since it constructs chains of reasoning rather than simply clustering points in space. One problem they do not address convincingly, is that of human knowledge re-organization and re-structuring. This has yet to be successfully modeled, and automated in any form in AI programs.

Slade reviews semantic network memory theory originally put forth as the basis for the use of human language [Slade 91]. He describes episodic memory proposed to replace the semantic network theory for memory. Episodic memory links temporal events with relations, scenes, occurrences and stories. Conceptual memory which is a lexical memory organization with information about words, idioms, common expressions and is language-free is described to be episodic memory. Scripts were proposed by Roger Schank to represent episodic memory. Slade argues that we also need the processes to acquire and access the structural representation of episodic memory. He describes his process model for case-based reasoning which includes: the assignment of indices, case retrieval, case modification (adapting), case testing, the assignment and storage of successful cases in the case base, and the explanation of case failure, repair and re-testing. Slade provides a good review of the beginnings, motivations and trends in CBR [Slade 91]. He lists 27 recently (1986-89) implemented CBR systems. Slade argues that we still need to develop a scientific model for human memory. He concludes that we need to build more robust programs that assimilate experiences and adapt to new situations.

Janet Kolodner is another key founder of CBR. Kolodner's philosophy for the use CBR is that, "people naturally reason using analogs" [Kolodner 91]. When people do not have previous knowledge, (are novices in an area) or do not have problem solving knowledge, they can turn to CBR tools. Such tools make assumptions to fill in missing and incomplete knowledge. To her, CBR is a method of analogical reasoning where old solutions are mapped to solve new problems. She also believes that CBR learns by remembering, but it can be combined with inductive learning or explanation-based learning, to improve its case structure. She describes CBR control through case indexing, and presents

a good research problem (e.g. what features of a case should be selected to typify it so that it can be accessed or adapted for re-use?). MEDIATOR is an early CBR system that Kolodner developed to resolve treaty disputes. She has described the lessons learned from the MEDIATOR project [Kolodner 89a]. MEDIATOR uses a planner, can recognize failures, and generate explanations based on the case errors by backtracking through the failure with a trace. Kolodner argues that CBR is useful since it:

- has broad applicability, since it can be used to classify problems, choose a planning strategy, and generate explanations as a mechanism for failure recovery,
- provides reasoning shortcuts on long chains of inferences, which can be re-used for quicker subsequent solutions, and
- anticipates, resolves, and can avoid future case base failures.

JULIA is a CBR system implemented by Kolodner for reasoning with failed cases in catering meals [Kolodner 87a]. Six steps are described to reason on the basis of failed cases:

1. recall reasoning sequence if failed case was not followed up on yet,
2. recall or find the reason the case failed (this could be missing/false information or missing/false problem solving strategy),
3. find the relation between the current focus and the previous failure, and re-focus reasoner,
4. recall inference rules, and justify conditions used, to infer focused-on portion of failed case,
5. check to see if the potential for failure exists in the new case, by reasoning through conditions on failed and new case, and
6. evaluate and re-work case based on results of step 5.

Kolodner provides a good summary of reasoning on the basis of case failures. She emphasizes elsewhere that case-based inferences can form shorter chains than rule-based inferences, since case-based reasoners can use failure for improving solutions [Kolodner 87b].

PARADYME is a case memory organization scheme for **JULIA** [Kolodner 89b].

PARADYME retrieves a small set of best matches from memory and uses 6 heuristics for deciding the best case. Kolodner provides a good summary of these six preference criteria for case selection:

1. goal-directed preference, which prefers those cases which meet the current goal,
2. salience, which uses experience to tell us what features to focus on,
3. specificity, which implies that less specific features are a subset of the more specific features attached to a case which in turn is a better case to match on, it also implies that a case with more parts with features that match is a better case to match on and it also implies that a case that is more general is placed higher in the generalization hierarchy and hence is preferred,
4. frequency, which implies that a case which is accessed more is preferred,
5. recency, which implies that a case which is more recently accessed is preferred, and
6. ease of adaptation heuristic, which implies that a case which is easier to fix is preferred to one which is harder to fix - this implies that a case is selected on the basis of it being more useful rather than similar.

Kolodner has discussed two somewhat novel applications of CBR [Kolodner 91].

One of these is in Battle Management instruction. The CBR system is in use at West Point and was developed by Cognitive Systems Inc. It has a case base of 600 battles which are used to instruct students.

In the AAAI-92 tutorial on CBR, Ashley and Simoudis give a detailed review of case-based reasoning methods and implementations [Ashley and Simoudis 92]. They review five paradigms with which CBR has been combined:

1. statistics, the MBRTalk system which pronounces novel words is used as an example,
2. model-based reasoning, as an example they describe CASEY implemented by Phyllis Koton which uses a causal model of the heart to support CBR [Koton 89],
3. planning and design, they describe two examples, CHEF, a planner implemented by Chris Hammond which prepares meals and PERSUADER implemented by Katia Sycara to mediate labor negotiations [Sycara 90; Sycara 91],

4. **exemplar-based reasoning**, they describe PROTOS by Bareiss and Porter which classifies patients with hearing problems and
5. **adversarial and precedent-based reasoning**, they describe HYPO by Ashley which is a system that analyses trade secret disputes [Ashley 89].

The tutorial also described CLAVIER [Hennessy and Hinkle 92], CASCADE [Simoudis 92] and CABER. CABER has been developed by Evangelos Simoudis as a means of recovering from milling machine failures. The case base was generated from a simulation of the milling machine. The tutorial shows the level of effort exploding in the field of CBR.

CASCADE is a novel case-based system implemented by Simoudis as part of his Ph.D. [Simoudis 92]. It uses CBR to advise on VMS operating system crashes. It uses both top-down and bottom-up case retrieval to advise a user. Its cases were derived from 200 problem reports on a help desk.

Another new application of CBR is KRITIK, a system for design [Goel and Chandrasekaran 90]. Its authors argue that design requires a task analysis structure, where an abstract task is characterized by the types and forms of knowledge used in addition to its sub-tasks and processing strategies. KRITIK has 17 modules that represent its task structure. It is used to design heat exchange systems and electric circuits. The design task is described as a behaviour-to-structure task. It is not clear how practical this approach would be to a complex real-world application. In an earlier paper, they apply device models to adapt cases for the design of a cooler [Goel and Chandrasekaran 89]. Their use of CBR is akin to *caching* the simulation run of a device model for future use. It is unusual to apply CBR in this manner. It would be more appropriate to use CBR in an attempt to

refine device models. Again, it is not clear whether the cooler problem would scale up to a more complex real-world problem.

CADET is another CBR design tool that aids in the design of hydromechanical systems [Navinchandra et al. 91a]. It uses knowledge of domain laws, design principles and behavioral constraints for a generative approach. It reasons from design goals to solutions. It has been applied to some simple problems such as the flow of water out of a tap. The combination of qualitative physics with CBR to achieve design goals is novel. However, CADET's scalability and application for complex real-world problems is questionable.

2.3.1 Noteworthy Papers

Table 3 is a summary of recent noteworthy approaches and applications in CBR. HYPO is an adversarial system which uses previous cases to justify its solution [Ashley 89]. It aids an attorney in arguing a case. It generates a chain-lattice to pull together relevant cases in the case base. Salient features of a case are formed by: (1) relevant case similarities defined as an intersection of features and (2) relevant differences defined outside any intersection. This work is novel in its definition of case salience and its approach to legal reasoning.

A novel problem that CBR has been applied to forms a collection of tools for sheet metal manufacturing [Cser et al. 91]. CBR is used in three different tasks. The first is in synthesis for metal bending, using a case base of process variables and metal profiles. The second is for the classification of sheet metal parts, using a semantic network which

encodes geometric structural features of metal. The third is for the diagnosis of the machine and tools in metal manufacturing, using extracted features in cases.

Table 3. Case-Based Reasoning (CBR) noteworthy papers

Who	System	CBR novelty	Application
[Ashley 89] IBM, NY USA	HYPO	Adversarial system that argues points of law.	American Law
[Csar et al. 91] U. Erlangen Germany	---	Three CBR tools: synthesis, classification and diagnosis.	Sheet Metal Manufacture
[Hennessy and Hinkle 92] Lockheed AI Center USA	CLAVIER	Establish local / global features for placing parts in oven.	Curing Aerospace Parts in Autoclave Oven
[Koton 89] MIT Cambridge, MA USA	CASEY	Modifies causal explanations generated from heart simulation to solve new problems.	Coronary Problems
[Pearce et al. 92] Galaxy Scientific and Georgia Institute for Technology USA	ARCHIE	Generates and critiques design cases. Uses primitive concepts and nearest neighbour to match goal constraints. Uses causal models to cluster cases.	Architectural Design of Office Buildings
[Rissland and Skalak 89a] U. of Massachusetts Amherst, MA USA	CABARET	Uses blackboard architecture to prioritize rule Vs. case base decisions.	Statutory Law Interpretation
[Simoudis 92] Lockheed AI Center USA	CASCADE	Uses top-down and bottom-up case retrieval to advise.	VMS Operating System Advisor
[Schmalhofer et al. 91] AI Research Center Kaiserslautern Germany	---	Uses manuals, previously solved cases and heuristics to learn process knowledge and generate expert system.	Production Process
[Sycara 90] [Sycara 91] Carnegie Mellon U. Pittsburgh, PA USA	PERSUADER	Combines CBR with decision theoretic models. Uses persuasive argumentation and threats.	Labor Negotiation

Another successful real-world application of CBR is CLAVIER [Hennessy and Hinkle 92]. CLAVIER is implemented by the Lockheed AI Center for curing aerospace composite materials in an autoclave oven. The Lockheed operator needs to match heating

rates of parts based on shape, mass, material, placement in the oven (front or back) and air currents around placed parts. The system has been in use since 1990 and gives the operator the right placement in 90% of the operations. It has built up a case base of 150 cases. It establishes both a global and a local context of features for where to place the parts in the oven. A major problem with CBR that the authors recognize, is the difficulty in validating a CBR system which requires knowledge of the problem to decide whether or not it can solve it.

A combined simulation and CBR approach is taken in CASEY. CASEY is a result from Koton's Ph.D. thesis. It uses a case-based reasoner and a simulation of the heart to diagnose coronary problems [Koton 89]. A case in the system is a causal explanation generated by the model. CASEY modifies causal explanations to solve new problem. If it fails, it defers to the model so that it may incorporate a new case. Phyllis Koton's work was the first to use model-based reasoning to generate cases for case-based reasoning. It is still often referenced as a key contribution in the field.

A novel real-world application of CBR for design is Archie. It represents, indexes and retrieves architectural design cases related to the construction of office buildings [Pearce et al. 92]. The system is used to generate and critique designs. It is based on the Remind shell from Cognitive Systems Inc. It has been applied to a real world problem of 20 architectural cases where each case has over 150 features. It uses primitive concepts and nearest neighbour to match goal constraints. It uses domain models (for example a causal model of building lighting) to cluster the cases.

Edwina Rissland is another key researcher in CBR. She has participated in the development of CABARET which is a CBR system that uses rule-based reasoning (RBR)

to interpret statutory laws based on previous legal interpretations [Rissland and Skalak 89a]. CABARET uses a blackboard architecture to prioritize decisions between CBR and RBR. The use of a blackboard architecture for CBR is novel. A related approach is described in [Howard et al. 89] for engineering design. The authors propose an integrated system that combines CBR and RBR for design in a blackboard architecture very similar to CABARET which would use opportunistic problem solving.

A system that formalizes production process knowledge and allows further elicitation of knowledge for mechanical parts manufacturing, is another novel CBR application [Schmalhofer et al. 91]. The system uses manuals, previously solved cases, and heuristics from experts to generate the expert system.

Katia Sycara is another key researcher in CBR, specifically in the area of labor negotiation and more recently in the use of CBR for design (see description in integrated diagnosis of CBR and model-based diagnosis for design in section 2.5 [Sycara and Navinchandra 89]). She has developed PERSUADER, a system designed to automatically negotiate labor disputes [Sycara 90]. The system combines case-based reasoning and decision theoretic models. Artificial Intelligence techniques are used to structure, represent, process and relax the human rationality assumption by using qualitative rather than quantitative knowledge. Cases are organized hierarchically based on important domain concepts. Generalized episodes are used to organize the concepts in memory as implemented by Kolodner in several of her CBR memory systems discussed previously in this section. PERSUADER is applied to labor negotiation in the airline and trucking industries. For each dispute PERSUADER:

- introduces the new goal,

- substitutes the new goal in the case base goals,
- abandons the goal if it is not achievable, and
- changes reservation prices of negotiating parties to re-introduce a goal.

PERSUADER combines: (1) case-based reasoning, (2) situation assessment using domain-independent problem restructuring strategies, (3) search of agent goal graphs, and (4) persuasive argumentation using threats and promises. Sycara concludes that case-based reasoning is ideal in negotiation since there is no strong domain structure in negotiation [Sycara 91]. **PERSUADER** is the earliest successful application of CBR in labor negotiation that combines CBR with decision theory practices.

Bidder's Associate is a novel application for a complex problem [Whitaker and Thorsden 90]. It has been implemented using the **PROSPER** CBR shell. **Bidder's Associate** is a case-based system that accesses a database of 1000 previously manufactured air-frame parts [Whitaker and Wiggins 89]. Cases are retrieved based on descriptors of previous products and subassemblies. Bids are adjusted based on economic factors (inflation), previous bid costs, and any already developed technology included in the proposed work. The system provides an audit trail (trace of reasoning) and is implemented using frames and rules. It highlights the problem of indexing for case utility and retrieval.

2.3.2 Summary

As detailed above, CBR attempts to “remember” past problem solutions in generating solutions for new problems. Research issues in case-based reasoners include case indexing, modification, repair and learning. CBR has been applied successfully to domains that range from labor negotiation to architectural design. Researchers are working on reasoning with cases locally (using case features) and globally (using hierarchically

organized cases). Case-based reasoners are being combined with other techniques in AI including, fault-based reasoners, model-based reasoners (and simulators), explanation-based learners and genetic algorithms, in an attempt to make CBR more flexible. Some researchers have combined CBR with rule-based reasoners in blackboard architectures. CBR has had limited application in diagnosis, since, as stated earlier, fault-based reasoning can be viewed as a form of organized case-based reasoning. Also, as Sycara has pointed out, CBR is ideal for poorly structured domains. FBR is used successfully in the diagnosis of inherently well engineered devices and processes (for example, the space shuttle main engine or a nuclear power plant). However, one could argue that diagnostic systems could reason more quickly if they had a case-based component, since CBR can be viewed as an approach to speeding up repetitive diagnoses. The CBR field will predictably continue to attract a large number of researchers who will continue to work on the open problem of truly representing human memory.

2.4 Machine Learning for Knowledge Acquisition⁴

Machine learning is a key approach in knowledge acquisition. Machine learning includes empirical and analytic learning. Empirical learning focuses on learning for classification, including learning rules from data for diagnosis. Analytic learning addresses learning for problem solving tasks. Such tasks include planning, design, natural language understanding, control and execution. There has been an explosion in the amount of work in machine learning in recent years. It is viewed as one of the key approaches of reducing the knowledge acquisition bottleneck [Boose 91; Gaines and Shaw 91].

4. Note machine learning for knowledge acquisition literature review summaries in [Abu-Hakima 94c].

Explanation-Based Learning, EBL, is a form of analytic learning that takes positive and negative examples, and uses background knowledge (domain theory) to generate and generalize an explanation (proof tree) for the example. This is a form of speed-up learning, which is used to derive generalized knowledge from specific knowledge. It is also useful in making a knowledge base more compact so that reasoning paths may be shortened. As such, EBL can be useful in generalizing diagnostic knowledge.

Mitchell and his colleagues have written a landmark paper about explanation-based generalization (EBG) [Mitchell et al. 86]. The approach they describe is simple, one is given:

- a goal,**
- the training examples (instances),**
- a complete domain theory (rules and facts), and**
- the operationality criterion (the form the concepts are to be generalized in).**

One can follow 3 steps to generalize the goal:

- 1. formulate a goal tree branching down to the leaves that cover positive instances of the training examples,**
- 2. backpropagate and substitute variables (if possible) for constants in the goal tree, and**
- 3. express the goal as a conjunct of goal tree leafs.**

Much research has followed Mitchell's especially in the area of relaxing the requirement for a complete domain theory for learning. According to EBG one cannot have an imperfect, incomplete or inconsistent domain theory. If one does not have a complete domain theory, one can assume that EBG would fail. One approach to the problem of generating a complete domain theory has been the use of inductive learning over the positive and negative instances so that a domain theory is learned.

Explanation-based learning (EBL) has been described [DeJong and Mooney 86] in response to EBG [Mitchell et al. 86]. DeJong and Mooney state that if one is given:

- a domain theory which consists of:
 - types of objects and their properties,
 - a set of inference rules for the properties of objects and their relations, and
 - a library of problem solving operators and general schemata,
- a goal state (or an incomplete world state), and
- an initial state.

Then if one does not have an observed operator state that can explain the goal one can generate a state sequence to explain the goal. The six step sequence is:

1. if no human operator can form an observed state, form a plan (generate a goal tree) by using the domain theory to include a list of operations that are generalized (operations with variables), and specialized (based on instances),
2. eliminate operations and states that do not causally support the goal, the remaining 'plan' forms the solution explanation,
3. identify the nominal instantiations of the known schema, and retract them,
4. remove the actions/states which support only the inferences to more abstract states,
5. if the explanation is from an observed solution, check if it can be made more efficient (i.e. look back to see where to specialize), and
6. generate the final generalized explanation, by applying substitutions from the generalized list of substitutions to the remaining explanation structure.

DeJong and Mooney (EBL) raise these problems with the Mitchell (EBG) approach:

- one needs to specialize as well as generalize concepts in EBL - EBG does not examine concept refinement (i.e. specialization),
- EBG paper does not describe how explanations are constructed - in EBL they are formed using theorem proving and observation,
- EBG generalizes too closely to training instances,
- the authors argue that the operability criterion should be related to the system performance and learning objectives - EBL eliminates its use since the goal of the system is to learn dynamically and operability,
- EBG does not allow the exploration of alternative explanations - EBL is based on structured/hierarchical relations, thus generating alternate explanations easily,

- goal regression in EBG is better replaced by a unification retractor that keep a list of specific/general substitutions in the goal tree so that specific/general explanations can be generated.

This landmark paper in EBL has been followed by quite a bit of research. EBL, unlike EBG, is presented as a more flexible dynamic approach to generating causal explanations.

One paper that follows on in EBL describes the use of abduction and analogy to compensate for incomplete domain theories [Duval 91]. EBL is first used to generate partial proofs, abduction is then used to generate hypotheses that can be tested as full EBL proofs. This approach is similar to several in the literature and is applied to simple horn clauses. Another approach to handling incomplete theories in explanation-based learning follows three steps [Genest et al. 90]. They are:

1. generate a partial explanation using traditional EBL and use abduction to extend the partial explanation,
2. use analogical reasoning to extend the explanation if possible, and
3. fallback to a case base and adapt a case if it is still not possible to generate a complete explanation.

This approach was applied to the problem of software specification. One problem in using this approach in other domains, could be the absence of cases to adapt.

Another author integrates explanation-based learning (EBL) and similarity-based learning (SBL) [Koppel 91]. EBL is viewed as the process by which an explanation E (proof) of the fact that some vector, v is a member of a relation r , and is used to generalize that fact to all vectors in r , whose membership can be similarly explained. EBL is appropriate when such an explanation can be formed. SBL is the process by which in the absence of an explanation, E , and in the presence of $v_1, \dots, v_n$, vector members of r , some set of features common to the vectors is presumed to explain their membership in r , and it

is tentatively coincided that all objects with these features are in r . SBL is appropriate when an EBL explanation cannot be formed. The author uses horn clause representations to integrate EBL and SBL and refers to it as ESDL. The paper gives a good theoretical description but does not convincingly apply it.

One of the key developers of classification algorithms in empirical learning is Quinlan. Classification algorithms take positive and negative instances and build classification trees which can be pruned to provide rules that represent the examples. In one of his papers Quinlan proposes 4 methods of pruning classification trees: (1) cost-complexity pruning, (2) reduced error pruning, (3) pessimistic pruning and (4) simplifying to production rules [Quinlan 87]. In another paper he describes the following divide and conquer algorithm:

- the training set is divided into subsets based on a test for a single attribute,
- the subsets are subsequently divided to learn on another attribute, and
- the process is repeated until the subsets can no longer be divided based on a single test [Quinlan 91].

He also describes the steps in the covering methods used in the inductive AQ classification algorithms:

- find the conjunction of conditions in some target class,
- append this conjunction as a disjunct of the rule,
- remove all objects that satisfy the conjunction, and
- if any objects of the target class remain repeat steps 1 to 3.

Quinlan has also implemented the FOIL algorithm [Quinlan 91]. The FOIL algorithm can learn multi-clause recursive definitions of relations by using the other relations as a kind of background knowledge. Pazzani and Brunk describe FOCL, their improvements to Quinlan's FOIL algorithm [Pazzani and Brunk 91]. FOCL learns despite a false

domain theory (wrong rules in expert system). FOCL uses explanation-based learning and induction to learn - the expert system acts as the domain theory for EBL. Their approach is novel but they only describe a toy application.

2.4.1 Noteworthy Papers

Table 4 presents some noteworthy approaches and applications in machine learning, which are discussed further in this section. Carnes and Fischer describe the novel use of supervised learning for the placement of sensors in a thermal model [Carnes and Fisher 92]. They use cluster analysis from COBWEB to group together observations for diagnosis. An explicit model of diagnosis (a model-based simulation) is used to place the sensors by generating the various states that need to be measured for design or diagnosis. Sensor placement is not a new issue tackled by machine learning.

Chen and Gelernter describe a somewhat novel use of a conceptual model of the domain to support EBL. This is used to avoid the need for a complete domain theory for acquiring knowledge [Chen and Gelernter 91]. They apply their approach to learning the molecular composition of organic compounds.

Harris presents a good review of real-time autonomous guided vehicle (AGV) fusion of sensor requirements [Harris 89]. These requirements include look up tables for classification, maps, feature models, alternative plans or strategies [Harris 89]. There are 3 levels in his hierarchy of multi-sensor data fusion: position estimation using bayesian probability; fusing conflicting data using distribution of data and confidence criteria rather than nearest neighbor or other statistical techniques; and assimilation of multi-valued input estimates using statistical (pairwise correlation, ranking, etc.), or symbolic (explana-

tion-based reasoning, truth maintenance and feature mapping) techniques. His method for assimilation is partial belief, where each knowledge source (e.g. radar source or mapped feature), can express itself at the most appropriate level, allowing decisions over one proposition or another to be suspended until further data is available.

Table 4. Machine Learning (ML) for Knowledge Acquisition (KA) noteworthy papers

Who	System	ML for KA novelty	Application
[Carnes and Fischer 92] Boeing Aerospace, and Vanderbilt U. USA	COBWEB	Use unsupervised learning to cluster diagnostic observations.	Sensor Placement in Thermal Model
[Chen and Gelemter 91] State U. Stony Brook, NY, USA	---	Use conceptual model of domain to support explanation-based learning.	Learning Molecular Composition of Compounds
[Harris 89] U. of Southampton Southampton, Hants UK	---	Suggest the use of explanation-based reasoning since a chain of justifications can be used to resolve conflicts in sensor data fusion.	Sensor Fusion for Autonomous Guided Vehicle
[Nakamura and Kobayashi 91] Japan Tobacco and Tokyo Institute of Technology Japan	---	Experts use cases and causal models when their experience fails them. Explanation-based learning is used on collected cases to augment diagnostic system.	Cigarette Making Machine
[Simoudis 90] Digital Equipment Corporation Marlboro, MA USA	---	System learns redesign knowledge for problem solving using analogical learning in a semi-automatic manner.	Troubleshooting Transmission Lines
[Turney and Halasz 92] National Research Council, Ottawa Canada	DITFED	Use nearest neighbour algorithm to cluster engine sensor data into features. Features can then be used for diagnosis.	Diagnosis of CF-5 Aircraft Engine

Harris uses Dempster's consensus rule to allow multiple information sources to be combined, in order to increase the belief in the propositions that are supported by the various bodies of evidence. The problem with the consensus rule, is that the complexity of the algorithm increases geometrically as the number of sensors increases, since the original

sensor declarations are used forever. The author suggests explanation based reasoning (EBR) as an alternative to resolving conflicts in sensor data assimilation. He believes that a chain of justifications would be built as part of the EBR algorithm. He argues that justifications can be viewed as maximizing the explanatory connectivity or coherence of beliefs.

Nakamura and Kobayashi argue that experts use cases and causal models when their experience fails them [Nakamura and Kobayashi 91]. They have used this approach to implement a system which uses EBL on collected cases to augment rules in an expert system which diagnoses problems in a cigarette making machine. Their approach and application are somewhat novel.

Simoudis describes a novel system that troubleshoots transmission lines [Simoudis 90]. The system learns re-design knowledge for problem solving using analogical learning in a semi-automated manner. It combines some ideas from LEAP (learns using explanation-based learning) and SALT (learns in a semi-automated manner by interviewing expert).

Jet engine sensor data has been collected and interpreted in comparing multivariate linear regression and instance-based learning [Turney and Halasz 92]. The results of the study indicate that instance-based learning can form the basis for a useful diagnostic tool for aircraft engine technicians. The application of instance-based learning for generalizing fault libraries for jet engine diagnosis is novel.

The AAAI-92 tutorial given by Pat Langley and Raymond Mooney concentrates on the use of analytical machine learning for problem solving tasks [Langley and Mooney

92]. The tutorial describes speedup learning as a method for improving the following in problem solving performance:

- accuracy (an objective in classification learners),
- reliability,
- quality, and
- efficiency of a solution (the latter three being objectives of task-oriented problem solvers).

The tutorial also describes the credit assignment problem (a term coined by Marvin Minsky in 1961) as a need to credit desirable decisions and blame (debit) undesirable decisions. The problem of learning search control knowledge is described for these techniques:

- reinforcement learning - Sutton in 1988 set up a reward function in his genetic algorithms work,
- supervised learning - induction from solution paths is used,
- explanation-based learning (EBL) - guaranteed to produce rules that lead to the goal rapidly since that is the objective in EBL, and
- genetic algorithms - the Pitt approach of learning that represents a control program as a set of production rules and uses crossover and mutation operations to evolve the program to an efficient one.

The problem of learning plan structures is described, highlighting the use of analytic construction of macro-operators, and the formation of iterative or disjunctive macro-operators in EBL. Plan structures are also generated by analogical re-use in case-based reasoning. The tutorial also describes the utility problem in machine learning: *how does one make sure that the macros learned in EBL speed up reasoning rather than slowing it down?* Also discussed is the operationality criterion: *how does one learn operator descriptions?* Quite a chunk of the tutorial concentrated on EBL and it provides a good reference for learners in task-oriented problem solving.

2.4.2 Summary

There has been an explosion of activity in machine learning in recent years, specifically in explanation-based learning, as exemplified by the annotated bibliography [Abu-Hakima 94c]. EBL has been applied to the problem of generating and refining rules for diagnosis amongst other tasks. EBL is speedup learning, it aims to learn knowledge that helps perform a task faster. Classification learning attempts to extract rules from positive and negative examples. I believe that neither approach addresses the problem of *knowledge-rich* learning, where structured knowledge is learned. Such knowledge could be learned from positive and negative examples, to produce hypothesis hierarchies such as those described in fault-based reasoning. In addition, *knowledge-rich* learning is learning from structured knowledge to produce new knowledge such as learning a device model from its fault hierarchy as the Diagnostic Remodeler does. Despite the flurry of activity in machine learning, I believe it remains in its infancy in producing results for well-structured diagnostic systems.

However, some hybrid systems (as described under the Integrated Diagnostic Techniques in section 2.5) are using machine learning to extract symptoms from sensor data, thus interpreting data to allow FBR diagnosis in an on-line mode. Such an approach simplifies device monitoring, since sensor data is interpreted, and then relayed to a failure-driven reasoner for a fast diagnosis. Other systems are combining machine learning with model-based reasoning. Some systems make use of a simulator to generate data and generate diagnoses based on the use of machine learning. These eliminate healthy components from being considered in a diagnosis and more quickly focus on components whose behaviours are unexpected.

2.5 Integrated Diagnostic Techniques⁵

Many researchers are developing hybrid systems, as reflected in over seventy papers included in the literature review of Integrated Diagnostic Techniques (IDT). Some systems are using model-based reasoning (MBR) to support a fault-based reasoning (FBR) system. MBR is used to detect novel faults, while FBR is used to quickly diagnose common faults. Some systems are using machine learning, to extract symptoms from sensor data using data interpretation, so that a fault-based reasoning system can be used for diagnosis in an on-line mode. Such an approach simplifies device monitoring, since sensor data is interpreted and then relayed to a failure-driven reasoner for a fast diagnosis. Other systems are combining sensor data interpretation with model-based reasoning, to eliminate healthy components from being considered in a diagnosis and more quickly zeroing in on components whose behaviour deviates from what is expected. Cases of previous failures are being indexed, and used to speed up diagnosis, while combining case-based with fault-based reasoning. Cases of previous failures, are also being used to speed up model-based diagnosis. Some systems are also refining diagnostic knowledge, using explanation-based learning. Many researchers are thus, combining fault, model and/or case-based reasoning, and are proposing integrated techniques to improve the diagnostic capability of their systems.

A single strategy for diagnosis does not seem to be suitable especially for complex problems. The reasons an integrated approach is superior are clear when one examines the weaknesses of each of the approaches.

5. Note that references for integrated diagnostic techniques are in literature review of [Abu-Hakima 94c].

- Fault-based reasoning systems do not learn new knowledge as they are used and thus are inadequate at detecting novel faults.
- Model-based reasoning systems are computationally very expensive and have an exponential increase in search as they attempt to detect a fault.
- Case-based reasoning systems are not very general and their utility becomes a function of indexing and searching the case base. Cases need adapting to detect novel faults.
- Machine learning for data interpretation requires the compilation of libraries of healthy and fault patterns representing the performance of a device. These libraries are not knowledge-rich structures or justifications for device behaviour.

Console and his colleagues describe the issues in model-based diagnosis, fault-based diagnosis, case-based reasoning and explanation-based learning [Console et al. 91]. They discuss the issues related to compiling solutions to speed up diagnosis in MBR. They also address the use of conditions on an entity for justifying its presence within a model. This helps them prune the search space in model-based diagnosis.

IDM (Integrated Diagnostic Model) is an approach which explicitly encodes heuristic (experiential) and fundamental (device oriented) knowledge [Fink and Lusth 87]. The authors developed functional primitives to describe the device model. They clearly define and classify diagnostic knowledge, and provide an alternate methodology for combining fault-based and model-based reasoning. They have not yet applied their approach to a complex real-world problem but it looks quite promising.

Walter Hamscher is a key researcher in diagnosis who developed XDE [Hamscher 90]. XDE is a system that uses fault-based diagnosis to bound the search in model-based diagnosis. XDE is an enhancement on De Kleer's GDE, an ATMS (Automatic Truth Maintenance System) for model-based diagnosis [de Kleer and Williams 87]. Hamscher's application is a board-scale digital circuit. He uses a physical hierarchy and a functional hierarchy of the system being diagnosed. He concludes by criticizing the control structure

of his implementation, which decomposes for hierarchic diagnosis, and refines for fault-based diagnosis. His paper is a landmark paper that combines model- and fault-based reasoning, to reduce the amount of search and computational complexity in diagnosis.

An interesting approach to diagnosis is a fault-based system for induction motor condition monitoring [Leith et al. 90]. The authors integrated a case-based reasoning component to collect cases in the form of scripts. They then applied machine learning, specifically Quinlan's ID3 classification algorithm, to induce rules from the case base. They conclude that inductive learning is poor technique, since the results of the knowledge added to the frame-based system were "flaky". It is possible that the performance of their diagnostic system would improve, if a simulation of the motor could be used to validate the fault-based system, thus generating more reliable cases which could induce more useful knowledge for the fault hierarchy. Combining fault-based and case-based reasoning with ID3 is novel. But the poor results the authors report demonstrate that a validated knowledge base is essential for their approach to succeed.

Original work for a model-based and a fault-based approach for the diagnosis of the same device has been reported [McDowell et al. 90]. The device is a black and white television set. The authors conclude that both techniques had access to the same knowledge of components and connections. They also conclude that fault-based reasoning provides an architecture which speeds up diagnosis and model-based reasoning does not scale up for complex devices. The diagnoses in fault-based reasoning took less than 60 seconds to generate while they took 10-15 minutes in model-based reasoning. Where the authors recommend model-based reasoning is in the area of novel faults. This is one of the few papers that compares FBR and MBR for diagnosing the same device. It provides convinc-

ing arguments for the selection of one technique over the other based on the problem being diagnosed as a novel (MBR) or an expected (FBR) one.

Sycara and Navinchandra combine case-based reasoning and qualitative modelling for design [Sycara and Navinchandra 89]. They apply their approach to simple problems such as designing a water faucet. It is not clear how well their methods would scale up for the design of a more complex device. However, combining CBR and qualitative modelling is novel.

Krishnamurthi and Philips describe the diagnostic model in relation to the cognitive model [Krishnamurthi and Philips 92]. The authors separate domain knowledge from the process of diagnosis based on the cognitive model. They provide a good description of the diagnostic cycle and the attempts by a technician to troubleshoot a device. They also present a good summary of processes for generating a generalized diagnostic shell on the basis of cognition. The shell combines a causal model with the fault model. Their approach is somewhat novel and provides further justification for hybrid techniques in diagnosis.

2.5.1 Noteworthy Papers

Table 5 provides a summary of papers describing noteworthy approaches and applications to integrated diagnostic techniques which are discussed further in this section.

Aamodt highlights the issues of combining case-based and model-based reasoning [Aamodt 90]. Aamodt reviews PROTOS, a system implemented by Bareiss as a Ph.D. thesis. PROTOS is used to diagnose hearing disorders and it combines case-based reasoning for concept learning, and fault-based reasoning for diagnosis. He also reviews CASEY, a

system implemented by Phyllis Koton as a result of her Ph.D. thesis. CASEY uses case-based and model-based reasoning for diagnosing heart disease. The model of the heart in CASEY is used to diagnose those problems that the case-based component cannot. A new case in CASEY is formed when the model makes a diagnosis that the case-base failed at (see [Koton 89] in the CBR literature review section 2.3). Aamodt then describes CREEK, an integrated architecture for diagnosis that combines case-based, fault-based and model-based reasoning with the intention of learning and adapting its knowledge structures to new diagnostic problems. The learning is achieved by using explanation-based, and case-based learning. The architecture is novel, and relevant, but the paper is not detailed enough in its description of the knowledge representation, to show how the knowledge structures will be adapted as new problems are solved. It is also surprising that the paper does not mention MOLTKE [Althoff et al. 90] which has already achieved many of the goals of CREEK.

Table 5. Integrated Diagnostic Techniques (IDT) noteworthy papers

Who	System	AI Techniques	Application
[Aamodt 90] U. of Trondheim Trondheim Norway	CREEK	CBR, MBR, FBR. Adapts knowledge structures for new diagnoses.	None
[Althoff et al. 90] U of Kaiserslautern Germany	MOLTKE	MBR, CBR, FBR and EBL. CBR used for knowl- edge acquisition and EBL used to improve rules.	Computerized Numerical Control (CNC) Machining Center
[Bizzari et al. 90] INFN Rome, Italy	---	MBR and FBR. FBR used to reduce search in MBR.	High Energy Physics Accelerator at CERN
[Berzillon and Fauquembergue 90] U of Paris VI and Electricite de France France	CEPT	Simulation and FBR. Simulator used to fully test FBR rules.	Diagnosis of Electric Power Lines

Table 5. Integrated Diagnostic Techniques (IDT) noteworthy papers

Who	System	AI Techniques	Application
[Carlson et al. 88] GE and Electric Power Res. Inst. USA	GEN-X, FEAT and TMGR	Feature extraction and analysis (FEAT) tool used to generate rules from data. GEN-X is the generic shell used to reason with rules. Test Man- ager (TMGR) used to view process control data graphically.	Detect Rotor Cracks in Power Generators
[Daniels and Foelders 91] Tilburg University Netherlands	---	Quantitative and qualitative models.	Diagnose Firms needing Financing from AMRO Bank
[Dewberry and Carnes 90] NASA Marshall and Boeing AI Center Schenectady, NY and Huntsville, Alabama USA	---	Describe an architec- ture for combined FBR (using Boeing's Acquinas shell for knowledge acquisi- tion) and MBR (using NASA's KATE tool) and sen- sor monitoring for diagnosis.	Environmental Con- trol and Support System on Space Station Freedom
[Dolce et al. 90] NASA Lewis Cleveland, Ohio USA	---	Describes an archi- tecture that com- bines APEX (FBR with data analysis testbed), MBR and Arbiter (multi-agent architecture) for scheduling resources using economic bidding model	Managing the Power Distribution on Space Station Freedom
[Dollman and Webster 89] NASA Marshall Huntsville, Alabama USA	---	Vehicle safety expert system that moni- tors/ trends teleme- try data. Uses DRS (device reasoning shell) based on FBR. Uses document retrieval assistant which manages 1000 multimedia docu- ments. Uses DART (design alternatives rationale tool) which provides a list of faults and relations during failure modes. Uses Hubble Safe-mode Trouble- shooting System which relies on MBR device models and rules.	Hubble Space Telescope
[Guo 91] NASA Lewis Cleveland, Ohio USA	---	3 layers of software for diagnosis: sensor data pre-processing, feature extraction using neural net and MBR, and FBR for troubleshooting.	Space Shuttle Main Engine

Table 5. Integrated Diagnostic Techniques (IDT) noteworthy papers

Who	System	AI Techniques	Application
[Hagar and Alcock 89] Pratt and Whitney Palm Beach, Florida USA	RECMS	Combine sensor data input from engine controller, vehicle monitoring system and ground mission system. Use feature extraction, FBR and MBR from simulation.	Monitoring Engines on F-15 and F-16 simulator.
[Hoffman et al. 91] U of Alabama and NASA Marshall Huntsville, Alabama USA	EDIS	Use MBR (qualitative and quantitative reasoning), FBR and data analysis to interpret engine data. Implemented using a blackboard architecture to coordinate tasks.	Space Shuttle Main Engine
[Kindler 91] U of Ulm Ulm, Germany	---	Implemented a qualitative (MBR) model with a FBR component to generate explanations.	Diagnosis of Nerve Lesions in Human Arm
[Marsh 88] MITRE Corp. Houston, Texas USA	ISA	Implemented an end-to-end testbed using sensor data, MBR and FBR for the diagnosis of power buses, cooling loops, data relay satellite system and down-link/uplink communication bands.	Space Station Freedom Prototype Testbed
[Meisner et al. 90] Honeywell Sys. Res. Minneapolis, Minn. USA	FCMDS	Implemented an MBR system (using numerical simulations) and evolved it to include FBR due to the computational efficiency of using diagnostic hierarchies.	F-16A Aircraft Control System
[Millett and Maby 88] McDonnell Douglas Helicopter Mesa, Arizona USA	---	Traditional FBR combined with statistical information derived from fault histories.	Apache Helicopter
[Modesitt 90] Western Kentucky U. Bowling Green, Ken. USA	Scotty	Data interpretation by rule induction and FBR.	Space Shuttle Main Engine Firing
[O'Connell and Ott 89] Texas Instruments Austin, Texas USA	GTA	Uses CBR to update FBR knowledge base.	Diagnosis of Manufacturing Assemblies
[Papenhausen and Hadley 86] Boeing Military Air. Seattle, Washington USA	CEPS	Proposes the use of avionics design knowledge, FBR and statistical data analysis.	B1B Aircraft Diagnosis Toolkit

Table 5. Integrated Diagnostic Techniques (IDT) noteworthy papers

Who	System	AI Techniques	Application
Pazzani 90] U of California Irvine, California USA	ACES	Combines FBR with EBL applied to simulation (MBR) to expand rules to include invalidation conditions.	DSCS-III Satellite Attitude Control System
[Pearce 88] Turing Institute Glasgow, Scotland	---	Use simulation (MBR) to generate examples from which rules are induced by AQ algorithm for FBR. Works better than hand-generated rule system.	Satellite Electrical Power System
[St. Clair and Wetterstroem 88] McDonnell Douglas and NASA Johnson St. Louis, Missouri and Houston, Texas USA	ESPAD	Combine sensor data, FBR and historical faults for power subsystem analysis and diagnosis.	Water to H2 and O2 conversion on Space Shuttle and Space Station Freedom
[Sonoda et al. 89] Toshiba Corp. Yokohama City, Japan	---	Combine sensor data, FBR, fuzzy logic to interpret transients in signal data from plant.	Boiling Water Reactor

The MOLTKE (MOdels, Learning and Temporal Knowledge in Expert systems) testbed for diagnosis is under development at the University of Kaiserslautern, Germany [Althoff et al. 90]. The system is designed to acquire device knowledge for diagnosis. It has a model-based reasoning mechanism for acquiring device models based on their components. A *component* of the model includes, a name, ports to other components (with optional test costs), possible internal states (with optional test costs), behaviour of the component (either in state tables or rules that represent the constraints the component sets up between its ports and states), subparts and their interconnections (if the component is non-atomic), typical malfunctions with name and effects (model typical behaviour when the component fails) and a priori probability of failure. MOLTKE uses case-based reasoning to acquire and refine knowledge that is generalized to a fault-based hierarchy. It also uses explanation-based learning to refine the rules in the fault-based hierarchy to get the

minimum paths for a solution. MOLTKE has been applied to a Computerized Numerical Control (CNC) machining center. It is also under investigation for the problem of driving mining machines. MOLTKE stands out as one of the more advanced systems that integrate several techniques for diagnosis.

An interesting IDT application is the use of fault-based diagnostic hierarchies to reduce the search in model-based reasoning [Bizzari et al. 90]. The authors have applied their approach to diagnose the high energy physics accelerator at CERN. Their work is somewhat novel but they do not provide many details. They present some good ideas on structuring the knowledge in model-based and fault-based diagnosis.

A novel system that uses a simulation, to validate FBR knowledge for diagnosis, is an expert system that diagnoses electric power lines [Brezillon and Fauquembergue 90]. A simulation of the system to be diagnosed was used to thoroughly test the diagnostic knowledge in the fault-based system before it was used. The validation of the diagnostic knowledge by using a simulator to fully exercise the knowledge base is a good idea. However, many complex devices cannot be accurately simulated. As a result this approach is only ideal for those devices and processes that can be accurately simulated.

Another IDT system detects rotor cracks in power generators using a combination of expert system and signature analysis software [Carlson et al. 88]. GEN-X (General Electric's GENeric eXpert system) and FEAT (General Electric's Feature Extraction and Analysis Tool) are used for this system. FEAT is used to convert the sensor data to rules which are then input to GEN-X to generate an expert system. The system has been applied to the interpretation of power plant vibration data. It operates automatically (by sending alarms on detecting a problem) or interactively with an operator.

Another interesting IDT application is one which diagnoses problems with firms requiring financial assistance from Banks [Daniels and Feelders 91]. The system uses quantitative and qualitative relations to explain symptoms and provides a recommendation on financing. The system is being tested at the AMRO Bank of the Netherlands.

Dewberry and Carnes describe an IDT systems approach to the problem of Environmental Control and Life Support System (ECLSS) of NASA's Space Station Freedom [Dewberry and Carnes 90]. The authors propose the integration of fault-based reasoning, model-based reasoning and sensor monitoring for the diagnosis of ECLSS. They propose using Aquinas from Boeing as a knowledge acquisition tool for the fault reasoner. They propose using G2 from Gensym for reasoning with simulations and KATE, NASA's tool for model-based reasoning.

Another IDT systems approach is used for managing NASA's space station power distribution [Dolce et al. 90]. These four main projects are underway:

1. security monitor and analysis software, which monitors the power load by taking critical measurements of currents and voltages, and propagating the values through the network topology and comparing it to known thresholds,
2. APEX (described under [Quinn and Walters 91]), is proposed for diagnosing faults with the power distribution system, using fault and model-based techniques,
3. TROUBLE III, is an implementation of APEX using the Automated Reasoning Tool (ART), and
4. Arbiter, a multi-agent architecture for coordinating and scheduling resources using the economic model for bidding.

It is expected that these systems will be integrated into a single power distribution management tool for use on the space station. This is a novel integration of multiple paradigms for diagnosis.

Other integrated diagnosis work at NASA includes several prototypes implemented to support the Hubble Space Telescope [Dollman and Webster 91]. The paper describing this work is unfortunately shallow and quite brief. The high speed peripheral processor is implemented to read data and deliver it to several knowledge-based systems in the Hubble verification testbed. The Hubble safemode troubleshooting system uses device models and rules to determine safemodes from telemetry data. This system is implemented using the Device Reasoning Shell (DRS), which is a tool that allows one to model devices and integrate the model with a rule-based system. The vehicle safety expert system, implemented by Lockheed, monitors and trends the space telescope telemetry data and provides advice when problems arise. The thermal control subsystem is an expert system that provides a hierarchy of interactive diagrams with sensor data and is useful for fault detection. The document retrieval assistant (DRA) allows NASA to manage more than 1000 documents written for the telescope. The DRA is a knowledge-based multimedia system, that uses hypermedia to link text, graphics and video clips from designers of the telescope. The design alternatives rationale tool (DART), is a knowledge-based tool that provides a list of possible faults and their relations during various possible failure modes. DART documents design alternatives and possible re-configurations in case of faults with the Hubble. If all these tools were integrated into one system that monitors the Hubble, the work would be quite novel in IDT.

In another novel NASA application, Guo describes a 3-layer approach to the diagnosis of the Space Shuttle Main Engine (SSME) [Guo 91]. The first layer is sensor data preprocessing, the second is feature extraction (using neural networks or model-based reasoning) to trigger component faults, and the third layer is classical troubleshooting. Both

component fault detection and troubleshooting are implemented in the G2 real-time shell from Gensym Corporation. The high pressure oxidizer turbopump seal system of the SSME, was used to try the 3-layer diagnostic approach. G2 was used because of its connectivity to the simulator, and its ability to accept data directly from the simulation. G2 was found to have many problems: its one second sampling rate proved too slow for SSME operation, modifying a class in G2 modifies all its instances (this is standard in the object-oriented programming paradigm) and their respective knowledge base components, and finally G2 has poor graphics support. Overall however, the author seems quite satisfied with the 3-layer approach to diagnosis, he just reports that G2 is not necessarily the tool for this IDT application.

An IDT approach is proposed for Pratt and Whitney's rocket engine condition monitoring (RECMS) [Hagar and Alcock 89]. The RECMS architecture is an integrated diagnostic systems approach which combines input from the engine controller, the vehicle monitoring system and ground mission systems for the F-15 and F-16 aircraft. They propose the identification and development of new sensors, monitoring and maintenance strategies. This includes the development of new algorithms (simulation-based) to detect fault signatures. These would perform signal conditioning (removal of noise from the data), feature extraction and pattern classification for expert system (fault-based reasoning) use. The ground support system would include a centralized database, an expert system and any additional hardware. Combining ground-based and in-flight diagnostic facilities is somewhat novel but the paper only presents an architecture not an actual system.

An IDT system that combines model-based reasoning, fault-based reasoning and data analysis (pattern recognition based on fault libraries) is applied for the space shuttle main engine (SSME) [Hofmann et al. 91]. The authors used the blackboard architecture to coordinate the various tasks in the diagnosis. They are re-implementing the system as a second prototype in NEXPERT with TOOLBOOK as the user interface tool. The work is continuing at the University of Alabama in Huntsville in collaboration with NASA. This is one of the first systems that combines qualitative and quantitative models with fault-based reasoning.

Kindler describes an interesting medical application [Kindler 91]. The author describes a model-based reasoning system with a fault-based reasoning component. It uses a qualitative model of the nerves in the arm and diagnoses the lesions that may occur. He argues for the use of fault-based reasoning for the generation of explanations. This is a new application and the successful integration of the techniques further demonstrates the need of model-based and fault-based reasoning for achieving justification in diagnosis.

Marsh describes interesting work at the MITRE Corporation's work for space station freedom (SSF) diagnostics for NASA's end-to-end testbed (the testbed includes power distribution management, payloads, thermal control, communications and tracking as well as the European Columbus work) [Marsh 88]. Integrated Status Assessment (ISA) monitors station status and configuration, manages configuration cautions/warnings and performs global fault management. The SSF operation is viewed as a hierarchy of controllers with decreasing authority as one moves down the chain of command. ISA has been prototyped for the power buses, cooling loops, tracking and Data Relay Satellite System and the Interface to the DMS (data management system) for the downlink and uplink com-

communications band of SSF. The knowledge in ISA is represented using objects for components (using entity-relation modelling), rules for how to perform the diagnosis and qualitative models to describe the states of the components. The implementation is on a Symbolics 3600 in ZetaLISP. What is interesting in the approach is that deep reasoning strategies are used to narrow down the fault (including knowledge on handling ambiguous sensor data from failed sensors) and shallow reasoning strategies (rule-based) are used to troubleshoot. ISA generates explanations and has a tiled user interface (much like JETA's division of the screen as described in [Abu-Hakima et al. 93b]).

An integrated diagnostic tool for the F-16A control system, is used by flightline technicians in troubleshooting the F-16A [Meisner et al. 90]. The tool was initially based on model-based techniques (numerical simulations), and evolved to a mixture with fault-based techniques due to the complexity of the problem. The system took 5 years to develop and is currently being field tested by Honeywell and the US Air Force. The lesson that the authors impart is, *the importance of using fault-based hierarchies in conjunction with models to accomplish diagnosis in a finite time*. Although the approach is not novel the application is a complex real-world problem that demonstrates the importance of integrating model-based and fault-based reasoning.

Another IDT system has been developed by McDonnell Douglas Helicopter, to diagnose faults aboard the Apache helicopter [Millett and Maby 88]. The system is rule-based and it uses the history of faults to prioritize recommended repair actions. The system uses extensive graphics to allow the user to interact with schematics rather than only text. This work is a good example of the use of previous cases for refining the knowledge in FBR.

Scotty is an interesting expert system under development at Rocketdyne for the Space Shuttle Main Engine (SSME) [Modesitt 90]. It is aimed at automating the analysis carried out in firing the shuttle engines. The FBR system is based on the experience of propulsion engineers. It consists of 125 manually derived rules, and 1400 automatically induced rules from example runs. The rules are embedded in a distributed software environment. The rules are poorly structured and will be moved to a relational database management system (RDBMS) to improve the knowledge representation. The author is also planning to enhance Scotty's graphics and generalize the approach to other Rocketdyne engines. Rule induction from data as a technique is not new but this integration with FBR knowledge and its application is somewhat novel.

The Generic Technician's Assistant (GTA) is another IDT tool [O'Connell and Ott 89]. It is a shell designed to assist in the diagnosis and repair of manufacturing assemblies at Texas Instruments. It uses case-based reasoning, or previously solved cases, to update its knowledge base. The cases clearly identify symptoms and faults which can be integrated with the fault-based diagnostic network of symptoms and their relations to faults.

Papenhausen and Hadley describe a system that integrates avionics design knowledge, maintenance expertise, statistical analysis of sensor data to improve fault detection and isolation [Papenhausen and Hadley 86]. CEPS is the B1B aircraft Central Integrated Test System Expert Parameter System. It is to be implemented by Boeing for the US Air Force. The architectural design of CEPS combines a database of cases supported by sensor data with an expert system that combines design engineering knowledge with the expertise of aircraft technicians to provide B1B technicians with a diagnostic toolkit. The

described system is in design stages and has not yet been prototyped. The work is novel in its proposed integration of diagnostic techniques in a toolkit.

ACES (Attitude Control Expert System) is an interesting system that diagnoses anomalies in the attitude control system of the DSCS-III satellite [Pazzani 90]. ACES is fault-based (rules represented as Prolog predicates). A fault is confirmed or denied, by comparing the observed behaviour, to that predicted with a simulator. In the case where the simulation denies the fault, the heuristic that proposed the fault is expanded to include the tests that the simulator performed to rule out the fault. ACES uses explanation-based learning to identify the conditions under which the heuristic will propose a fault that is denied. The author concludes that failure-driven learning finds sufficient conditions for ruling out a fault and success-driven learning finds sufficient conditions for establishing a fault (but not necessarily ruling others out). Pazzani's work is novel and very relevant to the refinement of fault-based knowledge using model-based reasoning and explanation-based learning.

Pearce describes two parallel and interesting approaches to acquiring knowledge for an expert system [Pearce 88]. The first is generated by a knowledge engineer, who hand codes rules from an expert. The second uses examples of failure from a simulation and applies AQ (one of Quinlan's classification algorithms, see section 2.4), an induction algorithm, to the data. The author concludes that the second approach gives much better results. The only drawback to the use of induction of rules from simulated data, is that one needs a good simulation of a device to generate the examples from which the expert system is induced. It is not always possible to find or generate such good simulations.

An IDT system has been implemented jointly by McDonnell Douglas and NASA's Johnson Space Center, for monitoring and diagnosing a subsystem that converts water to H_2 and O_2 , for the space shuttle and space station [St. Clair et al. 88]. The diagnostic program assumes that the subsystem has built-in redundancies, so that failed components may be swapped out for functioning ones. Sensor data triggers the fault-based reasoner which has classes of rules associated with each unit that can fail. The reasoner dynamically orders the rules, based on historical information of which unit is most likely to fail more frequently. The system has been tested with a simulation of the water conversion subsystem. Reprioritizing reasoning based on collected statistics and integrating sensor data with the fault-based reasoner for this type of application is somewhat novel.

A boiling water reactor IDT system has been developed [Sonoda et al. 89]. The authors use a simulator and a classical MYCIN-like [Clancey 86] expert system, with access to simulation data for diagnosis. They use fuzzy logic to interpret the abnormal transients in signal data from the plant. The described system is a prototype and not in actual use at the plant. Their prototype combines numerical and heuristic diagnostic techniques with fuzzy logic and presents a promising approach.

2.5.2 Summary

As illustrated above many researchers are developing hybrid systems. Model-based reasoning is used with fault-based reasoning, to integrate in a single system the experiential knowledge of diagnosing a device with its expected behaviour. This reduces the computational complexity of finding a diagnosis using MBR. MBR is also integrated with data interpretation to reduce the computational search. Explanation-based learning is

used to refine the reasoning chains in rule-based FBR. Rule induction is also used to generate rule-based FBR systems. FBR is integrated with CBR as a means of acquiring new knowledge and reducing the search for a diagnosis. Similarly MBR is integrated with CBR to accelerate diagnosis.

Some of the most active research in integrated techniques for diagnosis is occurring at NASA. This is not surprising since the problems they face in space systems are quite complex and require hybrid solutions. Complex space problems include the monitoring and diagnosis, of the Space Shuttle Main Engine (SSME), and the Hubble Space Telescope. Other complex tasks include, the power distribution and environmental control on the Space Station Freedom.

However, one area where no work has been done is in the re-use of FBR knowledge which is well structured and rich in its content. FBR knowledge has not been used to automatically acquire models for later input in MBR systems.

2.6 Literature Review Summary

This chapter reviews the state of the art in knowledge-based techniques applicable to diagnosis, to confirm that the Diagnostic Remodeler is a new algorithm. The review is intended to guide a reader as to the approaches, applications and issues in using AI techniques to research and develop diagnostic systems.

This is achieved with a thorough review of those techniques relevant to diagnosis. The review is supported by an annotated bibliography encompassing 262 papers aimed as a guide to significant and related work. The review is subdivided into eight sets with the

first five, the most relevant to diagnosis, analyzed and summarized in sections 2.1 to 2.5.

They are:

1. fault-based diagnostic techniques,
2. model-based diagnostic techniques,
3. case-based reasoning techniques,
4. machine learning for knowledge acquisition, and
5. integrated diagnostic techniques.

As described, Fault-based reasoning (FBR) is used in many diagnostic systems.

The knowledge in such systems is largely based on maintenance manuals and interactions with a human who is a domain expert. Two problems with FBR that are often cited are the difficulty in maintaining, or continuously validating, the knowledge base, and in dealing with novel faults. In response to these two problems researchers are working to improve the techniques for uncertainty handling, and for dealing with novel faults through approaches such as model-based and case-based reasoning. Despite its shortcomings FBR has proven to be an effective means of developing diagnostic tools. There have been many successful applications based on FBR as exemplified by the wide range of applications in the review.

Model-based reasoning (MBR) for diagnosis, as described, concentrates on reasoning about the expected versus the observed behaviour of a device. There are many active areas in MBR research as it is less mature an approach in diagnosis in relation to fault-based reasoning. One of the issues that remains in dispute is the definition of device structure and behaviour, versus device functionality. Uncertainty handling and bounding complexity in MBR are two other research issues. Representing and explaining causality

is another open question. Combining MBR with fault-based reasoning to bound computational complexity is an active research area. The use of classes of models to address various application tasks such as diagnosis, scheduling and control is a relatively new and promising area in MBR.

Case-based reasoning (CBR), as described, is the ability to use previously solved cases to guide problem solving. Key issues in achieving CBR include, indexing cases and representing features, as well as adapting cases to solve novel problems. The ability to repair a case that has failed in providing a solution, and learning in CBR are two additional issues. Research and new implementations of CBR systems, including CBR tools or shells, have enjoyed an explosion of interest, as exemplified by the number of publications and products for CBR since 1990. CBR attempts to "remember" past problem solutions in generating solutions for new problems. CBR has been applied successfully to domains that range from negotiation to design. Strategies being investigated for CBR include local (using case features) and global (using hierarchically organized cases) reasoning. Case-based reasoning is being integrated with other techniques in AI including, fault-based, model-based (including simulation), explanation-based as well as genetic reasoning in an attempt to make it more tractable and flexible. CBR has had limited application in diagnosis. One reason may be that fault-based reasoning can be viewed as a form of organized case-based reasoning. Another reason is that CBR is better suited for ill-structured domains and FBR is usually used in reasoning about well-engineered systems. However, one may argue that diagnostic systems that reason about repetitive problems, may do so more quickly, if they had a case-based component. The field will predictably continue to attract a large number of researchers, who will continue to work on the open problem of

truly representing human memory, which after all, is the problem that first attracted researchers to CBR.

Machine learning is another key approach in knowledge acquisition useful for diagnosis that is reviewed. There has been tremendous activity in machine learning in recent years. It is viewed as one of the key approaches to reducing the knowledge acquisition bottleneck. Machine learning includes empirical and analytic learning. Empirical learning focuses on learning for classification. Classification learning, often referred to as induction, attempts to extract rules from positive and negative examples. It has been applied to a variety of problems from planning to design. Analytic learning addresses learning for problem solving tasks such as planning, design, natural language understanding or device control. Explanation-based learning is a form of analytic learning that has been applied to the problem of generating and refining rules for diagnostic tasks. EBL is speedup learning implying that it is intended to learn knowledge that could help perform a task faster. As stated earlier, I believe that machine learning has not successfully addressed the problem of *knowledge-rich* learning, where structured knowledge is learned or re-used. Structured knowledge includes hypothesis hierarchies such as those needed for diagnosis in FBR. In addition, there is a need to develop algorithms that re-use fault knowledge for the generation of behavioural and functional models for diagnosis.

Many researchers as reflected in the literature review of Integrated Diagnostic Techniques, are developing hybrid systems. Some are using model-based reasoning to support a fault-based reasoning system. MBR is used to detect novel faults while FBR is used to quickly diagnose common faults. Other systems are using machine learning to extract symptoms from sensor data, thus interpreting data to allow FBR diagnosis in an

on-line mode. Such an approach simplifies device monitoring, since sensor data is interpreted, and then relayed to a failure-driven reasoner for a fast diagnosis. Other systems are combining sensor data interpretation with model-based reasoning. These eliminate healthy components from being considered in a diagnosis and more quickly focus on components whose behaviours are unexpected. Cases of previous failures are indexed and used to speed up diagnosis in combining case- with fault-based reasoning. Cases of previous failures are also being used to speed up model-based diagnosis. Some systems are refining diagnostic knowledge using explanation-based learning. Many researchers are thus combining fault, model and/or case-based reasoning, and are proposing integrated techniques to improve the diagnostic capability of their systems.

As discussed in the IDT section, a single strategy for diagnosis is not suitable, especially for complex problems. Reasons why an integrated approach is superior are clear when one examines the weaknesses of each of the approaches:

- Fault-based reasoning systems are inadequate at detecting novel faults, and do not learn new knowledge as they are used.
- Model-based reasoning systems have exponential computational complexity as the number of components and resultant behaviours increase in a model.
- Case-based reasoning system utility is a function of indexing and searching the case base and as a result, are not very general.
- Machine learning for diagnostic data interpretation requires the compilation of large, expensive to compile, libraries of healthy and fault patterns. Machine learning also does not generate well-structured knowledge or that can be used to justify device behaviour.

As a result of the shortcomings with the individual techniques, many researchers are developing hybrid systems. To advance the re-use of knowledge in hybrid systems, the Diagnostic Remodeler algorithm was developed as the subject of this thesis and it is described in the remaining chapters.

3.0 Model Acquisition: Diagnostic Remodeler Design

3.1 Hypothesis of Diagnostic Remodeler (DR) Algorithm

Humans use failure-driven reasoning for successful device diagnosis and repair. As humans reason about diagnosis and repair, they build causal mental models of how a device functions and fails [Krishnamurthi and Philips 92]. The hypothesis for the Diagnostic Remodeler algorithm is that explicit, and some implicit knowledge of failure and repair embodied in most structured diagnostic knowledge bases, can be re-used to derive black box device component models. The DR algorithm extracts these device component models from fault knowledge and they represent device structure, component behaviour, and function (see chapter 1.0 for exact definitions).

3.1.1 Utility of DR Algorithm

A great deal of effort is expended hand-coding complex knowledge bases for diagnostic fault-based reasoning [Abu-Hakima 93a]. The artifacts these diagnostic systems are developed for are often expensive machines which have been designed, continuously modified, and continually deteriorate. As a result, no existing accurate design schematic of their behaviour, or resultant explanation for their function remains. Such an application of the Jet Engine Troubleshooting Assistant (JETA), was to a J85-CAN-15 jet engine [Halasz et al. 92]. JETA is a typical structured fault-based reasoning (FBR) system similar to MDX [Chandrasekaran et al. 79]. The J85-CAN-15 engine was designed in the 1950's and has easily had at least one modification a year since its launch. As a result of modifications and the deterioration from use (flying in the arctic and flying in desert heat), the jet engine

is a very different device than was originally designed. Moreover, it sometimes displays behaviour inexplicable by current troubleshooting procedures. No existing design schematics usable by a technician can completely capture the engine's behaviour or completely predict its function. Historically, it is also a very difficult device to diagnose (its design preceded some of the modular engine designs that are currently used to facilitate diagnosis and maintenance). For these reasons, a tool such as JETA was developed. As is typical with FBR systems, JETA does not diagnose novel faults. Learning the device component model, its behaviour, and functionality by re-using the FBR knowledge, provides the technician with a model that can be used to check the FBR knowledge. Such a black box component model may potentially be used as input to a model-based diagnostic tool. Such a tool may detect novel faults since the behaviour, rather than the known failures, of the device would be the basis of model knowledge representation and reasoning.

3.2 Novelty of DR Algorithm

The DR algorithm is a hybrid algorithm, that combines fault-based knowledge with device dependent and device independent background knowledge, to generate a black box component behavioural model. This component model can then be re-used to find gaps and inconsistencies in the fault knowledge. The DR algorithm is unique in that it makes use of fault-based knowledge to generate model-based knowledge. In the extensive literature review of chapter 2.0 and the additional annotated bibliography of over 250 references in [Abu-Hakima 94c], there is not any similar published work.

The DR algorithm is novel in that it re-uses fault-based knowledge to acquire a model. The model can then be used as input to an off-the shelf model-based diagnosis

toolkit. For example, the Intelligent Diagnostic Expert Assistant (IDEA) is a tool distributed by Microprocessor Systems Inc. IDEA is an extension of Randall Davis' work at MIT [Davis 84] and follows a 'black box component model approach (hence DR output could be easily input to IDEA) [Orchard and Shevel 94]. Another off-the-shelf commercial system is Service Power from ServiceSoft Corporation (formerly Rosh Intelligent Systems). Service Power is a set of diagnostic tools with hybrid capabilities (fault-based and model-based) [Orchard and Shevel 94].

There is some work in the published literature on the re-use of knowledge in diagnosis. Specifically, the notion of context for selecting a model for diagnosis is described in section 2.2 [Abu-Hanna et al. 92 b]. However, this work does not address the automatic acquisition of model knowledge from fault knowledge as DR does.

3.3 DR Algorithm

3.3.1 Background

Here, we follow the de Kleer and Williams [de Kleer and Williams 87] and Sticklen [Sticklen et al. 88] approaches which model devices as a causal network of black box components. These components are nodes in the network that represents a full device model. The components are linked through their input/output behaviours. A device can be diagnosed by assuming a faulty component, and enumerating the behavioural states that the fault propagates in the remainder of the device [Davis 84; Hamscher and Struss 90; Struss 89]. This is compared to the behaviour that a technician observes when attempting to isolate a problem. Model-based reasoning for diagnosis can detect novel faults, since

the behaviour of the device is the basis of its knowledge representation and reasoning. Fault-based reasoning uses the faults in the functioning of a device, rather than its actual behaviour, hence FBR cannot detect novel faults. However, model-based reasoning can lead to a combinatorial explosion in producing a diagnosis for complex systems, and it does not easily allow causal explanations [Struss and Dressler 89].

The DR algorithm has been implemented for the re-use of fault knowledge. DR automates the generation of black box component models with explicit input-to-output behaviours [Abu-Hakima 94a; 94b; 93a]. Connections between component models are extracted from fault knowledge to provide full subsystem or device model connectivity. Device dependent and independent background knowledge is used to generate component model descriptions that include function, inputs, outputs, regulation, and explicit behavioural relations.

By extracting a behavioural model, both fault and model-based diagnosis can eventually be pursued in a single system, gaining from the advantages of the two approaches while minimizing the disadvantages. In addition, the generated black box component models can be used to uncover gaps, or inconsistencies, in the fault knowledge. The hybrid approach is supported by many systems described under integrated diagnosis approaches in section 2.5. The DR algorithm has been tested for the modelling of complex hydromechanical devices, specifically with jet engine fuel controls. The DR algorithm has also been tested with a simpler application, modelling a coffee maker device.

3.3.2 Method

DR is an algorithm that takes as input fault knowledge of a device and some background knowledge about device components to attempt to learn full component behaviour and function. DR initially extracts from the fault knowledge base all references to device components which could be related to a subsystem of interest. Given these components, the algorithm backtracks through a diagnostic hierarchy of nodes (such a hierarchy is a classical approach to representing knowledge for FBR as described in section 2.2) to generate hypotheses for component connectivity. To establish component connectivity, DR examines symptomatic, or parametric knowledge that activates the diagnostic nodes. Upon completion, DR outputs a black box component model in the form of a meaningful symbolic description (see resultant output in chapter 5.0).

A generalization one can make about a troubleshooting hierarchy, shown in Figure 3.1, is that as one moves upward, the knowledge describes generic faults related to phases of device operation. These can be refined to faults in operational modes that may be related to subsystems on a device. As one moves down in the hierarchy and is closer to the nodes that represent device components, the knowledge is more symptomatic in nature. Symptomatic knowledge is knowledge of specific component parameter failures which can be re-used to generate hypotheses about parameters, or behaviours, attached to specific components.

Figure 3.1. Well-structured diagnostic hierarchy with layers of knowledge

Approach

The top level design of the DR algorithm is shown in Figure 3.2. Two phases clearly divide the operation of the algorithm. In the first phase, an existing fault knowledge base (for example, one that diagnoses a complex hydromechanical system) is used as input to DR-1. Two types of background knowledge, device dependent and device independent knowledge are used as inputs to the second phase of the algorithm, DR-2. Device independent background knowledge is in the form of a component library and is general in nature. For example, it includes knowledge that a pump delivers some liquid from a source to a sink and needs a control signal (such as pressure for example), to increase or decrease the flow of liquid. It also includes some knowledge about feedback control in moderating the flow of a liquid to a source, based on the level of the liquid at the sink. The device dependent knowledge includes a list of nested triplets. Each triplet identifies a control mode, respective input(s), and respective output(s). The inputs and outputs may include a sign (plus for increasing value or minus for decreasing value), that indicates the change in the value of the variable for a particular mode.

Figure 3.2. The Phases, DR-1 and DR-2, of the Diagnostic Remodeler Algorithm

3.3.3 DR Objective Statement¹

The objective of the DR algorithm is to discover and refine a component behavioural model with explicit function, by re-using device fault knowledge. In the most general sense, the algorithm must identify the components of the device, generate links between those components, and generate hypotheses in the form of component descriptions, for the behaviour and function of the components.

1. Definitions for terms used in describing DR throughout this document are in the introduction chapter 1.0.

3.3.4 DR Algorithm Steps

To achieve its objectives the DR algorithm must perform five steps²:

- 1. Identify the terminal nodes in the diagnostic hierarchy**
-these represent component nodes that have no child or sibling refinements
- 2. Identify the component nodes in the diagnostic hierarchy related to the subsystem to be modelled (if required)**
(a model can be constrained to the components of a subsystem rather than a full device)
-identify the subsystem of a component through a component node slot (if possible)
-or perform a pattern match with the known subsystem name or its derivatives, possibly acronyms (do this only if subsystem name embedded within component name)
- 3. Identify the parents and siblings of the nodes**
-backtrack from terminal node to parent node and tag
-tag parents that are shared between terminal nodes
-tag siblings that share a parent
- 4. extract relations (behaviours) between the sibling nodes**
(movement from terminal nodes to root node implies symptomatic or parametric information)
-cluster nodes related by parental nodes
- 5. match device model against background knowledge and output gaps for verification to the user**
-map out the identified components of the subsystem
-relate the components through shared parameters
-match derived component model with device dependent knowledge to derive exact parameter behaviours
-match derived component model with library component model to extract function and uncover gaps in original fault knowledge

Step 1 in the DR algorithm identifies the component nodes in the diagnostic network. This is achieved by searching the full fault knowledge base for nodes that have no child or sibling refinements. In some frame representations, for example JETA's, node

2. For implementation details of each step refer to chapter 4.0 and the results in chapter 5.0.

types are identified explicitly as combinations of replace and repair types (implementation described in section 4.1). Such node types are terminal nodes which map directly to components or actions directly related to components.

Step 2 in the DR algorithm is an optional step which is useful for complex devices that have many (more than ten) components. Devices with many components are generally decomposed into subsystems which perform specific functions in a device. DR identifies such components by extracting the subsystem relation from an explicitly represented frame slot, or from decomposing the name of the component itself which implicitly encodes the subsystem name (i.e. the name of the terminal node frame). In applications where no explicit subsystem frame slot exists, as in JETA, the name of the node is decomposed to identify the subsystem (implementation described in section 4.2). If the name of the node is not decomposable, then some background knowledge or semantic parsing of the name to identify its relation to a subsystem, may be needed.

Step 3 in the DR algorithm backtracks through the identified terminal nodes to their respective parent nodes to identify sibling nodes. The parent node generally has explicit representation of its children. This may be in the form of a slot in the parent frame. Some terminal nodes may have several parents which in turn may lead to more siblings for the terminal node. In JETA's case, siblings are identified through a slot that prioritizes the children in the diagnosis (implementation described in section 4.3).

Step 4 in the DR algorithm extracts links that represent behaviours between sibling nodes. Movement from the terminal nodes up to the root nodes represents a generalization of symptomatic knowledge to operational faults of a particular system (see layers of diagnostic hierarchy in figure 3.1). The parents directly linked to the terminal nodes often rep-

resent parametric tests on the components directly. Nodes that are siblings are clustered according to the parent node. The symptomatic information that the parent node represents is used to infer a parameter that relates sibling nodes. The inference may be based on the name of the parent node, or explicitly represented information about the symptoms in the frame knowledge of the parent. In the case of JETA, the parent node name is decomposed to extract the parametric link (implementation described in section 4.4). To constrain the parameter extraction, a list of the valid parameters for the entire device is used.

Step 5 in the DR algorithm matches the components and their extracted behaviours from the fault knowledge to device dependent and device independent component knowledge. This background knowledge is used to generate explicit behaviours (whether an input variable value increases or decreases with respect to the output variable, and whether an output variable value increases or decreases in relation to the input variable), between the components, and the relationships between the inputs and outputs of the components. Any control variables that directly relate inputs to outputs are identified by the presence of a plus or minus (+/-) sign in their device dependent background knowledge description. A relation between inputs and outputs is derived. This relation describes whether the inputs and outputs are changing with respect to each other in a proportional or inversely proportional manner. In addition, some of the inputs and outputs change in a piecewise linear manner. Thus, the relations between the component inputs and outputs can be proportional or inversely proportional, in a linear, or a piecewise linear relation. These complex behaviours are useful for describing components such as control components which require the modelling of feedback behaviour. Such modelling is essential in representing the behav-

aviours of complex real-world devices, such as jet engines, which have control loops between various components.

3.3.5 DR Assumptions

The DR algorithm makes two important assumptions for the re-use of fault knowledge:

1. The first assumption that DR-1 makes is that terminal nodes in a fault knowledge base can be identified (specific methods are described in section 3.4), and represent component knowledge. In the case of inconsistent fault knowledge, terminal nodes may not represent components. In such a case, DR can be used to identify terminal nodes that do not lead to components, and tag them as inconsistencies in the fault knowledge. These can then be corrected interactively with a domain expert.
2. The second assumption that DR-1 makes is that movement from the terminal nodes up to the root nodes represents a generalization of symptomatic knowledge to operational faults of a particular device (as illustrated in figure 3.1). This symptomatic knowledge can be used to extract parameters that are directly related to component behaviour. Thus, the symptoms encoded at the component parent nodes map to behaviours on the components.

3.4 DR-1 Input: Fault Based Knowledge

DR Fault-Based Knowledge

To achieve the knowledge-rich modelling proposed as the output of DR, one requires the use of a well-structured knowledge base. Such a knowledge base has an explicit knowledge representation and adequately represents diagnostic causality. DR extracts the terminal nodes that represent device components. DR then extracts the connections represented by shared parents of the terminal nodes. These connections are extracted between components in a subsystem or for a full device. The connections are also used to infer the variables that typify the behaviour between components (for example, temperature, flow, speed, pressure, etc.).

It is interesting to note that movement further up (one level or more in figure 3.1) from the parents of the terminal nodes does not extract a direct behavioural relation. It is possible to identify more behaviours nearer to the root nodes, but there is no pattern that would directly relate these behaviours to the component nodes at the bottom of the hierarchy. Examples of direct and indirect behaviours of the components are discussed in the results chapter 5.0.

In typical troubleshooting systems frames offer a great deal of flexibility in constructing and reasoning about problem solving knowledge. As an example, the frame template for a JETA diagnostic node is shown in figure 3.3. It has fourteen slots. DR generally uses four slots to extract components and their behaviours. It uses the name of a frame, the type of frame (to identify it as a terminal node), its parental knowledge (to identify any shared parents between terminal nodes) and its sibling knowledge (to identify components that share parents).

In JETA's case, a fifth slot named the activating-observation slot may also be used. This slot can be used by DR to identify component parameter information when the parent node does not. In experimenting with DR, it was found that this slot was useful for identifying a single behaviour for one of the components (see chapter 5.0 results). The fifth slot was judged to be too JETA-specific, and not likely to be present in other, well-structured frame-oriented fault knowledge bases. Given its limited utility, the fifth slot is not used to extract behaviour in DR, (for example, it is not used in the coffee maker application).

2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

defnode: node name (not a slot)
display-name: displayed node name for user
time-estimate: time estimate for recommended actions (displayed on node entry)
line-of-maintenance: severity of repair (1st, 2nd or 3rd line)
help-info: help information on node including hypertext markers
node-type: type of frame
 - general (default), / - all-ok,
 - snag,
 - functional test / - incorrect test setup,
 - repair,
 - replace,
 - repair/replace,
child-node-of: kinship to parent nodes
activating-observation: rule expression which could be used to activate node
child-node-ranking: relative child-node priority list (default = equally likely)
blocking-if-user-selected: check to see if other conditions which may block entry to node (if user uses direct selection to move in network).
entry-actions: sequence of operations carried out upon node entry.
 examples
 - general advice,
 - advice which depends upon path through network, status of other nodes or parameter values,
 - asking for intermediate parameter values,
 - intermediate servicing of another node,
 - setting states of other nodes,
 - ask user to make selections from the list of sub-nodes.
after-fixup-actions: sequence of operations to be executed after having carried out entry-actions.
completion-actions: sequence of operations to be executed before returning to the invoking node
functional-tests-required: validation tests to check that repair was carried out correctly
component-affected: component in engine parts database with serial numbers, etc.

Figure 3.3 An example diagnostic frame template representing explicit fault knowledge

3.5 DR-1 Outputs

The following are the three outputs of DR-1 in the Diagnostic Remodeler algorithm:

1. **Components:** these are the identified terminal nodes that represent components which are extracted in step 1 (extract components), and step 2 (if a subsystem is to be modelled), of the DR algorithm.
2. **Component Connections:** these are component-to-component links that represent connections that lead to shared behaviours in DR-2. The links are identified by backtracking to the parent node and clustering related siblings. The parent node is used to identify the behaviour that relates the siblings. The component connections are extracted in steps 3 and 4 of the DR algorithm.
3. **Component Parameters:** these are used to deduce specific component behaviours. Parameters are extracted by grouping all the connections that represent behaviours of a particular component. They are extracted in steps 3 and 4 of the algorithm.

3.6 DR-2 Input: Background Knowledge

Two types of background knowledge are used by the second phase of the DR algorithm. These typify the behaviours and identify the relationships between component input and output behaviours. The two types of background knowledge are device dependent and device independent background knowledge.

3.6.1 Device Dependent Background Knowledge

Device dependent background knowledge is used to identify the type of a component (for example a pump, a filter, a variable control, a vessel, a source, etc.), and any specifics about inputs or outputs related to a particular mode of operation (component operational mode). These are any increases/decreases (+/-) in the values of the input or output variables interpreted as component behaviours for a particular component operational mode. This level of detail is required by DR since in modelling component feedback it is essential to identify the modes of operation where the feedback happens. The traditional approach used in modelling feedback in electro- or hydromechanical systems requires that modes of component operation, and parameters that act as control variables in a particular mode, be explicitly identified. These control variables are interpreted to typify component behaviour [Treager 79].

The explicit description that is used for device dependent knowledge is:

```
glossary(KB_name,Component_id,Proper_name,  
        [Component_type,Modes],  
        [[Possible_inputs_related_to_mode],[Possible_outputs_related_to_mode]]).
```

These descriptions are instantiated as Quintus Prolog facts specific to the system to be modelled. Examples of these are included in the descriptions of DR's empirical evaluation applications for an engine and a coffee maker in section 3.8.

3.6.2 Device Independent Background Knowledge

The second type of background knowledge is device independent component model descriptions. These descriptions form a re-usable library of components. It contains the building blocks for modelling or designing a subsystem or device. In concept, it is similar to that of a CAD/CAM (computer aided design/computer aided manufacturing) library used in applications such as digital circuit design. The major difference is that the DR component models are textual descriptions rather than visual icons. At some future date DR may be extended to generate visual icons.

For each of the components in the device independent background knowledge the function of the component is described. The definition of function used in the background knowledge follows the definitions put forth by Sticklen and his colleagues [Sticklen et al. 88]. Function is taken to mean the goal or purpose of the device component in relation to its input and output behaviours (for example, a pump pumps fluid based on an input pressure, a vessel accepts and outputs liquid, etc.).

In addition to their identification, the inputs and the outputs of the component have explicit descriptions relating them to the component in the background knowledge. In the case of a regulated component that has a control signal¹, a regulation input parameter is also identified. Finally, the behaviour relation mapping the inputs and outputs of the component is identified. In the case of a proportional relation (increasing input and increasing

output or decreasing input and decreasing output), a behaviour is identified. More complex components which have complex behavioural relations dependent on specific operational modes, can also be described. The generalized explicit description that is used for device independent knowledge is:

```

component(Component_type,Component_name,Inputs,Outputs,Modes,F,I,O,R,B):-
    F = function(Component_name, Action(Input)),
    I = input(Component_name,Type(Input)),
    O = output(Component_name,Type(Output)),
    R = regulator(Component_name,Main_input),
    behaviour_proportional(Input,Output,Behaviour),
    B = behaviour(for(Component_name),
        behaviour_is_proportional(Input,Output,Behaviour)).

```

Thus, when an extracted component from DR-1 is matched to its respective device dependent and independent background knowledge, an expression that represents function, input, output, regulation behaviour (if it exists) and the overall input/output component behaviour relation is generated (see chapter 4.0 for implementation details). In the case of simple behaviour, the following expression generates a description of the proportional input to output behaviour.

```

behaviour_proportional(In,Out,Behaviour):-
    Behaviour = [increase_in(In),increases(Out),
        decrease_in(In),decreases(Out)].

```

In the case of complex behaviours, the behaviour is described as piecewise linear and the inputs, outputs, and control modes, are used to derive an expression that relates the control variables and their respective behavioural relations.

```

B = behaviour(Component_name,
    behaviour_is_pieewise_linear(Inputs,Main_Output,Outs,Modes,
        Control_var_list,Behaviours_list)).

```

To generate piecewise linear behaviour, each input and output variable are examined to infer the relation based on a positive or negative (+/-) change in their value. In control components, an input variable will increase up to a certain limit and then have to decrease in reaction to feedback from the output variable [Treager 79]. In some cases, inputs and outputs will increase or decrease in opposite directions. In these cases, the behaviour is interpreted as inverse proportional behaviour between the input and output. The generalized DR Prolog expressions that represent piecewise linear behaviour are:

```
get_individual_behaviour(proportional,Input_control_variable,Regulated_output_variable,Mode,B):-
```

```
    B = [the_behaviour_is_proportional_for(C_Mode),
         the_input_control_variable(In),results_in_regulated_output_variable(Out),
         at_limit(In), decrease(Out),
         increases(In),increases(Out)].
```

```
get_individual_behaviour(inverse_proportional,In,Out,C_Mode,B):-
```

```
    B = [the_behaviour_is_inverse_proportional_for(C_Mode),
         the_input_control_variable(In), results_in_regulated_output_variable(Out),
         at_limit(In), decrease(Out),
         increases(In),decreases(Out)].
```

These descriptions are used in combination with the device dependent knowledge to extract specific component models as described for the engine and coffee maker applications in section 3.8

3.7 DR-2 Outputs

DR-2 has four specific outputs. They are:

1. A symbolic expression describing the component connections. The description includes the name of the first component, its type and its function, and the name of the second component with the names of the parameter(s) linking the two components. Specifically, the expression is of the form:

```
[Component_name1,
 is_a([Component_type,for,[Component_function]]),
 and_is_connected_to(Component_name2),
 with_connectivity_parameter([Parameters_linking_component_1_and_2])].
```

2. A symbolic expression describing each extracted component, and its respective parameters. The expression is specifically:

```
[[Component_name, has_parameters([Parameters])],
```

3. The third output is the explicit description of each component behavioural model. This description is based on the input expression, describing the generalized component. The generated expression is:

```
{function(Component_name,Action(Output)),  
  input(Component_name,Type(Input)),  
  output(Component_name,Type(Output)),  
  regulator(Component_name,Main_input),  
  behaviour(for(Component_name),  
    behaviour_is_proportional(Input,Output,  
      [increase_in(Main_input),increases(Output),  
        decrease_in(Input),decreases(Output)]))},
```

4. The fourth output of DR-2 is the intersection between the knowledge extracted from the fault model and that generated by matching it with the background knowledge on the device components. This intersection represents gaps and possible inconsistencies in the fault knowledge. Gaps can be used to derive novel fault information for later integration back into the original fault knowledge. The list of gaps output by DR-2 explicitly names the component, the functional mode the behavioural gap was found for, the fault-based extracted parameters, and the respective input and output parameters generated through the component background knowledge. For each input and respective mode, a list of gaps is generated. In addition, any parameters found in the fault knowledge but not in the model are also output as gaps. These directly show possible inconsistencies in the fault knowledge. The expression denoting gaps is:

```
[[Component_name,  
  [[gap_for_mode,Mode,extracted,Parameter,  
    input,[Parameter_from_model],output,[Parameter_from_model]]]],
```

3.8 DR Empirical Evaluation: Fault Knowledge Input to DR-1

3.8.1 JETA

Objectives

The objective behind the development of the Jet Engine Troubleshooting Assistant (JETA) was to develop a tool that can be used to troubleshoot complex electro- or hydro-mechanical devices [Halasz et al. 92].

Approach

In JETA, domain knowledge is organized into network nodes³ that lead to other network nodes. The knowledge is organized into a directed network which is organized hierarchically around the root node, before branching off in directed paths based on various relations between the nodes. Reasoning proceeds by activating nodes whose root node contains the most general diagnosis of the system. Guided by a reasoning algorithm during a user session, network nodes are activated and trigger the more specific nodes based on parameter values provided by the user.

When troubleshooting mechanical systems [Halasz et al. 92] one generally follows a procedure which attempts to:

- validate that a problem with the device exists,
- execute tests which help localize a problem to a fault in a component,
- repair or replace the suspected component,
- repeat some tests to ensure correct operation of the components, and
- determine whether or not the device is fully operational.

3.8.1.1 How a Jet Engine Works⁴

A jet engine works on the mechanical principle, Sir Isaac Newton's third law of motion, which states that every action has an equal, and opposite reaction [How 90]. To generate the action in a jet engine, air is sucked into the engine, compressed to heat and expanded for exhaust at high velocity.

3. See Appendix B.1 for the graphical representation of the JETA fault network.

4. Based on a simplified description excerpted from the book "How Things Work" by the editors of Consumer Guide, Publications International Ltd., Illinois, 1990, pages 12-13 and 214-215.

The principle of compression is a simple one. Gas molecules move very quickly and for greater distances when heated. If a gas is heated in a contained area, without additional area for expansion, the gas molecules hit the walls of the contained area harder resulting in a build-up of pressure. Reducing the size of the contained area, without allowing any of the molecules to escape, will result in a buildup of pressure. Energy is expended in building up the pressure, thus the stored energy is increased which in turn can be released as motion or thrust in a jet engine. The compressor in a turbojet engine is usually an axial compressor. It has alternating sets of rotating and stationary blades attached to disks mounted on a shaft. The action of each set of blades compresses the air and slightly heats it. There may also be inlet guide vanes that direct the flow of the first row and air bleed valves, which balance the flow through the compressor.

Fuel is then burned with the hot compressed air output by the compressor. The fuel combustion results in a hot mass of gases that expands through a turbine. A turbine has a similar configuration as a compressor, but extracts energy from the gases. This energy extraction from the turbine rotates the compressor which is mounted on a connecting shaft.

Since fuel was added, the gases have some remaining energy in the form of pressure and temperature. These gases create a powerful thrust that moves the plane forward. A nozzle at the rear of the engine efficiently converts the pressure of the escaping gases to higher velocity. The jet engine nozzle is formed by the converging section at the rear of the outer cone. The area of the jet engine nozzle is critical since it affects the back pressure of the turbine and hence the speed in rpm (rotations per minute), thrust, and exhaust gas temperature which in turn affect overall engine operation.

Afterburning of the engine exhaust gases is a method of augmenting the basic thrust of an engine for military aircraft [Treager 79]. This improves the aircraft take-off, climb and maneuverability, without adding weight by increasing the basic size of an engine. Afterburning consists of introducing and burning fuel in a duct between the engine turbine and the jet engine nozzle using the unburned oxygen in the exhaust gas to support combustion. Combustion of the exhaust gases increases the temperature of the exhaust gas, the speed of the jet leaving the propelling nozzle, and as a result, the engine thrust. In summary, these five steps describe jet engine afterburning operation:

1. A compressor pulls air into the engine.
2. The compressor's blades compress the air and cause some air heating.
3. In the combustion chamber, fuel enters through spray nozzles, and mixes with the compressor exit air. The mixture ignites because of the high temperatures.
4. The hot gases expand through the turbine which drives the compressor.
5. The exhaust gases are channeled with an adjustable jet engine nozzle, which may be adjustable in the nozzle area. Fuel may be added, and ignited in an afterburner duct, to provide additional thrust.

3.8.1.2 Reasoning Strategies in JETA

The reasoning algorithm developed for JETA is quite flexible:

"It interprets locally encoded commands in the network. This allows for context-sensitive branching depending on values of the different parameters used at each node. Advice displayed at a node is tailored for special circumstances, such as the mode of entry into the node or the path taken in the network to get to the node. This allows JETA to behave flexibly and does not force the user through fixed sequences of questions. Logical "jumps" in the network are accommodated so that a current line of reasoning can be aborted should new observations lead to other nodes becoming more probable. Postponement of certain procedures is allowed when a technician may not have time to carry them out, or a parts shortage exists and observations are not conclusive enough for the particular failure, or the technician simply has a "hunch" that this path is not where they really want to be." [Halasz et al. 92]

Causality in JETA

At the top level knowledge of operation of the device is represented. At a lower level the symptomatic level of the device problems is represented. At the lowest level the components which are repaired or replaced in the device are represented. Causality is difficult to discern in JETA. One reason for this is the use of intermediate nodes to affect the control strategies. Such intermediary nodes are not links in the causality chain for a diagnosis but are needed to maintain the flexibility in reasoning. JETA's reasoning strategies navigate its network in three possible ways:

- through sibling refinement by traversing the network from left to right
- through activating conditions or rules that allow the reasoner to navigate the network
- through entry actions which require the evaluation of various parameters which in turn activate new nodes not immediately related to the current node

3.8.1.3 Frame-Based Knowledge Representation of JETA

JETA's frame-based knowledge representation was evolved carefully:

"The design of the knowledge representation evolved from careful analysis of the manuals and many discussions with an expert maintenance technician. A frame representation was selected over rules since the structure of the domain knowledge is made more explicit. Although the size of the J85 knowledge-base could not be predicted at the outset, the complexity of the troubleshooting task pointed the developers clearly towards a representation that promotes structure, flexibility and organization.

The troubleshooting knowledge appears as a diagnostic decision network which is a general directed graph. Each node in the network corresponds to a decision point in the troubleshooting process while the connecting links represent the flow of control. The network is organized into four distinct levels, each made up of nodes of a particular type. For example, a major symptom (or snag) node, is the main reason for servicing the engine and is one to two levels from the top. It is common to have varying depths of nodes along a path within a level. The overall network is much broader than it is deep since there are many components and associated symptoms. The depth of nodes along a network path varies from a minimum of four to a maximum of approximately twelve. The possible next moves in the network are represented as the children of a node. Any node can have multiple parents which permits more than one cause for a component malfunction or several means of entering a node. The troubleshooting

knowledge is encoded at each node or decision point through the use of a custom command language. The reasoning strategy, while interpreting this language, allows for the many aspects inherent in complex diagnosis." [Halasz et al. 92]

3.8.1.4 DR-1 Input: JETA's Fault-Based Knowledge

A frame in JETA has the following general form:

```
(frame-header frame-name
 (slot 1)
 (slot 2)
 ....
 (slot n)
 )
```

The frame-header specifies the type of frame. In JETA, three types are used: diagnostic network, parameter and glossary frames. The frame-header for each is *defnode*, *defparam*, and *defglossary* respectively. The frame-name which labels a particular frame is only used by the reasoning module and therefore not displayed to the user. The slots contain the information pertaining to the particular frame.

Diagnostic Network Frames

A JETA diagnostic network frame template designed specifically for the diagnosis of mechanical equipment is given in figure 3.2. Of the fourteen network node slots shown, four are relevant to the DR algorithm.

The slots used are highlighted in Figure 3.2 and are the node name, the node type, the childnode-of, and the child-node-ranking. The node-type slot is used to identify the terminal nodes. Specifically, for JETA the replace node types are used to extract the components. Knowledge of the node type is combined with knowledge of the subsystem (in JETA's case this is extracted from the node name). The child-node-of slot is used to determine the parent of a terminal (component) node. The child node-ranking is used to determine the siblings of a terminal node.

A fifth slot, the activating-observation slot was used for the JETA application of DR to identify parameters when the parent node information did not identify any. As stated earlier, this slot was useful for identifying a single behaviour for one of the components (see chapter 5.0 results) and this was judged as too JETA-specific. Such a slot is not likely to be present in other well-structured fault knowledge bases. Given this limited utility, the activating-observation slot is not assumed to extract behaviours in DR and is not used in other applications.

Parameter Frames

Parameters in JETA are variables which represent the user's observations about the current state of the device and the environment it operates in. The values of the variables are used by the rule expressions in the activating-observation slot of a diagnostic network node frame. It is this rule expression which characterizes the symptoms of a particular node. For example, the actual temperature (a particular parameter), is compared to a value and from this the symptom high operating temperature or low operating temperature is determined and used for reasoning. DR does not use the parameter frames and they are described here as background for JETA.

Glossary Frames

The glossary frames are used to provide hypertext information in JETA. Hypertext basically provides an automated indexing system. For example, if a certain term in a paragraph of text required clarification or a definition, one would simply select it (with the mouse) and the information would be displayed. If there were more clarifications required to help interpret something in the text, it could be accessed in the same manner.

In the case of JETA, if there are any common abbreviations required by the help-info slot of either the diagnostic network or parameter frames, the definition slot of a glossary frame is accessed through a hypertext link. DR does not use the glossary frames, although some of DR's glossary fact contents are directly hand coded from JETA's glossary.

Diagnostic Network

JETA's knowledge base is designed to mainly troubleshoot aircraft engine acceleration faults. Such faults involve most of the subsystems of the aircraft engine and are more difficult for technicians to troubleshoot. JETA's knowledge base is neither complete nor fully consistent. This is demonstrated by several gaps in the knowledge which are apparent as the knowledge is analysed. The identification of knowledge gaps is in itself a contribution by the Diagnostic Remodeler algorithm in its attempt to build component models as detailed in the results of chapter 5.0.

JETA's knowledge network of approximately 200 nodes starts as a hierarchy which is organized into two main branches as illustrated in figure 3.4. The root of the hierarchy is a J85-CAN-15 jet engine fault. This splits into a fault during phases of engine operation or a 'snag'. Snag is a term used to indicate a symptom of a fault and snags are used to identify faults directly rather than through operational tests. The phases of engine operation faults branch into seven paths: engine start-up, acceleration, steady state, deceleration, shut-down, general engine operation and indication. JETA's knowledge for steady state, shut-down, general engine operation, and indication faults is incomplete, hence these nodes are represented as grounded-out in figure 3.4. Start-up, deceleration and general engine operation faults are refined to snags. Acceleration faults are refined to four faults

during a burst of acceleration. The acceleration burst faults are subsequently refined to snag faults. Steady state faults are refined to idle, part-speed and military speed or afterburner speed (AB) faults which are refined to snag faults.

Figure 3.4. JETA's top level knowledge organization and child node refinements to snags

These thirteen snags are encoded in JETA:

- 1) engine will not start,
- 2) slow acceleration to idle,
- 3) RPM hangup below idle,
- 4) high exhaust gas temperature (EGT) during start-up,
- 5) nozzle dance,
- 6) nozzle jitter
- 7) no hesitation at acceleration area
- 8) slow acceleration
- 9) variable exhaust nozzle (VEN) hangup at acceleration position
- 10) VEN hangup at lockout position
- 11) engine flameout
- 12) compressor stall
- 13) excessive rollback on afterburner initiation

These nodes are further refined to indicate specific component faults. As an example the snags from the RPM hangup that are refined to actions on engine components on the main fuel system are illustrated in Figure 3.5 below.

The RPM hangup snag is refinable from the acceleration speed bursts, the general engine operation, or the start up faults. The RPM hangup snag is refinable to three nodes identifying the hangup at an engine speed that is idle, that is below idle or that is between 51% and military speed limit. The RPM hangup at idle refines into Wf (fuel flow) within or not within limits. If Wf is within limits it is refined to a main fuel nozzles or main fuel control replacement and repair end nodes. Similarly, the RPM hangup at idle with Wf not in limits is refined to main fuel nozzles or main fuel control actions.

Figure 3.5. Subset of JETA's acceleration RPM hang up and engine flameout nodes

The RPM hangup between 51% and MIL is refined to RPM hangup where the exhaust gas temperature (EGT) is greater than military (MIL) limits or the RPM hangup is within or not within 3% engine speed differential. The EGT > MIL fault is refined to engine component actions other than those of the main fuel system (for example, the variable exhaust nozzle repair node). The RPM hangup within/ not within 3% is refined to an overspeed governor (OSG) replacement fault (note that the intermediate refinements to OSG operation and OSG tests 1 and 2 are omitted from the diagram). The RPM hangup below idle is refined to a low fuel flow fault which in turn is refined to the replacement of the pressurizing and drain valve.

A second snag is illustrated in Figure 3.5, mainly an engine flameout fault. This fault is refinable to a loss of fuel flow which is refined to the main fuel pump replacement node. In addition, the engine flameout is refinable to a decrease in fuel flow fault which implies that the main fuel pump or main fuel control filters need replacing. In addition, the engine flameout fault is refinable to a pressurizing and drain valve problem which may indicate its replacement.

Terminal Nodes: JETA's Repair and Replacement Nodes

At the lowest level of JETA's diagnostic hierarchy are 78 repair or replacement nodes. Each JETA node has two names: a defnode name used for reasoning and a less cryptic display name used in user interaction. The defnode names have the structure *N-SYS-xx, where SYS is an acronym for a subsystem of the engine and xxx is related to the component or procedure in question. Thus, the repair and replacement node names can be used to group the nodes into 13 subsystems. As an example of JETA's terminal nodes the main fuel system terminal nodes are shown in table 6. They include replacement nodes for

the main fuel nozzles, the main fuel control, the main fuel pump, the overspeed governor and the pressurizing and drain valve. In addition, the main fuel system repair nodes include the main fuel pump and main fuel control filters, MFC feedback cables, the overspeed governor test lever position, the fuel density setting and a signal line between the main fuel control and the afterburner fuel control.

Table 6. Main Fuel System repair and replacement nodes

repair/ replace main fuel noz- zles (FN)	replace main fuel control (MFC)	replace over- speed gover- nor (OSG)	replace main fuel pump (MFP)	replace Pres- surizg. and Drain valve (PDV)	repair fuel filter (200 mesh)	repair- fuel filter (120 micron)	repair- MFC cable	repair- MFC/ ABFC Signal Line	repair OSG test lever pos.	repair Fuel Den- sity Setting
--	---	---	--	---	---	--	-------------------------	---	--	---

List of Engine Parameters

In addition to the JETA fault knowledge the full list of engine parameters is used to constrain the extraction of the parameters from the parental node names. The expression used is input to the Prolog knowledge base of the DR algorithm as the following fact:

```
parameters(engine, ['N1','RPM','CDP','P3','EGT','T5','CIT','T2',
                    'PA','PLA','Ta','WF','WFM1','WFT','NPI','Thrust','IGV']).5
```

These parameters have meaningful symbolic text translations that are used as input to DR-2 as described below.

3.8.1.5 DR-2 Input: Engine Device Dependent Background Knowledge

Two types of device dependent knowledge are input to DR. The first is knowledge from the JETA glossary frames that translates acronyms into meaningful symbolic text. The first type of translation is for parameter names. The full set of these parameters is included in the implementation chapter 4.0. Some of the expressions for the main fuel sys-

5. Note that some parameters are actual while others are sensed, the pairs are (N1, RPM), (CDP,P3), (EGT,T5), (CIT,T2).

term that are input to the DR Prolog knowledge base are (note that % is a comment marker in Prolog):

```
% glossary for engine parameters: glossary(KB_name, Parameter, Meaningful_name)
glossary('JETA','N',engine_speed).           % measured by RPM indicator
glossary('JETA','CDP',compressor_discharge_pressure). % also P3
glossary('JETA','EGT',exhaust_gas_temperature). % also T5
glossary('JETA','CIT',compressor_inlet_temperature). % also T2
glossary('JETA','Pa',fuel_pump_inlet_pressure). % also 'Pa'
glossary('JETA','PLA',power_lever_angle).
glossary('JETA','WF',weight_of_fuel_flow).
```

The second set of acronyms are related to the component names. Their translation is included in the expression that represents the device dependent component model. For example, one of the components of the main fuel system is the main fuel control (MFC). The MFC is a control with 6 fuel scheduling modes that vary from acceleration to speed cutback with a variety of modes in between. For each mode there are key parameters that represent component behaviours. They include engine speed (N), pilot demanded speed (Nd), throttle position or power lever angle (PLA), compressor inlet temperature (T2), fuel flow (Wf), compressor discharge pressure (P3) and inlet guide vanes (IGV) which indicate guide vane angles and air bleed valve positions. As shown in the excerpt of device dependent background knowledge for the MFC, each mode has a specific set of behaviours represented as lists of input-output pairs (note that % is a comment marker in Prolog):

```
% glossary(KB,Component, ProperName,[ComponentType,for,[Modes]], [InputOutputPairs]).
% main fuel control terms from JETA's Glossary Frames and J85 Control Parameters/Modes
glossary('JETA','MFC',main_fuel_control,
[control,for,[acceleration_fuel_limit_control, proportional_speed_control, speed_cutback_control,
variable_geometry_scheduling, fru_fuel_selection, fuel_flow_selection]],
[[['N+', 'PLA'], ['acceleration_WF/P3+']], %%% Nd+-->f(PLA+), WF/P3+---->f(WF+)
[['N+', 'PLA+'], ['proportional_speed_WF/P3+']] % PLA+ -> Nd+, Nd-N-->adjusted WF, +ve, fru inc., -ve dec.
[['N+', 'T2', 'T2_limit'], ['speed_cutback_T2-']], %%% T2 less than or equal T2_limit, -->Nd-
[['N+', 'T2'], ['IGV-', 'bleed valve positions']], %%% IGV kept between min/max values (fully open to closed)
[['acceleration_WF/P3', 'proportional_speed_WF/P3', 'P3'], ['Wf']], %%% reflect selec'n between 2 modes
[['Wf', 'WF_min'], ['WF_mfc']])). %%% proportional until WF_min
```

3.8.1.6 DR-2 Input: Engine Device Independent Background Knowledge

The second type of input into DR-2 is device independent background knowledge. The complete set of expressions for the engine application is in chapter 4.0. As an illustration of the structure of the knowledge the device independent model for a pump and a variable nozzle are shown below. These generalized component descriptions can be used for any of the engine subsystems and are not specific to the main fuel system. The component is matched based on the following predicate call from DR-2:

```
component(Component_type,Component_name,Inputs,Outputs,Modes,F,I,O,R,B)
```

Thus, for a pump, the function is to deliver fluid which is accepted as its input variable. The output is the fluid (at higher pressure) and the control variable controls the flow rate. The behaviour relation between the inputs and outputs is proportional. The specific device independent component model for a pump is:

```
component(pump,Pump_name,Fluid,Control,_F,I,O,R,B):-  
    F = function(Pump_name, delivers(Fluid)),  
    I = input(Pump_name,fluid(Fluid)),  
    O = output(Pump_name,fluid(Fluid)),  
    R = regulator(Pump_name,Control),  
    behaviour_proportional(Fluid,Control,Behaviour),  
    B = behaviour(for(Pump_name),  
        behaviour_is_proportional(Fluid,Control,Behaviour)).
```

A complex engine has a variable exhaust nozzle. This nozzle has a variable number of inputs and outputs and its overall function is controlled by specific input/output behaviours. Once an output is at a particular limit, then a signal is fed back so that the input is decreased. Thus, the variable nozzle has a variable number of inputs and outputs which may act as control variable(s) at different modes of operation. This variable behav-

behaviour is interpreted as piecewise-linear behaviour. The variable nozzle device independent component model is:

```
component(variable_nozzle, Nozzle_name, Inputs, Outputs, Modes, F, I, O, R, B):-  
    Outputs = [Main_Output|Outs],  
    F = function(Nozzle_name, controls(Main_Output)),  
    I = input(Nozzle_name, control(Inputs)),  
    O = output(Nozzle_name, control(Outputs)),  
    R = regulator(Nozzle_name, regulation_control(Inputs)),  
    typify(Inputs, Outputs, Modes, Control_var_list, Behaviours_list),  
    B = behaviour(Nozzle_name,  
        behaviour_is_piecewise_linear(Inputs, Main_Output, Outs, Modes,  
            Control_var_list, Behaviours_list)).
```

3.8.2 Coffee Maker

To test algorithm generality and demonstrate that DR is applicable to both complex and simple domains, the coffee maker fault knowledge base was developed. Through the coffee maker application the DR algorithm assumptions described in 3.3.5 were tested. Namely all terminal nodes were used as input to DR-1 instead of simply the replace component nodes as was the case with the engine application. The coffee maker has 29 fault nodes of which 10 are terminal nodes. Thus, the entire coffee maker device was modelled using DR. The knowledge representation for the coffee maker was frame-based with a subset of the JETA slots. The device dependent background knowledge consisted of only the component descriptions since the fault base was written in meaningful symbolic text. In addition, the engine device independent component knowledge library was added to, to include descriptions for a holder, heater and a switch.

3.8.2.1 Frame-Based Knowledge Representation of Coffee Maker Faults

The coffee maker frames have the following slots:

```
defnode(NodeName,  
        [[NodeName],  
         [display-name, NamewithDetail],  
         [help-info, DescriptionOfFault]  
         [node-type, Type],  
         [child-node-of, [Parent]],  
         [child-node-ranking, [ChildrenRankings]]]).
```

The list structure of JETA's⁶ frames was maintained to allow the full re-use of the DR code implemented for the engine. Although the frames have 6 slots, only 4 are used by the DR algorithm, namely, the node name, the node type, the child-node-of, and the child-node-ranking. All node names are Prolog strings of meaningful symbolic text to facilitate program readability. A help-info slot was used to provide some explicit explanation for the existence of the node and may be re-usable at some future time as justification in describing model component details (or details at any point in the hierarchy - see [Abu-Hakima and Oppacher 90] for the importance of explicit knowledge in knowledge representation and acquisition for justifying reasoning).

The node type slot included repair, replace and general node types. The DR algorithm for the coffee maker application was run with all terminal nodes, specifically all repair and replace nodes. The child-node of slot indicates the parent of the node and is used by DR-1 to extract links between components. The child-node-ranking slot listed the children of the node in the order of likelihood of fault behaviour and ease of diagnosis (much as JETA's diagnostic network was organized).

6. Note that JETA is implemented in LISP and its fault knowledge was converted to Prolog for re-use in DR.

3.8.2.2 DR-1 Input: Coffee Maker Fault-Based Knowledge

Diagnostic Network

The diagnostic hierarchy of the coffee maker device faults is shown in figure 3.6. The hierarchy includes 29 faults. At the root node which represents a general coffee maker fault, there is a refinement to four branch nodes. The first branch is the bad coffee taste node which can be refined to a foul coffee taste problem, and coffee too strong or too weak problems. The foul coffee taste indicates a possible problem with the water in the water reservoir. The coffee too strong is the parent node to the three siblings coffee filter, coffee holder and coffee pot problems. The coffee too weak parent node indicates a possible fault with the type of coffee filter or coffee grounds in the coffee holder which are again sibling nodes.

The second branch from the root node is the grounds in the coffee problem. This is refinable to a missing coffee filter, a filter that is too porous for the type of coffee grounds and a possible overflow of the grounds in the coffee holder.

The third branch from the root is the wrong coffee temperature node which is refinable to four nodes indicating that the coffee or the water are either too hot or too cold. In the case of a coffee temperature problem, a fault in the coffee heating element is indicated. In the case of a water temperature problem, a fault in the water heating element may be the cause.

The fourth branch from the root is a problem with the coffee not dripping. This may indicate a problem with the power to the coffee maker device which can be manifested as one of two siblings, a coffee maker plug or a coffee maker switch problem. The

coffee could also not be dripping because of absence of water in the water reservoir or that the water is not dripping from the water reservoir. In the case of the water not dripping it could indicate a problem with one of two siblings, a plugged coffee drip problem or a problem with the water temperature control which is not allowing the water to drip.

Figure 3.6. Coffee Maker Device Fault Knowledge

Terminal Nodes

The coffee maker device has 10 terminal nodes which are a combination of repair and replacement nodes. The names of the nodes encoded in the knowledge base are the proper meaningful symbolic names listed in table 7. The repair nodes include the water reservoir, the coffee holder, the switch, the power plug, the coffee pot and the coffee drip. The replace nodes include the coffee filter, the coffee and water heating elements, as well as the water temperature heat control.

Table 7. Coffee maker repair and replacement nodes

repair water reserv.	replace coffee filter	repair coffee holder	replace coffee heating elemnt	replace water heating elemnt	repair coffee maker switch	repair coffee maker plug	repair coffee pot	replace water temp control	repair coffee drip
----------------------------	-----------------------------	----------------------------	--	---------------------------------------	-------------------------------------	-----------------------------------	-------------------------	-------------------------------------	--------------------------

List of Coffee Maker Device Parameters

In addition to the coffee maker fault knowledge the full list of device parameters is used to constrain the extraction of the parameters from the parental node names. The device had four parameters with which to model behaviour. They are input to DR as the following Prolog fact:

```
parameters(coffee,[water,coffee,power,heat,]).
```

3.8.2.3 DR-2 Input: Coffee Maker Device Dependent Background Knowledge

The device dependent background knowledge consisted of only the component descriptions since the fault base was written in meaningful symbolic text. The general expression that defines the knowledge is included below as a Prolog comment. In addition, the component facts for the coffee maker plug and the drip are included below. The full device dependent background knowledge is in the implementation chapter 4.0.

```
% glossary(KB,Component_id,Proper_name,[Component_type,Modes],
%          [[Possible_inputs_rel_mode],[Possible_outputs_rel_mode]]).

glossary(coffee,'coffee maker plug',[regulator,for,[power]],[[['power+'],['power+']]]).

glossary(coffee,'coffee drip',[regulator,for,[coffee]],[[['coffee+'],['coffee+']]]).
```

3.8.2.4 DR-2 Input: Coffee Maker Device Independent Background Knowledge

In addition, the engine device independent component knowledge library was expanded to include descriptions for a holder, heater and a switch. The control component

model for the engine was renamed as a regulator model for the coffee maker application but it essentially is the same (see chapter 4.0 for full details).

A switch component model has an input described as a state to make it very general for any type of switch (for example a dip switch on a computer card). From the device dependent knowledge for the coffee maker, the switch input is 2-state (on/off). The output is a state dependent on the input. The output behaviour of the switch is dependent on the state of the input.

```
component(switch,Switch_name,Input,Output,_F,I,O,R,B):-
  F = function(Switch_name, has_states(Input)),
  I = input(Switch_name,input_states(Input)),
  O = output(Switch_name,output_states(Output)),
  R = regulator(Switch_name,none),
  B = behaviour(for(Switch_name),
    behaves_according_to_state(Input,implies_respective(Output))).
```

The coffee maker has a couple of heating elements for water and for coffee. The function is for it to heat a fluid. It has input behaviour (power) and output behaviour (heat). The output is regulated by the input behaviour in a proportional relation.

```
component(heater,Heater_name,Input,Output,Fluid,F,I,O,R,B):-
  F = function(Heater_name, heat(Fluid)),
  I = input(Heater_name,is(Input)),
  O = output(Heater_name,is(Output)),
  R = regulator(Heater_name,Input),
  behaviour_proportional(Input,Output,Behaviour),
  B = behaviour(for(Heater_name),
    behaviour_is_proportional(Behaviour)).
```

Another component added to the library is that of a holder. The function of a holder is to act as a vessel. It has an input and an output behaviour which is proportional in nature.

```
component(holder,Holder_name,Input,Output,_F,I,O,R,B):-
  F = function(Holder_name, vessel_for(Input)),
```



```

I = input(Holder_name,is(Input)),
O = output(Holder_name,is(Output)),
R = regulator(Holder_name,none),
behaviour_proportional(Input,Output,Behaviour),
B = behaviour(for(Holder_name),
    behaviour_is_proportional(Behaviour)).

```

3.9 DR Algorithm Evaluation

The DR algorithm was implemented according to the following 5 steps:

1. The 5 steps of the algorithm were coded as 10 individual modules, in traditional software engineering fashion. The modules were first specified in a high level design based on the initially proposed algorithm requirements. This implies that the inputs and outputs to each module were clearly identified, the code was fully commented and each module was designed and implemented separately. The modules were tested individually and integrated in an iterative fashion to facilitate debugging. DR-1 modules included *get components*, *get component connections* and *get parameters*. DR-2 modules included, *translate-1* component connections to meaningful symbolic text expressions, *translate-2* component parameters to meaningful symbolic text, *device dependent background knowledge* (one for engine and another for coffee application), *device independent background knowledge* (component library for engine and second extended one for coffee) and *match*.
2. The fault knowledge for JETA was converted to Prolog from LISP.
3. The fault knowledge for the coffee maker device was formulated and coded in Prolog.
4. Once the automatically acquired DR engine subsystem model was generated, it was analysed to ensure that gaps or inconsistencies could be identified from the JETA fault knowledge. In addition, the results were validated by an engine expert from NRC's Institute for Aerospace Research's Propulsion Laboratory (as described in section 3.9.1), which has test cells of aircraft engines such as that for which JETA's fault knowledge was acquired. These engines are used to study various engine performance modelling techniques.
5. After the engine results were validated, the algorithm's generality was tested with a coffee maker application. The coffee maker fault knowledge was input to DR-1. In addition, device independent component models formulated for the coffee maker device were used as input to DR-2. The coffee maker device was successfully mod-

elled as described in the results chapter 5.0. The coffee application model was verified through analysis of the components, connections and resultant behaviours. Since it was a simpler application (in comparison to the engine), and a commonly existing device in daily use in people's homes and offices, its resultant model was more easily verified, and did not seem to require specialized domain expertise.

3.9.1 Validation of Engine application

The background knowledge, and the results of the application of the DR algorithm to the jet engine fault knowledge, to generate subsystem component models, was validated by Jeff Bird, an expert from the Propulsion Laboratory, at NRC's Institute for Aerospace Research. The evaluation was typical of knowledge-based systems, where the expert was asked to evaluate the output of the algorithm based on his domain expertise. As much as possible, this evaluation was done in an independent manner so that the expert's input was not biased by the developer. This was accomplished by corresponding with the expert by fax and by providing the expert with raw (unmassaged) output.

The DR algorithm was described to the expert in August 1993, in an informal two hour discussion, before it was implemented. He believed that it would be useful in bridging the gap between fault and model knowledge (logbook notes from this meeting are in Appendix A.2, page 1).

The black box background knowledge component models for the engine were first generated, and faxed to the expert in December 1993. The expert returned these by fax, with annotations, and attached additional clarifications (this correspondence is in Appendix A.2, pages 2 to 7). The additional clarifications on pages 6 and 7 of Appendix A.2 included a device dependent description for the Main Fuel Control Component (MFC),

the Overspeed Governor (MFC/OSG), the Variable Exhaust Nozzle (VEN) and the Afterburner Fuel Control (ABFC).

DR raw output model descriptions were then sent to the expert in May 1994, who subsequently annotated the output directly (this output is in Appendix A.3). He understood the output component model descriptions, and gave direct feedback on it, as well as the expressions output as a result of matching with the background knowledge. Specifically, on page 1 of the output in Appendix A.3, he pointed out that a connection was missing in the fault knowledge between the main fuel nozzles and the main fuel pump. He also pointed out that the parameter engine speed was missing in relation to the main fuel pump. He also pointed out that the MFC acts like a processor and fuel is increased until an alarm signal is fed back so that it may be decreased again. On page 2, he pointed out that JETA does not likely encode control component mode knowledge. He also appended 3 pages (pages 6 to 8 in Appendix A.3) explaining the operation of fuel filters and fuel nozzles as processors and the main fuel pump as a transformer. He also had questions about the syntax of the code in describing operations of various components and elaborated with more explanation on the MFC control modes. Thus, he was able to independently understand, and make use of the DR output descriptions to hypothesize about the missing fault knowledge in JETA and the goodness of the background knowledge. This confirmed that the DR algorithm could produce useful descriptions of subsystem model components, that were practical.

The expert then had a three hour session with the developer of the DR algorithm, evaluating, and explaining the results he understood from the raw output. Specific expla-

nations included, why DR uncovered gaps in JETA, and how the background knowledge formulation described the links between the components.

In the first case, the expert believed that JETA encodes knowledge based on failure rather than operation. In some cases, knowledge of operation is used in JETA to identify faults. But generally, JETA identifies faults by testing for component failures, this is why operational knowledge which is important in linking components may be missing from JETA. In addition, JETA does not include explicit control mode faults that map directly to components. This is why this layer of knowledge (see figure 3.1) was found to be missing in JETA. For future engine fault bases, the expert agreed that including the missing knowledge discovered by DR, would simplify the re-use of fault knowledge, for the generation of black box component models.

4.0 Implementation of Diagnostic Remodeler Algorithm

4.1 High Level Design

The Diagnostic Remodeler algorithm has 10 modules used in two distinct phases, DR-1 and DR-2. The outputs of DR-1 are used as inputs to DR-2 as the flow diagram of Figure 4.1 illustrates.

Figure 4.1. High level view of Diagnostic Remodeler algorithm showing DR-1, DR-2 modules.

All DR modules are shown in boxes and generated outputs are shown in highlighted ellipses. Ellipses that are not highlighted constitute fault-based knowledge used as input to DR. Note that the background knowledge modules are highlighted in bold to indicate their implementation for both the engine and coffee device applications.

DR-1 extracts the components from the knowledge base in its *get components* module. The extracted components are then used as input with the list of device parameters to the *get component connections* module. The inferred component connections are then used as input to the *get component parameters* as well as the DR-2 *translate-1* module. The output from the *get parameters* module is all the extracted component parameters. Thus, DR-1 has three outputs: components, component-component connections and component parameters.

DR-2 uses device dependent knowledge as input, to refine and translate the component connections using *translate-1*, and the component parameters using *translate-2*, to meaningful symbolic text. The refinements in the translate modules are simply the removal of duplicates and the flattening of nested lists. In addition, *device dependent* and *independent background knowledge*, with the refined component parameters, are used as input to the *match* module, to generate the component behaviour model output. Also, *match* generates a second output, the gaps in the fault knowledge. These gaps represent the intersection of the component parameter knowledge extracted in DR-1 from the original fault knowledge with the device dependent background knowledge. All gaps in the fault or model knowledge are output, so that they may be analyzed for inconsistencies or gaps in the original fault knowledge.

4.2 DR Algorithm Structure

The DR algorithm has the following pseudo code:

For nodes 1 to n in fault knowledge base

 identify and print out terminal nodes 1 to t

For terminal nodes 1 to t

 identify parent 1 to p in fault knowledge base

 For parent 1 to p (and terminal node t)

 identify sibling 1 to s

 extract parameter encoded in parent description

 output component-component-connection through parameter description

For component-to-component-connection 1 to cc and valid list of device parameters

 collect individual parameters for each component

 output component-parameter description

For component-component-connection 1 to cc and device dependent background K

 output meaningful component-component connection text

For component-parameter description 1 to cp and device dependent background K

 output meaningful component-parameter text

For meaningful component-parameter text 1 to cpt, device dependent and independent K

 output component behaviour model

 output gaps between component behaviour model and meaningful parameter text

- Thus, the 10 modules of DR implement the above pseudo code. The *get components* module implements the first loop. The *get components connections* implements the second loop. The *get component parameters* implements the third loop. The fourth and fifth loops are output by modules *translate-1* and *translate-2* respectively with *device dependent background knowledge* as input. The sixth loop is implemented by the *match* module with *device dependent* and *independent knowledge* as input as well as the output of the *translate-2* module (parameter text).

4.3 Get Components Module

The first of DR-1's 3 modules is the *get components* module. It searches the knowledge base for those frames representing components relevant to a particular subsystem. The inputs to the module are:

1. a fault knowledge base of frames,
2. a subsystem to collect frames for (naming a subsystem is *optional*, and is intended to constrain modelling in DR to components of a specified subsystem for complex applications, such as the engine application), and
3. the type of terminal nodes to collect.

The output of the module is a list of all terminal node names or terminal nodes of a particular type for a named device subsystem.

As described in the DR assumptions (section 3.3.5) extracting the components related to a subsystem was used to constrain the component model extracted for the aircraft engine. This assumption was relaxed for the coffee application, and all the terminal nodes of the coffee faults were used in modelling the behaviours relating the device components.

To achieve its objectives, the *get components* module first reads in the specified fault knowledge base of frames that represents the nodes. It then checks the name of the frame in the node name slot. For a named subsystem (in the case of the engine application) the module validates that the node name can be decomposed to include the name of the specified subsystem. Note that in a more general knowledge base the name of a subsystem

would ideally be a separate frame slot. Since DR was aimed at the re-use of knowledge in an existing complex fault base, it makes use of JETA's encoding of subsystems within the names of relevant nodes. After the *get components* module establishes that a frame is related to a named subsystem, the node type is checked to validate the frame as a terminal node. In the case of the engine, only replace terminal nodes are used to further constrain the search. Other terminal nodes related to a subsystem in the engine application are collected in the *get component connections* module. In the case of the coffee maker device, all terminal nodes are collected, namely repair and replace node types. The list of components is returned as the output of the *get components* module. The main predicate of *get components* module:

```
get_component_nodes(KB_name,Subsystem,TerminalNodeType,Components):-
    kb_frames(KB_name,KB),
    build_list(KB,Subsystem,TerminalNodeType,Components).
```

The predicate that checks the subsystem in the case of the engine application, builds the component list by checking that the node belongs to the user specified subsystem. It then checks that the node type slot, also matches the terminal node type specified. The list of components is built by the predicate *build_list*, which recurses on the list of frames that form the fault knowledge base:

```
build_list([],_,_).
build_list([F|Fs],Subsystem,Type,[C|Comps]):-
    node_type(F,Type),
    check_subsys(F,Subsystem,C),
    build_list(Fs,Subsystem,Type,Comps).
build_list([_Fs],Subsystem,Type,Comps):- build_list(Fs,Subsystem,Type,Comps).
```

The *node_type* predicate recurses through the slots of the frame to find the specified node:

```
node_type([],_):- fail.
node_type([S|Ss],Type):- check_it(S,Type).
```

```
node_type([S|Ss],Type):- node_type(Ss,Type).
```

```
check_it(S,Type):- memberchk(Type,S).
```

In the case of the engine, the predicate that checks if the node name slot identifies the specified subsystem is `check_subsys`. It calls `check_type` which in turn calls `check_if_same_subsys`. The subsystem name is converted from a character string to a list representing its equivalent ASCII code. The subsystem ASCII code is then pattern matched against the node name. If the list of subsystem ASCII code is identified as a proper sublist of the node name ASCII code, the node name is returned as a subsystem component. If the predicate fails, the next frame in the list of frames recursed through by the `build_list` predicate is advanced to.

```
check_subsys([Frame1|_],Subsystem,Comp):- check_type(Frame1,Subsystem,Comp).
```

```
check_type([Name_slot|_],Subsystem,Comp):- check_if_same_subsys(Name_slot,Subsystem,Comp).
```

```
check_if_same_subsys(Name_slot,_,_):- Name_slot = [],fail.
```

```
check_if_same_subsys(Name_slot,Subsystem,Comp):-
```

```
    name(Name_slot,N),
```

```
    name(Subsystem,SubN),
```

```
    proper_sublist(N,S:'bN'),           % check if frame is of subsystem
```

```
    Comp = Name_slot.                   % return component name
```

In the case of the coffee maker application, only the node type is checked for a match and the `check_subsys` predicate is commented out.

4.4 Get Component Connections Module

The *get component connections* module is the second module of DR-1 and it uses the components output from the *get components* module to extract connections between nodes in a subsystem. This is achieved by backtracking from the terminal component node to its parent and identifying siblings that share the parent node.

A direct parental link can be extracted by building a list of parents for the sibling components. In the case of the engine application, where a subsystem is used to constrain the grouping of terminal nodes, only those siblings common to the named subsystem are collected.

The inputs to the module are the fault knowledge base of frames and the previously extracted components list. In the case of the engine, the named subsystem is also included to constrain the selection of siblings to those within the subsystem. The output of the module is a nested list of triplets that includes a pair of components and their shared parent.

The parent information is extracted through explicit inheritance knowledge. DR assumes (as described in the assumptions section 3.3.5), that well-structured fault-based knowledge is traditionally organized as a hierarchy (or a directed network) of nodes. These hierarchies are represented by frames with slots that explicitly relate nodes as parents, children, or siblings. In the case of the engine and coffee applications, parent-to-child knowledge is explicitly represented in the child-node-of slot. Sibling component information is also explicitly represented through the parent child-node-ranking slot.

The main module predicate is `get_components_connections`. It calls `check_child_node_of`. This predicate recurses through the list of previously extracted components, and calls `access_child_node_of` to get the parent(s) of a component. Once the parent or parents are identified, the `access_child_node_ranking` predicate collects the respective siblings recursively.

```

get_component_connections(KB,Subsystem,Components,Component_conns):-
    check_child_node_of(KB,Subsystem,Components,Component_conns).

check_child_node_of(.,_,[]).
check_child_node_of(KB,Subsystem,[C|Cs],[CCL|CCLs]):-
    defnode(C,Frame),
    access_child_node_of_slot(Frame,Parents_list),
    access_child_node_ranking(KB,Subsystem,C,Parents_list,CCL),
    check_child_node_of(KB,Subsystem,Cs,CCLs).
check_child_node_of(KB,Subsystem,[_|Cs],CCLs):-
    check_child_node_of(KB,Subsystem,[C|Cs],CCLs).

%% Build up list of parents for each component node
access_child_node_of_slot([],_).
access_child_node_of_slot([FS|_],Parents_list):-
    memberchk(child-node-of,FS),
    del_element(child-node-of,FS,[Parents_list]).

access_child_node_of_slot([FS|FSs],Parents_list):-
    access_child_node_of_slot(FSs,Parents_list).

% Build up list of siblings for each component node
% 'child-node-ranking' slot indicates sibling rankings
access_child_node_ranking(.,_,[],_).
access_child_node_ranking(KB,Subsystem,C,[PL|PLs],Comp_conns_list):-
    build_connection(KB,PL,Subsystem,C,Comp_conns_list),
    access_child_node_ranking(KB,Subsystem,C,PLs,Comp_conns_list).

```

The list of siblings for each component node is built by calling the recursive predicate `build_connection`. `Build_connection` recurses on the list of frames in the knowledge base and checks the parent of a node to see if it matches the name of the parent of the component in the component connections list. The predicate `is_frame_parent` then calls the `siblings` predicate to collect the component siblings. Once the siblings are collected, the predicate `build_sib_con` is called to build a connection between the component siblings and their parent. In this manner, additional siblings may be found. Furthermore, newly found siblings may have multiple parents, and require that their triplet representing a com-

ponent connection, be appended to the list that forms the output of the build_connection list.

```
build_connection([],_,_,_).
build_connection([F|Fs],PL,Subsystem,C,Comp_conecs_iist):-
    is_frame_parent(F,PL,Subsystem,C,Comp_conecs_list),
    build_connection(Fs,PL,Subsystem,C,Comp_conecs_list).
build_connection([F|Fs],PL,Subsystem,C,Comp_conecs_list):-
    build_connection(Fs,PL,Subsystem,C,Comp_conecs_list).

is_frame_parent([FS|FSs],PL,Subsystem,C,Comp_conecs_list):-
    FS = [PL],
    siblings([FS|FSs],[Sibling_list]),
    build_sib_con(Sibling_list,C,PL,Subsystem,Comp_conecs_list).
```

The siblings predicate checks if the slot is the child-node-ranking slot, if it is it returns the list of siblings.

```
% get siblings
siblings([],_).
siblings([S|_],Sibling_list):-
    memberchk(child-node-ranking,S),
    del_element(child-node-ranking,S,Sibling_list).
siblings([S|Ss],Sibling_list):- siblings(Ss,Sibling_list).
```

The predicate build_sib_con recurses through the list of siblings and builds a component connection. In the case of the sibling actually being the component itself, the next sibling is recursed to. For the engine (as described under the assumptions of section 3.3.5) the subsystem is checked to constrain the sibling components to the pre-specified subsystem. As stated earlier, for the get components module in section 4.2, this assumption is relaxed for the coffee application and all device components are used in generating the list of component connections.

```
%%%%%%%%%%%%%% Build C1-C2-connection lists
build_sib_con([],_,_,_).
build_sib_con([SL|SLs],C,PL,Subsystem,Comp_conecs_list):-
    check_if_same(SL,C),
    check_if_same_subsys(SL,Subsystem,Comp), %%%% get_frames module!
```

```

add_element(Comp,[C,PL],Co_co),
add_element(Co_co,Comp_connecs_list,SCL), %%% [C,SL,PL] comp-con
build_sib_con(SLs,C,PL,Subsystem,SCL).

```

```

build_sib_con([SL|SLs],C,PL,Subsystem,Comp_connecs_list):-
    build_sib_con(SLs,C,PL,Subsystem,Comp_connecs_list).

```

```

check_if_same(SL,C):- SL \== C.                %%% ensure diff comps

```

4.5 Get Components Parameters Module

The *get components parameters* module is the third module of DR-1 and it uses the parents of the component modules, identified through the *get components* module, to extract parameters that represent component behaviours.

The inputs to the module include the knowledge base of faults, the previously extracted components list, and the previously constructed component-connections list. In the case of the engine, the named subsystem is used to constrain the search (again, as described under the assumptions of section 3.3.5). In addition, device dependent background knowledge in the form of a list of device parameters shown below is used. This list is used to ensure that the extracted device parameters exist and are valid for the device.

The output of the module is a list of nested pairs of components and respective parameters. In the case of the engine, the components are subsystem components. In the case of the coffee maker, the components are for the full device.

The parameters used as device dependent background knowledge for the gas turbine engine are:

```

% parameters of gas turbine engine
parameters(engine,[N,'Nd','RPM','CDP','P3','EGT','T5','CIT','T2','Pa','PLA',
    'Ta','WF','WFM1','WFT','NPI','Thrust','IGV']).

```

The list of parameters used for the coffee maker device are:

```
parameters(coffee,[water,coffee,power,heat,]).
```

The main predicate in the module is `get_component_parameters` which recursively calls `build_param_list` for each member of the component-connections list and builds another list of paired components with their respective parameters:

```
get_component_parameters(_,[],[]).
get_component_parameters(Param_list,[CC|CCs],[CP_list|CP_lists):-
    build_param_list(CC,Param_list,CP_list),!,
    get_component_parameters(Param_list,CCs,CP_lists).

get_component_parameters(Param_list,[_|CCs],CP_lists):-
    get_component_parameters(Param_list,CCs,CP_lists).
```

The `build_param_list`, recurses through the individual component-connections list, to collect parameters:

```
build_param_list([],_,[]).
build_param_list([CCCL|CCCLs],Param_list,[CP_list|CP_lists):-
    collect_params(CCCL,Param_list,CP_list),!,
    build_param_list(CCCLs,Param_list,CP_lists).

build_param_list(_|CCCLs,Param_list,CP_lists):-
    build_param_list(CCCLs,Param_list,CP_lists).
```

The `collect_parameters` predicate, first makes sure that the individual component-connections list is not empty. The list of extracted parameters for component 1 and component 2 are then combined since both components share the full list of parameters (the parameters having been derived from the names of the shared parent in the component-component connections list). Note that sibling components with multiple parents imply that additional parameters are extracted.

In the case of the engine, if no component parameters are found through parental nodes, the activating-observation slot of the frame may be used to check for a parameter

relation. This slot is too JETA-specific to be useful for DR on general fault knowledge bases. The activating-observation exists as a meta-rule, attached to a node through which parameters on the engine may be checked. This is not as reliable as using the parent information, since not all parental or component frames have activating-observation slots. In the case of the coffee application, this predicate is commented out to illustrate that DR can produce component parameters without meta-rules (such as activating-observation) attached to fault nodes.

```
collect_params([A,B,C],Param_list,PList):-
    check_if_valid(C,Param_list,PList),
    nonvar(PList),
    append(PList,[A],APPList),
    append([B],PList,BPPList),
    defnode(A,Frame),
    access_child_node_of_slot(Frame,Parents_list),
    check_A_list(Param_list,Parents_list,A_params),
    flatten(A_params,A_flat),
    remove_dups(A_flat,A_ps),
    append(A_ps,APPList,A_big_list),
    remove_dups(A_big_list,A_clean_list),
    check_for_activating_observations(B,Param_list,BPPList,BACList), %commented out for coffee maker
    PList = [A_clean_list,BACList].
```

The check_A_list predicate recurses through the list of parent names and extracts any parameters that exist as part of the parent name. It calls check_if_valid to achieve this.

```
check_A_list(_,[],[]).
check_A_list(Param_list,[P1|Ps],[A_param|A_params]):-
    check_if_valid(P1,Param_list,A_param),!,
    check_A_list(Param_list,Ps,A_params).

check_A_list(Param_list,[_|Ps],A_params):-
    check_A_list(Param_list,Ps,A_params).
```

The check_if_valid predicate recurses through the list of device parameters and extracts any parameters that exist as part of the parent name.


```

check_if_valid([], []).
check_if_valid(Param,[P1|Ps],[PLLlist|PLLlists]):-
    name(Param,NParam),
    name(P1,NP1),
    proper_sublist(NParam,NP1),
    P1 = PLLlist,
    check_if_valid(Param,Ps,PLLlists).

check_if_valid(Param,[_|Ps],PLLlists):-
    check_if_valid(Param,Ps,PLLlists).

```

If no parameters are extracted in the engine application, the `check_for_activating_observations` predicate checks for possible parameters. The coffee application does not use this.

```

check_for_activating_observations(B,Param_list,BPPLlist,BACList):-
    length(BPPLlist,1),
    defnode(B,Frame),
    access_activation_slot(Frame,A_slot),
    flatten(A_slot,A_flat),
    look_for_params(Param_list,A_flat,AP_list),
    remove_dups(AP_list,APL),
    flatten(APL,APPL),
    append(BPPLlist,APPL,BACList).
check_for_activating_observations(B,Param_list,BPPLlist,BACList):-
    length(BPPLlist,X),
    X >= 2,
    BPPLlist = BACList.

```

The `access_activation_slot` recurses through the slots of the frame for the activating-observation slot and returns the contents of the slot once it is found.

```

access_activation_slot([],_).
access_activation_slot([FS|_],A_slot):-
    memberchk(activating-observation,FS),
    del_element(activating-observation,FS,A_slot).

access_activation_slot([FS|FSs],A_slot):-
    access_activation_slot(FSs,A_slot).

```

The look_for_params predicate recurses through the activating observation slot contents and extracts any parameters.

```
look_for_params([], []).
look_for_params(Param_list, [A|As], [AList|ALists]):-
    atom(A),
    check_if_valid(A, Param_list, AList),
    look_for_params(Param_list, As, ALists).

look_for_params(Param_list, [_|As], ALists):-
    look_for_params(Param_list, As, ALists).
```

4.6 Device Dependent Background Knowledge

The fifth and sixth modules of the DR algorithm are the device dependent background knowledge used as input to DR-2. One is used for the engine application and the other is used for the coffee maker device application. The Diagnostic Remodeler algorithm uses background knowledge of components based on the input from an expert at NRC's propulsion lab. Device dependent and device independent knowledge can be combined to form a partial domain theory on engine component behaviour. The device dependent knowledge includes:

1. a complete list of parameters for the device (used in get parameters module),
2. glossary information of component and parameter acronyms needed only for the engine application and is derivable from the JETA glossary frames, and
3. in the case of specific components, a black box component model is used where the component is taken as the basic unit of a subsystem (in engine application) or a device (in coffee application) and its input and output parameter behaviours are modeled.

For the engine application, the component models were generated based on the input from an expert at NRC's propulsion laboratory at the Institute for Aerospace Research. These include relational descriptions of device inter-subsystem behaviour,

based on feedback modes. For some, piece-wise linear models of components, with variable control modes, are used so that parameter feedback can be represented.

4.6.1 Engine Device Dependent Background Knowledge

The fourth DR module, the *engine device dependent knowledge* module, is mostly composed of knowledge for the main fuel system. Some additional afterburner and variable exhaust nozzle knowledge is also included so that their respective subsystems may be modelled at some future time if desired.

The module contains four types of knowledge. The first is the list of parameters instantiated in the *get parameters* module of DR-1. The *get parameters* module, as described earlier, extracts valid parameter names from parents (representing behavioural links) of terminal nodes (representing components). In the aircraft engine case, the list of device parameters are:

```
% parameters of gas turbine engine - *** instantiated in get parameters module
parameters(engine,['N1','RPM','CDP','P3','EGT','T5','CIT','T2','PA','PLA','Ta',
                  'WF','WFM1','WFT','NPI','Thrust','IGV']).
```

The second type of knowledge in the module, is also glossary knowledge used to extract component acronym names from JETA nomenclature. For example, for the JETA node *N-SUB-XXX, the component acronym is XXX. The following fact is used to constrain the XXX in a node name of a component extracted in DR-1, and subsequently translated to a meaningful symbolic name in DR-2:

```
% components of gas turbine engine - used for extracting names from JETA nomenclature
components(engine,['FP-FF','MFC-FF','MFC','FP','FN','OSG','PDV']).
components(engine,['AFS'],['ABFC','ABS-FP','AFS-FN']).
```

The third type of knowledge is for parameter names. The engine application has 22 parameters whose acronyms are encoded in JETA. To make DR's extracted component model comprehensible, these meaningful symbolic names were input as facts, so that they may be used in describing parameters for the DR output model:

```
% glossary for engine parameters
glossary('JETA','N',engine_speed).
glossary('JETA','Nd',pilot_demanded_engine_speed).
glossary('JETA','N1',single_spool_engine_speed).
glossary('JETA','RPM',measured_rpm_engine_speed).
glossary('JETA','CDP',compressor_discharge_pressure).
glossary('JETA','P3',compressor_discharge_pressure).
glossary('JETA','EGT',exhaust_gas_temperature).
glossary('JETA','T5',exhaust_gas_temperature).
glossary('JETA','T5B',biased_exhaust_gas_temperature).
glossary('JETA','T5H',unbiased_exhaust_gas_temperature).
glossary('JETA','T5X',trimmed_exhaust_gas_temperature).
glossary('JETA','CIT',compressor_inlet_temperature).
glossary('JETA','T2',compressor_inlet_temperature).
glossary('JETA','RTD',resistance_temperature_detector). % equivalent to T2
glossary('JETA','OAT',outside_air_temperature).
glossary('JETA','Pa',fuel_pump_inlet_pressure).
glossary('JETA','PLA',power_lever_angle).
glossary('JETA','Ta',ambient_temperature).
glossary('JETA','WF',weight_of_fuel_flow).
glossary('JETA','WFM1',main_weight_of_fuel_flow).
glossary('JETA','WFAB',afterburner_weight_of_fuel_flow).
glossary('JETA','WFT',total_weight_of_fuel_flow).
glossary('JETA','A8',nozzle_area).
glossary('JETA','A8d',demanded_nozzle_area).
glossary('JETA','NPI',nozzle_position_indicator).
glossary('JETA','IGV',inlet_guide_vane_position_indicator). % also bleed valve position indicator
```

The fourth type of knowledge in the module is a *device dependent* component behaviour model. This is input as a glossary fact that represents a black box model of the component. This fact identifies:

1. the knowledge base the component is in,
2. the acronym the component is known by,
3. the type of the component,

4. the modes the component has which is essential for control components that have multiple operational modes, and
5. the inputs and outputs related to each operational mode (simple components generally have a single mode whereas more complex ones may have two or more).

The general form of this glossary fact is:

```
glossary(KB,Component_id,Proper_name,[Component_type,Modes],
        [[Possible_inputs_rel_mode],
         [Possible_outputs_rel_mode]]).
```

For the main fuel control (MFC) of the engine main fuel system, 6 operational control modes are possible. For the first mode, `acceleration_fuel_limit_control`, an increase in power lever angle and thus demanded speed causes an acceleration fuel ratio output increase. In the second mode, the `proportional_speed_control`, a small increase in power lever angle (throttle position) causes a proportional fuel ratio output increase. In the third mode, `speed_cutback_control`, an increase in engine speed coupled with a decrease in compressor inlet temperature, below a limit, will cause a decrease in the proportional fuel ratio output. The `variable_geometry_scheduling` mode, implies that an increase in speed is inversely related to inlet guide vane position, and the air bleed valve positions. For the `fru_selection_mode`, the demanded fuel is selected based on the minimum between the acceleration and proportional speed fuel ratios. For the `fuel_flow_selection` mode of the MFC, the demanded fuel is scheduled based on it being above a minimum fuel flow limit (this is necessary to sustain combustion).

```
% main fuel control terms from JETA's Glossary Frames and J85 Control Parameters/Modes
glossary('JETA','MFC',main_fuel_control,
        [control,for,
         {acceleration_fuel_limit_control,
          proportional_speed_control,
          speed_cutback_control,
          variable_geometry_scheduling,
          fru_fuel_selection,
          fuel_flow_selection}],
        [[[ 'N+', 'PLA' ], [ 'acceleration_WF/P3+' ]], %%%% Nd+ -> f(PLA+), WF/P3+ -> f(WF+)
```

```

[['N+', 'PLA+', ], ['proportional_speed_WF/P3+', ]], % PLA+ > Nd+, Nd-N=ajstd WF, +ve fru inc., -ve fru dec.
[['N+', 'T2', 'T2_limit'], ['speed_cutback_T2-']], %%% T2 less than or equal T2_limit -->Nd-
[['N+', 'T2'], ['IGV-', 'bleed valve positions']], %%% IGV kept between min/max values (fully open to closed)
[['acceleration_WF/P3', 'proportional_speed_WF/P3', 'P3'], ['WFd']], %%% reflect selec'n min of 2 modes
[['WFd', 'WF_min'], ['WF_mfc']]). %%% proportional until WF_min

```

The overspeed governor (OSG) has 3 functional control modes. The first is the **overspeed_regulation_run**, which takes as inputs engine speed and overspeed switch position, and generates a reduced fuel flow. In the second mode, the **overspeed_regulation_test**, the engine speed and an engine overspeed test switch, supply reduced fuel flow at a decreased limit for engine speed. The outputs of the first and second modes of the OSG and the OSG_test_switch, are coupled with the fuel flow from the MFC to schedule the main fuel flow.

```

glossary('JETA', 'OSG', overspeed_governor,
[control, for,
[overspeed_regulation_run,
overspeed_regulation_test,
fuel_flow_selection]],
[[['N', 'N_overspeed'], ['WF_overspeed']], %%% N > N_overspeed, (105% N) --> Wf reduced
[['N', 'N_overspeed_test'], ['WF_overspeed_test']], %%% N > N_overspeed_test (95.5% N) --> Wf reduced
[['OSG_test_switch', 'WF_overspeed', 'WF_overspeed_test', 'WF_mfc'], ['WF']]]).

```

The main fuel pump pumps fuel and is regulated by engine speed.

```

glossary('JETA', 'FP', main_fuel_pump,
[pump, for, [fuel_pump_with_speed_regulator]], [[['N+', 'WF'], ['WF+']]]).

```

The pressurizing and drain valve takes the fuel pump inlet pressure, and keeps it above a minimum limit. If the limit is exceeded, it reduces the fuel flow.

```

glossary('JETA', 'PDV', pressurizing_and_drain_valve,
[control, for, [fuel_overflow]], [[['Pa+', 'WF', 'Pa_limit'], ['WF-']]]).

```

The main fuel control filter passes fuel through, and filters out particles.

```

glossary('JETA', 'MFC-FF', main_fuel_control_fuel_filter,
[filter, for, [filtering_fuel]], [[['WF+', 'WF+']]]).

```

Similarly, the main fuel pump filter passes fuel through, and traps particles.

```
glossary('JETA','FP-FF',main_fuel_pump_fuel_filter,  
[filter,for,[filtering_fuel]],[['WF+'],['WF+']])).
```

The main fuel nozzles spray fuel, at the rate it is input.

```
glossary('JETA','FN',main_fuel_nozzles,  
[nozzle,for,[fuel_flow_control]],[['WF+'],['WF+']])).
```

4.6.2 Coffee Maker Device Dependent Background Knowledge

The fifth DR module is the *coffee maker device dependent knowledge*. It includes knowledge for modelling the full device components. In this manner, it was possible to test DR's generality if no device subsystem was specified and all terminal nodes were used to model the device. Only two types of background knowledge were needed since no knowledge base syntax translation was needed. The two types include the device parameter list and the component black box behaviour models.

The coffee maker parameters instantiated in the get parameters module are:

```
parameters(coffee,[coffee,water,heat,power]). % instantiated in get parameters module
```

The device component models are represented by a glossary fact. It includes the name of the fault knowledge base, the name of the component, the component type, the modes and the inputs and outputs to the component. There are 10 coffee maker device dependent component models:

1. The coffee maker plug provides power.

```
glossary(coffee,'coffee maker plug',[regulator,for,[power]],[['power+'],['power+']])).
```

2. The coffee drip is a flow regulator for coffee.

```
glossary(coffee,'coffee drip',[regulator,for,[coffee]],[['coffee+'],['coffee+']])).
```

3. The coffee maker switch is a 2-state (on/off) switch for power.

```
glossary(coffee,'coffee maker switch',[switch,for,[power]],[['on/off'],['power/no power']])).
```

4. The water heating element is a heater for water and it takes power as input.

```
glossary(coffee,'water heating element',[heater,for,[water]],[['power+'],['heat+']])).
```

5. The coffee pot heating element is also a heater.

`glossary(coffee,'coffee pot heating element',[heater,for,[coffee]],[[['power+'],['heat+']]]).`

6. The water reservoir¹ is a holder for water.

`glossary(coffee,'water resevoir',[holder,for,[water]],[[['water'],['water']]]).`

7. The water temperature heat control regulates the flow of hot water.

`glossary(coffee,'water temperature heat control',[regulator,for,[water]],[[['heat+'],['water+']]]).`

8. The coffee holder holds the coffee grounds and passes water through them to generate coffee.

`glossary(coffee,'coffee holder',[holder,for,[grounds]],[[['water+'],['coffee+']]]).`

9. The coffee pot is a holder for coffee.

`glossary(coffee,'coffee pot',[holder,for,[coffee]],[[['coffee'],['coffee']]]).`

10. The coffee filter passes water through coffee grinds to generate coffee.

`glossary(coffee,'coffee filter',[filter,for,[grounds]],[[['water+'],['coffee+']]]).`

4.7 Translate 1 Module: Component Connections Meaningful Text

The sixth DR module is the *translate 1* module. It uses the device dependent knowledge of the component glossary information to translate the JETA acronyms of the component connections list, derived in DR-1, into meaningful symbolic text. This is achieved for both parameters and components. In addition, any duplicates are removed and the refined text is output for DR-2. The general description derived for component-component connectivity is:

```
[[Component1,is_a(Component1_type),
connected_to(Component2),
with_connectivity_parameter(Parameters)],
```

The translate predicate takes as input, the name of the fault knowledge base and the component-connection list. In the case of the engine application, it also takes as input the subsystem name. It then recurses through the list of component connections and decouples the pairs extracted in DR-1.

1. The term 'reservoir' has been misspelled in the DR background knowledge (as 'resevoir') and for consistency has not been corrected in the input or output of the algorithm.


```

translate(_[],_[]).
translate(KB_name,[CC|CCs],Subsystem,[Text_list|Text_lists]):-
    decouple(CC,KB_name,Subsystem,Text_list),!,
    translate(KB_name,CCs,Subsystem,Text_lists).

```

The decouple predicate calls clean_list_of_comps on components 1 and 2, which in turn extracts the component name from the JETA nomenclature. This predicate is commented out in the case of the coffee application. The glossary facts of the respective components are then used to generate the proper meaningful symbolic names and modes of the components. In addition, the list of component-component connections is parsed to the clean_list_of_params to extract the parameter list. The build_description predicate, is then called, with the meaningful symbolic names of the components and the parameters.

```

decouple([C1,C2,P12],KB_name,Subsystem,Text_list):-
    clean_list_of_comps(Subsystem,C1,C1_name),
    clean_list_of_comps(Subsystem,C2,C2_name),
    glossary(KB_name,C1_name,Name1,C1_mode,_),
    glossary(KB_name,C2_name,Name2,C2_mode,_),
    clean_list_of_params(KB_name,P12,PList),
    expand(KB_name,PList,Text_param),
    build_description(Name1,C1_mode,Name2,C2_mode,Text_param,Text_list).

```

In the engine application, the clean_list_of_comps uses the components fact in the device dependent knowledge to extract the acronym of the component from the JETA syntax component name.

```

clean_list_of_comps(Subsystem,C,Clean):-
    components(engine,Subsystem,CompL),
    extract_name(C,CompL,Clean).

```

The extract_name predicate, recurses through the list of component acronyms, and attempts to intersect each one with the JETA syntax component name, until an intersection is found. It then returns the acronym that is a part of the JETA syntax name.

```

extract_name(_[],_).
extract_name(C,[CL_|_],Clean):-
    name(CL,CL_name),
    name(C,C_name),

```

```

proper_sublist(C_name,CL_name),
Clean = CL.
extract_name(C,[_CLs],Clean):-
    extract_name(C,CLs,Clean).

```

The `clean_list_of_params` predicate uses the parameter fact that lists the parameters of the device to extract the name(s) of the parameter(s) from the connection linking the two components. This is used for both the engine and coffee applications.

```

clean_list_of_params(KB_name,P12,List_of_params):-
    parameters(KB_name,Param_list),
    check_if_valid(P12,Param_list,List_of_params).

```

The predicate `expand` recurses through the list of parameters and uses the glossary facts of the respective knowledge bases to generate the meaningful symbolic text for the parameters.

```

expand(_,[],[]).
expand(KB_name,[P|Ps],[PN|PNs]):-
    glossary(KB_name,P,PN),
    expand(KB_name,Ps,PNs).

```

The `build_description` predicate takes as input the meaningful symbolic names of the components, the mode of the first component, and the respective parameters to generate the meaningful symbolic text describing the connection.

```

build_description(Name1,C1_mode,Name2,_,Text_param,Text_list):-
    remove_dups(Text_param,TPs),
    Text_list = [Name1,is_a(C1_mode),
                and_is_connected_to(Name2),
                with_connectivity_parameter(TPs)].

```

4.8 Translate 2 Module: Component Parameters Meaningful Text

The seventh DR module is the *translate 2* module. It also uses the device dependent knowledge of the component glossary information, to form a description of the com-

ponent parameter list derived in DR-1, into meaningful symbolic text. This is the second DR-2 output. The general description derived for the component parameter relation is:

[[Component, has_parameters (Parameters)]

The **translate2** predicate takes as input, the name of the knowledge base, the component parameter list inferred in DR-1, and generates the description text. In the case of the engine application, it also uses the subsystem name to extract the acronyms from the JETA syntax. The predicate recurses through the list of component-parameter pairs.

```
translate2(_,_,[]).
translate2(KB_name,[CP|CPs],Subsystem,[C_P_text|C_P_texts]):-
    double_decompose(KB_name,CP,Subsystem,C_P_text),!,
    translate2(KB_name,CPs,Subsystem,C_P_texts).
```

The predicates **double_decompose** and **ddcouple** separate out the nested pair of lists generated in DR-1.

```
double_decompose(_,_,[]).
double_decompose(KB_name,[CPL|CPLs],Subsystem,[C_P_ts|C_P_text]):-
    ddcouple(KB_name,CPL,Subsystem,C_P_ts),!,
    double_decompose(KB_name,CPLs,Subsystem,C_P_text).
```

```
ddcouple(_,_,[]).
ddcouple(KB_name,[CCPL|CCPLs],Subsystem,[C_P_t|C_P_ts]):-
    decompose_again(KB_name,CCPL,Subsystem,C_P_t),!,
    ddcouple(KB_name,CCPLs,Subsystem,C_P_ts).
```

For the engine application, the predicate **decompose_again** uses the glossary components fact to get a list of the component acronyms, and uses the **extract_name** predicate of the **translate 1** module to get the name embedded in the JETA syntax component name. It then uses the glossary fact to match the acronym with its meaningful symbolic text name. The predicate call to **decompose_again** is commented out in the coffee application since the parameter names are already in meaningful symbolic text form.

As in module *translate 1*, the extracted parameters are converted to their meaningful symbolic text names, using the *expand* predicate. The *describe* predicate then generates the component-parameter relation.

```
decompose_again(KB_name,[C|Parameters],Subsystem,C_P_t):-
    components(KB_name,Subsystem,CompL),
    extract_name(C,CompL,Component),      % defined in Translate 1
    glossary(KB_name,Component,C_name,_,_),
    expand(KB_name,Parameters,P_text),    % defined in Translate 1
    describe(C_name,P_text,C_P_t).
```

The *describe* predicate generates the relation between the component and its parameter(s).

```
describe(C_name,P_text,C_P_text):-
    C_P_text = [C_name,has_parameters(P_text)].
```

4.9 Device Independent Background Knowledge

The eighth and ninth modules of DR, are the *device independent background knowledge* modules for the engine and coffee maker applications respectively. These are used as inputs to DR-2. The DR algorithm uses device independent knowledge as encyclopedic knowledge. This is library component knowledge intended to be the building blocks for devices. This knowledge is general in nature and re-usable for different devices. It includes general descriptions of pumps, filters, controls, nozzles, vessels, etc. One can imagine that such component libraries could be used as inputs to design tools such as those used for CAD (computer aided design) / CAM (computer aided manufacturing) tools in digital circuit design.

The input to output behavioural relations for each component are either, proportional, inverse proportional, or piece-wise linear.

1. Proportional behaviour implies, that for a particular component, as the input increases the output increases proportionately. If the input decreases, the output decreases in the same manner.
2. Inverse proportional behaviour implies, that as a component input increases, the respective component output decreases proportionately.
3. Piece-wise linear behaviour, is used to model components which have multiple modes, with variable inputs and outputs. In a certain mode, an input variable may increase and cause the output to increase to a certain maximum. A second fixed input variable may represent an input maximum. Once the first input reaches this second input maximum, the component output behaviour is adjusted, until an output minimum is reached. This allows the input variable to start rising again. In a manner, this becomes similar to a gradual step function, where the input and output signals are adjusted to maintain a minimum or a maximum. Such modelling is essential for control components such as the main fuel control and the overspeed governor in the engine main fuel system. This is also a simpler, but valid approach similar to that used in the traditional modelling of control components, so that input/output feedback behaviour may be understood [Treager 79].

4.9.1 Engine Device Independent Background Knowledge

Five, generalized engine component models are used in the DR algorithm as device independent background knowledge. For each a component function (function is defined as the goal of a component or its *raison d'être*), input parameter(s), output parameter(s), regulator variable for each mode, and overall component behaviour are generated. The behaviours are either proportional, inverse proportional or piece-wise linear. The component models are:

1. For a pump: a pump delivers fluid and has a control variable that regulates its input-output behaviour in a proportional manner.

%%%% Generalized Description of a pump whose input/output/control variable behaviour is linear component(pump,Pump_name,Fluid,Control,_,F,I,O,R,B):-

```
F = function(Pump_name, delivers(Fluid)),
I = input(Pump_name, fluid(Fluid)),
O = output(Pump_name, fluid(Fluid)),
R = regulator(Pump_name, Control),
behaviour_proportional(Fluid, Control, Behaviour),
B = behaviour(for(Pump_name),
behaviour_is_proportional(Fluid, Control, Behaviour)).
```

2. For a filter: A filter allows an input fluid to pass and has proportional behaviour. It does not have a regulation variable.

%%% filter passes fluid

```
component(filter,Filter_name,Fluid,Control,_,F,I,O,R,B):-
  F = function(Filter_name, filters(Fluid)),
  I = input(Filter_name,fluid(Fluid)),
  O = output(Filter_name,fluid(Fluid)),
  R = regulator(Filter_name,none),
  behaviour_proportional(Fluid,Control,Behaviour),
  B = behaviour(for(Filter_name),
    behaviour_is_proportional(Fluid,Control,Behaviour)).
```

3. For a control: a control component may have several inputs and outputs, and displays variable control function behaviours. It also has several modes, and within specific modes, particular variables act as control variables. To generate the various input and output behavioural relations, the typify predicate described later in the section is called.

%%%%%%%% Generalized Description of a control component with a variable number of inputs and outputs

% whose input/output/control variable(s) behaviour is piecewise-linear

% note that first element of Outputs is main output

```
component(control,Control_name,Inputs,Outputs,Modes,F,I,O,R,B):-
  Outputs = [Main_Output|Outs],
  F = function(Control_name,controls(Main_Output)),
  I = input(Control_name,control(Inputs)),
  O = output(Control_name,control(Outputs)),
  R = regulator(Control_name,regulation_control(Inputs)),
  typify(Inputs,Outputs,Modes,Control_var_list,Behaviours_list),
  B = behaviour(Control_name,
    behaviour_is_piecewise_linear(Control_var_list,Behaviours_list)).
```

4. For a nozzle: a nozzle acts in a manner similar to a filter, in that it simply passes an input on as an output. Its function is flow control.

%%%%%%%% Generalized Description of a nozzle whose input/output/control variable behaviour is linear

```
component(nozzle,Nozzle_name,Flow_var,Control_Var,_,F,I,O,R,B):-
  F = function>Nozzle_name, flow_control(Flow_var)),
  I = input>Nozzle_name,flow(Flow_var)),
  O = output>Nozzle_name,flow(Flow_var)),
  R = regulator>Nozzle_name,regulation_control(Control_Var)),
  behaviour_proportional(Flow_var,Control_Var,Behaviour),
  B = behaviour>Nozzle_name,
    behaviour_is_proportional(Flow_var,Control_Var,Behaviour)).
```

5. For a variable nozzle: a variable nozzle is similar to a control component in that it has variable inputs and outputs, and multiple control modes. For a specific mode, a particular input variable may act as a control variable. The overall component behaviour is piecewise linear.

```

%%%%%%%% Generalized Description of a variable nozzle component with a variable
%%%%%%%% number of inputs and outputs
%   whose input/output/control variable(s) behaviour is piecewise-linear

```

```

component(variable_nozzle, Nozzle_name, Inputs, Outputs, Modes, F, I, O, R, B):-
    Outputs = {Main_Output|Outs},
    F = function(Nozzle_name, ccntrls(Main_Output)),
    I = input(Nozzle_name, control(Inputs)),
    O = output(Nozzle_name, control(Outputs)),
    R = regulator(Nozzle_name, regulation_control(Inputs)),
    typify(Inputs, Outputs, Modes, Control_var_list, Behaviours_list),
    B = behaviour(Nozzle_name,
        behaviour_is_piecewise_linear(Inputs, Main_Output, Outs, Modes,
            Control_var_list, Behaviours_list)).

```

The behaviour_proportional predicate generates a description to indicate that an increase in an input will cause an increase in an output. Conversely, a decrease in the input will cause a decrease in the output.

```

%%%%%%%% Generalized Description of proportional behaviour of a control variable and the resultant output

```

```

behaviour_proportional(Out, Control_Var, Behaviour):-
    Behaviour = {increase_in(Control_Var), increases(Out),
        decrease_in(Control_Var), decreases(Out)}.

```

The typify predicate generates a description for components such as a control or a variable nozzle which are characterized by the behaviour of their inputs and outputs for particular modes. The predicate returns this control variable description:

```

[mode(M), input_control_variable(In)]

```

It also returns a behaviour description that relates the inputs and outputs. The proportional behaviour description is:

```

(the_behaviour_is_proportional_for(C_Mode),
the_input_control_variable(In),
results_in_regulated_output_variable(Out),

```

```

at_limit(In), decrease(Out),
increases(In), increases(Out)))

```

The inverse proportional behavioural description is:

```

[the_behaviour_is_inverse_proportional_for(C_Mode),
the_input_control_variable(In),
results_in_regulated_output_variable(Out),
at_limit(In), decrease(Out),
increases(In), increases(Out))]

```

The typify predicate recurses through the inputs and outputs to derive the control variables and builds a behavioural description as a behaviour list.

```

typify([],[],[],[]).
typify([In|Ins],[Out|Outs],[M|Ms],[Control_var|Control_var_list],[Behaviour|Behaviour_list]):-
    build_relational_description(In,Out,M,Control_var,Behaviour),
    typify(Ins,Outs,Ms,Control_var_list,Behaviour_list).

```

To generate the behaviour, the build_relational_description predicate first identifies the control variable. It then identifies the relation between inputs and outputs. Lastly, it generates the behavioural description.

```

build_relational_description(In,Out,M,Control_var,Behaviour):-
    get_control_var(M,In,Control_var),
    get_relation(In,Out,Relation),
    get_behaviour(Relation,Out,M,Behaviour).

```

The inputs to a control component may have several control variables. If an input has a plus or minus (+/-) sign, it is defined as an input control variable. The get_control_var recurses through the inputs and checks the input sign, if an input has a sign, it is tagged as a control variable.

```

get_control_var(_,[],[]).
get_control_var(M,[In|_],[Cv]):-
    get_sign(In,Sign),
    memberchk(Sign,['+','-']),
    Cv = [mode(M),input_control_variable(In)].

```



```

get_control_var(M,[_|Ins],Control_vars):-
    get_control_var(M,Ins,Control_vars).

```

The `get_sign` predicate intersects the ASCII of the input variable with that of a negative sign or a positive sign, and returns the respective sign to `get_control_var`. If no sign is found, this is also indicated.

```

% check if Input name includes a '-' sign
get_sign(In,Sign):-
    name(In,In_name),
    name('-',Neg_name),
    proper_sublist(In_name,Neg_name),
    Sign = '-'.

```

```

% check if Input name includes a '+' sign
get_sign(In,Sign):-
    name('+',Pos_name),
    name(In,In_name),
    proper_sublist(In_name,Pos_name),
    Sign = '+'.

```

```

% if Input sign is neither +/- return Sign = none
get_sign(_,Sign):-
    Sign = none.

```

The relation information between inputs and outputs is accumulated by the `get_relation` predicate. If both the input and output share the same sign, the relation is proportional. If they have opposite signs, the relation is inversely proportional. The inputs are recursed through with the outputs, to generate relations between each of the individual variables.

```

get_relation([],_,[]).
get_relation([In|Ins],Outs,[R|Relation_list]):-
    check_relation(In,Outs,R),
    get_relation(Ins,Outs,Relation_list).

```

The `check_relation` predicate takes an input variable, and recurses through the output variable list to get the relation. The relation is either proportional or inverse proportional.

```
% check_relation(____,Rel_list).

check_relation(____,[]).
check_relation(In,[O|Os],[Rel|Rel_list):-
    check_individual(In,O,Rel),
    check_relation(In,Os,Rel_list).

% check if the Input/Output signs match
check_individual(In,Out,R):-
    get_sign(In,S_in),
    get_sign(Out,S_out),
    S_in = S_out,
    R = [proportional,In,Out].

% since the Input/Output signs do not match they are inverse_proportional
check_individual(In,Out,R):-
    R = [inverse_proportional,In,Out].
```

The predicate `get_behaviour` takes in the relations, the outputs, the mode and generates individual behaviours. It first calls the predicate `decompose_relation` to get at the specific behaviour between input and output. It then generates a textual description of behaviour.

```
get_behaviour([____,____]).
get_behaviour([R|Rs],Outs,M,[Behaviour|Behaviours]):-
    decompose_relation(R,Outs,M,Behaviour),
    get_behaviour(Rs,Outs,M,Behaviours).

% parse back Relation and respective Input
decompose_relation([____,____]).
decompose_relation([Rel|Rels],Outs,M,Behaviour):-
    Rel = [RelIn,In,____],
    cycle_through_outputs_behaviour(RelIn,In,Outs,M,Behaviour),
    decompose_relation(Rels,Outs,M,Behaviour).

% cycle through Outputs for a particular Input and respective I/O relation
cycle_through_outputs_behaviour(____,____,____).
cycle_through_outputs_behaviour(RelIn,In,[O|Os],M,[BiBehaviour]):-
```

```

get_individual_behaviour(ReIn,In,O,M,B),
cycle_through_outputs_behaviour(ReIn,In,Os,M,Behaviour).

```

The `get_individual_behaviour` description for proportional input-output behaviour is generated by:

```

get_individual_behaviour(proportional,In,Out,C_Mode,B):-
    B = [the_behaviour_is_proportional_for(C_Mode),
         the_input_control_variable(Input_Control_variable),results_in_regulated_output_variable(Out),
         at_limit(Input_Control_variable), decrease(Out),
         increases(Input_Control_variable),increases(Out)].

```

The `get_individual_behaviour` description for inverse proportional input-output behaviour is generated by:

```

get_individual_behaviour(inverse_proportional,In,Out,C_Mode,B):-
    B = [the_behaviour_is_inverse_proportional_for(C_Mode),
         the_input_control_variable(In), results_in_regulated_output_variable(Out),
         at_limit(In), decrease(Out),
         increases(In),decreases(Out)].

```

4.9.2 Coffee Maker Device Independent Background Knowledge

The coffee maker device application has 5 generalized component behaviour models. All the coffee maker models are simple linear models which have proportional behaviour. The component models are:

1. For a regulator: a regulator controls an output based on some input and it exhibits proportional behaviour.

```

%%%%% Generalized Description of a regulator whose input/output variable behaviour is linear
component(regulator,Regulator_name,Input,Output,_,F,I,O,R,B):-
    F = function(Regulator_name, regulates(Input)),
    I = input(Regulator_name,Input),
    O = output(Regulator_name,Output),
    R = regulator(Regulator_name,Input),
    behaviour_proportional(Input,Output,Behaviour),
    B = behaviour(for(Regulator_name),
                  behaviour_is_proportional(Behaviour)).

```

2. For a switch: a switch has a multi-state input. Its input to output behaviour varies according to the state it is in.

```
component(switch,Switch_name,Input,Output,_,F,I,O,R,B):-  
  F = function(Switch_name, has_states(Input)),  
  I = input(Switch_name,input_states(Input)),  
  O = output(Switch_name,output_states(Output)),  
  R = regulator(Switch_name,none),  
  B = behaviour(for(Switch_name),  
    behaves_according_to_state(Input,implies_respective(Output))).
```

3. For a heater: a heater's function is to heat fluid. Its output is proportional to its input.

```
component(heater,Heater_name,Input,Output,Fluid,F,I,O,R,B):-  
  F = function(Heater_name, heat(Fluid)),  
  I = input(Heater_name,is(Input)),  
  O = output(Heater_name,is(Output)),  
  R = regulator(Heater_name,Input),  
  behaviour_proportional(Input,Output,Behaviour),  
  B = behaviour(for(Heater_name),  
    behaviour_is_proportional(Behaviour)).
```

4. For a holder: a holder functions as a vessel for a certain input. It has proportional input to output behaviour.

```
component(holder,Holder_name,Input,Output,_,F,I,O,R,B):-  
  F = function(Holder_name, vessel_for(Input)),  
  I = input(Holder_name,is(Input)),  
  O = output(Holder_name,is(Output)),  
  R = regulator(Holder_name,none),  
  behaviour_proportional(Input,Output,Behaviour),  
  B = behaviour(for(Holder_name),  
    behaviour_is_proportional(Behaviour)).
```

5. For a filter: a filter's function is to filter some material. Its output behaviour is proportional in relation to its input.

```
component(filter,Filter_name,Input,Output,Filter_material,F,I,O,R,B):-  
  F = function(Filter_name, filters(Output)),  
  I = input(Filter_name,filter(Filter_material),fluid(Input)),  
  O = output(Filter_name,fluid(Output)),  
  R = regulator(Filter_name,Input),  
  behaviour_proportional(Input,Output,Behaviour),  
  B = behaviour(for(Filter_name),  
    behaviour_is_proportional(Behaviour)).
```

As in the case of the engine application (section 4.8.1), the proportional behaviour description indicates that an increase in an input implies an increase in an output. Similarly, a decrease in an input implies a decrease in output.

```
behaviour_proportional(In,Out,Behaviour):-
    Behaviour = [increase_in(In),increases(Out),
                decrease_in(In),decreases(Out)].
```

4.10 Match Module

The tenth DR module is *match*. It takes as input the results of DR-1 and generates the final output of DR-2. Thus, it is the main module of DR-2. It calls the translate 1 and 2 modules to generate the meaningful symbolic text of the component connections and the component parameters. A key action that *match* performs is to intersect the knowledge acquired from DR-1, the components and their extracted parameters, with the background knowledge from DR-2. This intersection produces description text for gaps or inconsistencies in the fault knowledge. The final Model that is output is a list that combines the component behaviour and gap lists resulting from the *derive_behaviour* predicate.

```
match_components_to_known_models(KB_name,Susystem,Comp_conecs_list,Comp_param_list,
                                Text_comp_con,Text_comp_param,Model):-
    collapse_list(Comp_conecs_list,Comp_con_list),
    translate(KB_name,Comp_con_list,Subsystem,Text_comp_con),
    refine_description(Text_comp_con,Refined_text),
    nl,write(refined_text_comp_cons),nl,write(Refined_text),
    translate2(KB_name,Comp_param_list,Subsystem,Text_comp_param),
    refine_param_description(Text_comp_param,Refined_param_text),
    nl, write(refined_text_parameters),nl,write(Refined_param_text),nl,
    derive_behaviour(KB_name,Subsystem,Refined_param_text,Behaviour_gaps,Behaviour_list),
    nl,write(behaviour_gaps),nl,write(Behaviour_gaps),nl,
    nl,write(behaviour_list),nl,write(Behaviour_list),nl,
    Model = [Behaviour_list,Behaviour_gaps].
```

The *derive_behaviour* predicate takes in the name of the knowledge base that is used to extract the model and recurses through the list of refined component-parameter

text from *transalte2*. It decomposes the component-parameter description to arrive at a component name and its parameters. In the case of the engine application, it uses the `get_parameters` predicate to get the JETA-encoded parameters (this is not needed in the coffee application, since the parameters are meaningful symbolic names). The glossary fact on the component is then used to extract the device dependent background knowledge which provides the parameters the component should have attached to it. The parameters (extracted from the faults and looked up from the background knowledge), are passed with the respective modes to `decompose_params`. This predicate returns the inputs, outputs and any gaps between the 2 sets of parameters. The `build_behaviour` predicate is then called to generate a description of the behaviour text and gaps.

```

derive_behaviour(____,[]).
derive_behaviour(KB_name,Subsystem,[RePT|RePTs],[Gaps|Behaviour_gaps],[B_text|Behaviour_list):-
    RePT = [C1_name,has_parameters(P_text)],
    get_parameters(P_text,Params),
    glossary(KB_name,_,C1_name,C1_mode,C1_parameters),
    C1_mode = [_,C_modes],
    decompose_params(C1_name,C_modes,C1_parameters,Params,Inputs,Outputs,Modes,Gaps),
    build_behaviour(KB_name,C1_name,_,_,Gaps,B_text),
    derive_behaviour(KB_name,Subsystem,RePTs,Behaviour_gaps,Behaviour_list).

derive_behaviour(KB_name,Subsystem,[_RePTs],Behaviour_gaps,Behaviour_list):-
    derive_behaviour(KB_name,Subsystem,RePTs,Behaviour_gaps,Behaviour_list).

```

The `get_parameters` predicate extracts the meaningful names of the parameters, based on the glossary facts of the device dependent background knowledge.

```

get_parameters([],[]).
get_parameters([PT|PTs],[P|Ps]):-
    glossary(____,P,PT),!,
    get_parameters(PTs,Ps).

```

The `decompose_params` predicate recursively calls the `characterize_param` predicate which builds a description of any gaps.

```

decompose_params(_,[],[],[],[],[]).
decompose_params(C1_name,[C_m|C_ms],[C_p|C_ps],Extracted_Params,
                  [In|Inputs],[Out|Outputs],[Mode|Modes],[Gap|Gaps]):-
    characterize_param(Extracted_Params,C_m,C_p,In,Out,Mode,Gp),
    Gap = [C1_name,Gp],
    decompose_params(C1_name,C_ms,C_ps,Extracted_Params,Inputs,Outputs,Modes,Gaps).

```

The `characterize_param` predicate recurses through the extracted parameters and calls `get_in_out_gap`.

```

characterize_param([],[],[],[],[]).
characterize_param([EP|EPs],C_m,[C_in,C_out],[Inpt|Inpts],[Outpt|Outpts],[C_m|Modes],[Gp|Gps]):-
    get_in_out_gap(EP,C_m,C_in,C_out,Inpt,Outpt,Gp),
    characterize_param(EPs,C_m,[C_in,C_out],[Inpts,Outpts,Modes,Gps]).

```

The first clause of the `get_in_out_gap` predicate, checks if the extracted parameter is a member of the list of device dependent input variables, for the component in the background knowledge. If it is, it generates a description, including the extracted parameter, for the input and uses the background knowledge, to generate an output variable description and returns an empty list for gaps.

```

get_in_out_gap(EP,C_m,C_in,C_out,Input,Output,Gap):-
    memberchk(EP,C_in),
    Input = [input_for_mode,C_m,extracted_input,EP],
    Output = [output_for_mode,C_m,model_output,C_out],
    Gap = [].

```

The second clause of the `get_in_out_gap` predicate, checks if the extracted parameter from the fault knowledge is a member of the output list. If it is, it generates a description including it for the output parameter. It also generates a description for the input parameter using the background knowledge. It again returns an empty list for the gap.

```

get_in_out_gap(EP,C_m,C_in,C_out,Input,Output,Gap):-
    memberchk(EP,C_out),
    Input = [input_for_mode,C_m,model_input,C_in],
    Output = [output_for_mode,C_m,extracted_output,EP],
    Gap = [].

```

The third clause for the `get_in_out_gap` predicate, generates a description for a gap indicating that the extracted parameter was neither an input or output. For the input and output descriptions it again generates descriptions using the background knowledge.

```
get_in_out_gap(EP,C_m,C_in,C_out,Input,Output,Gap):-
    Input = [input_for_mode,C_m,model_input,C_in],
    Output = [output_for_mode,C_m,model_output,C_out],
    Gap = [gap_for_mode,C_m,extracted,EP,input,C_in,output,C_out].
```

The `build_behaviour` predicate calls the component glossary fact to construct the instantiated component model. In the case of components which do not have piece-wise linear behaviour, the first clause is used. In the case of more complex components with a variable number of inputs, the second clause is used since the input and output pairs must be decoupled so that the behaviour text can be generated.

```
build_behaviour(KB,C1_name,_,_,Gaps,B_text):-
    glossary(KB,_,C1_name,[Component_type|_],[[Inz,Outz]]),
    memberchk(Component_type,Simple),!,
    Inz = [Control_var|_], Outz = [Out1],
    component(Component_type,C1_name,Out1,Control_var,_,F,I,O,R,B),
    B_text = [F,I,O,R,B,Gaps].
```

```
build_behaviour(KB,C1_name,_,_,Gaps,B_text):-
    glossary(KB,_,C1_name,[Component_type,_,M],In_outs),
    flatten(M,Modes),
    memberchk(Component_type,[control]),
    decouple(In_outs,Ins,Outs),
    component(Component_type,C1_name,Ins,Outs,Modes,F,I,O,R,B),
    B_text = [F,I,O,R,B,Gaps].
```

```
decouple([],[],[]).
decouple([IO|IOs],[I|Ins],[O|Outs]):-!
    IO = [I,O],
    decouple(IOs,Ins,Outs).
```


5.0 Results of Diagnostic Remodeler (DR) Implementation

5.1 General Results and Testing of DR Assumptions

The DR algorithm was successfully tested with two very different applications. The first, was the use of DR to model a subsystem of a complex device, an engine Fuel Control System. This application was selected since it required that DR be used on already existing complex real-world fault knowledge. The resultant black box component model was validated by a domain expert of the Propulsion Laboratory at NRC's Institute for Aerospace Research. The second, was the application of DR to model a full device, for a simpler application, a coffee maker. This second application was selected to demonstrate that DR can be re-used for modelling other fault knowledge. The more simple coffee maker application also allowed DR to be tested for modelling a full device thus using all terminal nodes (rather than a specific type of terminal node, used to constrain the engine application). Finally, the coffee maker application is one that can be more easily related to by other AI researchers and thus, its model could be more easily evaluated. The results of the second model were verified by comparing the expected model and its respective behaviours for a coffee maker device with those that DR generated. DR generated a model that accurately reflected the expected model.

As described in chapter 3.0, the DR algorithm makes these two key assumptions:

1. Terminal nodes in a fault knowledge base can be identified and represent components of the device whose fault knowledge is represented.
2. Movement up the diagnostic hierarchy (or directed network), in the fault knowledge to the root node, as shown in Figure 5.1, is a generalization of symptomatic knowledge on device components (terminal nodes), to operational mode faults which in turn can be generalized to phases of device operation faults.

The first assumption was relaxed from the identification of a specific terminal node (a replace node in JETA's engine knowledge), to the identification of all terminal nodes (both repair and replace nodes in the coffee application). For complex devices, with many components (more than 10), it is better to constrain DR to model subsystems of the device. In this manner, the black box component model descriptions output by DR are understandable. Even in the case of the coffee maker device, which has 10 components, the black box notation (shown for the component models of the two applications in Figures 5.2 and 5.3), was found easier to follow than the raw output (for raw output, see Appendix A.3 for engine and Appendix B.2 for coffee maker). The engine application generated 18 connections between its initial 4 replace terminal nodes. Through the parent node information from the connections, 3 more terminal nodes were found to be sibling components to the initial 4. These formed the 7 components output as the engine Main Fuel System (MFS) model. The coffee application generated 47 connections between its 10 terminal nodes, the connections extracted increase significantly with an increase in the number of terminal nodes. The engine fault knowledge includes 78 terminal nodes, generating a full device model without constraining it to subsystems would generate DR output that would be very difficult to understand.

To apply the DR algorithm for the modelling of the full engine device represented by the 78 terminal node components would have required that the full background knowledge, describing the 78 engine components, be generated. The full generation of this knowledge would have further evaluated the scalability of the DR algorithm, but would not have been any more revealing in evaluating the hypothesis behind the DR algorithm. The hypothesis of DR is that fault-based knowledge with component background knowl-

edge can be used to automatically generate black box component model descriptions, and behaviours linking them.

Figure 5.1. Layers of knowledge in JETA showing the operational modes missing layer

One of the key objectives of DR was to determine if the component models output by the algorithm could be used to identify gaps or inconsistencies in the fault knowledge. These gaps or inconsistencies would be interpreted as novel faults, faults not originally encoded in the fault knowledge base. Gaps and inconsistencies were identified in both the engine and the coffee maker application (described in section 5.5). These identify novel faults in the engine knowledge and were validated by the domain expert. The gaps and inconsistencies identified in the coffee maker fault knowledge were based on deliberately omitted or deliberately encoded fault knowledge which was used to test the premise that the DR algorithm could uncover them.

The second assumption of the DR algorithm is key in discovering the relation between the components and the various layers of knowledge above them, thus identifying any gaps or inconsistencies. Specifically, in the engine fault knowledge, it is clear that a

significant layer of knowledge is missing. If one examines the knowledge extracted by DR, the lowest layer of knowledge (represented by the terminal nodes in Figure 5.1), is the component knowledge. The layer above that knowledge is, as assumed by DR, symptomatic knowledge that maps directly to component parameters. These parameters represent model behavioural variables. Some minor inconsistencies were found in JETA at this level. However, the most significant discovery is the layer of missing knowledge above the symptomatic knowledge for components that have multiple functional modes in JETA. These are specifically control components (e.g. the Main Fuel Control, the Overspeed Governor, etc.). These components have associated with them functional modes with a different number of respective control variables. Thus, in a specific mode, a fault in JETA may be manifested and it would be indicated by some variable. In another mode, a completely different variable may be the indicator of a fault with the component. At a higher level in the JETA hierarchy, the functional modes are related to phases of engine operation. However, the current JETA fault knowledge does not explicitly relate these component operational modes to the phases of engine operation. Given the DR acquired component models, it would not be very difficult to add this new layer of knowledge above the component symptom layer, and explicitly relate it to the phases of engine operation. Some of the missing knowledge may be implicit in the advice and instructions that JETA gives the technician performing the diagnosis. However, the knowledge in JETA's advice information slot is neither explicit nor consistent.

5.2 Impact of Background Knowledge in relation to Fault Knowledge

Two types of background knowledge are needed to achieve the DR algorithm results (described in detail in section 3.6). The first type of knowledge is device dependent background knowledge (DDBK). This knowledge is in the form of a component-dependent description. It provides glossary knowledge, mapping the fault-based encoded name of the component to a meaningful symbolic name (this was necessary for the engine application to decode JETA syntax, but was not used for the coffee application which encoded meaningful symbolic text in its fault knowledge). The DDBK also represented any component-specific modes of operation and their respective input/output variables with a plus/minus (+/-) sign indicating a direction for any changes in value.

The second type of background knowledge is device independent background knowledge (DIBK). This knowledge is in a form of a generic description for a component type that gets instantiated according to the extracted component types that model a device. These generic component descriptions are designed to be placed in a design or modelling library so that they may be re-used for different devices. Their description includes a function (the purpose or goal of the component, e.g. to pump, to control, to act as a source, to filter, etc.), inputs, outputs, regulation inputs, and a behavioural relation describing how the inputs change with respect to the outputs for particular modes of operation (proportional, inverse proportional or piece-wise linear).

The statistics comparing the quantity of background knowledge (BK) to fault-based knowledge (FBK) used by DR is in Table 8. The total fault knowledge used for the engine application is 3972 lines of code or 197 fault nodes. Out of this fault knowledge, in

modelling the Main Fuel System (MFS), approximately 60 fault nodes are used. The average number of lines of code per node is 20. Thus, the DR algorithm uses 1200 lines of JETA fault knowledge code, to model the MFS components and connections. DR also uses 168 lines of code of total background knowledge, of which 101 is device dependent (DDBK) and 67 is device independent (DIBK). Thus, the ratio of background knowledge to total fault knowledge is only 4.23%, and to fault knowledge used by DR for modelling the MFS is 14%.

Similarly, for the coffee application, 330 lines of fault knowledge code or 29 fault nodes are used by DR. The total number of lines of code of background knowledge used for modelling the device are 91, of which 35 are device dependent and 56 are device independent. Thus, the ratio of background, to total fault knowledge used by DR for modelling the full coffee maker device is 28%.

Table 8. Statistics: background knowledge (BK) versus fault-based knowledge (FBK) used by DR

	FBK (total lines of code)	FBK (total number of nodes)	DR (number FBK nodes used)	DR lines of FBK used (20 lines of code per node)	BK (total lines of code)	DDBK (lines of code)	DIBK (lines of code)	ratio BK/FBK (total lines of code)	ratio BK/FBK (used lines of code)
engine	3972	197	~60	1200	168	101	67	4.23%	14%
coffee	330	29	29	330	91	35	56	28%	28%

It is interesting to note that as the application gets more complex (i.e. the engine application), the ratio of total background knowledge to fault knowledge is smaller. What is more significant is that the background knowledge used by DR for modelling a subsystem for a complex application requires 50% less (14% versus 28% in table 8), than for a simpler application. This further justifies the hypothesis that DR works well in modelling complex fault knowledge bases (such as that of the engine), whereas its use may not be justified for simple fault bases. This use can also be qualified by the level of effort

required to encode the background knowledge. Admittedly however, formulating the background knowledge required to model a simple device is far easier than that required to model a complex device. This is supported by the fact that a domain expert participated in ensuring that the engine background knowledge was accurate, whereas no specialized expertise was required to formulate the background knowledge for the coffee maker device. However, the resultant black box component models with explicit descriptions of behaviour and function are far easier to understand than reading the fault knowledge in a complex application. The same may not be necessarily true for the simpler application, since its models were not constrained to subsystems and as a result, the raw output reflected a far greater number of behaviours between components that require a graphical, rather than a textual description for better comprehension. However, if the simpler device modelling was restricted to a smaller number of components (for example, 5 rather than 10), the DR output would more likely be easily understood.

One interesting question in the use of the DR algorithm is: *how gracefully will the component models degrade, if pieces of the background knowledge were omitted?*

To answer this question, it is important to first identify and categorize what the device dependent (DDBK), and device independent (DIBK), background knowledge descriptions provide. Table 9, lists 11 descriptors for device dependent, and 6 descriptors for device independent, background knowledge (BK). They are organized from the minimum, to the maximum BK useful to the DR algorithm. The minimum BK required for the DR algorithm is the list of valid device parameters (descriptor 1) of the DDBK and *no* DIBK. Descriptor 1 is required as input to DR-1 so that the connections extracted as the shared parents of the component terminal nodes can be decoded to identify the parameters

that link the components. Omitting descriptor 1 from the DDBK, implies that no links would be extracted from the parent names that are shared by sibling components.

Table 9. Explicit device dependent and device independent background knowledge descriptors

	Device Dependent Background Knowledge	Device Independent Background Knowledge
1	list of valid parameters for full device	component type
2	list of valid device component identifiers	explicit component function
3	fault knowledge base name	inputs related to component mode of operation
4	component identifier	regulating inputs related to component mode
5	component name	outputs related to component mode of operation
6	component type	behaviours relating inputs and outputs
7	modes of operation related to component	—
8	inputs related to component mode of operation	—
9	change of value (+/-), in input related to mode	—
10	outputs related to component mode of operation	—
11	change of value (+/-), in output related to mode	—

If DDBK descriptor 1 is omitted then only the terminal node names (encoded identifiers in the engine application, and meaningful symbolic names in the coffee application), would be returned. From the terminal node names it is possible to identify using the DDBK, the component type. Descriptors 2, 3, and 4, are used by DR-2 to decode the component name so that a pattern match can be performed with the respective DDBK of a component. Thus, if descriptor 1 is omitted, but descriptors 2 to 4 are not, the DDBK for a component can be output. This DDBK could *not* be used to check what links are missing from the fault knowledge, since in omitting descriptor 1, no fault knowledge links are identified. The DDBK would simply list the component name, its type, its inputs and outputs. These in turn form descriptors 5 to 11.

Thus, *without* any DIBK and with DDBK descriptors 2 to 4, DR can generate DDBK descriptors 5 to 11. Any of descriptors 5 to 11 can be dropped out of the DDBK, and

the DR output would simply reflect their absence. For example, if descriptors 5 or 6 were dropped, then no respective component name or type would be output. Similarly, if descriptors 7, 8 or 9 were dropped, then the respective component modes, inputs, or outputs, would be missing from the DDBK expression output by DR.

If *no* DDBK were included in DR, then *no* DIBK descriptors that form the output expressions of DR would be generated. If only descriptors 2 to 4 of the DDBK are input to DR, then the component type (descriptor 6 in the DDBK, and descriptor 1 in the DIBK), could be identified. From descriptor 1 in the DIBK, the function of the component (descriptor 2 of the DIBK) could be identified. This is the *minimum* DIBK that could be output based on the availability of the *least* descriptors from the DDBK.

The other descriptors in the DIBK would be output based on their presence in the DDBK. Thus, if descriptors 2 to 4, 6, 7, 8, 9, and 10 of the DDBK were added individually, their reflection would be respectively expanded DIBK expressions. These DIBK expressions would include component type (from DDBK 2 to 4), function (from DDBK 6), inputs (from DDBK 8), regulating inputs (from DDBK 8 and 9), and outputs (from DDBK 10). If the sign descriptors (DDBK 9 and 11) are omitted from the BK, then neither the regulating inputs nor the behaviours descriptors DIBK 4 or 6 could be output. The DIBK behaviour descriptor 6 requires that the DDBK descriptors 8 to 11 be present.

Thus, with a minimum amount of DDBK (the list of device parameters are 1 line of code) and the fault knowledge as input, DR can still have a useful output, namely the links between the component terminal nodes. As descriptors are added to the DDBK, the descriptors output by the DIBK increase. DIBK in turn makes the output of DR more expressive. For a complex application, DR's formulation of background knowledge is thus

quite worthwhile. DR can produce black box component models that can assist someone in understanding, verifying, and if necessary, expanding the fault knowledge to include any gaps or corrections to inconsistencies.

5.3 DR-1 Output

5.3.1 Engine Application

An analysis of the JETA fault knowledge shows layers of knowledge represented as a directed network (see Appendix A.1), which can be reduced to leaves of diagnostic trees. The topmost layer is an entry point to jet engine faults and subsequent layers organize the faults into various branches. The phases of operation branches lead to various symptomatic nodes labelled as *snags*. These snags in turn are refinable down to repair and replacement nodes which represent the terminal nodes of the diagnostic hierarchy¹. If one examines the knowledge encoded in these terminal nodes more closely one discovers that they represent faults directly on physical engine components. These physical component fault nodes can be grouped into those affecting one of thirteen subsystems by their nomenclature. One can follow the steps of the DR algorithm to extract in DR-1 the components, their connections and respective parameters for the main fuel system of the aircraft engine.

5.3.1.1 Extracted Components

As described in chapters 3.0 and 4.0, the first step of DR is to extract the terminal nodes. In the case of the aircraft engine, the replace node-type slot is found for the frames

1. The diagnostic hierarchy is sometimes referred to as a network, since it includes relations that are not directly inherited. These allow the JETA reasoner to jump around between nodes, thus forming more of a directed network than a hierarchy.

of the main fuel system (MFS). This outputs 4 MFS components, specifically, the main fuel pump, the main fuel control, the overspeed governor and the pressurizing and drain valve. The output of the get component module in JETA nomenclature is:

components	
[*N-MFS-FP-R,	% ² main fuel system pump
*N-MFS-MFC-R,	% main fuel system fuel control
*N-MFS-OSG-R,	% main fuel system overspeed governor
*N-MFS-PDV-R]	% main fuel system pressurizing and drain valve

5.3.1.2 Inferred Component-to-Component Connections

The second module of DR-1 is *get component connections*. It uses the list of extracted components to search for parental nodes that link components. It is possible to extract additional components in this step since parental nodes of components may lead to siblings that are not necessarily of the replace node-type. This is illustrated below since additional components such as the main fuel pump fuel filter (*N-MFS-FP-FF), the main fuel nozzles (*N-MFS-FN), and the main fuel control fuel filter (*N-MFS-MFC-FF) have connections to the already extracted components. The components are related by fuel flow (WF), by indicated engine speed (RPM), actual engine speed (N1), exhaust gas temperature (EGT) and parents which do not explicitly encode parametric knowledge (*N-VGS-BV-CL, *N-S-FLOUT).

The extracted connections are grouped according to the components from which the parental nodes are found. Thus, the first group is from the main fuel pump (MFP), the second is from the main fuel control, the third is from the overspeed governor, OSG, (note that it has no parental nodes explicitly related to its sibling components - this is indicated

2. '%' is the comment operator in Prolog.

by the variable output instead of the connection for the third nested list), and the fourth is from the pressurizing and drain valve (PDV) component. Note that the third variable in the list is a parent to the terminal nodes and its JETA-encoded name may include GT (greater than), LE (less than), OK, NOK (not ok), etc. in the name. This knowledge is not used in the DR algorithm and was a naming convention selected by the JETA knowledge engineer to indicate tests that a node may represent.

component_connections³

```
[[[*N-MFS-MFC-R,*N-MFS-FP-R,*N-WF-LOSS]_],
[ [*N-MFS-FP-R,*N-MFS-MFC-R,*N-WF-LOSS],
[ *N-MFS-MFC-FF,*N-MFS-MFC-R,*N-WF-DECR],
[ *N-MFS-FP-FF,*N-MFS-MFC-R,*N-WF-DECR],
[ *N-MFS-OSG-IL-12,*N-MFS-MFC-R,*N-RPMH-IDLE-WF-OK],
[ *N-MFS-FN,*N-MFS-MFC-R,*N-RPMH-IDLE-WF-OK],
[ *N-MFS-FN,*N-MFS-MFC-R,*N-RPMH-IDLE-WF-NOK],
[ *N-MFS-OSG-TL,*N-MFS-MFC-R,*N-RPMH-N1-GT-3],
[ *N-MFS-OSG-IL-12,*N-MFS-MFC-R,*N-RPMH-N1-GT-3],
[ *N-MFS-OSG-CHK-1,*N-MFS-MFC-R,*N-RPMH-N1-GT-3],
[ *N-MFS-FN,*N-MFS-MFC-R,*N-RPMH-N1-GT-3],
[ *N-MFS-OSG-CHK-1,*N-MFS-MFC-R,*N-RPMH-N1-LE-3],
[ *N-MFS-OSG-IL-12,*N-MFS-MFC-R,*N-RPMH-N1-LE-3],
[ *N-VGS-MFS-MFC-R,*N-MFS-MFC-R,*N-VGS-BV-CL],
[ *N-MFS-OSG-IL-12,*N-MFS-MFC-R,*N-WF-LOW],
[ *N-MFS-PDV-CHK-STRT,*N-MFS-MFC-R,*N-WF-LOW],
[ *N-MFS-FP-R,*N-MFS-MFC-R,*N-WF-LOW]_], _],
[[*N-MFS-FN,*N-MFS-PDV-R,*N-S-FLOUT]_]]
```

5.3.1.3 Extracted Component Parameters

The component connection lists are used to generate pairs of component-parameter relations constrained by the valid list of device parameters. In some cases if no parameters are found, the parent nodes are again examined for encoded parameters. Such is the case for the main fuel nozzles to pressurizing and drain valve connection where the node (*N-S-FLOUT) leads to the parameters exhaust gas temperature(EGT), engine speed (RPM

3. The '_' character is a variable in Prolog, in this case a tail list variable which appears as _NNNNNN, where N is a number between 0 and 9, in the raw output shown in Appendix A.3.

is sensed engine speed, and N1 is actual), and fuel flow(Wf). In addition, in the case of the PDV, the activation observation is examined for parameter relations since no parental nodes lead to parameters. From the activation observation, the fuel pump inlet pressure (Pa), is extracted. Upon analysis it is evident from the connections (especially the fuel nozzles) that some of the higher level parents hold the symptomatic or parameter knowledge while in one other component (specifically the PDV) the parametric knowledge is embedded in the activating conditions. Note that component parameters are grouped in the translate 2 module (described in section 5.3.2.3) and redundancies removed.

component_parameters⁴

```
[[ [ *N-MFS-MFC-R,N1,RPM,Wf],[ *N-MFS-FP-R,Wf]], ...
[ [ *N-MFS-MFC-FF,Wf],[ *N-MFS-MFC-R,Wf]],
[ [ *N-MFS-FP-FF,Wf],[ *N-MFS-MFC-R,Wf]],
[ [ *N-MFS-OSG-IL-12,N1,RPM,Wf],[ *N-MFS-MFC-R,RPM,Wf]],
[ [ *N-MFS-FN,EGT,N1,RPM,Wf],[ *N-MFS-MFC-R,RPM,Wf]], ...
[ [ *N-MFS-PDV-CHK-STRT,Wf],[ *N-MFS-MFC-R,Wf]], ...
[[ [ *N-MFS-FN,EGT,N1,RPM,Wf],[ *N-MFS-PDV-R,Pa,Wf]]]]
```

Figure 5.2 shows the graphical black box component models for each of the connections. These provide more of an idea of what the extracted component connections signify. The first shows the main fuel control (MFC), connected to the main fuel pump (FP), through the fuel flow (Wf) parameter. Connections two to four show the MFC connected to the main fuel control fuel filter (MFC-FF), the main fuel pump fuel filter (MFP-FF), and the pressurizing and drain valve (PDV), through the fuel flow parameter. Connection five shows the MFC connected to the main fuel nozzles (FN) through the sensed engine speed (RPM) and the fuel flow parameters. It also shows that FN has an exhaust gas temperature (EGT) and the actual speed (N1) parameter. Connection six shows the main fuel control

4. The full listing of component_parameters is in Appendix A.3.

connected to the overspeed governor through sensed (RPM) and actual (N1) engine speed parameters as well as the fuel flow parameter. Connection seven shows the main fuel nozzles connected to the pressurizing and drain valve through the fuel flow parameter. It also shows PDV as having the inlet pressure parameter (Pa) and FN having the EGT and engine speed parameters RPM and N1.

Figure 5.2. Black box view of component connections for engine application

5.3.2 Coffee Maker Application

To test the generality of the DR algorithm and relax the assumption that a subset of the terminal nodes (relevant to a subsystem) be used rather than the full set (relevant to the entire device), a 29-node knowledge base for the diagnosis of a coffee maker was generated (a very different device than an aircraft engine). The coffee maker device had a variety of terminal nodes (not only replace types). DR selected all terminal nodes and

assumed that they represent device components. Then, as before parental nodes were used to identify sibling nodes and connections between them. Thus, only 3 of the node slots of the fault-based frame knowledge were used, the node name, the child node of, and the child node ranking slots.

5.3.2.1 Extracted Components

Ten components were extracted from the full coffee maker device fault knowledge which is input to the *get components module*. The components had a 1-1 mapping with the terminal nodes since all terminal nodes were assumed to represent components. Thus, the output of the *get components module* for the coffee application is:

```
components
[coffee drip,
 coffee filter,
 coffee holder,
 coffee maker plug,
 coffee maker switch,
 coffee pot,
 coffee pot heating element,
 water heating element,
 water reservoir,
 water temperature heat control]
```

5.3.2.2 Inferred Component-to-Component Connections

As in the case of the engine application, DR backtracks from each terminal component to identify its parent nodes, and subsequently, any terminal node siblings it may have (see description in section 4.3). These parent nodes are assumed to encode the symptomatic and hence component parametric knowledge. Ten sets of component connection lists were generated, thus mapping to the parental knowledge of the ten components extracted in the first module. Redundancies in component connections are collapsed in the translate

1 module. Forty seven connections are generated, and fourteen of them are shown below. For this explanation they have been re-grouped according to their shared parent node (for the raw output, refer to Appendix B.2). The subset of output of the *get component connections* module for the coffee maker is described below.

component_connections⁵

The first set of terminal nodes share the parent 'coffee taste too strong'. This parent is at the component symptom layer of the fault knowledge (see coffee hierarchy in section 3.8.2.2, figure 3.6). The parent is shared by three sets of siblings. These sets of siblings are coffee (filter,holder), (drip, holder), and (pot,drip). This implies that the taste of the coffee being too strong is directly related to a problem with the filter (perhaps the filter is the wrong type for the coffee?), holder (perhaps it has too many coffee grounds in it?), or the coffee pot (perhaps the coffee pot has grounds in it?).

```
[[[coffee filter,coffee holder,coffee taste too strong],...  
[coffee drip,coffee holder,grounds blocking coffee drip],...  
[coffee pot,coffee drip,grounds blocking coffee drip]]],...
```

The second set of terminal nodes are related by the 'heat problem: water too hot parent'. Two sets of siblings share this parent, namely (water heating element, water reservoir) components, and (water temperature heat control, water reservoir) components. This implies that these components are directly affected by a problem with heating the water.

```
[water heating element,water resevoir,heat problem: water too hot],  
[water temperature heat control,water resevoir,heat problem: water too hot]]],
```

5. This is a partial listing of the component_connections, the full list is included in Appendix B.2.

The third set of terminal nodes are related by the 'no power' parent. The siblings are the (plug, switch) component nodes. These components are directly affected by a 'no power' problem.

[[coffee maker plug,coffee maker switch,no power],

The fourth set of terminal nodes are related by the parent, 'heat problem: water too cold'. There are five sets of siblings that directly share this parent. These identify four specific components, switch, plug, water heating element, and water temperature heat control. As expected, these components would all create a problem in heating the water if they were faulty.

[coffee maker switch,coffee maker plug,heat problem: water too cold],
[water heating element,coffee maker switch,heat problem: water too cold],
[water heating element,coffee maker plug,heat problem: water too cold],
[water temperature heat control,coffee maker plug,heat problem: water too cold],
[water temperature heat control,coffee maker switch,heat problem: water too cold],

The fifth set of component connections share another heat problem, namely, 'coffee too cold'. Three components, plug, switch, and coffee pot heating element, are all affected by a heat problem which causes the coffee to become too cold.

[coffee maker switch,coffee maker plug,heat problem: coffee too cold],
[coffee pot heating element,coffee maker plug,heat problem: coffee too cold]_|,...
[coffee pot heating element,coffee maker switch,heat problem: coffee too cold]_|,

5.3.2.3 Extracted Component Parameters

From the component connections and constrained with the list of valid parameters for the device, the parameters are inferred. The parameters represent parents that encode symptomatic, and as a result parametric knowledge. Parameters are grouped for individual components and redundancies removed in translate 2. The forty seven component connec-

tions in turn generate forty seven lists of nested pairs, that show the component name and its respective parameters (for the full raw output see Appendix B.2). From the list of connections, it is possible to generate twenty explicit pairs of components with shared behaviours. From the fourteen connections described in the previous section (5.3.2.2), twelve sets of component pairs with parameters relating them can be extracted. Figure 5.3 shows the respective black box component models of the connections extracted from the shared parents, and described below.

component_parameter_pairs

From the three sets of siblings that shared the parent 'coffee tastes too strong', a nested list of three pairs is extracted. The valid coffee maker device parameter identifiable from the shared parent is 'coffee'. Thus the filter, holder, drip and pot, all have the behavioural parameter 'coffee' (note that this term is taken to be the fluid output by the device). In addition, the coffee pot component had multiple parents, (including 'heat problem: coffee too hot'), this results in pot also having a parameter, heat, associated with it.

```
[[[coffee.coffee filter],[coffee holder,coffee]],
 [[coffee,coffee drip],[coffee holder,coffee]],
 [[coffee,coffee pot,heat],[coffee drip,coffee]]].
```

The second shared parent is 'heat problem: water too hot'. This parent results in the extraction of the 'heat' and 'water' parameters that directly affect the water heating element, water reservoir, and the water temperature control.

```
[[heat,water,water heating element],[water resevoir,water,heat]],
 [[heat,water,water temperature heat control],[water resevoir,water,heat]]].
```

Figure 5.3 Black box view of some component connections for coffee maker device application

The third parent is 'no power' and it results in the parameter 'power' being extracted for the plug and switch components. It is interesting to note, that both have multiple parents which lead to the extraction of coffee, water and heat, as additional behaviours from direct parents (specifically, the parents previously introduced, 'heat problem: water too cold', and 'heat problem: coffee too cold'). These indirect parents were deliberately introduced into the fault knowledge, to investigate their effect at being introduced at the component symptom layer, versus the operational mode layer. For DR not to extract them as direct behaviours, they should be introduced at a higher layer that would typify, modes of operation of the coffee maker device. Such modes could be, for example 'wrong water temperature problems' (similar to the 'wrong coffee temperature' problem shown in the fault hierarchy of figure 3.6). Note that the problem with the coffee temperature

(which causes the coffee parameter to be extracted), was introduced as a direct symptom to the switch, and plug components, to investigate the results by DR.

```
[[[coffee,coffee maker plug,heat,power,water],[coffee maker switch,power]],  
[[coffee,coffee maker switch,heat,power,water],[coffee maker plug,power]]].
```

The fourth set of connections from the shared parent, 'heat problem: water too cold', results in the extraction of, water, and heat, parameters for the water heating element, water temperature heat control, switch, and plug components.

```
[[heat,water,water heating element],[coffee maker switch,water,heat]],  
[[heat,water,water heating element],[coffee maker plug,water,heat]],  
[[heat,water,water temperature heat control],[coffee maker switch,water,heat]],  
[[heat,water,water temperature heat control],[coffee maker plug,water,heat]].
```

The fifth set of connections from the shared parent 'heat problem: coffee too cold', results in the generation of the coffee and heat parameters for the coffee pot heating element, switch and plug components.

```
[[coffee,coffee pot heating element,heat],[coffee maker switch,coffee,heat]]],  
[[coffee,coffee pot heating element,heat],[coffee maker plug,coffee,heat]]],
```

5.4 DR-2 Output

5.4.1 Engine Application Resultant Model

Main Fuel System Model

The partial view of the extracted MFS subsystem is shown in Figure 5.4. This diagram is not produced by DR (but certainly can be in future work), and it maps the expected component descriptions and their links. Note that the main fuel pump filter and main fuel control filter extracted by DR were omitted to simplify the diagram. Thus, DR

succeeds in extracting the 7 components (5 shown excluding the 2 filters and the fuel tank which is not a component in the fault knowledge but is shown for clarity) and their respective connections in DR-1. In DR-2, the device dependent and device independent background knowledge is used to derive the direction and relations between the extracted parameters. Any gaps between JETA and the background knowledge are highlighted, as described in 5.4 so that the fault-based knowledge can be made consistent or modified.

Figure 5.4. The Main Fuel System model expected by the DR algorithm output

5.4.1.1 Meaningful Text: Component Connection Relations

A topological network is the basis for the hypothesized component-component relations that are generated as described in chapters 3.0 and 4.0. These relations are directly extracted from the JETA fault knowledge. It is output in meaningful symbolic text by the translate 1 module of DR-2. Sibling nodes are clustered based on shared parent

links. Note that no parameter relating the main fuel nozzles and the pressurizing and drain valve can be found from their respective parental information. The rest of the components have expected relations that identify the parameters between them. One notable gap at this point is the main fuel pump to main fuel control connection which should include engine speed (N), in addition to the extracted fuel flow (WF) parameter.

Another interesting inconsistency is that engine speed (N), is extracted as a main fuel nozzles control variable. In the operation of the engine, the fuel nozzles simply spray fuel. In detecting the failures of the fuel nozzles, the technician checks the nozzles at different engine speeds (idle or not idle speed). This is confirmed by checking the JETA fault knowledge sub-hierarchy in figure 3.4. From the figure, one can see that a refinement of RPM hangup (RPM is the sensor reading of engine speed), leads to a fuel nozzles fault. In structuring fault knowledge so that it can be re-used for model knowledge, it may be useful to make explicit references to parameters that are component-specific, versus those that reflect the mode of operation of the device. Thus, in the case of the fuel nozzles, fuel flow would be a component-specific parameter, and engine speed would not be component-specific, rather, a parameter related to the operational mode of the component. Thus, fuel flow would be a parameter encoded at the component symptoms layer in figure 5.1, and engine speed would be encoded at the operational modes layer. These component-component connection descriptions generated by DR, form the first output of DR-2:

```
refined_text_comp_cons
[[main_fuel_control,
 is_a([control,for,[acceleration_fuel_limit_control,proportional_speed_control,
                    speed_cutback_control,variable_geometry_scheduling,
                    fru_fuel_selection,fuel_flow_selection]]),
 and_is_connected_to(main_fuel_pump),
 with_connectivity_parameter([weight_of_fuel_flow])],
```

```
[main_fuel_control_fuel_filter,  
is_a([filter,for,[filtering_fuel]]),  
and_is_connected_to(main_fuel_control),  
with_connectivity_parameter([weight_of_fuel_flow]),
```

```
[main_fuel_control_fuel_filter,  
is_a([filter,for,[filtering_fuel]]),  
and_is_connected_to(main_fuel_control),  
with_connectivity_parameter([weight_of_fuel_flow]),
```

```
[main_fuel_nozzles,  
is_a([nozzle,for,[fuel_flow_control]]),  
and_is_connected_to(main_fuel_control),  
with_connectivity_parameter([measured_rpm_engine_speed, single_spool_engine_speed,  
weight_of_fuel_flow]),
```

```
[main_fuel_nozzles,  
is_a([nozzle,for,[fuel_flow_control]]),  
and_is_connected_to(pressurizing_and_drain_valve),  
with_connectivity_parameter([])],
```

```
[main_fuel_pump,  
is_a([pump,for,[fuel_pump_with_speed_regulator]]),  
and_is_connected_to(main_fuel_control),  
with_connectivity_parameter([weight_of_fuel_flow]),
```

```
[main_fuel_pump_fuel_filter,  
is_a([filter,for,[filtering_fuel]]),  
and_is_connected_to(main_fuel_control),  
with_connectivity_parameter([weight_of_fuel_flow]),  
[overspeed_governor,  
is_a([control,for,[overspeed_regulation_run,  
overspeed_regulation_test,  
fuel_flow_selection]]),  
and_is_connected_to(main_fuel_control),  
with_connectivity_parameter([measured_rpm_engine_speed,single_spool_engine_speed,  
weight_of_fuel_flow]),
```

```
[pressurizing_and_drain_valve,  
is_a([control,for,[fuel_overflow]]),  
and_is_connected_to(main_fuel_control),  
with_connectivity_parameter([weight_of_fuel_flow])]
```

5.4.1.2 Meaningful Text: Component Parameters

The component parameters are grouped and their description output by the *translate 2* module. Again, the parameters are extracted directly from the JETA knowledge. As in the case of the component connections some of the parameter knowledge is missing. Notable amongst the missing relations is that of the main fuel control to the power lever angle (PLA indicating throttle position), compressor inlet temperature (T2) and compressor discharge pressure (P3). These are missing from the DR output, since they are encoded at a higher level in the fault hierarchy, specifically, as snags (which are at the same layer as phases of operation faults), and are not at the component symptom layer. As mentioned previously, the engine speed (N) is not found related to the main fuel pump as expected. The reason for this is that the engine speed is not used as a component symptom to troubleshoot the fuel pump.

In addition, an unexpected parameter is found related to the main fuel nozzles, specifically, the exhaust gas temperature (EGT). Upon closer examination of the main fuel nozzles failures, the EGT parameter, is found to be (as was the case with the unexpected engine speed connection to the fuel nozzles in section 5.4.1.1), related to the component-specific symptom layer, in describing the failures rather than, the operational mode layer (shown in figure 5.1). This is another example, where knowledge of failure would have to be represented more explicitly, so that operational model knowledge can be derived without incorrectly introducing or omitting behaviours.

refined_text_parameters

```
[[main_fuel_control,  
has_parameters([measured_rpm_engine_speed,single_spool_engine_speed,  
weight_w_fuel_flow])],
```



```
[main_fuel_control_fuel_filter,  
has_parameters([weight_of_fuel_flow])),
```

```
[main_fuel_nozzles,  
has_parameters([exhaust_gas_temperature,  
                measured_rpm_engine_speed,single_spool_engine_speed,  
                weight_of_fuel_flow])),
```

```
[main_fuel_pump,  
has_parameters([weight_of_fuel_flow])),
```

```
[main_fuel_pump_fuel_filter,  
has_parameters([weight_of_fuel_flow])),
```

```
[overspeed_governor,  
has_parameters([measured_rpm_engine_speed,single_spool_engine_speed,  
                weight_of_fuel_flow])),
```

```
[pressurizing_and_drain_valve,  
has_parameters([fuel_pump_inlet_pressure,  
                weight_of_fuel_flow]))]
```

5.4.1.3 Component Behaviour Models

DR-1 output is matched against device independent and dependent background knowledge and gaps identified in DR-2. The component function, type, inputs, outputs, regulator inputs, and behaviours between inputs and outputs are generated. These form the black box component model descriptions that can then be matched against the extracted knowledge of DR-1 to uncover gaps. In the case of DR-1, inconsistencies such as parameters which are not explicitly related to components, as identified by the background knowledge, point to inaccuracies in JETA's fault knowledge. These inaccuracies can be used to acquire new fault knowledge that can be interactively added to JETA by a domain expert who would further validate the new fault knowledge before integrating it. In all knowledge-based applications that may affect the safety of people in the military, knowledge in a system has to be validated with a human in the loop.

Note that some of the component models have fairly lengthy behavioural descriptions. This is due to the components (e.g. control type components), having a variable number of inputs and outputs, that are typified by piece-wise linear behaviour. Within such behaviour the inputs and outputs may vary in a proportional or inverse proportional manner. The DR generated component models for the main fuel system are described below.

component_models

The main fuel control fuel filter function is to filter out particles. It has the same input and output variables, namely fuel. It has no regulation input and its input-to-output behaviour is proportional.

```
{function(main_fuel_control_fuel_filter,filters(WF+)),
input(main_fuel_control_fuel_filter,fluid(WF+)),
output(main_fuel_control_fuel_filter,fluid(WF+)),
regulator(main_fuel_control_fuel_filter,none),
behaviour(for(main_fuel_control_fuel_filter),
behaviour_is_proportional(WF+,WF+,[increase_in(WF+),increases(WF+),
decrease_in(WF+),decreases(WF+)]))},
```

The function of the main fuel nozzles is to spray fuel. The input and output variables are fuel, and the flow is regulated by the amount of fuel that is input. The input-to-output behaviour relation is proportional.

```
{function(main_fuel_nozzles,flow_control(WF+)),
input(main_fuel_nozzles,flow(WF+)),
output(main_fuel_nozzles,flow(WF+)),
regulator(main_fuel_nozzles,regulation_control(WF+)),
behaviour(main_fuel_nozzles,
behaviour_is_proportional(WF+,WF+,[increase_in(WF+),increases(WF+),
decrease_in(WF+),decreases(WF+)]))},
```

The function of the main fuel pump is to deliver fuel. The input and output variables are fuel. The regulation variable for the pump is the engine speed (N) signal. The behaviour relation between the control variable and the output is proportional.

```
[function(main_fuel_pump,delivers(WF+)),
input(main_fuel_pump,fluid(WF+)),
output(main_fuel_pump,fluid(WF+)),
regulator(main_fuel_pump,N+),
behaviour(for(main_fuel_pump),
behaviour_is_proportional(WF+,N+,[increase_in(N+),increases(WF+),
decrease_in(N+),decreases(WF+)]))],
```

The main fuel pump fuel filter function is to filter out particles. It has the same input and output variables, namely fuel. It has no regulation input and its input-to-output behaviour is proportional. Note that both MFS filters share the same generic description of the device-independent background knowledge.

```
[function(main_fuel_pump_fuel_filter,filters(WF+)),
input(main_fuel_pump_fuel_filter,fluid(WF+)),
output(main_fuel_pump_fuel_filter,fluid(WF+)),
regulator(main_fuel_pump_fuel_filter,none),
behaviour(for(main_fuel_pump_fuel_filter),
behaviour_is_proportional(WF+,WF+,[increase_in(WF+),increases(WF+),
decrease_in(WF+),decreases(WF+)]))],
```

The pressurizing and drain valve controls the fuel flow by decreasing it, upon detecting a high inlet pressure. It takes as inputs a pressure signal (Pa), a pressure signal limit and fuel flow. Its behaviour relating inputs and outputs is piecewise linear. It has a single operational mode, namely, a fuel_overflow mode. It lets the fuel pass through until it detects that the inlet pressure is at its limit (Pa_limit). When Pa_limit is detected, the fuel flow is reduced.

```

[function(pressurizing_and_drain_valve,controls([WF-])),
input(pressurizing_and_drain_valve,control([Pa+,WF,Pa_limit])),
output(pressurizing_and_drain_valve,control([WF-])),
regulator(pressurizing_and_drain_valve,regulation_control([Pa+,WF,Pa_limit])),
behaviour(pressurizing_and_drain_valve,
behaviour_is_pieewise_linear([
[(mode fuel_overflow),input_control_variable(Pa+)])),
[[[the_behaviour_is_proportional_for(fuel_overflow),
the_input_control_variable(Pa+),
results_in_regulated_output_variable(WF-),
at_limit(Pa+),decrease(WF-),increases(Pa+),increases(WF-)]],

[[the_behaviour_is_proportional_for(fuel_overflow),
the_input_control_variable(WF)
,results_in_regulated_output_variable(WF-),
at_limit(WF),decrease(WF-),increases(WF),increases(WF-)]],

[[the_behaviour_is_proportional_for(fuel_overflow),
the_input_control_variable(Pa_limit),
results_in_regulated_output_variable(WF-),
at_limit(Pa_limit),decrease(WF-),increases(Pa_limit),increases(WF-)]])]]),

```

The function of the overspeed governor is to control engine speed. It has multiple input and output variables that are controlled in one of three modes, `overspeed_regulation_run`, `overspeed_regulation_test`, and `fuel_flow_selection`. The `overspeed_regulation_run` mode has engine speed as the input control variables, engine speed (N) and `N_overspeed`. The overspeed fuel flow output is controlled by maintaining the engine speed under the `N_overspeed` limit. The `overspeed_regulation_test` mode has the input variables N and `N_overspeed_test`. They result in proportional behaviour in relation to the output variable fuel flow overspeed test. The `fuel_flow_selection` mode has as its inputs the fuel flow and the fuel flow for the main fuel control. Its outputs are the overspeed governor test switch and fuel flow. At the input limits, the outputs are decreased proportionately.

```

[[function(overspeed_governor,controls([[WF_overspeed]]),
input(overspeed_governor,control([[N,N_overspeed],[N,N_overspeed_test],
[OSG_test_switch,WF_overspeed,WF_overspeed_test,
WF_mfc]])),
output(overspeed_governor,control([[WF_overspeed],[WF_overspeed_test],[WF]])),
regulator(overspeed_governor,regulation_control([[N,N_overspeed],[N,N_overspeed_test],
[OSG_test_switch,WF_overspeed,
WF_overspeed_test,WF_mfc]])),
behaviour(overspeed_governor,behaviour_is_pieewise_linear([[]],[[]],
[[[the_behaviour_is_proportional_for(overspeed_regulation_run),
the_input_control_variable(N),
results_in_regulated_output_variable(WF_overspeed),
at_limit(N),decrease(WF_overspeed),increases(N),increases(WF_overspeed)]]],

[[the_behaviour_is_proportional_for(overspeed_regulation_run),
the_input_control_variable(N_overspeed),
results_in_regulated_output_variable(WF_overspeed),
at_limit(N_overspeed),decrease(WF_overspeed),
increases(N_overspeed),increases(WF_overspeed)]]],

[[[the_behaviour_is_proportional_for(overspeed_regulation_test),
the_input_control_variable(N),results_in_regulated_output_variable(WF_overspeed_test),
at_limit(N),decrease(WF_overspeed_test),increases(N),increases(WF_overspeed_test)]]],

[[the_behaviour_is_proportional_for(overspeed_regulation_test),
the_input_control_variable(N_overspeed_test),
results_in_regulated_output_variable(WF_overspeed_test),
at_limit(N_overspeed_test),decrease(WF_overspeed_test),
increases(N_overspeed_test),increases(WF_overspeed_test)]]],

[[[the_behaviour_is_proportional_for(fuel_flow_selection),
the_input_control_variable(OSG_test_switch),
results_in_regulated_output_variable(WF),
at_limit(OSG_test_switch),decrease(WF),increases(OSG_test_switch),increases(WF)]]],

[[the_behaviour_is_proportional_for(fuel_flow_selection),
the_input_control_variable(WF_overspeed),
results_in_regulated_output_variable(WF),
at_limit(WF_overspeed),decrease(WF),increases(WF_overspeed),increases(WF)]]],

[[[the_behaviour_is_proportional_for(fuel_flow_selection),
the_input_control_variable(WF_overspeed_test),
results_in_regulated_output_variable(WF),
at_limit(WF_overspeed_test),decrease(WF),increases(WF_overspeed_test),increases(WF)]]],

[[the_behaviour_is_proportional_for(fuel_flow_selection),
the_input_control_variable(WF_mfc),
results_in_regulated_output_variable(WF),
at_limit(WF_mfc),decrease(WF),increases(WF_mfc),increases(WF)]]]]],

```

The main fuel control is the heart of the main fuel system. It controls the acceleration to compressor pressure fuel ratio. It has four operational control modes and an overall piecewise linear behaviour. The first is acceleration fuel limit control, with the control input variables engine speed (N) and power lever angle (PLA). The resultant output variable is an acceleration fuel to compressor discharge pressure ratio. The behaviour relation in this mode is proportional.

```
[function(main_fuel_control,controls([acceleration_WF/P3+])),
input(main_fuel_control,control([([N+,PLA],[N+,PLA+],[N+,T2,T2_limit],[N+,T2],
[acceleration_WF/P3,proportional_speed_WF/P3,P3],
[WFd,WF_min]))),
output(main_fuel_control,control([([acceleration_WF/P3+],[proportional_speed_WF/P3+],
[proportional_speed_WF/P3-],[IGV-,bleed valve positions],
[WFd],[WF_mfc]))),
regulator(main_fuel_control,regulation_control([([N+,PLA],[N+,PLA+],
[N+,T2,T2_limit],[N+,T2],
[acceleration_WF/P3,
proportional_speed_WF/P3,P3],
[WFd,WF_min]))),
behaviour(main_fuel_control,behaviour_is_piecewise_linear([
[[(mode acceleration_fuel_limit_control),input_control_variable(N+)]],
[[(mode proportional_speed_control),input_control_variable(N+)]],
[[(mode speed_cutback_control),input_control_variable(N+)]],
[[(mode variable_geometry_scheduling),input_control_variable(N+)]],

[[[the_behaviour_is_proportional_for(acceleration_fuel_limit_control),
the_input_control_variable(N+),
results_in_regulated_output_variable(acceleration_WF/P3+),
at_limit(N+),decrease(acceleration_WF/P3+),increases(N+),increases(acceleration_WF/P3+)]],

[[the_behaviour_is_proportional_for(acceleration_fuel_limit_control),
the_input_control_variable(PLA),
results_in_regulated_output_variable(acceleration_WF/P3+),
at_limit(PLA),decrease(acceleration_WF/P3+),
increases(PLA),increases(acceleration_WF/P3+)]],
```

The second operational mode for the MFC is the proportional speed control. Again the input variables engine speed and throttle position (PLA) generate the proportional speed fuel to compressor discharge pressure (P3) ratio.

```

[[[the_behaviour_is_proportional_for(proportional_speed_control),
the_input_control_variable(N+),
results_in_regulated_output_variable(proportional_speed_WF/P3+),
at_limit(N+),decrease(proportional_speed_WF/P3+),
increases(N+),increases(proportional_speed_WF/P3+)]]],

```

```

[[[the_behaviour_is_proportional_for(proportional_speed_control),
the_input_control_variable(PLA+),
results_in_regulated_output_variable(proportional_speed_WF/P3+),
at_limit(PLA+),decrease(proportional_speed_WF/P3+),
increases(PLA+),increases(proportional_speed_WF/P3+)]]],

```

The third operational mode for the MFC is the speed cutback control. The input variables are engine speed and compressor inlet temperature (T2). They generate a proportional speed fuel to discharge compressor pressure ratio. As the T2 limit is reached the ratio signal (which will ultimately affect fuel flow to the MFC) is decreased.

```

[[[the_behaviour_is_proportional_for(speed_cutback_control),
the_input_control_variable(N+),
results_in_regulated_output_variable(proportional_speed_WF/P3-),
at_limit(N+),decrease(proportional_speed_WF/P3-),
increases(N+),increases(proportional_speed_WF/P3-)]]],

```

```

[[[the_behaviour_is_proportional_for(speed_cutback_control),
the_input_control_variable(T2),
results_in_regulated_output_variable(proportional_speed_WF/P3-),
at_limit(T2),decrease(proportional_speed_WF/P3-),
increases(T2),increases(proportional_speed_WF/P3-)]]],

```

```

[[[the_behaviour_is_proportional_for(speed_cutback_control),
the_input_control_variable(T2_limit),
results_in_regulated_output_variable(proportional_speed_WF/P3-),
at_limit(T2_limit),decrease(proportional_speed_WF/P3-),
increases(T2_limit),increases(proportional_speed_WF/P3-)]]],

```

The fourth operational mode for the MFC is the variable geometry scheduling. The input variables are engine speed, the inlet guide vane signal and the compressor inlet temperature. They control the bleed valve positions output.

```

[[[the_behaviour_is_proportional_for(variable_geometry_scheduling),
the_input_control_variable(N+),
results_in_regulated_output_variable(IGV-),
at_limit(N+),decrease(IGV-),increases(N+),increases(IGV-)],
[the_behaviour_is_proportional_for(variable_geometry_scheduling),
the_input_control_variable(N+),
results_in_regulated_output_variable(bleed valve positions),
at_limit(N+),decrease(bleed valve positions),increases(N+),increases(bleed valve positions)]]],

```

```

[[the_behaviour_is_proportional_for(variable_geometry_scheduling),
the_input_control_variable(T2),
results_in_regulated_output_variable(IGV-),
at_limit(T2),decrease(IGV-),increases(T2),increases(IGV-)],
[the_behaviour_is_proportional_for(variable_geometry_scheduling),
the_input_control_variable(T2),
results_in_regulated_output_variable(bleed valve positions),
at_limit(T2),decrease(bleed valve positions),increases(T2),increases(bleed valve positions)]]],

```

The fifth operational mode for the MFC is the fuel ratio selection. The input variables include the acceleration, and the proportional speed fuel ratio to compressor discharge pressure, the compressor discharge pressure itself, the demanded fuel flow (Wfd) and the minimum fuel flow. They generate a proportional main fuel control signal.

```

[[[the_behaviour_is_proportional_for(fru_fuel_selection),
the_input_control_variable(acceleration_WF/P3),
results_in_regulated_output_variable(WFd),
at_limit(acceleration_WF/P3),decrease(WFd),increases(acceleration_WF/P3),increases(WFd)]]],

```

```

[[the_behaviour_is_proportional_for(fru_fuel_selection),
the_input_control_variable(proportional_speed_WF/P3),
results_in_regulated_output_variable(WFd),
at_limit(proportional_speed_WF/P3),decrease(WFd),
increases(proportional_speed_WF/P3),increases(WFd)]]],

```

```

[[the_behaviour_is_proportional_for(fru_fuel_selection),
the_input_control_variable(P3),
results_in_regulated_output_variable(WFd),
at_limit(P3),decrease(WFd),increases(P3),increases(WFd)]]],

```

```

[[[the_behaviour_is_proportional_for(fuel_flow_selection),
the_input_control_variable(WFd),
results_in_regulated_output_variable(WF_mfc),
at_limit(WFd),decrease(WF_mfc),increases(WFd),increases(WF_mfc)]]],

```

```

[[the_behaviour_is_proportional_for(fuel_flow_selection),

```


3 of/de 3

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

```

the_input_control_variable(WF_min),
results_in_regulated_output_variable(WF_mfc),
at_limit(WF_min),decrease(WF_mfc),increases(WF_min),increases(WF_mfc))]]]])).

```

5.4.2 Coffee Maker Resultant Model

From the 5 steps (with step 1 relaxed) of the DR algorithm, the DR-2 model below was generated. A regulator, a switch, a heater, a holder and a filter were the device independent component models used for background knowledge. In addition, 10 expressions that represented the device dependent background knowledge giving the type of component and the input/output behaviour parameters were used by DR. Thus, it was possible to successfully generate a component behaviour model for a full device (rather than only a subsystem) with explicit function and behaviour descriptions for each of the coffee maker components.

5.4.2.1 Meaningful Text: Component Connection Relations

The component connection relations extracted in DR-1 are translated into meaningful symbolic text descriptions by translate 1. Notable inconsistencies are the extra parametric links between the coffee pot and the coffee filter, holder, pot heating element and water reservoir (see Appendix B.2 for raw output). Of all these connections, that of the coffee pot and its heating element is valid. These inconsistencies were placed in the fault knowledge so that they may be trapped by the gaps and inconsistencies described in section 5.4. Refer to figure 5.5 for the black box model of the 8 connections out of the 47 original ones.

The first output of DR-2 is:

refined_text_comp_cons

As shown in figure 5.5, the coffee filter is connected to the coffee holder, and the holder is connected to the filter, through the coffee parameter.

```
[coffee drip,is_a([regulator,for,[coffee]]),  
and_is_connected_to(coffee holder),  
with_connectivity_parameter([coffee]]),  
[coffee filter,is_a([filter,for,[grounds]]),  
and_is_connected_to(coffee holder),  
with_connectivity_parameter([coffee]]).
```

The coffee maker plug is linked to the switch, the coffee pot and water heating element elements, and the water temperature heat control. These links are through the power, heat, water, and coffee, device parameters respectively. The coffee maker switch connections are not shown, (refer to the full output in Appendix B.2), but reflect those of the coffee maker plug connections.

```
[coffee maker plug,is_a([regulator,for,[power]]),  
and_is_connected_to(coffee maker switch),  
with_connectivity_parameter([coffee,heat,power,water]]).
```

```
[coffee maker plug,is_a([regulator,for,[power]]),  
and_is_connected_to(coffee pot heating element),  
with_connectivity_parameter([coffee,heat]]).
```

```
[coffee maker plug,is_a([regulator,for,[power]]),  
and_is_connected_to(water heating element),  
with_connectivity_parameter([heat,water]]).
```

```
[coffee maker plug,is_a([regulator,for,[power]]),  
and_is_connected_to(water temperature heat control),  
with_connectivity_parameter([heat,water]]).
```

The water heating element is connected to the water reservoir, which in turn is connected to the water temperature heat controller. Both connections are through water and heat parameters.

```
[water heating element,
is_a([heater,for,[water]]),
and_is_connected_to(water reservoir),
with_connectivity_parameter([heat,water])),
```

```
[water reservoir,
is_a([holder,for,[water]]),
and_is_connected_to(water temperature heat control),
with_connectivity_parameter([heat,water])),
```

5.4.2.2 Meaningful Text: Component Parameters

The extracted parameters in DR-1 are grouped and output as descriptions by translate 2. Notable inconsistencies due to extra connections, are the relation of the coffee maker plug and switch to all the device parameters. As described in section 5.3.2.3, these 'extra' parameters are the result of shared parents, who do not have a direct behavioural link to the components. These extra fault relations were encoded in the fault knowledge, to investigate their impact on the resultant component model. In addition, the relation of the water reservoir to the coffee parameter is unexpected in the model. This relation is extracted based on the parent 'plugged coffee drip'. This is another indirect fault which can be generalized to an operational mode layer with the mode being, 'coffee not dripping' being related to the water reservoir component rather than the component symptom ('plugged coffee drip'), since this is an *indirect* rather than a *direct* behavioural fault. Again, these are flagged by the inconsistencies and gaps described in section 5.4. The second output of DR-2 is:

refined_text_parameters

```
[[coffee drip,has_parameters([coffee])],
[coffee filter,has_parameters([coffee])],
[coffee holder,has_parameters([coffee])],

[coffee maker plug,has_parameters([coffee,heat,power,water])],
[coffee maker switch,has_parameters([coffee,heat,power,water])],
```

```

[coffee pot,has_parameters([coffee,heat])),
[coffee pot heating element,has_parameters([coffee,heat])),

[water heating element,has_parameters([heat,water])),
[water temperature heat control,has_parameters([heat,water]]]

[water resevoir,has_parameters([coffee,heat,water])),

```

5.4.2.3 Component Behaviour Models

Below are the DR-2 generated component models for the 10 components (the model for the coffee maker device is shown in figure 5.5). The behaviours of all the components are proportional with only that of the switch being a 2-state behaviour. The coffee maker input/output relations are much simpler than those of the engine application since no feedback modelling is required.

Figure 5.5. Coffee Maker device model mapped from DR output

behaviour_text

The coffee drip regulates the flow of coffee. The input regulates the output for a proportional behaviour.

```
[function(coffee drip,regulates(coffee+)),  
input(coffee drip,coffee+),  
output(coffee drip,coffee+),  
regulator(coffee drip,coffee+),  
behaviour(for(coffee drip),  
behaviour_is_proportional([increase_in(coffee+),increases(coffee+),  
decrease_in(coffee+),decreases(coffee+)]))],
```

The coffee filter filters the coffee grounds and lets the coffee fluid through. The water input regulates the coffee that is output in a proportional manner.

```
[function(coffee filter,filters(coffee+)),  
input(coffee filter,filter([grounds]),fluid(water+)),  
output(coffee filter,fluid(coffee+)),  
regulator(coffee filter,water+),  
behaviour(for(coffee filter),  
behaviour_is_proportional([increase_in(water+),increases(coffee+),  
decrease_in(water+),decreases(coffee+)]))],
```

The coffee holder is a vessel whose input is water and output is coffee. The flow of coffee is proportional to the flow of water.

```
[function(coffee holder,vessel_for(water+)),  
input(coffee holder,is(water+)),  
output(coffee holder,is(coffee+)),  
regulator(coffee holder,none),behaviour(for(coffee holder),  
behaviour_is_proportional([increase_in(water+),increases(coffee+),  
decrease_in(water+),decreases(coffee+)]))],
```

The coffee maker plug has power as its input and output which is related proportionately.

```
{function(coffee maker plug,regulates(power+)),
input(coffee maker plug,power+),
output(coffee maker plug,power+),
regulator(coffee maker plug,power+),
behaviour(for(coffee maker plug),
behaviour_is_proportional([increase_in(power+),increases(power+),
decrease_in(power+),decreases(power+)]))},
```

The coffee maker switch has two input states of on/off which result in the output power/no power. Its behaviour is state-related.

```
{function(coffee maker switch,has_states(on/off)),
input(coffee maker switch,input_states(on/off)),
output(coffee maker switch,output_states(power/no power)),
regulator(coffee maker switch,none),
behaviour(for(coffee maker switch),
behaves_according_to_state(on/off,implies_respective(power/no power)))},
```

The coffee pot is a vessel for coffee. Its input and output is coffee.

```
{function(coffee pot,vessel_for(coffee)),
input(coffee pot,is(coffee)),
output(coffee pot,is(coffee)),
regulator(coffee pot,none),
behaviour(for(coffee pot),
behaviour_is_proportional([increase_in(coffee),increases(coffee),
decrease_in(coffee),decreases(coffee)]))},
```

The function of the coffee pot heating element is to heat coffee. Its input is power and output is heat.

```
{function(coffee pot heating element,heat([coffee])),
input(coffee pot heating element,is(power+)),
output(coffee pot heating element,is(heat+)),
regulator(coffee pot heating element,power+),
behaviour(for(coffee pot heating element),
behaviour_is_proportional([increase_in(power+),increases(heat+),
decrease_in(power+),decreases(heat+)]))},
```

The function of the water heating element is to heat water. Its input is power and output is heat.

```
[function(water heating element,heat([water])),  
input(water heating element,is(power+)),  
output(water heating element,is(heat+)),  
regulator(water heating element,power+),  
behaviour(for(water heating element),  
behaviour_is_proportional([increase_in(power+),increases(heat+),  
decrease_in(power+),decreases(heat+)]))],
```

The water reservoir is a vessel for water. Its input and output is water.

```
[function(water resevoir,vessel_for(water)),  
input(water resevoir,is(water)),  
output(water resevoir,is(water)),  
regulator(water resevoir,none),  
behaviour(for(water resevoir),  
behaviour_is_proportional([increase_in(water),increases(water),  
decrease_in(water),decreases(water)]))],
```

The function of the water temperature control is to heat water and regulate its flow.

Its input is heat and its output is water which flows proportionately.

```
[function(water temperature heat control,regulates(heat+)),  
input(water temperature heat control,heat+),  
output(water temperature heat control,water+),  
regulator(water temperature heat control,heat+),  
behaviour(for(water temperature heat control),  
behaviour_is_proportional([increase_in(heat+),increases(water+),  
decrease_in(heat+),decreases(water+)]))],
```

5.5 Gaps, Inconsistencies and Novel Faults

The gaps that are extracted by the DR algorithm in DR-2 as a result of the match module represent the intersection of the fault knowledge with the device dependent background knowledge. As described in the results section 5.1 for the engine application, many parametric gaps were uncovered in addition to the absence of the operational mode

fault knowledge layer in JETA for the control components encoded in the diagnostic network.

5.5.1 Engine Application

The DR algorithm uncovered some missing and unexpected behaviours in the generation of its black box component models. These gaps and inconsistencies in JETA's fault knowledge have one of three explanations. Fault knowledge is represented as explicit device failures, not, correct device operation. Thus, some key knowledge in describing the behaviours of components, and their impact on correct overall device operation, is not encoded, as it is necessary only to reason about, known and expected device failures, in a fault-based system. To allow an algorithm such as DR to re-use fault knowledge, it would be essential to identify (through DR), and encode component behaviours and their impact on correct device operation. As discussed previously, fault knowledge could be encoded to include four important layers (shown in figure 5.1). The layers would be the phases of device operation, the operational modes specific to components (essential for control components that have multiple modes of operation based on feedback signals from the outputs to the inputs), component symptoms (which map component-specific behaviours) and the components (terminal nodes).

Other inconsistencies in the models derived from the fault knowledge, include the unexpected derivation of behaviours, not directly related to a component. As discussed in section 5.4.1.1, these inconsistencies arise, since JETA does not explicitly separate component symptom, (direct relation to component behaviours), from operational mode faults, (indirect relation to component behaviours in figure 5.1).

The third explanation for the gaps and inconsistencies uncovered by DR in JETA, is that the full device fault knowledge is not represented. The fault knowledge of some engine phases of operation, were not acquired. The reason for this is that JETA was developed, to assist a technician diagnosing acceleration faults on the engine, since these faults are the most difficult to troubleshoot. Other faults could be diagnosed from the engine maintenance practices (manuals), easily. The acceleration faults impact most, but not all, of the engine subsystems. This is why the JETA fault knowledge is representative of most, but not all, of the device.

For each of the components extracted by DR, gaps are described in relation to the functional mode(s) of the component, the extracted parameters, and the behavioural model inputs, and outputs. Each of the component gaps are explained below.

behaviour_gaps

The main fuel control has a gap in the JETA fault knowledge in relation to its acceleration fuel limit control mode. Specifically, the power lever angle (PLA) and acceleration fuel flow to compressor discharge pressure (P3) fuel ratio is missing. The PLA parameter is encoded in JETA's fault knowledge at a higher level. It can be extracted by DR as an indirect relation to the MFC component, however, it would be preferable to correct JETA's knowledge so that component-specific symptoms are at the layer directly above the terminal nodes (refer to figure 5.1). The fuel ratio relation is not part of the troubleshooting procedure of the engine, rather it is operational knowledge, it would have to be explicitly encoded if DR is to find it.

```
[[[main_fuel_control,
[[gap_for_mode,acceleration_fuel_limit_control,
extracted,RPM,input,[N+,PLA],output,[acceleration_WF/P3+]],
```

```
[gap_for_mode,acceleration_fuel_limit_control,
extracted,N1,input,[N+,PLA],output,[acceleration_WF/P3+]],
[gap_for_mode,acceleration_fuel_limit_control,
extracted,WF,input,[N+,PLA],output,[acceleration_WF/P3+]]],
```

The main fuel control also has a gap in the fault knowledge in relation to its proportional speed control mode. Again, the power lever angle (PLA) and proportional speed fuel flow to compressor discharge pressure (P3) fuel ratio is missing.

```
[main_fuel_control,
[[gap_for_mode,proportional_speed_control,
extracted,RPM,input,[N+,PLA+],output,[proportional_speed_WF/P3+]],
[gap_for_mode,proportional_speed_control,
extracted,N1,input,[N+,PLA+],output,[proportional_speed_WF/P3+]],
[gap_for_mode,proportional_speed_control,
extracted,WF,input,[N+,PLA+],output,[proportional_speed_WF/P3+]]],
```

Another main fuel control gap in the fault knowledge is in relation to its speed cutback control mode. The compressor inlet temperature (T2) and proportional speed fuel flow to compressor discharge pressure (P3) fuel ratio is missing. T2 is another parameter that is encoded as an indirect relation to the MFC and it should be more directly related.

```
[main_fuel_control,
[[gap_for_mode,speed_cutback_control,
extracted,RPM,input,[N+,T2,T2_limit],output,[proportional_speed_WF/P3-]],
[gap_for_mode,speed_cutback_control,
extracted,N1,input,[N+,T2,T2_limit],output,[proportional_speed_WF/P3-]],
[gap_for_mode,speed_cutback_control,
extracted,WF,input,[N+,T2,T2_limit],output,[proportional_speed_WF/P3-]]],
```

For the main fuel control variable geometry scheduling mode, the compressor inlet temperature (T2) and inlet guide vane (IGV) parameter relations which lead to bleed valve positions are missing. IGV is encoded at a higher level than a component-specific symptom and should be moved to the component symptom layer in figure 5.1.

```
[main_fuel_control,
[[gap_for_mode,variable_geometry_scheduling,
extracted,RPM,input,[N+,T2],output,[IGV-,bleed valve positions]],
```

```
[gap_for_mode,variable_geometry_scheduling,  
extracted,N1,input,[N+,T2],output,[IGV-,bleed valve positions]],
```

```
[gap_for_mode,variable_geometry_scheduling,  
extracted,WF,input,[N+,T2],output,[IGV-,bleed valve positions]]],
```

In addition, for the main fuel control fuel ratio fuel selection mode, the acceleration and proportional speed fuel flow to compressor discharge pressure (P3) ratios are missing.

```
[main_fuel_control,  
[[gap_for_mode,fru_fuel_selection,  
extracted,RPM,input,[acceleration_WF/P3,proportional_speed_WF/P3,P3],output,[WFd]],  
[gap_for_mode,fru_fuel_selection,  
extracted,N1,input,[acceleration_WF/P3,proportional_speed_WF/P3,P3],output,[WFd]],  
[gap_for_mode,fru_fuel_selection,  
extracted,WF,input,[acceleration_WF/P3,proportional_speed_WF/P3,P3],output,[WFd]]],
```

The fuel flow selection control mode parameters are fuel flow which is represented in the fault knowledge, however no explicit relation is made to the parameter relation to the operational mode is made.

```
[main_fuel_control,  
[[gap_for_mode,fuel_flow_selection,  
extracted,RPM,input,[WFd,WF_min],output,[WF_mfc]],  
[gap_for_mode,fuel_flow_selection,  
extracted,N1,input,[WFd,WF_min],output,[WF_mfc]],  
[gap_for_mode,fuel_flow_selection,  
extracted,WF,input,[WFd,WF_min],output,[WF_mfc]]],
```

The main fuel control, main fuel pump fuel filters and pressurizing and drain valve show a gap in behaviour since their models generate a proportional input-output relation which is not in the fault knowledge. This is an acceptable gap since the fault knowledge can only indicate parametric knowledge, not the increasing or decreasing relations of those parameters.

```
[main_fuel_control_fuel_filter,  
[[gap_for_mode,filtering_fuel,extracted,WF,input,[WF+],output,[WF+]]],
```

```
[[main_fuel_pump_fuel_filter,
[[gap_for_mode,filtering_fuel,extracted,WF,input,[WF+],output,[WF+]]]],

[[pressurizing_and_drain_valve,
[[gap_for_mode,fuel_overflow,extracted,Pa,WF,input,[Pa+,WF,Pa_limit],output,[WF-],[ ]]]]]
```

The main fuel nozzles show a relation to both engine speed and exhaust gas temperature in the fault knowledge which is inconsistent with the behavioural model. This is a JETA inconsistency, which is related to their encoding as component-specific, rather than at the operational mode level, and it can be corrected.

```
[[main_fuel_nozzles,
[[gap_for_mode,fuel_flow_control,extracted,EGT,input,[WF+],output,[WF+]],
[gap_for_mode,fuel_flow_control,extracted,RPM,input,[WF+],output,[WF+]],
[gap_for_mode,fuel_flow_control,extracted,N1,input,[WF+],output,[WF+]],
[gap_for_mode,fuel_flow_control,extracted,WF,input,[WF+],output,[WF+]]]]],
```

The overspeed governor had both engine speed (N) and fuel flow (WF) extracted from the fault knowledge. The gap in its knowledge indicates the absence of the functional control mode relation in the knowledge base. This could be added within a layer above the component symptoms so that it may related to phases of engine operation to improve the reflection of the control mode failures in the fault knowledge.

```
[[overspeed_governor,
[[gap_for_mode,overspeed_regulation_run,
extracted,RPM,input,[N,N_overspeed],output,[WF_overspeed]],
[gap_for_mode,overspeed_regulation_run,
extracted,N1,input,[N,N_overspeed],output,[WF_overspeed]],
[gap_for_mode,overspeed_regulation_run,
extracted,WF,input,[N,N_overspeed],output,[WF_overspeed]]]],

[overspeed_governor,
[[gap_for_mode,overspeed_regulation_test,
extracted,RPM,input,[N,N_overspeed_test],output,[WF_overspeed_test]],
[gap_for_mode,overspeed_regulation_test,
extracted,N1,input,[N,N_overspeed_test],output,[WF_overspeed_test]],
[gap_for_mode,overspeed_regulation_test,
extracted,WF,input,[N,N_overspeed_test],output,[WF_overspeed_test]]]]],
```

```
[overspeed_governor,
[[gap_for_mode,fuel_flow_selection,
extracted,RPM,
input,[OSG_test_switch,WF_overspeed,WF_overspeed_test,WF_mfc],
output,[WF]],
[gap_for_mode,fuel_flow_selection,
extracted,N1,
input,[OSG_test_switch,WF_overspeed,WF_overspeed_test,WF_mfc],
output,[WF],[])]]].
```

5.5.2 Coffee Maker Application

The coffee maker gaps are more of the form of inconsistencies, and missing connections in the fault knowledge, that do not accurately reflect the behavioural component models. These gaps were deliberately implanted to check how they would affect the resultant model compared to that expected (shown in figure 5.5). In some ways, they change the model by adding links between components but generally they have little effect.

behaviour_gaps

The coffee drip and the water temperature heat control correctly identify the parameters that indicate the input-output relations. Again, they do not show the increasing or decreasing relations between the parameters, since such knowledge cannot be extracted from the fault knowledge.

```
[[[coffee drip,
[[gap,extracted,coffee,input,[coffee+],output,[coffee+]]]]],
```

```
[[water temperature heat control,
[[gap,extracted,heat,input,[heat+],output,[water+]],
[gap,extracted,water,input,[heat+],output,[water+]]]]]]]
```

The coffee filter and the coffee holder are missing their input parameters. This can be traced to the deliberate absence of parental node information (specifically, the parent 'water not dripping'), in the fault knowledge.

```
[[coffee filter,  
[[gap,extracted,coffee,input,[water+],output,[coffee+]]]]],
```

```
[[coffee holder,  
[[gap,extracted,coffee,input,[water+],output,[coffee+]]]]],
```

The coffee maker plug and the coffee maker switch have extra parameters in addition to their required power parameter. This is due to deliberately added parental nodes in their fault knowledge, (specifically, the parents 'heat problem: coffee too cold' and 'heat problem: water too cold').

```
[[coffee maker plug,  
[[gap,extracted,coffee,input,[power+],output,[power+]],  
[gap,extracted,heat,input,[power+],output,[power+]],  
[gap,extracted,power,input,[power+],output,[power+]],  
[gap,extracted,water,input,[power+],output,[power+]]]]],
```

```
[[coffee maker switch,  
[[gap,extracted,coffee,input,[on/off],output,[power/no power]],  
[gap,extracted,heat,input,[on/off],output,[power/no power]],  
[gap,extracted,power,input,[on/off],output,[power/no power]],  
[gap,extracted,water,input,[on/off],output,[power/no power]]]]],
```

The coffee pot deliberately has an extra link to a parental node that provides it with a heat parameter, (specifically, the parent 'heat problem: coffee too hot').

```
[[coffee pot,[[]],[gap,extracted,heat,coffee,input,[coffee],output,[coffee]]]]],
```

The heating elements are deliberately missing their parameter that provides them with a power behaviour, (specifically, the parent 'no power').

```
[[coffee pot heating element,  
[[gap,extracted,coffee,input,[power+],output,[heat+]],  
[gap,extracted,heat,input,[power+],output,[heat+]]]]],
```

```
[[water heating element,  
[[gap,extracted,heat,input,[power+],output,[heat+]],  
[gap,extracted,water,input,[power+],output,[heat+]]]]],
```

The water reservoir has extra parameters, specifically coffee and heat due to deliberately added links and its missing heat link, (specifically, the parents 'heat problem: water too hot' and 'coffee too weak').

```
[[water resevoir,  
[[gap,extracted,coffee,input,[water],output,[water]],  
[gap,extracted,heat,input,[water],output,[water]],[]]]],
```


6.0 Concluding Remarks

AI researchers in diagnosis share a common goal, developing the most accurate and efficient means of identifying and troubleshooting a problem with a device. However, they disagree on the approach. Some researchers embrace a fault-based approach which makes use of the known failures of a device to diagnose it. Other researchers endorse a model-based approach which makes use of expected device operation to detect anomalies between observations of a device, and its respective model. Fault-based diagnosis (FBD) is criticized as inadequate for identifying novel faults. Model-based diagnosis (MBD) is criticized as intractable, computationally complex, and as a result impractical for diagnosing complex devices. Both MBD and FBD share problems in knowledge acquisition, be it for acquiring models or faults. Some MBD researchers have argued for a balance between complexity and completeness in diagnosis [Davis 84]. MBD is computationally complex, while FBD knowledge is generally incomplete since it cannot diagnose novel faults. The Diagnostic Remodeler algorithm, the subject of this dissertation, is intended to form a bridge between FBD and MBD, by relating and re-using fault knowledge for automated model acquisition.

The Diagnostic Remodeler algorithm is a new algorithm which has been formulated, implemented, and its results presented [Abu-Hakima 93a; Abu-Hakima 94a; Abu-Hakima 94b]. DR illustrates that well-structured fault knowledge can be mapped and re-used as model knowledge. DR addresses the re-use of existing complex device fault knowledge in conjunction with device dependent and independent background knowledge for the generation of black box component models of a device. These component models

represent device structure, behaviour, and function. DR thus maintains two views for the diagnostic knowledge of a single device or subsystem, a fault-based view and a model-based one. These two views help illustrate that fault and model knowledge for the same device are related and given one view, the second can be extracted. DR illustrates that the model view can be extracted from the fault view in conjunction with background knowledge. DR thus forms a bridge between the MBD and FBD communities in its automated acquisition of model knowledge from fault knowledge. Other MBD researchers have generated fault knowledge by compiling diagnoses based on the model view [Bizzari et al. 90; Meisner et al. 90; Pearce 88]. These two approaches support the DR hypothesis that fault and model knowledge is strongly related.

DR's black box component models provide a powerful, more simple and more easily generated alternative to full numerical simulations. Numerical simulations tend to be more difficult to generate and perfect, especially for complex devices. In addition, for complex devices, DR's component models are easier to understand than their fault knowledge.¹ The models have also been shown to be useful in validating the original fault knowledge.²

The DR algorithm has two phases, DR-1 and DR-2. In DR-1, fault-based knowledge from a typical troubleshooting system is used to extract component-connection relations. In DR-2, the extracted components are matched against background knowledge to derive specific component input/output behaviours. In addition, any gaps in the original

1. As an illustration, one can compare the DR model descriptions in chapter 5.0, section 5.4, for the engine application with their respective fault knowledge base graphically illustrated in Appendix A.1.

2. See the discussion of the engine gaps and inconsistencies in chapter 5.0, section 5.5 for specifics.

fault knowledge component-connection links are tagged. These gaps and inconsistencies uncover novel faults in the original fault knowledge.

Two applications are used to test the generality of the DR algorithm. The first is to complex well-structured fault-knowledge, namely, the Jet Engine Troubleshooting Assistant, JETA [Halasz et al. 92; Abu-Hakima et al. 93b]. JETA was chosen to demonstrate how DR can successfully extract a model, validated manually by a domain expert, of the main fuel system from an existing real-world knowledge base that encoded faults for a jet engine.

The second application of DR is to model a simpler device, a coffee maker. The coffee device fault knowledge was formulated to re-use DR for a very different domain. The coffee domain is more easily understood by other researchers. Thus, its DR generated model does not require evaluation by a domain expert. A full device, black box component model, is acquired by DR from the coffee fault and background knowledge. This model was evaluated manually by tracing through the fault knowledge and reviewing the extracted model.

6.1 Addressing the Thesis Objective Statement

The objective of the thesis has been to develop and test an algorithm that demonstrates that automated model acquisition, from fault-based knowledge is practical, in that it can be understood, and validated, by a domain expert. To this end, the Diagnostic Remodeler algorithm has been developed. The DR algorithm is used to discover and refine a component behavioural model with explicit function. The objective is to generate a behaviour model from fault knowledge which may be suitable, but its use not necessary

in this dissertation, for diagnosis. In the first phase, DR-1 identifies the components of the device and generates links between them from fault-based knowledge. In the second phase, DR-2 uses device dependent and device independent background knowledge to generate a component behavioural model for the originally extracted components and their connections. DR-2 transforms the component connections extracted by DR-1 into explicit inter-component behaviours.

DR has been successfully tested and validated by an engine expert with real-world jet engine fault knowledge from JETA. From JETA, DR produces a component behavioural model of the main fuel system by visiting ~60, out of a possible 197 nodes³ (30% of encoded fault nodes). The model is easier to understand than JETA's fault-based knowledge (JETA graphical network is in Appendix A.1). This model was manually checked and verified by the expert.

In the case of the coffee maker, a 30 node fault base was formulated. The assumption that constrained DR to select specific types of terminal nodes for modelling only a subsystem was relaxed and the full device model was output. DR successfully generated a ten component full device model for the coffee maker. It took as input fault knowledge as well as device dependent and independent background knowledge. It is important to note that most of the device independent component knowledge for the engine example was re-used for the coffee maker application (see chapter 3.0 for details). The coffee maker model was verified by manually comparing it to the expected model⁴.

3. The 60 nodes represent 1200 of the 3972 lines of JETA fault knowledge code.

4. Refer to chapter 5.0, section 5.4.2 for the discussion of the DR model of the coffee maker device.

DR had as one of its objectives the discovery of novel faults, defined as those not encoded in the fault knowledge base. These were assumed to be identifiable through the black box component models and their respective behaviours. Gaps and inconsistencies were identified in both the engine and the coffee maker application⁵. These identify novel faults in the engine knowledge and were validated by the domain expert. Gaps and inconsistencies in the coffee maker fault knowledge were based on deliberately omitted, or deliberately encoded fault knowledge, used to test the hypothesis that the DR algorithm could identify them.

The DR algorithm has two assumptions⁶. First, terminal nodes map to device components and can be identified in the fault knowledge. Second, movement up the diagnostic hierarchy in the fault knowledge to the root node⁷ is a generalization of symptomatic knowledge on device components (terminal nodes), to operational mode faults, which in turn can be generalized to phases of device operation faults. This second assumption is important in mapping the relation between the components and the various layers of knowledge above it as well as identifying any gaps or inconsistencies. Specifically, in the engine fault knowledge it is clear that a significant layer of knowledge is missing. If one examines the knowledge extracted by DR, the lowest layer of knowledge is the component knowledge. The layer above that knowledge is, as assumed by DR, symptomatic knowledge that maps directly to component parameters. These parameters represent model behavioural variables. Some minor inconsistencies were found in JETA at this

5. Refer to chapter 5.0, section 5.5. for details on the gaps and inconsistencies, in the engine, and coffee applications, uncovered by the DR algorithm.

6. Refer to chapter 3.0, section 3.3 for discussion of the two DR assumptions.

7. Refer to chapter 5.0, section 5.1 and Figure 5.1 for detailed discussion on the diagnostic hierarchy and DR.

level. However, the most significant discovery is the layer of missing knowledge above the symptomatic knowledge for components that have multiple functional modes in JETA. These are specifically control components (e.g. the main fuel control, the overspeed governor, etc.). These components have associated with them functional modes with a different number of respective control variables. Thus, in a specific mode, a fault in JETA may be manifested and it would be indicated by some variable. In another mode, a completely different variable may be the indicator of a fault with the same component. At a higher level in the JETA hierarchy, the functional modes are related to phases of engine operation. However, the current JETA fault knowledge does not explicitly relate these component operational modes to the phases of engine operation. Given the DR acquired component models, it would not be very difficult to add this new layer of knowledge above the component symptom layer and explicitly relate it to the phases of engine operation.

In formulating the device dependent and device independent background knowledge, it was essential to address some practical real-world modelling problems such as representing inter-component feedback and multiple component inputs and outputs, both common in control problems. To resolve these problems, the background knowledge representation makes use of a common technique in engine control operation and performance modelling [Treager 79]. Specifically, a component that has multiple inputs and is a control component has a variety of control modes. Each control mode has a set of inputs and a set of outputs. Thus, in determining inter-component behaviour, it is essential to identify the mode a component is operating in and match that with the modes the other components have. Thus, for an engine which is in acceleration mode, a main fuel control

component has a primary control signal (specifically, the pilot selected throttle position measured as the power lever angle control signal), which will affect its output (demand for more fuel), and control signals to other components. Other engine modes will dictate other primary control signals such as engine speed, compressor inlet temperature or compressor discharge pressure (see chapter 3.0 for specifics), for the main fuel control component.

A relatively small amount of background knowledge was used as input to DR. For the engine application, the percentage of background to fault knowledge was only 14% (or exactly 168 lines of code (LOC) compared to the 1200 in JETA that encode the faults of the main fuel system). Out of this 8.42% (or 101 lines of code) was device dependent knowledge, and 5.58% (or 67 lines of code) was device independent. This further supports the use of DR to generate a component model that is more comprehensible than the original engine fault knowledge.

For the coffee maker application, the ratio of background to fault knowledge was 91 to 330 lines of code or 28% (11% or 35 LOC device dependent, and 17% or 56 LOC device independent). The ratio is higher for the coffee maker device since the background knowledge is for the full device rather than just a subsystem (as in the engine application).

It is important to note that as the application gets more complex (i.e. the engine), the ratio of total background knowledge to fault knowledge gets smaller. What is more significant, is that the background knowledge used by DR for modelling a subsystem for a complex application requires 50% less code (14% versus 28%), than for a simpler application. This justifies the hypothesis that DR works well in modelling complex fault knowledge bases whereas its use may not be justified for simple fault bases. Formulating the background knowledge for a simple device requires less effort than for a complex device.

This is supported by the fact that a domain expert participated in ensuring that the engine background knowledge was accurate whereas no specialized expertise was required to formulate the background knowledge for the coffee maker device. However, the resultant black box component models with explicit descriptions of behaviour and function are far easier to understand than reading the fault knowledge in a complex application. The same may not be necessarily true for the simpler application, since its model was not constrained to a subsystem (for example, a subsystem with 5 components rather than the 10 that make up the device), and as a result, the raw output reflected a far greater number of connections and resultant behaviours between components that require a graphical, rather than a textual description for better comprehension. However, if the simpler device modelling was restricted to a smaller number of components the DR output would more likely be easily understood.

It is important to examine the impact of background knowledge in relation to fault knowledge⁸. One finds that with a minimum amount of device dependent background knowledge (specifically, the *list of device parameters* descriptor, which is 1 line of code), and the fault knowledge as input⁹, DR can still have a useful output, namely the links between the component terminal nodes extracted in DR-1. As descriptors are added to the device dependent background knowledge in DR-2 (a total of 11 can be included), the descriptors output by the device independent background knowledge increase (a total of 6 can be instantiated for any generic component). Device independent background knowledge in turn makes the output of DR more expressive. For a complex application, DR's

8. Refer to chapter 5.0, section 5.2 for a detailed analysis and discussion of fault to background knowledge.

9. Fault knowledge input to DR consisted of, 1200 lines of code for the engine, and 330 LOC for the coffee applications, respectively.

formulation of background knowledge is thus quite worthwhile. DR can produce black box component models that can assist someone in understanding, verifying, and if necessary, expanding the fault knowledge to include any gaps or corrections to inconsistencies.

Another significant point is that DR does not rely greatly on the background knowledge for its results. As such, if only the list of device parameters were used as background knowledge, DR could uncover gaps and inconsistencies in connections between components. It can achieve this by presenting a user with the encoded links in the fault knowledge between components which the user can then validate. Admittedly, it would be more useful for DR to compare the extracted links to those it can extract from its device dependent background knowledge, as it does in DR-2.

6.2 Usability of DR Output

This section addresses specific questions on the usability of the DR algorithm. To facilitate this explanation, the DR methodology is related to modelling the coffee maker device.

- *How do you use DR for a new application?* DR re-uses fault knowledge in generating component behaviour models. Thus, for a new application, the fault knowledge must exist. This fault knowledge must be well-structured. If no fault knowledge exists, it can be generated as in the case of the coffee maker fault knowledge (section 3.8.2). If the fault knowledge exists, but it is not well-structured, a pre-processing phase (possibly DR-0) must be implemented to convert the unstructured knowledge. In the case of the coffee maker, a device with 10 components, a well-structured 30-node fault knowledge base was generated (section 3.8.2.2). The knowledge base (KB) was organized as a hierarchy of frames where each frame has a slot that explicitly identified the node name, its parent(s), any siblings and any children. In addition to the fault knowledge, both device dependent and independent background knowledge needs to be formulated

as in the case of the coffee maker device (section 3.8.2.3 and 3.8.2.4 respectively). The device dependent knowledge is an expression with the component name, its type, input(s), output(s) and respective signs (+/-) that indicate the change in value of the specific I/O parameters. Ten expressions are formulated for the coffee device. One of these is a coffee filter whose device dependent expression is:

```
glossary(coffee,'coffee filter',[filter,for,[grounds]],[['water+'],['coffee+']])).
```

The device independent background knowledge is formulated as template that can be re-used across domains for component types. Five templates are formulated for the 10 coffee maker device components. One is for a filter-type of component and the device independent description is:

```
component(filter,Filter_name,Input,Output,Filter_material,F,I,O,R,B):-
    F = function(Filter_name, filters(Output)),
    I = input(Filter_name,filter(Filter_material),fluid(Input)),
    O = output(Filter_name,fluid(Output)),
    R = regulator(Filter_name,Input),
    behaviour_proportional(Input,Output,Behaviour),
    B = behaviour(for(Filter_name),
        behaviour_is_proportional(Behaviour)).
```

- *What assumptions does DR make related to the fault knowledge?* Given an existing fault knowledge base, DR has two assumptions. First, that terminal nodes in the fault hierarchy are assumed to be components. Such is the case for the coffee maker fault hierarchy (Figure 3.6 in section 3.8.2.2). Second, DR assumes that the fault hierarchy is well-structured and that parental nodes relate siblings (defined as more than one terminal node that shares a parent) through explicitly identified parameters.
- *How are DR steps followed?* DR has five steps (described in section 3.3.4). First, DR extracts the terminal nodes which are the device component nodes (following DR assumption 1). In the case of the coffee maker, the 10 device component names are returned after step 1. Second, DR backtracks to parent(s) of terminal nodes and searches for siblings. For example, DR extracts that the *coffee filter* and *coffee holder* share the parent *coffee taste too strong*. Third, DR relates siblings based on parameters that are explicit at parent nodes. These form the component-to-component connections.

Thus, the *coffee filter* and *coffee holder* share the parameter *coffee* (this parameter is identified as one of 3 that the coffee device has - specifically, *coffee*, *water* and *power* - see section 3.8.2.3). Fourth, DR matches the component-to-component connections to device dependent knowledge that provides the input and output and parametric relations for each component in addition to identifying the component type (see section 5.4.2.1). Fifth, match the components based on their type to device independent descriptions so that the complete component behaviour model could be output (5.4.2.3).

- *What is the form of DR output?* The specific DR output for the 30 node FK of the coffee maker device is (fully described in for DR-1 output in 5.3.2 and for DR-2 output in 5.4.2 and a full test run is in Appendix B.2). However, the 10 components extracted from the fault knowledge are:

```
[coffee drip,coffee filter,coffee holder,coffee maker plug,
coffee maker switch,coffee pot,coffee pot heating element,
water heating element,water reservoir,water temperature heat control]
```

For each of the 10 device components, the following description is output (see Appendix B.2 for the 10 component behaviour descriptions).

```
[function(coffee filter,filters(coffee+)),
input(coffee filter,filter([grounds]),fluid(water+)),
output(coffee filter,fluid(coffee+)),
regulator(coffee filter,water+),
behaviour(for(coffee filter),
behaviour_is_proportional([increase_in(water+),increases(coffee+),
decrease_in(water+),decreases(coffee+)])),
[[coffee filter,[gap,extracted,coffee,input,[water+],output,[coffee+]]]]].
```

- *How does the modification of fault knowledge affect the output of DR?* The modification of the fault knowledge could affect the components or the connections between the components that DR extracts. For example, if 11 terminal nodes exist for the coffee maker but the device dependent knowledge only identifies 10, then a gap will be output to indicate that extra fault knowledge was extracted that has no background knowledge to map to. In addition, if the fault knowledge identifies several shared parents that lead to several parameters for a device whose background knowledge only indicates a single input, an inconsistency between the extracted fault knowledge and the background knowledge parameters is output as a gap. Finally, if the fault knowledge is missing a

shared parent that identifies a specific parameter, a gap is output. Such is the case for the coffee filter example above where only the coffee parameter was extracted but no water parameter was extracted. This implies that a parent of the coffee filter terminal node is missing.

- *How does the modification of device dependent knowledge change DR output?* The device dependent background knowledge identifies the component name, its type and I/O parameters and their respective signs that indicate changes in value. Thus, any of these descriptors can be modified. Input/Output parameters can be added or dropped thus modifying the final component behaviour output after matching the device dependent to the device independent knowledge. Also, the signs on the inputs and outputs can be modified. Sign changes will affect the final component behaviour which is proportional, inverse proportional or piecewise linear depending on the signs and the complexity of their I/O relations.
- *How does the modification of device independent knowledge alter DR output?* Device independent knowledge has descriptors for component function (based on component type from the device dependent knowledge), inputs, outputs, control variables and behaviours. Thus, if inputs and outputs are added or removed in the device dependent knowledge, they will be reflected in the behavioural model. If I/O signs are changed in the device dependent knowledge, the behaviours inferred will change (for example, if an input and its respective output has a positive direction, and if the input is changed to have a negative direction while the output is kept positive, then the component behaviour changes from proportional to inversely proportional). The device independent background knowledge can be expanded to include additional descriptors (for example, for more complicated behaviour functions or inverse rather than proportional control variables). Finally, new component types and their respective descriptors can be added (for example, an inductance, a capacitance or a resistance component can be added as templates and their respective I/O, control and behaviour descriptors be formulated).
- *How is the device independent knowledge re-usable and how does it make DR output predictable?* The device independent background knowledge is formulated as a template. As such, it can be re-used easily across domains if the component types match.

For example, the filter component type can be re-used across domains since its function is simply to filter some substance (identified in the device dependent background knowledge input) from a fluid (also identified but as an output). Thus, for a previously identified component type (directly extracted from the device dependent knowledge), the respective device independent template is filled with an input, output and its behaviour inferred from the I/O signs from the device dependent background knowledge.

- *How is the device independent knowledge searched?* The device independent knowledge is searched based on the component type. The type acts as a key to identify the template to match on. This is akin to the use of an index in a database search.
- *How can one decide when to expand the background knowledge?* In some cases the device dependent knowledge may be missing an input or output parameter. This can simply be added to the respective list. In addition, the device independent template for a component type may be missing a key descriptor (for example, an inverse control variable or perhaps a logarithmic behaviour function relating inputs and outputs). This can simply be added to the respective component type with any new related inputs or outputs. It is important to note that the component descriptions output by DR may be missing relations and can very simply be expanded to include them.
- *Can DR be used iteratively?* DR can be used iteratively to first generate the component-connections from the fault knowledge. If known connections are missing, the fault knowledge should be examined and corrected to avoid inconsistencies or gaps. In the case of an assumed correct set of component-connections extracted from the fault knowledge, a match with the background knowledge in DR-2 may uncover missing connections. In such a case, it is possible to modify the fault knowledge iteratively until no gaps or inconsistencies are identified in DR-2.

6.3 Thesis Contributions

Five main contributions have resulted from this Ph.D. thesis:

- The definition, implementation and testing of the general Diagnostic Remodeler algorithm which can automatically acquire a component behaviour model with explicit function for a device or device subsystem has been achieved.
- The Diagnostic Remodeler algorithm results (the black box component models) can be used to validate, and uncover novel faults in the fault knowledge base. DR has successfully generated a model that has been validated by a domain expert for a complex real-world device, the main fuel system for the J85-CAN-15 jet engine. The input to DR-1 (the first DR phase), is the real-world Jet Engine Troubleshooting Assistant (JETA) fault knowledge. The input to DR-2 (the second DR phase), is device dependent and independent background knowledge acquired from an aircraft engine Expert at NRC's Propulsion Laboratory. The background knowledge includes explicit representation of inter-component feedback and multiple component inputs and outputs, common in control problems, but not traditionally addressed in acquiring or using models in the AI community. The generated model has been used to uncover gaps and inconsistencies in JETA's fault knowledge, and as a result, has uncovered some novel faults not encoded in JETA.
- The Diagnostic Remodeler software has been successfully re-used for a second very different, but simpler problem, the automatic model acquisition of a coffee maker full device component model from fault knowledge. The model again uncovers some gaps and inconsistencies in the fault knowledge. The coffee maker application demonstrates the possible generality of the algorithm and its potential wide applicability across various application domains.
- In implementing the DR algorithm's device independent knowledge, it was possible to formulate generalized component descriptions which were successfully re-used with minor modifications between the engine and the coffee maker domains. These descriptions present a novel manner by which to describe component function (what a component is designed to do), inputs, outputs, primary control variables and behaviours. Behaviours are specifically represented as proportional, inversely proportional, or piecewise linear, input-to-output behaviour as dictated by operating modes for the variable inputs and outputs of a component.

- The fifth contribution is the thorough review of techniques relevant to diagnosis in Artificial Intelligence [Abu-Hakima 94c]. This review was completed to confirm that the DR algorithm concepts are unique and have not been addressed in the related AI literature. The review is supported by an extensive annotated bibliography [Abu-Hakima 94c] intended to guide the reader to significant and related work. The review is subdivided into eight sets, with the five most relevant to diagnosis, discussed in chapter 2.0.

6.4 Issues Addressed by DR

There are several issues that the DR algorithm addresses.

Firstly, what is the exact form of the automatically acquired model when some or no background knowledge is used? If no background knowledge is used, is the model much more than a topological rather than a component behaviour model with explicit representation of function?

The answer here is that a minimum amount of device dependent knowledge, namely the list of device parameters, is required. These parameters are used to constrain DR-1 from extracting parameters that are irrelevant in modelling the device components. If this minimal device dependent knowledge is not used as input to DR-1 with the fault knowledge, then the specific parameters extracted from the shared parents, for sibling component nodes could not be identified. The topological knowledge would only indicate that sibling components are connected, but the parameters connecting them could not be explicitly extracted.

If no device dependent background knowledge is used as input to DR-2, then gaps and inconsistencies between the fault-based knowledge and the generated component

model could not be identified. Also, the increasing or decreasing relations between the inputs and outputs of a component could not be identified.

If no device independent knowledge (component library knowledge) is used as input to DR-2, then the primary control variables for the components could not be identified. In addition, explicit component function (e.g. a pump pumps or a control regulates or a holder is a vessel that holds a substance) could not be identified. Also, the behavioural relations between the behavioural input and output paths (e.g. proportional behaviour or piecewise linear behaviour) could not be identified.

Another question that had to be addressed in the implementation of the DR algorithm is how general an algorithm is it?

The DR algorithm has been implemented as an algorithm that can generate black box component models for devices from fault and background knowledge. The DR algorithm has been applied for modelling two very different devices. The first is complex and is based on existing real-world well-structured fault knowledge. The second is simple and is based on formulated well-structured fault knowledge. The first is a subsystem of a jet engine and the second is the full device representing a coffee maker. All the DR algorithm software was re-used in modelling the coffee maker after modelling the engine main fuel system. Specifically, DR's *get components*, *get component connections*, *get component parameters (translate 1)*, *translate parameters (translate 2)* and *match* modules, were all re-used. The only difference was that the engine application used the JETA fault knowledge and engine dependent background knowledge. The coffee application used the coffee maker fault knowledge and coffee dependent background knowledge. Importantly, for the device independent background knowledge, the coffee application had 5 component

descriptions of which 4 had proportional input-to-output behaviour, which essentially re-used 3 of the 5 engine component descriptions, with very minor modifications. The 2 engine components not re-used for the coffee maker device, as they are more applicable for complex devices and describe control-type components with a variable number of inputs and outputs, related to variable operating modes described by piecewise linear behaviour.

6.5 Future Work

Several directions of future work can be taken in advancing the use of the Diagnostic Remodeler algorithm:

1. The DR algorithm requires a well-structured fault knowledge base with explicit representation of terminal nodes, parental nodes, and sibling nodes as slots (or contents) of a frame. It would be interesting to determine DR's applicability to a rule-based representation of faults. Rule-based systems are not often used in complex real-world troubleshooting as their knowledge is implicit rather than explicit which makes them difficult to debug and modify. Rule-based systems mix control knowledge for affecting inference with data or knowledge about the domain (this is why in rule-based systems rule order affects reasoning [Clancey 86; Abu-Hakima 88]). However, a pre-DR algorithm step could be implemented that could attempt to separate control from domain knowledge so that DR can be re-used. Such a step may be an algorithm itself, since it would have to identify rules and group them according to their use in the domain as data or control knowledge. As an intermediate step it would be interesting to re-use DR on a meta-rule-based system. Meta rules attempt to group rules, and separate control from domain knowledge [Clancey 86].
2. Another interesting direction that would further confirm the utility of DR is to use the extracted black box component models as input into an off-the-shelf model-based diagnosis tool. For example, the Intelligent Diagnostic Expert Assistant (IDEA) is a

tool distributed by Microprocessor Systems Inc. IDEA is an extension of Randall Davis' work at MIT [Davis 84] and follows a black box component model approach (hence DR output could be easily input to IDEA) [Orchard and Shevel 94]. IDEA could be used to further illustrate that the model extracted by DR is a behavioural model in MBD. Thus, additional novel faults could be identified for the fault base that DR is used to extract a device model from.

3. As described in chapters 2.0 and 3.0, structured fault-based systems have a knowledge base with a wealth of knowledge that could be mined for acquiring models. The DR algorithm only uses three explicit slots in DR-1 to extract component-to-component connections. A knowledge base such as JETA has fourteen slots which hold a variety of knowledge including help information to guide the technician as well as background knowledge justifying steps in the troubleshooting procedures. At the higher level in the hierarchically-based network there also exists some minimal overall device function operation knowledge which may be extracted and explicitly described, (such as the phases that an engine operates in and hence fails in, such as start-up, acceleration, etc.). It may also be possible to expand DR-1 to further interpret help information (possibly through natural language parsing), or to directly extract explicit component operation knowledge and relate it to the affected components.
4. A direction of research in diagnosis is the generation of fault-based knowledge from model-based knowledge. Pearce was one of the earliest implementors of an automated approach to generate a fault base by inducing rules from a simulator model of an electrical power system [Pearce 88]. In a related manner, it would be interesting to write a third phase to the DR algorithm, namely DR-3, to automatically generate faults that could be added to the source fault base the DR model was generated from. What would be unique is that DR-3 would generate fault-based frames to simplify their integration into the domain fault knowledge base. One must caution that the predictability and practicality of such an approach is uncertain.

5. Finally, the DR algorithm has been successfully tested for two different domains that represent physical systems namely an aircraft engine and a coffee maker. A key problem today is the problem of software diagnosis. Software is not a physical, but rather is an abstract domain. It would be interesting to examine whether the DR approach is applicable to abstract domains.

Software is generally written to provide some function or solution to a problem. Structured software programs generally have a main program that calls subprograms which can each be separate modules or procedures (functions in LISP, predicates in Prolog, routines in C, etc.). The subprograms often have identifiable inputs and outputs in addition to internal variables, with each representing particular subprogram behaviour. The subprograms themselves may have their own subprograms with respective input, output, or internal variables.

Thus to apply DR, it would first be necessary to break down a software program into individual subprograms. It would also be essential to identify the familial relations between the program, its first level subprograms (these can be considered the children of the program), and all nested subprograms called by the first level subprograms (the children and their position in the subprogram family relations). It would also be important to identify the types of relations between the various program units (which calls which, what function is each doing - is it sorting a list, is it passing a parameter value, is it making a calculation, etc.). In this manner, the connections that DR-1 generates for a physical system could be generated for a software system. However, the connections between the software subprograms would likely be more complex and variable than those of a physical system even a complex one such as an aircraft engine.

Once the various program and subprogram connections are identified, it should be possible to match the individual units (program or subprograms), to background knowledge of component-like libraries that are both program dependent, and program independent, as done in DR-2 with physical systems. The program dependent background knowledge would serve to constrain subprogram descriptions based on the program and language constraints (language syntax and program semantics such as

function or goal of program). It would also establish subprogram type (for example, a list sort subprogram, a pretty print subprogram, a numeric value calculation subprogram, etc.). To facilitate establishing the type of subprogram, language constructs and library routines can be used to generate a partial set of subprogram types. It is obvious that each program and subprogram will have unique elements that cannot typically be categorized. However, as new types are found, the program dependent background knowledge can be expanded to include them.

Program independent subprogram descriptions are abstractions, or more general descriptions, of program dependent background knowledge. They would serve to simplify the generation of descriptions for similar subprograms that have the same function but variable inputs and outputs. They would be useful in mapping the subprogram description libraries between languages, (C, Prolog, LISP, etc.), as the component libraries were in generating device models for different DR domains.

Thus, it may be possible to expand the DR algorithm's functionality to extract a model of subprogram software behaviour. One must be cautious however not to assume that the problem of generating the complex relations between program and subprogram elements is a simple one. However, if one can identify such relations, and enumerate anomalies in program-subprogram interactions in a fault base, then it should be possible to use an algorithm like DR to generate a behavioural model of the software units.

7.0 References

[Aamodt 90] Aamodt, A., "Knowledge-intensive case-based reasoning and sustained learning." ECAI 90, Proceedings of the 9th European Conference on Artificial Intelligence, Stockholm, Sweden, pp. 1-6, (August 6-10, 1990).

[Abu-Hakima 94a] Abu-Hakima, S., Automatic Extraction of Device Models from Fault Knowledge: the Diagnostic Remodeler Algorithm. To be published in the Proceedings of the AAAI-94 Workshop on Device Function. To be held in August 1994, Seattle, Washington, USA. AAAI Press(1994).

[Abu-Hakima 94b] Abu-Hakima, S., Automated Model Acquisition: the Diagnostic Remodeler Algorithm. Submitted to the Fifth International Workshop on Principles of Diagnosis. To be held in October 1994, New Paltz, New York, USA. (1994).

[Abu-Hakima 94c] Abu-Hakima, S., Diagnostic techniques in knowledge-based systems: a review of approaches, applications and issues. 100 p. NRC ERA-381 (1994).

[Abu-Hakima 93a] Abu-Hakima, S., Automatic Knowledge Acquisition in Diagnosis. Proceedings of DX-93, Fourth International Workshop on Principles of Diagnosis. Aberystwyth, Wales. 236-250. (1993).

[Abu-Hakima 93b] Abu-Hakima, S., Halasz, M., and Phan, S., An Approach to Hypermedia in Diagnostic Systems. Chapter 10 in Intelligent Multimedia Interfaces, Edited by Mark Maybury, AAAI Press (1993).

[Abu-Hakima and Oppacher 90] Abu-Hakima, S., and Oppacher, F., Improving explanations in knowledge-based systems: RATIONALE. Knowledge Acquisition journal (2). 301-343. December 1990.

[Abu-Hakima 88] Abu-Hakima, S., RATIONALE: A Tool for Developing Knowledge-Based Systems that Explain by Reasoning Explicitly. Masters thesis, Carleton University, Ottawa, Canada. May 1988.

[Abu-Hanna et al. 92b] Abu-Hanna, A., Benjamins, R., and Jansweijer, W., "Integrating multiple model types in model-based diagnosis", DX-92, The 3rd International Workshop on Principles of Diagnosis, Rosario, Washington, pp. 179-184, (October 12-14, 1992).

[Abu-Hanna and Benjamins 90] Abu-Hanna, A. and Benjamins, R., "Knowledge classification of models in model-based diagnosis", Expert Systems and their Applications: 10th International Workshop, Avignon, France, pp. 97-110, (May 28 - June 1 1990).

[Althoff et al. 90] Althoff, K-D., Maurer, F., and Rehbold, R., "Multiple knowledge acquisition strategies in MOLTKE." Current trends in knowledge acquisition, Published by IOS, Amsterdam, Netherlands. pp. 21-40, (1990).

[Ashley 89] Ashley, K.D., "Defining salience in case-based arguments", Proceedings of IJCAI-89, Detroit, Michigan, pp. 537-542, (1989).

[Ashley and Simoudis 92] Ashley, K.D. and Simoudis, E., "Case-based reasoning", [1992] AAAI-92 Tutorial Notes, Received at the tenth national conference of artificial intelligence, San Jose, California. July 12, 1992. pp. 1-172, (1992).

[Attias and Sveitis 90] Attias, L. and Sveitis, J., "Inertial navigation system intelligent diagnostic expert (INSIDE)", Proceedings of the 1990 National Aerospace and Electronics Conference: NAECON 90. Dayton, OH, pp. 1250-53, (21-25 May 1990).

[Azmoodeh 91] Azmoodeh, M., "Representation of generic structure and behaviour of networks for model based diagnostic applications", British Telecom Technology Journal, vol. 9, no. 3, pp. 52-60, (1991).

[Bizzari et al. 90] Bizzari, I., Corazziari, F., Faciano, S., Gualaccini, P.G., Luminari, L., Savarese, M. and Trasatti, E., "Fault diagnosis of electronic circuits", Tenth International Workshop: Expert Systems and their Applications. Specialized Conference: Artificial Intelligence and Electrical Engineering, Avignon, France, pp.115-23, (May 28-June 1, 1990).

[Bonarini and Sassaroli 92] Bonarini, A., and Sassaroli P., "Opportunistic multi-model-based diagnosis: using all the knowledge we have", DX-92, The 3rd International Workshop on Principles of Diagnosis, Rosario, Washington, pp. 205-211, (October 12-14, 1992).

[Boose 91] Boose, J.H., "Knowledge acquisition tools, methods and mediating representations", Knowledge Acquisition for Knowledge-Based Systems, Edited by H. Motoda, R. Mizoguchi, J. Boose and B. Gaines. Published by IOS Press, the Netherlands. pp. 25-62, (1991).

[Bramucci et al. 89] Bramucci, F., Aquilani, S. and Donelli, C., "SADE - a support tool for the diagnosis of liver diseases", AIME 89: Second Generation Conference on Artificial Intelligence in Medicine, London, UK. pp. 199-203, (29-31 August 1989).

[Brezillon and Fauquembergue 90] Brezillon, P. and Fauquembergue, P., "La tache de diagnostic dans le systeme expert SEPT", Tenth International Workshop: Expert Systems and their Applications, Specialized Conference: Artificial Intelligence and Electrical Engineering, Avignon, France. pp. 283-293, (May 28-June 1, 1990).

[Carlson et al. 88] Carlson, G.J., Imam, I., Azzaro, S.H., Scheibel, J.R., "Arti Expert System On-Line Crack Monitoring and Diagnostic System", Proceedings of the 1988 ASME International Computers in Engineering Conference and Exhibition, San Francisco, CA, (31 July - 4 August 1988).

[Carnes and Fisher 92] Carnes, J.R. and Fisher, D., "Inductive learning approaches to fault isolation and sensor placement", Notes of the Third International Workshop on the Principles of Diagnosis, October 12-14, Rosario, Washington pp. 132-138, (1992).

[Chandrasekaran and Mittal 83] Chandrasekaran, B. and Mittal, S. "Deep versus compiled knowledge approaches to diagnostic problem-solving". International Journal of Man-Machine Studies, 19(5), 425-436.(1983).

[Chandrasekaran et al. 82] Chandrasekaran, B., Mittal, S. and Smith, J.W., "Reasoning with uncertain knowledge: the MDX approach". Proceedings of 1st Annual Joint Conference American Informatics Association, 335-339.(1982).

[Chandrasekaran et al. 79] Chandrasekaran, B., Gomez, F., Mittal, S. and Smith, J.W., "An approach to medical diagnosis based on conceptual structures". Proceedings of the Joint Conference on Artificial Intelligence, 134-142.(1979).

[Chandrasekaran and Tanner 86] Chandrasekaran, B. and Tanner, M.C., "Uncertainty handling in expert systems: uniform vs. task-specific formalisms". In Kanal L.N. and Lemmer J., editors, *Uncertainty in Artificial Intelligence*, North Holland Publishing.(1986).

[Chen and Gelernter 91] Chen, C. and Gelernter, H., "IREF - An interactive theory-driven knowledge refinement tool", Seventh IEEE Conference on Artificial Intelligence Applications, Miami Beach, Florida, February 24-28, 1991, pp. 40-46, (1991).

[Clancey 86] Clancey, W.J., "From GUIDON to NEOMYCIN and HERACLES in twenty short lessons: ORN final report 1979-1985". *The AI Magazine*, pp. 40-60,(1986).

[Console et al. 91] Console, L., Portinale, L. and Dupre, D.T., "Focussing abductive diagnosis", *AI Communications*, 4(2-3). pp. 88-97, (June-September 1991).

[Cser et al. 91] Cser, L., Greiger, M., Greska, W. and Hoffmann, M., "Three kinds of case-based learning in sheet metal manufacturing", *Computers in Industry*, Volume 17, Number 2-3, Elsevier Science Publishers, Amsterdam, North Holland, pp. 195-206, (November 1991).

[Daniels and Feelders 91] Daniels, H.A.M. and Feelders, A.J., "Combining qualitative and quantitative methods for model-based diagnosis of firms". *Decision Support Systems and Qualitative Reasoning: Proceedings of the IMACS International Workshop*, Toulouse, France 13-15 March, 1991. North-Holland Publishing, New York, USA, pp. 255-260, (1991).

[Davis 84] Davis, R., "Diagnostic reasoning based on structure and behaviour", *Artificial Intelligence*, Vol. 24, pp. 347-410 (1984).

[de Kleer and Williams 87] de Kleer, J., and Williams, B.C., "Diagnosing Multiple Faults", Artificial Intelligence, vol. 32, (1987).

[De Jong and Mooney 86] DeJong, G., and Mooney, R., "Explanation-based learning: an alternative view", Machine Learning, Volume 1, pp. 145-176. Kluwer Academic Publishers, (1986).

[Dewberry and Carnes 90] Dewberry, B.S. and Carnes, J.R., "Intelligent monitoring and diagnosis systems for the space station freedom ECLSS", Fourth annual workshop on space operations automation and robotics (SOAR '90), Albuquerque, New Mexico. NASA Goddard technical report NASA CP-3103. (March 90).

[Dolce et al. 90] Dolce, J.L., Mellor, P.A. and Kish, J., "Automated electric power management and control for space station freedom", Proceedings of the 25th intersociety energy conversion engineering conference: IECEC-90, Reno, Nevada. pp. 269-274, (August 12-17, 1990).

[Dollman and Webster 91] Dollman, T. and Webster, K., "Monitoring and analysis of data from complex systems". Technology 2000, Washington, DC. pp. 119-126, (27-28 November, 1991).

[Duval 91] Duval, B., "Abduction for explanation-based learning", Proceedings of the European Working Session on Learning, Porto, Portugal, March 6-8, 1991, Published by Springer-Verlag, pp. 348-360, (1991).

[Ebrahimi and Perry 88] Ebrahimi, M. and Perry, J., "FMEA -IMRL correlation expert system", Proceedings of the fourth annual artificial intelligence and advanced com-

puter technology conference, Long Beach, CA, Published by Tower Conference Management, Glen Ellyn, IL. pp 494-8, (4-6 May 1988).

[Fink and Lusth 87] Fink, P.K., Lusth, J.C., "Expert Systems and Diagnostic Expertise in the Mechanical and Electrical Domains", IEEE Transactions on Systems, Man and Cybernetics, Vol SMC-17, No.3, (May/June 1987).

[Gaines and Shaw 91] Gaines, B. and Shaw, M., "Foundations of Knowledge Acquisition", Knowledge Acquisition for Knowledge-Based Systems, Edited by H. Motoda, R. Mizoguchi, J. Boose and B. Gaines. Published by IO3 Press, the Netherlands. pp. 3-23, (1991).

[Genest et al. 90] Genest, J., Matwin, S. and Plante, B., "Explanation-based learning with incomplete theories: a three-step approach", Proceedings of the Seventh International Conference on Machine Learning, University of Texas, Austin, June 21-23, pp. 287-294, (1990).

[Goel and Chandrasekaran 90] Goel, A. and Chandrasekaran, B., "A task structure for case-based design", Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Washington, D.C. IEEE Computer Society Press, pp. 587-592, (1990).

[Goel et al. 87] Goel, A., Soundararajan, N. and Chandrasekaran, B., "Complexity in classificatory reasoning", Proceedings of the 6th National Conference on Artificial Intelligence: AAAI-87, Menlo Park, CA, pp. 421-25, (1987).

[Goker and Kuru 90] Goker, M.N., Kuru, S., "The TSE_310 Troubleshooting Expert Prototype for the Airbus A-310 Commercial Aircraft", Proceedings of the Third

International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Charleston, SC, (15-18 July 1990).

[Guo 91] Guo, T-H., "An SSME high pressure oxidizer turbopump diagnostic system using G2 real-time expert system". NASA Lewis technical report NASA-TM-105328. Presented at the third annual health monitoring conference for space propulsion systems sponsored by the University of Cincinnati, OH, (13-14 November 1991).

[Hagar and Alcock 89] Hagar, S.K., Alcock, J.F., "Rocket Engine Condition Monitoring System", AIAA/ASME/SAE/ASEE 25th Joint Propulsion Conference, Monterey, CA, (10-12 July 1989).

[Halasz et al. 92] Halasz, M., Davidson, P., Abu-Hakima, S., and Phan, S., JETA: A Knowledge-based Approach to Aircraft Gas Turbine Engine Maintenance. Journal of Applied Intelligence, 2, pp. 25-46, Kluwer Academic Publishers, NRC 31832, (1992).

[Hamscher 90] Hamscher, W., "XDE: Diagnosing devices with hierarchic structure and known component failure modes", Proceedings of the Sixth Conference on Artificial Intelligence Applications. Santa Barbara, CA, pp. 48-54, (5-9 May, 1990).

[Hamscher and Struss 90] Hamscher, W. and Struss, P., "Model-Based Diagnosis", AAAI-90 Tutorial Notes, Received at the eighth national conference of artificial intelligence, Boston, Massachusetts. July 29, 1990. pp. 1-179, (1990).

[Harris 89] Harris, C.J., "Distributed estimation, inferencing and multi-sensor data fusion for real time supervisory control", Artificial Intelligence in Real-Time Control 1989. Proceedings of the IFAC Workshop, Shenyang, China, Published by Pergamon Press, Oxford, UK, 1990. pp. 19-24, (19-21 September 1989).

[Hennessy and Hinkle 92] Hennessy, D. and Hinkle, D., "Applying case-based reasoning to autoclave loading", *IEEE Expert Magazine: Intelligent Systems and their Applications*. Volume 7, Number 5. Published by the IEEE Computer Society. pp. 21-26, (October 1992).

[Hofmann et al. 91] Hofmann, M.O., Cost, T.L. and Whitley, M., "A hybrid model for SSME diagnosis", *Proceedings of SOUTHEASTCON '91*, Williamsburg, VA. pp. 239-242, volume 1, (7-10 April, 1991).

[Horak 90] Horak, D.T., "System Failure Isolation in Dynamic Systems", *Journal of Guidance, Control and Dynamics*, vol. 13, no. 6, (November/December 1990).

[How 90] "How Things Work" by the editors of Consumer Guide, Publications International Ltd., Illinois, 1990, pages 12-13 and 214-215.

[Howard et al. 89] Howard, H.C., Wang, J., Daube, F., and Rafiq, T., "Applying design-dependent knowledge in structural engineering design", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AI EDAM*, 3(2), Academic Press Limited Publishing. pp. 111-23 (1989).

[Henrion et al. 91] Henrion, M., Breese, J.S., Horvitz, E.J., "Decision Analysis and Expert Systems", *AI Magazine*, (Winter 1991).

[Hughes 89] Hughes, P.M., "CLEAR: Automating control centers with expert system technology", *Third annual workshop on space operations automation and robotics (SOAR '89)*, Houston, TX, NASA Goddard technical report NASA CP-3059, (March 90).

[Kindler 91] Kindler, H., "Controlling qualitative reasoning by a cognitive problem solving model for decision support in electromyography." *Decision Support Systems*

and Qualitative Reasoning: Proceedings of the IMACS International Workshop, Toulouse, France, North-Holland Publishing, New York, USA, pp. 241-246, (13-15 March 1991).

[Kolodner 91] Kolodner, J.L., "Improving human decision making through case-based decision aiding", AI Magazine, Volume 12, Number 2, Published by American Association of Artificial Intelligence, (Summer 1991).

[Kolodner 89a] Kolodner, J.L., "The MEDIATOR: analysis of an early case-based problem solver", Cognitive Science 13. pp. 507-549, (1989).

[Kolodner 89b] Kolodner, J.L., "Selecting the best case for a case-based reasoner", Proceedings of the Eleventh Annual Conference of the Cognitive Science Society. Hillsdale New Jersey. Lawrence Erlbaum Publishers. pp. 155-162, (1989)

[Kolodner 87a] Kolodner, J.L., "Capitalizing on failure through case-based inference", [1987a] Proceedings of the 1987 Conference on the Cognitive Science Society. Seattle, Washington pp. 715-726, (1987).

[Kolodner 87b] Kolodner, J.L., "Extending problem solver capabilities through case-based inference", Proceedings of the Fourth Machine Learning Workshop. San Mateo, California Morgan Kaufmann Publishers. pp. 167-178, (1987).

[Koppel 91] Koppel, M. "ESBL: An integrated method for learning from partial information", Annals of Mathematics and Artificial Intelligence, Volume 4, pp. 323-344, (1991).

[Koton 89] Koton, P., "A medical program that improves with experience", Computer Methods and Programs in Biomedicine, 30, Elsevier Science Publishers Biomedical Division. pp. 177-184, (1989).

[Krishnamurthi and Phillips 92] Krishnamurthi, M. and Phillips, D.T., "An expert system framework for machine fault diagnosis", *Computers and Industrial Engineering*, 22(1). pp. 67-84, (January 1992).

[Kumar 94] Kumar, A., "Reasoning about function: AAAI-93 Workshops Summary Report", *AI Magazine*, Spring 1994, Vol. 15, No. 1, (1994).

[Langley and Mooney 92] Langley, P. and Mooney, R., "Machine Learning for Problem Solving", AAAI-92 Tutorial Notes, Received at the tenth national conference of artificial intelligence, San Jose, California. July 13, 1992. pp. 1-104, (1992).

[Lee et al. 90] Lee, M.H., Hunt, J.E., Price, C.J., and Long, F.W., "REPAIR: a model-based diagnosis system", UK IT 1990 Conference, Aberystwyth, UK, pp. 266-270, (19-22 March 1990).

[Leitch and Stefanini 89] Leitch, R., and Stefanini, A., "Task dependent tools for intelligent automation", *Artificial Intelligence in Engineering*, vol. 4, no. 3, pp. 126 - 143, (1989).

[Leith et al. 90] Leith, R.G.I.T., Deans, N.D. and Stewart, I.D., "Electrical machine monitoring using a learning expert system", *Proceedings of the 25th Universities Power Engineering Conference*, Published by Robert Gordon's Institute of Technology, Aberdeen, UK, pp 707-10, vol 2, (12-14 September 1990).

[MacIsaac and Muir 91] MacIsaac, B.D., Muir, D.E., "Less is Learned in Gas Turbine Performance Analysis", *Proceedings of the Canadian Gas Association Symposium on Industrial Application of Gas Turbines*, Banff, Alberta, (16-18 October 1991).

[Malin, and Liefken 91] Malin, J.T., and Liefken, D.B., "Representing functions/ procedures and processes/structures for analysis of effects of failures on functions and operations", Proceedings of 1991 NASA Goddard Conference on Space Applications of Artificial Intelligence, Greenbelt, Maryland, pp. 141-151, (May 13-15 1991).

[Marsh 88] Marsh, C.A., "The ISA expert system: a prototype for failure diagnosis on the space station" Proceedings of the 1st International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE '88, University of Tennessee Space Institute, Tennessee. Volume 1, pp. 60-74, (1-3 June 1988).

[McBryan et al. 90] McBryan, B., Meyer, S.M. and Knabach, M.D., "A sensor management expert system for multiple sensor integration (MSI)", Proceedings of the IEEE 1990 National Aerospace and Electronics Conference (NAECON 1990), Dayton, OH, pp. 1107-12, (21-25 May 1990).

[McDowell et al. 90] McDowell, J.K., Davis, J.F., Abd-Allah, M.A. and Aref, M., "A comparison of compiled reasoning systems and model-based reasoning systems and their applicability to the diagnosis of avionics systems", Proceedings of the 1990 National Aerospace and Electronics Conference: NAECON 90, Dayton, Ohio, pp. 1243-1249, (May 21-25, 1990).

[Meisner et al. 90] Meisner, J., Bursch, P. and Funk, H., "Evolution of a maintenance diagnostic system", Proceedings of the IEEE 1990 National Aerospace and Electronics Conference: NAECON 1990, Dayton, Ohio, vol. 2, pp. 520-525, (21-25 May 1990).

[Millett and Maby 88] Millett, B.P. and Maby, R.K., "A hybrid knowledge-based approach to AH-64A fault isolation." Received through personal communication August 1988.

[Mitchell et al. 86] Mitchell, T.M., Keller, R.M., Kedar-Cabelli, S.T., [1986] "Explanation-based generalization: a unifying view", Machine Learning, Volume 1, pp. 47-80. Kluwer Academic Publishers, (1986).

[Modesitt 90] Modesitt, K.L., "Inductive knowledge acquisition experience with commercial tools for space shuttle main engine testing", Proceedings of the fifth conference on Artificial Intelligence for space applications, NASA Marshall Flight Center, pp. 487-496, (May 1990).

[Mozetic 92] Mozetic, I., "A polynomial-time algorithm for model-based diagnosis", Report from the Austrian Research Institute for Artificial Intelligence, TR-91-26, 39p. (1992).

[Nakamura and Kobayashi 91] Nakamura, K. and Kobayashi, S., "Knowledge acquisition from machine adjustment cases using causal model and operationality criteria", Knowledge Acquisition for Knowledge-Based Systems, Edited by H. Motoda, R. Mizoguchi, J. Boose and B. Gaines. Published by IOS Press, the Netherlands. pp.261-276, (1991).

[Navinchandra et al. 91a] Navinchandra, D., Sycara, K.P. and Narasimhan, S., "Behavioral synthesis in CADET, a case-based design tool", Proceedings of the 7th Conference on the Applications of Artificial Intelligence. Miami Beach, Florida. pp. 217-221, (1991).

[O'Connell and Ott 89] O'Connell, P.F., Ott, J.F., "Technician's Assistant, A Generic Knowledge Based Diagnostic Tool", Proceedings of TEST 1989: ATE and Instrumentation Conference West, Anaheim, CA, (23-26 Jan 1989).

[Orchard and Shevel 94] Orchard, R., Shevel, R., "A review of commercial diagnostic development systems", NRC Internal Report, 54 pages, (1994).

[Papenhausen and Hadley 86] Papenhausen, D.W., Hadley, B.L., "CEPS: An Artificial Intelligence Approach to Avionics Maintenance", Proceedings of the Air Force Workshop on Artificial Intelligence Applications for Integrated Diagnostics, Boulder, CO, (29-31 July 1986).

[Pazzani 90] Pazzani, M.J., "Learning in order to avoid search in logic programming", Computers & Mathematics with Applications, 20(9-10), pp. 101-10, (1990).

[Pazzani and Brunk 91] Pazzani, M.J. and Brunk, C.A., "Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning", Knowledge Acquisition, Volume 3, pp. 157-173, (1991).

[Pearce 88] Pearce, D.A., "The Induction of Fault Diagnosis Systems from Qualitative Models", AAAI-88, Seventh National Conference on Artificial Intelligence, Saint Paul, MN, (21-26 August 1988).

[Pearce et al. 92] Pearce, M., Goel, A.K., Kolodner, J.L., Zimring, C., Sentosa, L. and Billington, R., "Case-based design support", IEEE Expert Magazine: Intelligent Systems and their Applications. Volume 7, Number 5. Published by the IEEE Computer Society. pp. 14-20, (October 1992).

[Quinlan 91] Quinlan, J.R., "Inductive knowledge acquisition from structured data", Knowledge Acquisition for Knowledge-Based Systems, Edited by H. Motoda, R. Mizoguchi, J. Boose and B. Gaines. Published by IOS Press, the Netherlands. pp. 97-111, (1991).

[Quinlan 87] Quinlan, J.R., "Simplifying decision trees", International Journal of Man-Machine Studies, Volume 27, Number 3, pp. 221-234, (1987).

[Quinn and Walters 91] Quinn, T.M. and Walters, J.L., "Autonomous fault detection, isolation and recovery system for a 20KHz electric power distribution testbed", NASA technical report NASA-TM-104344, 18 pages (June 1991).

[Rissland and Skalak 89a] Rissland, E.L. and Skalak, D.B., "Interpreting statutory predicates", Proceedings of the second international conference on artificial intelligence and law. Vancouver, BC, 13-16 June, 1989. Published by ACM. pp. 46-53. (1989).

[Schank and Slade 91] Schank, R.C. and Slade, S.B., "The future of artificial intelligence: learning from experience", Applied Artificial Intelligence, Volume 5, pp. 97-107. Published by Hemisphere Publishing (1991).

[Schmalhofer et al. 91] Schmalhofer, F., Kuhn, O. and Schmidt, G., "Integrated knowledge acquisition from text, previously solved cases and expert memories", Applied Artificial Intelligence, Volume 5, pp. 311-337. Published by Hemisphere Publishing Corporation. (1991).

[Scott et al. 77] Scott, A.C., Clancey, W.J., Davis, R. and Shortliffe, E.H., "Explanation capabilities of production-based consultation systems". Stanford University, Dept. of Computer Science, Report No. STAN-CS-77-593,(1977).

[Simoudis 92] Simoudis, E., "Using case-based retrieval for customer technical support", IEEE Expert Magazine: Intelligent Systems and their Applications. Volume 7, Number 5. Published by the IEEE Computer Society. pp. 7-12, (October 1992).

[Simoudis 90] Simoudis, E., "Learning redesign knowledge", IEEE Transactions on Computer-Aided Design, volume 9, Number 10, pp. 1047-1062, (1990).

[Slade 91] Slade, S., "Case-based reasoning: a research paradigm", AI Magazine, Volume 12, Number 1, Published by the American Association of Artificial Intelligence, pp. 42-55, (Spring 1991).

[Sonoda et al. 89] Sonoda, Y., Kanemoto, S., Imaruoka, H., "A Diagnostic Expert System for a Boiling Water Reactor Using a Dynamic Model", Proceedings of Artificial Intelligence in Nuclear Power Plants, IAEA/IWG NPPCI Specialists Meeting, Helsinki, Finland, (10-12 October 1989).

[St. Clair et al. 88] St. Clair, D.C., Wetterstoem, A. and Johnson, V.M., "ESPAD: An adaptively controlled rule-based expert system for monitoring and diagnosing space vehicle subsystems", Proceedings of the ACM sixteenth annual computer science conference, Atlanta, GA, pp. 667-72 (23-25 February 1988).

[Sticklen et al. 88] Sticklen, J., Chandrasekaran, B., and Bond, W.E., "Distributed causal reasoning for knowledge acquisition: a functional approach to device understanding", 3rd AAAI sponsored Knowledge Acquisition Workshop for Knowledge-Based Systems, Banff, Canada, pp. 34-1 to 34-18, (1988).

[Struss 89] Struss, P., "New Techniques in Model-based Diagnosis", Proceedings of Knowledge-based Computer Systems, Bombay, India, (11-13 December 1989).

[Struss and Dressler 89] Struss, P., Dressler, O., "Physical Negation - Integrating Fault Models into the General Diagnostic Engine", Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89), Detroit, MI, (20-25 August 1989).

[Sycara 90] Sycara, K.P., "Negotiation planning: an AI approach", European Journal of Operational Research. 46. Elsevier Science Publishers. pp. 216-234, (1990).

[Sycara 91] Sycara, K.P., "Problem Restructuring in Negotiation", Management Science. Vol. 37, Number 10. Institute of Management Sciences Publishers. pp. 1248-1268, (October 1991).

[Sycara and Navinchandra 89] Sycara, K.P. and Navinchandra, D., "Integrating case-based reasoning and qualitative reasoning in engineering design", Proceedings of the Fourth International Conference on the Applications of Artificial Intelligence in Design, Cambridge, UK, pp. 231-250, (10-13 July 1989).

[Sykes et al. 88] Sykes, D.J., Cochran, J.K., Young, H.H., "Development of Diagnostic Expert Systems using Qualitative Simulation", Proceedings of the SCS Multiconference on Artificial Intelligence and Simulation: The Diversity of Applications, San Diego, CA, (3-5 February 1988).

[Treager 79] Treager, I.E. Aircraft Gas Turbine Engine Technology, Edited by D.E. Gilmore and G. McCloskey, Second Edition, McGraw-Hill Book Company, 1979, p. 586.

[Turney and Halasz 92] Turney, P. and Halasz, M., "Instance-based learning and multivariate linear regression applied to J85 transient performance data", Internal Report, Knowledge-Systems Laboratory, IIT, NRC, pp. 1-25. (1992).

[Wagner 91] Wagner, E.A., "Onboard automatic aid and advisory for pilots of control-impaired aircraft", *Journal of guidance, control and dynamics: a publication of the American Institute of Aeronautics and Astronautics*, vol. 14, no. 4, pp. 823-33, (July-August 1991).

[White 91] White, M.F., "Expert systems for fault diagnosis of machinery", *Measurement: Journal of International Measurement Confederation*, vol. 9, no. 4, pp. 163-71, (1991).

[Widman and Loparo 89] Widman, L.E., Loparo, K.A., "Artificial Intelligence, Simulation and Modeling: A Critical Survey", *Artificial Intelligence, Simulation and Modeling*, John Wiley and Sons, (1989).

[Whitaker and Thorsden 90] Whitaker, L.A. and Thorsden, M.L., [1990] "Aerospace applications of case-based reasoning", *Proceedings of the 1990 IEEE National Aerospace and Electronics Conference (NAECON 1990)*, Dayton, Ohio. Published by the IEEE. pp. 113-116, (1990).

[Whitaker and Wiggins 89] Whitaker, L.A. and Wiggins, S.L., "Bidder's Associate: a case-based reasoning system to improve cost estimates of manufactured products", *Proceedings of the 1989 IEEE Conference on Systems, Man and Cybernetics*. Published by IEEE. pp. 521-522, (1989).

Appendix A: Engine Application

A.1 Graphical View of JETA Network in Browser

179 of 179 nodes in box

Frame Editor

Slot

Refresh Topology Editor

'N-JETA-ROOT

J85-C engine

Slot

CHILD-NODE-OF

HAS-CHILD-NODES

DISPLAY-NAME

TIME-ESTIMATE

EVENT-ACTIONS

COMPLETION-ACTIONS

MODE-TYPE

TYPE

FLASHING

Trouble-shooting information for the J85-CAN-15 jet engine.

engine with afterburner which derives thrust from the change of momentum of gases going through it. The afterburner provides additional thrust for take-off, climb and brief periods high speed performance.

SL2 The engine

consists basically of an eight-stage axial flow compressor, an annular combustion section, a two stage turbine, a diffuser casing and an exhaust section. The afterburner casing is attached to the diffuser casing (Ref 817B Figure

3-14F - engine cutaway view)

A.2 Correspondence with Engine Domain Expert

Aug 29/93

- newer engines measure thrust
- fuel system controlled by speed + temperature
- controls SS + accel. up to temperature limit
- 15 controls open/closing of nozzles based off EG

Where Measurements
Compressor

* not used
for fuel
control
metered

compression $\rightarrow$

"in" $\rightarrow$ $\square$ $\rightarrow$ out

National Research
Council Canada

Conseil national
de recherches Canada

Institute for Institut de
Aerospace Research recherche aérospatiale

Ottawa, Canada
K1A 0R6

Telefax: 1-613-957-3281

TELEFAX COVER SHEET

TO:	S. Abu-Hakima IIT	FROM:	Jeff Bird
		PHONE NO.:	1-613-993-2214
		DATE:	5 January 1994
PHONE NO.:	991-1231	PAGES TO FOLLOW:	5
FAX NO.:	952-7151		

REFERENCE:

As requested, here is a review of your models for the J85-CAN-15 control modes. There is more detail that can be added from the physical modelling that has been done, if you want?
Sorry for the delay.

MFC modes:

1) steady speed control

- more demand for N_d ,
and W_F and FRU
based on increase in PLA

$$\text{nominal } N_d = f(PLA)$$

✓ 2) speed cutback control

Holds mechanical speed at ~ 16500 at low T_2

- T_2 below $-15^\circ F$ implies
cutback on N_d

✓ 3) acceleration fuel limit control

-? shouldn't
this be
based on
 N_d^+ not N^+ ?

3a) proportional speed control. (See attachment)

3b) see attach

✓ 4) deceleration fuel limit control

-? same question
as 3) - shouldn't
it be N_d^-

✓ 5) variable-geometry scheduling

Keeps fuel flow at minimum value
for adequate combustor
operation

- proportional
variation
of N/T_2
control
- IGV + bleed valves
within tolerance

ABFC modes:

~ 1) afterburner ignition

~ 2) afterburner fuel reduction (really included in 3) see note)

✓ 3) afterburner fuel scheduling (see note)

- increase fuel based on PLA^+ as long as A_8 , EGT , P_3 are within tolerance

✓ MFC / OSG mode:

- decrease (drain off) fuel if P_a exceeds P_a max tolerated

reduces fuel flow if mechanical speed exceeds $105\% \pm 1\%$

VEN modes

✓ 1) steady-state-control

✓ 2) engine acceleration control

IF ACCEL SET equivalent to a fast rate of PLA demands jump to 50% AR.

✓ 3) EGT limit

* A_{80} "floating" to maintain T_5 under limit

— T_5 proportional to

- limited below 70% N

* does floating imply that AR opens/closes based on T_5 increasing? T_2 increasing changes

? 4) inhibit dry operation limit vent ... affect T_5 limiting value

Larger A_{80T} set to maintain T_{5LIM} but T_{5LIM} may be affected by T_2 .

adjustments for attitude, effects on nozzle area (minor effect)

DME/DGM Engine Laboratory	SUBJ: JBS-CAN-15 CONTROL	NAME: JWB
	TITLE: LOGIC	PROJECT NO.:
		DATE: 9 JAN 93
		PAGE: 1 OF: 2

ADDITIONAL MODES/NOTES

- A) **A: MFC**
 3 a) Proportional Speed Control
 (when measured and demanded speeds are nearly equal ~ at end of acceleration or during very slow acceleration)

$$N_s = f(PLA)$$

3 b) FRU selection / fuel selection

MFC/OSG

Test switch determines whether target max speed is 105% (normal operation) or test mode 95.5%

DME/DGM Engine Laboratory	SUBJECT:	EBS CONTROL LOGIC CONT'D	
	TITLE:		
	NAME:	JWB	
	PROJECT NO.:		
	DATE:		
	PAGE	2	OF 2

C: VEN

Not sure about mode 4: there is a mode where in dry operation there is a max rattle area

D: ABFC

The logic of the LFAB selection can be given in more detail like 3b suggested. I can give you more details if desired.

A.3 Engine Test Run Annotated by Domain Expert

71 May 3 15:31:19 1994

weight_of_fuel_flow))),

{pressurizing and drain valve,
is a(control_for_fuel_overflow)),
and is connected to main_fuel_control),
with_connectivity_parameter((weight_of_fuel_flow)))

refined text parameters

{main_fuel_control,
has_parameters((measured_rpm_engine_speed,
engine_spoil_engine_speed,
weight_of_fuel_flow))),

{main_fuel_control_fuel_filter,
has_parameters((weight_of_fuel_flow))),

{main_fuel_nozzles,
has_parameters((exhaust_gas_temperature,
measured_rpm_engine_speed,
engine_spoil_engine_speed,
weight_of_fuel_flow))),

{main_fuel_pump,
has_parameters((weight_of_fuel_flow))), + 2

{main_fuel_pump_fuel_filter,
has_parameters((weight_of_fuel_flow))),

{overspeed_governor,
has_parameters((measured_rpm_engine_speed,
engine_spoil_engine_speed,
weight_of_fuel_flow))),

{pressurizing and drain valve,
has_parameters((fuel_pump_inlet_pressure,
weight_of_fuel_flow)))

behaviour_gaps

{[main_fuel_control,
{gap_for_mode,steady_speed_control},
extracted,RPM,input,[N,T2],output,[Nd+WF/P3]],
[gap_for_mode,steady_speed_control},
extracted,RPM,input,[N,T2],output,[Nd+WF/P3]],
[gap_for_mode,steady_speed_control},
extracted,RPM,input,[N,T2],output,[Nd+WF/P3]]],

{main_fuel_control,
{gap_for_mode,speed_cutback_control},
extracted,RPM,input,[N,T2_limit],output,[Nd-]],
[gap_for_mode,speed_cutback_control},
extracted,N1,input,[N,T2_limit],output,[Nd-]],
[gap_for_mode,speed_cutback_control},
extracted,WF,input,[N,T2_limit],output,[Nd-]]],

{main_fuel_control,
{gap_for_mode,acceleration_fuel_limit_control},
extracted,RPM,input,[N,T2],output,[WF/P3]],
[gap_for_mode,acceleration_fuel_limit_control},

extracted,N1,input,[N,T2],output,[WF/P3+]],
[gap_for_mode,acceleration_fuel_limit_control},
extracted,WF,input,[N,T2],output,[WF/P3+]]],
[main_fuel_control,
{gap_for_mode,deceleration_fuel_limit_control},
extracted,RPM,input,[N-],output,[WF/P3-WF_min]],
[gap_for_mode,deceleration_fuel_limit_control},
extracted,N1,input,[N-],output,[WF/P3-WF_min]],
[gap_for_mode,deceleration_fuel_limit_control},
extracted,WF,input,[N-],output,[WF/P3-WF_min]]],

{main_fuel_control,
[gap_for_mode,variable_geometry_scheduling},
extracted,RPM,input,[N,T2],output,[IGV_bleed_valve_positions]],
[gap_for_mode,variable_geometry_scheduling},
extracted,N1,input,[N,T2],output,[IGV_bleed_valve_positions]],
[gap_for_mode,variable_geometry_scheduling},
extracted,WF,input,[N,T2],output,[IGV_bleed_valve_positions]]],

{main_fuel_control,
[gap_for_mode,proportional_speed_control},
extracted,RPM,input,[N,PLA],output,[WF/P3]],
[gap_for_mode,proportional_speed_control},
extracted,N1,input,[N,PLA],output,[WF/P3]],
[gap_for_mode,proportional_speed_control},
extracted,WF,input,[N,PLA],output,[WF/P3]]],

{main_fuel_control,
[gap_for_mode,fuel_selection},
extracted,RPM,input,[WF/P3,P3],output,[WF]],
[gap_for_mode,fuel_selection},
extracted,N1,input,[WF/P3,P3],output,[WF]],

{[main_fuel_control_fuel_filter,
[gap_for_mode,filtering_fuel},
extracted,WF,input,[WF+],output,[WF+]]],

{[main_fuel_nozzles,
[gap_for_mode,fuel_flow_control},
extracted,EGT,input,[N,T2],output,[WF+]],
[gap_for_mode,fuel_flow_control},
extracted,RPM,input,[N,T2],output,[WF+]],
[gap_for_mode,fuel_flow_control},
extracted,N1,input,[N,T2],output,[WF+]]],

{[main_fuel_pump,{[[]]],

{[main_fuel_pump_fuel_filter},
[gap_for_mode,filtering_fuel},
extracted,WF,input,[WF+],output,[WF+]]],

{[pressurizing and drain valve,{[gap_for_mode,fuel_overflow},
extracted,Pa,input,[Pa,WF,Pa_limit],output,[WF-]]},
[[]]]]

behaviour_list

- * JETA does not explicitly encode operational modes although the control K (ie reasoning) implicitly takes this into account

parameters

```
results in regulated output variable(Nd-),
at limit(T1_limit),decrease(Nd-),increases(T2_limit),increases(Nd-)]],
```

Fri May 6 15:31:19 1994

```

[[main fuel control,
 [[gap for mode, steady speed control,
  extracted, RPM, input, [N, P1A+], output, [Nd+, WF/P3+]],
 [[gap for mode, steady speed control,
  extracted, N1, input, [N, P1A+], output, [Nd+, WF/P3+]],
 [[gap for mode, steady speed control,
  extracted, WF, input, [N, P1A+], output, [Nd+, WF/P3+]]]],

```

```

[[main fuel control,
 [[gap for mode, speed cutback control,
  extracted, RPM, input, [N, T2 limit], output, [Nd-]],
 [[gap for mode, speed cutback control,
  extracted, N1, input, [N, T2 limit], output, [Nd-]],
 [[gap for mode, speed cutback control,
  extracted, WF, input, [N, T2 limit], output, [Nd-]]]],

```

```

[[main fuel control,
 [[gap for mode, acceleration fuel limit control,
  extracted, RPM, input, [N+, T2], output, [WF/P3+]],
 [[gap for mode, acceleration fuel limit control,
  extracted, N1, input, [N+, T2], output, [WF/P3+]],
 [[gap for mode, acceleration fuel limit control,
  extracted, WF, input, [N+, T2], output, [WF/P3+]]]],

```

```

[[main fuel control,
 [[gap for mode, deceleration fuel limit control,
  extracted, RPM, input, [N-], output, [WF/P3- WF_min]],
 [[gap for mode, deceleration fuel limit control,
  extracted, N1, input, [N-], output, [WF/P3- WF_min]],
 [[gap for mode, deceleration fuel limit control,
  extracted, WF, input, [N-], output, [WF/P3- WF_min]]]],

```

```

[[main fuel control,
 [[gap for mode, variable geometry scheduling,
  extracted, RPM, input, [N, T2], output, [IGV, bleed valve positions]],
 [[gap for mode, variable geometry scheduling,
  extracted, N1, input, [N, T2], output, [IGV bleed valve positions]],
 [[gap for mode, variable geometry scheduling,
  extracted, WF, input, [N, T2], output, [IGV, bleed valve positions]]]],

```

```

[[main fuel control,
 [[gap for mode, proportional speed control,
  extracted, RPM, input, [N, P1A], output, [WF/P3]],
 [[gap for mode, proportional speed control,
  extracted, N1, input, [N, P1A], output, [WF/P3]],
 [[gap for mode, proportional speed control,
  extracted, WF, input, [N, P1A], output, [WF/P3]]]],

```

```

[[main fuel control,
 [[gap for mode, fuel selection,
  extracted, RPM, input, [WF/P3, P3], output, [WF]],
 [[gap for mode, fuel selection,
  extracted, N1, input, [WF/P3, P3], output, [WF]]], [ ]]],

```

```

[[function (main fuel control fuel filter, filters (WF+)),
 input (main fuel control fuel filter, fluid (WF+)),
 output (main fuel control fuel filter, fluid (WF+)),
 regulator (main fuel control fuel filter, none),

```

```

behaviour (for (main fuel control fuel filter),
 behaviour is proportional (WF+, WF+),
 [[increase_in (WF+), increases (WF+), decrease_in (WF+), decreases (WF+)])),
 [[main fuel control fuel filter,
 [[gap for mode, filtering fuel,
  extracted, WF, input, [WF+], output, [WF+]]]]]],

```

```

[[function (main fuel nozzles, flow control (WF+)),
 input (main fuel nozzles, flow (WF+)),
 output (main fuel nozzles, flow (WF+)),
 regulator (main fuel nozzles, regulation control (N+)),
 behaviour (main fuel nozzles, behaviour is proportional (WF+, N+,
 [[increase_in (N+), increases (WF+), decrease_in (N+), decreases (WF+)])),

```

```

[[main fuel nozzles,
 [[gap for mode, fuel flow control,
  extracted, RPM, input, [N+, WF], output, [WF+]],
 [[gap for mode, fuel flow control,
  extracted, N1, input, [N+, WF], output, [WF+]],
 [[gap for mode, fuel flow control,
  extracted, WF, input, [N+, WF], output, [WF+]]], [ ]]],

```

```

[[function (main fuel pump, delivers (WF+)),
 input (main fuel pump, fluid (WF+)),
 output (main fuel pump, fluid (WF+)),
 regulator (main fuel pump, N1+),
 behaviour (for (main fuel pump), behaviour is proportional (WF+, N1+,
 [[increase_in (N1+), increases (WF+), decrease_in (N1+), decreases (WF+)])),
 [[main fuel pump, [ ]]]]],

```

```

[[function (main fuel pump fuel filter, filters (WF+)),
 input (main fuel pump fuel filter, fluid (WF+)),
 output (main fuel pump fuel filter, fluid (WF+)),
 regulator (main fuel pump fuel filter, none),
 behaviour (for (main fuel pump fuel filter),
 behaviour is proportional (WF+, WF+),
 [[increase_in (WF+), increases (WF+), decrease_in (WF+), decreases (WF+)])),
 [[main fuel pump fuel filter,
 [[gap for mode, filtering fuel,
  extracted, WF, input, [WF+], output, [WF+]]]]]],

```

```

[[function (pressurizing and drain valve, controls (WF-)),
 input (pressurizing and drain valve, control ([[Pa+, WF, Pa_limit]])),
 output (pressurizing and drain valve, control ([[WF-]])),
 regulator (pressurizing and drain valve, regulation control ([[Pa+, WF, Pa_limit]])),
 behaviour (pressurizing and drain valve,
 behaviour is piecewise linear ([[mode fuel_overflow],
 input control variable (Pa+)])),
 [[the behaviour is proportional for (fuel_overflow),
 the input control variable (Pa+),

```

```

results in regulated output variable(WF-),
at limit(Pa-),decrease(WF-),increases(Pa+),increases(WF-)),
[[the behaviour is proportional for(fuel_overflow),
the_input_control_variable(WF),
results in regulated output variable(WF-),
at limit(WF),decrease(WF-),increases(WF),increases(WF-)),
[[the behaviour is proportional for(fuel_overflow),
the_input_control_variable(Pa_limit),
results in regulated output variable(WF-),
at_limit(Pa_limit),decrease(WF-),increases(Pa_limit),increases(WF-))]])),
[[pressurizing and drain valve,
[[gap for mode,fuel_overflow,
extracted_Pa,input,[Pa+WF,Pa_limit],output,[WF-]],{}}]]]]

```

```

yes
| ?-

```

SUBJECT: J85 CONTROL MODE REVISED	NAME: JWB → SAH
TITLE:	PROJECT NO.:
	DATE: 27 MAY 74
	PAGE: 1 OF: 1

Notes:

- ① Some of the lesser items might be closed as processors or transformers, i.e.

fuel filter: processor : unplugged screen : just passes flow
 plugged screen : increased pressure loss as
 fuel bypass screen and indicator light?

fuel nozzles : present pressure drop to fuel system.
 require pressure in a fixed range to allow proper burning in combustor.

fuel pump transformer - increase fuel pressure $\propto$ engine speed

- ② Connections - are they to the source of the connectivity parameter or to source + sink

- ③ Parameters - should these include input as well as output
 then add inputs to MFC : T_{inlet}
 P_3

Fuel pump: N

I had trouble with the system of your code so I will give you functional relationships which should be known about the components, + I/O parameters.

won't affect MFC perf. as much
 E.D.V.

SUBJECT:

TITLE:

MFC modes

NAME:

PROJECT NO.:

DATE:

PAGE: 2 OF:

In all N is N_{measured}
 N_d is an internally calculated value

1) Speed Cut back

2) Proportional Control

3) Acceleration Control

4) Deceleration Control

$FRU_d = \text{constant}$.

5) MFC Fuel Selection

6) OSG

7) IGV / Bleed Valve

email any questions jeff.bird@nrc.ca

Appendix B: Coffee Maker Application

B.1 Coffee Maker Device Knowledge Base

```
defnode('coffee problem',
  [['coffee problem'],
    {display-name:'Problem with coffee maker.'},
    {help-info,
      'The coffee maker is used by the knowledge systems lab to brew coffee.'},
    {node-type,general},
    {child-node-of}}).

%%%% branch 1 of coffee filter problems based on taste problems %%%%

defnode('bad coffee taste',
  [['bad coffee or water taste'],
    {display-name:'Coffee has a bad taste problem.'},
    {help-info,
      'Someone has noted that the taste of the coffee is awful but does not know why.'},
    {node-type,general},
    {child-node-of,['coffee problem']},
    {child-node-ranking,['foul coffee taste','coffee taste too strong','coffee taste too weak']}).

%%%% sub-branch of bad taste problems %%%%

defnode('foul coffee taste',
  [['foul coffee or water taste'],
    {display-name:'Coffee has a foul taste problem.'},
    {help-info,
      'Someone has noted that the coffee taste is foul but does not know why.'},
    {node-type,general},
    {child-node-of,['bad coffee taste']},
    {child-node-ranking,['water resevoir']}).

%%%% sub-branch 1.1.1 of foul taste problem %%%%

defnode('water resevoir',
  [['water resevoir'],
    {display-name:'Change water in water resevoir for a new brew.'},
    {help-info,
      'Empty out water resevoir if needed and add new water before brewing new coffee batch.',
      'Use the basic recipe of 2 scoops of coffee to each cup of water for a medium strong',
      'brew and modify coffee to water ratio to strengthen or weaken as desired.'},
    {node-type,repair},
    {child-node-of,['foul coffee taste','grounds blocking coffee drip',
      'no water in water holder','heat problem: water too hot','coffee taste too weak: reduce water']},
    {child-node-ranking}}).

%%%% sub-branch 2.1 of bad taste problems %%%%

defnode('coffee taste too strong',
  [['coffee taste too strong'],
    {display-name:'Coffee is too strong.'},
    {help-info,'Someone has noted that the coffee is too strong.'},
    {node-type,general},
    {child-node-of,['bad coffee taste']},
    {child-node-ranking,['coffee filter','coffee holder']}).
```

```

defnode('coffee holder',
  [['coffee holder'],
    [display-name,'Change coffee in coffee holder for a new brew.'],
    [help-info,
      'Empty out coffee holder if needed and add new coffee and level it off before brewing.',
      'Use the basic recipe of 2 scoops of coffee to each cup of water for a medium strong ',
      'brew and modify coffee to water ratio to strengthen or weaken as desired.',
      'Make sure coffee is ground to a medium level, otherwise it may pass through filter.'],
    [node-type,repair],
    [child-node-of,['coffee taste too strong','coffee taste too weak: reduce water',
      'coffee holder overflowing','grounds blocking coffee drip']],
    [child-node-ranking]]).

defnode('coffee filter',
  [['coffee filter'],
    [display-name,'Change coffee filter before adding coffee.'],
    [help-info,
      'Empty out coffee holder if needed, put in a new filter suitable to the ',
      'coffee grind fineness and add new coffee and level it off before brewing.'],
    [node-type,replace],
    [child-node-of,['coffee taste too strong','no coffee filter','grounds blocking coffee drip']],
    [child-node-ranking]]).

%%%% sub-branch 3.1 of bad taste problems %%%

defnode('coffee taste too weak: reduce water',
  [['coffee taste too weak'],
    [display-name,'Coffee is too weak.'],
    [help-info,'Someone has noted that the coffee is too weak.'],
    [node-type,general],
    [child-node-of,['bad coffee taste']],
    [child-node-ranking,['coffee holder','water reservoir']])).

%%%% branch 2 of coffee maker problems based on wrong coffee temperature problems %%%

defnode('wrong coffee temperature',
  [['wrong coffee temperature'],
    [display-name,'Coffee temperature is off.'],
    [help-info,
      'Someone has noted that the temperature of the coffee is off but does not know why.'],
    [node-type,general],
    [child-node-of,['coffee problem']],
    [child-node-ranking,['heat problem: coffee too hot',
      'heat problem: water too hot','heat problem: coffee too cold','heat problem: water too cold']])).

%%%% sub-branch of wrong coffee temperature problems %%%

defnode('heat problem: coffee too hot',
  [['heat problem: coffee too hot'],
    [display-name,'Coffee temperature is too hot.'],
    [help-info,
      'Someone has noted that the temperature of the coffee is too hot.'],
    [node-type,general],
    [child-node-of,['wrong coffee temperature']],
    [child-node-ranking,['coffee pot heating element','coffee pot']])).

defnode('coffee pot heating element',
  [['coffee pot heating element'],
    [display-name,'Coffee heating element needs changing.'],
    [help-info,
      'The coffee heating element is no longer functioning properly. Buy a new element, unplug',
      'the old one and plug in the new one.'],
    [node-type,replace],
    [child-node-of,['heat problem: coffee too hot','heat problem: coffee too cold']],
    [child-node-ranking]]).

```

```

defnode('heat problem: water too hot',
  [['heat problem: water too hot'],
    [display-name,'Water temperature is too hot.'],
    [help-info,
      'Someone has noted that the temperature of the water is too hot.'],
    [node-type,general],
    [child-node-of,['wrong water temperature']],
    [child-node-ranking,['water reservoir','water heating element','water temperature heat control']]]).

defnode('water temperature heat control',
  [['water temperature heat control'],
    [display-name,'Water temperature control is not working properly. It needs changing.'],
    [help-info,
      'Water temperature control is no longer functioning properly. Buy a new control',
      'and install it. Note that this control makes sure the water drips when heated',
      'into the coffee holder.'],
    [node-type,replace],
    [child-node-of,['heat problem: water too hot','heat problem: water too cold']],
    [child-node-ranking]]).

defnode('water heating element',
  [['water heating element'],
    [display-name,'Water heating element needs changing.'],
    [help-info,
      'The water heating element is no longer functioning properly. Buy a new element, unplug',
      'the old one and plug in the new one.'],
    [node-type,replace],
    [child-node-of,['heat problem: water too hot','heat problem: water too cold']],
    [child-node-ranking]]).

%%%% sub-branch of wrong coffee temperature problems %%%

defnode('heat problem: coffee too cold',
  [['heat problem: coffee too cold'],
    [display-name,'Coffee temperature is too cold.'],
    [help-info,
      'Someone has noted that the temperature of the coffee is too cold.'],
    [node-type,general],
    [child-node-of,['wrong coffee temperature']],
    [child-node-ranking,['coffee maker plug','coffee maker switch','coffee pot heating element']]]).

defnode('heat problem: water too cold',
  [['heat problem: water too cold'],
    [display-name,'Coffee temperature is too cold because water is not heated.'],
    [help-info,
      'Someone has noted that the temperature of the coffee is too cold.'],
    [node-type,general],
    [child-node-of,['wrong coffee temperature']],
    [child-node-ranking,['water heating element','coffee maker plug','coffee maker switch',
      'water temperature heat control']]]).

%%%% branch 3 of coffee maker problems based on grounds in the coffee problems %%%

defnode('grounds in the coffee',
  [['grounds in the coffee'],
    [display-name,'Coffee has grounds in it.'],
    [help-info,'Someone has noted that there are grounds floating in the coffee.'],
    [node-type,general],
    [child-node-of,['coffee problem']],
    [child-node-ranking,['no coffee filter','coffee filter too porous','coffee holder overflowing']]])

```

%%%% sub-branch of coffee no filter problems %%%%

```
defnode('no coffee filter',
  [['no coffee filter'],
    [display-name,'Coffee holder is missing filter.'],
    [help-info,'Someone has noted that there is no filter in coffee holder.'],
    [node-type,general],
    [child-node-of,['grounds in the coffee']],
    [child-node-ranking,['coffee filter']]])
```

```
defnode('coffee filter too porous',
  [['coffee filter too porous'],
    [display-name,'Coffee filter is not fine enough.'],
    [help-info,'Someone has noted that the coffee grounds are passing through the filter.'],
    [node-type,general],
    [child-node-of,['grounds in the coffee']],
    [child-node-ranking,['coffee filter']]])
```

```
defnode('coffee holder overflowing',
  [['coffee holder overflowing'],
    [display-name,'Coffee holder is overfull.'],
    [help-info,'Someone has noted that the coffee grounds are spilling over the top of the ',
      'coffee holder.'],
    [node-type,general],
    [child-node-of,['grounds in the coffee']],
    [child-node-ranking,['coffee holder']]])
```

%%%% branch 4 of coffee maker problems based on coffee not dripping problems %%%%

```
defnode('coffee not dripping',
  [['coffee not dripping'],
    [display-name,'Coffee is not dripping through filter.'],
    [help-info,'Someone has noted that the coffee is not dripping but does not know why.'],
    [node-type,general],
    [child-node-of,['coffee problem']],
    [child-node-ranking,['no power','no water in water holder','water in water holder not dripping',
      'grounds blocking coffee drip']]])
```

%%%% sub-branch of no power problems %%%%

```
defnode('no power',
  [['no power'],
    [display-name,'Coffee power light is off.'],
    [help-info,'Someone has noted that the coffee maker light is off.'],
    [node-type,general],
    [child-node-of,['coffee not dripping']],
    [child-node-ranking,['coffee maker plug','coffee maker switch']]])
```

```
defnode('coffee maker plug',
  [['coffee maker plug'],
    [display-name,'Coffee maker is not plugged in.'],
    [help-info,'Someone has noted that the coffee maker light is off and it is unplugged.'],
    [node-type,repair],
    [child-node-of,['no power','heat problem: water too cold','heat problem: coffee too cold']],
    [child-node-ranking]])
```

```
defnode('coffee maker switch',
  [['coffee maker switch'],
    [display-name,'Coffee maker is plugged in and power light is off.'],
    [help-info,'Someone has noted that the coffee maker is plugged in but the light is off.'],
    [node-type,repair],
    [child-node-of,['no power','heat problem: water too cold','heat problem: coffee too cold']],
    [child-node-ranking]])
```

%%%% sub-branch of no no water in water holder problems %%%%

```
defnode('no water in water holder',
  [['no water in water holder'],
    {display-name,'Coffee drip is not working and water resevoir is empty.'},
    {help-info,'Someone has noted that the coffee maker is not dripping and that the ',
      'water resevoir is empty.'},
    {node-type,general},
    {child-node-of,['coffee not dripping']],
    {child-node-ranking,['water resevoir','coffee drip']}]])
```

```
defnode('water in water holder not dripping',
  [['water in water holder not dripping'],
    {display-name,'Water resevoir is full but water is not drnpping to coffee holder.'},
    {help-info,'Someone has noted that the water is not dripping and that the ',
      'water resevoir is full '},
    {node-type,general},
    {child-node-of,['coffee not dripping']],
    {child-node-ranking,['water resevoir','coffee drip','water temperature heat control']}]])
```

%%%% sub-branch of plugged drip problems %%%%

```
defnode('grounds blocking coffee drip',
  [['grounds blocking coffee drip'],
    {display-name,'Coffee drip needs unstopping.'},
    {help-info,'Someone has noted that the coffee maker is not dripping and that the ',
      'drip is plugged.'},
    {node-type,general},
    {child-node-of,['coffee not dripping']],
    {child-node-ranking,['coffee holder','coffee drip','coffee filter','water resevoir','coffee pot']}]])
```

```
defnode('coffee drip',
  [['coffee drip'],
    {display-name,'Unclog coffee drip.'},
    {help-info,'Unclog drip by inserting toothpick or pin in hole through which coffee passes.'},
    {node-type,repair},
    {child-node-of,['grounds blocking coffee drip']],
    {child-node-ranking[]}]])
```

```
defnode('coffee pot',
  [['coffee pot'],
    {display-name,'Make sure coffee pot is on pot heating element, otherwise coffee will not drip.'},
    {help-info,
      'Empty coffee pot and place on heating element..'},
    {node-type,repair},
    {child-node-of,['coffee not dripping','heat problem: coffee too hot']},
    {child-node-ranking[]}]])
```

B.2 DR Coffee Maker Test Run

Coffee Example ---> May 18/94, 18:28

```
[41]ksl0cjsuhayya* cd prolog/DR
/home/ksl00/suhayya/prolog/DR
[42]ksl0cjsuhayya* prolog
Quintus Prolog Release 3.1.3 (Sun-4, SunOS 4.1)
Copyright (C) 1993, Quintus Corporation. All rights reserved.
2100 Geng Road, Palo Alto, California U.S.A. (415) 813-3800

| ?- ensure_loaded(match_2).
% compiling file /home/ksl00/suhayya/prolog/DR/match_2.pl
% compiling file /home/ksl00/suhayya/prolog/DR/get_param_2.pl
% compiling file /home/ksl00/suhayya/prolog/DR/get_frames_2.pl
% compiling file /home/ksl00/suhayya/prolog/DR/get_prolog_2_frames.pl
% compiling file /home/ksl00/suhayya/prolog/DR/kb_2_prolog.pl
% kb_2_prolog.pl compiled in module user, 1.884 sec 13,948 bytes
% get_prolog_2_frames.pl compiled in module user, 2.000 sec 15,012 bytes
% loading file /home/SOFTWARE/quintus/generic/qplib3.1.3/library/basics.qof
% basics.qof loaded, 0.100 sec -3,412 bytes
% module basics imported into user
% loading file /home/SOFTWARE/quintus/generic/qplib3.1.3/library/sets.qof
% sets.qof loaded, 0.184 sec 8,556 bytes
% module sets imported into user
% loading file /home/SOFTWARE/quintus/generic/qplib3.1.3/library/listparts.qof
% listparts.qof loaded, 0.100 sec 4,604 bytes
% module list_parts imported into user
% get_frames_2.pl compiled in module user, 2.667 sec 27,906 bytes
% compiling file /home/ksl00/suhayya/prolog/DR/get_con_2.pl
% loading file /home/SOFTWARE/quintus/generic/qplib3.1.3/library/lists.qof
% loading file /home/SOFTWARE/quintus/generic/qplib3.1.3/library/types.qof
% types.qof loaded in module lists, 0.067 sec 7,196 bytes
% lists.qof loaded, 0.334 sec 20,348 bytes
% module lists imported into user
* Singleton variables, clause 3 of check_child_node_of/4: C
* Approximate line: 57, file: /home/ksl00/suhayya/prolog/DR/get_con_2.pl
% get_con_2.pl compiled in module user, 0.767 sec 23,492 bytes
% loading file /home/SOFTWARE/quintus/generic/qplib3.1.3/library/flatten.qof
% flatten.qof loaded, 0.133 sec 4,760 bytes
% module flatten imported into user
* Singleton variables, clause 2 of check_for_activating_observations/4: Param_list, B
* Approximate line: 140, file: /home/ksl00/suhayya/prolog/DR/get_param_2.pl
* Singleton variables, clause 3 of access_activation_slot/2: FS
* Approximate line: 150, file: /home/ksl00/suhayya/prolog/DR/get_param_2.pl
% get_param_2.pl compiled in module user, 4.117 sec 50,704 bytes
* Singleton variables, clause 1 of test_match/0: Text_comp_con, Text_comp_param
* Approximate line: 71, file: /home/ksl00/suhayya/prolog/DR/match_2.pl
% match_2.pl compiled in module user, 5.850 sec 69,652 bytes

yes
| ?- test_match.
```

components

[coffee drip,coffee filter,coffee holder,coffee maker plug,coffee maker switch,coffee pot,coffee pot heating element,
water heating element,water reservoir,water temperature heat control]

component_connections

```
[[[coffee filter,coffee holder,coffee taste too strong],
[coffee drip,coffee holder,grounds blocking coffee drip],
[coffee filter,coffee holder,grounds blocking coffee drip],
[water reservoir,coffee holder,grounds blocking coffee drip],
[coffee pot,coffee holder,grounds blocking coffee drip]]_12144],

[[coffee holder,coffee drip,grounds blocking coffee drip],
[coffee filter,coffee drip,grounds blocking coffee drip],
[water reservoir,coffee drip,grounds blocking coffee drip],
[coffee pot,coffee drip,grounds blocking coffee drip]]_11798],
```

[[coffee holder,coffee filter,coffee taste too strong],
 [coffee holder,coffee filter,grounds blocking coffee drip],
 [coffee drip,coffee filter,grounds blocking coffee drip],
 [water reservoir,coffee filter,grounds blocking coffee drip],
 [coffee pot,coffee filter,grounds blocking coffee drip]]_11968],

 [[coffee holder,water reservoir,grounds blocking coffee drip],
 [coffee drip,water reservoir,grounds blocking coffee drip],
 [coffee filter,water reservoir,grounds blocking coffee drip],
 [coffee pot,water reservoir,grounds blocking coffee drip],
 [coffee drip,water reservoir,no water in water holder],
 [water heating element,water reservoir,heat problem: water too hot],
 [water temperature heat control,water reservoir,heat problem: water too hot]]_13092],

 [[coffee maker plug,coffee maker switch,no power],
 [water heating element,coffee maker switch,heat problem: water too cold],
 [coffee maker plug,coffee maker switch,heat problem: water too cold],
 [water temperature heat control,coffee maker switch,heat problem: water too cold],
 [coffee maker plug,coffee maker switch,heat problem: coffee too cold],
 [coffee pot heating element,coffee maker switch,heat problem: coffee too cold]]_12498],

 [[coffee maker switch,coffee maker plug,no power],
 [water heating element,coffee maker plug,heat problem: water too cold],
 [coffee maker switch,coffee maker plug,heat problem: water too cold],
 [water temperature heat control,coffee maker plug,heat problem: water too cold],
 [coffee maker switch,coffee maker plug,heat problem: coffee too cold],
 [coffee pot heating element,coffee maker plug,heat problem: coffee too cold]]_12321],
 [[coffee pot,coffee pot heating element,heat problem: coffee too hot],
 [coffee maker plug,coffee pot heating element,heat problem: coffee too cold],
 [coffee maker switch,coffee pot heating element,heat problem: coffee too cold]]_12746],

 [[coffee pot heating element,coffee pot,heat problem: coffee too hot]]_12612],

 [[water reservoir,water heating element,heat problem: water too hot],
 [water temperature heat control,water heating element,heat problem: water too hot],
 [coffee maker plug,water heating element,heat problem: water too cold],
 [coffee maker switch,water heating element,heat problem: water too cold],
 [water temperature heat control,water heating element,heat problem: water too cold]]_12898],

 [[water reservoir,water temperature heat control,heat problem: water too hot],
 [water heating element,water temperature heat control,heat problem: water too hot],
 [water heating element,water temperature heat control,heat problem: water too cold],
 [coffee maker plug,water temperature heat control,heat problem: water too cold],
 [coffee maker switch,water temperature heat control,heat problem: water too cold]]_13246]]

component_parameter_pairs

[[['coffee,coffee filter],[coffee holder,coffee]],
 [[coffee,coffee drip],[coffee holder,coffee]],
 [[coffee,coffee filter],[coffee holder,coffee]],
 [[coffee,heat,water,water reservoir],[coffee holder,coffee]],
 [[coffee,coffee pot,heat],[coffee holder,coffee]],
 [[['coffee,coffee holder,water],[coffee drip,coffee]],
 [[coffee,coffee filter],[coffee drip,coffee]],
 [[coffee,heat,water,water reservoir],[coffee drip,coffee]],
 [[coffee,coffee pot,heat],[coffee drip,coffee]],
 [[['coffee,coffee holder,water],[coffee filter,coffee]],
 [[coffee,coffee holder,water],[coffee filter,coffee]],
 [[coffee,coffee drip],[coffee filter,coffee]],


```

[[coffee,heat,water,water reservoir],[coffee filter,coffee]],
[[coffee,coffee pot,heat],[coffee filter,coffee]],
[[coffee,coffee holder,water],[water reservoir,coffee]],
[[coffee,coffee drip],[water reservoir,coffee]],
[[coffee,coffee filter],[water reservoir,coffee]],
[[coffee,coffee pot,heat],[water reservoir,coffee]],
[[coffee,coffee drip,water],[water reservoir,water]],
[[heat,water,water heating element],[water reservoir,water,heat]],
[[heat,water,water temperature heat control],[water reservoir,water,heat]],
[[coffee,coffee maker plug,heat,power,water],[coffee maker switch,power]],
[[heat,water,water heating element],[coffee maker switch,water,heat]],
[[coffee,coffee maker plug,heat,power,water],[coffee maker switch,water,heat]],
[[heat,water,water temperature heat control],[coffee maker switch,water,heat]],
[[coffee,coffee maker plug,heat,power,water],[coffee maker switch,coffee,heat]],
[[coffee,coffee pot heating element,heat],[coffee maker switch,coffee,heat]],
[[coffee,coffee maker switch,heat,power,water],[coffee maker plug,power]],
[[heat,water,water heating element],[coffee maker plug,water,heat]],
[[coffee,coffee maker switch,heat,power,water],[coffee maker plug,water,heat]],
[[heat,water,water temperature heat control],[coffee maker plug,water,heat]],
[[coffee,coffee maker switch,heat,power,water],[coffee maker plug,coffee,heat]],
[[coffee,coffee pot heating element,heat],[coffee maker plug,coffee,heat]],
[[coffee,coffee pot,heat],[coffee pot heating element,coffee,heat]],
[[coffee,coffee maker plug,heat,power,water],[coffee pot heating element,coffee,heat]],
[[coffee,coffee maker switch,heat,power,water],[coffee pot heating element,coffee,heat]],
[[coffee,coffee pot heating element,heat],[coffee pot,coffee,heat]],
[[coffee,heat,water,water reservoir],[water heating element,water,heat]],
[[heat,water,water temperature heat control],[water heating element,water,heat]],
[[coffee,coffee maker plug,heat,power,water],[water heating element,water,heat]],
[[coffee,coffee maker switch,heat,power,water],[water heating element,water,heat]],
[[heat,water,water temperature heat control],[water heating element,water,heat]],
[[coffee,heat,water,water reservoir],[water temperature heat control,water,heat]],
[[heat,water,water heating element],[water temperature heat control,water,heat]],
[[heat,water,water heating element],[water temperature heat control,water,heat]],
[[coffee,coffee maker plug,heat,power,water],[water temperature heat control,water,heat]],
[[coffee,coffee maker switch,heat,power,water],[water temperature heat control,water,heat]]]

```

refined_text_component_connections

```

[[coffee drip,is_a([regulator,for,[coffee]]),and_is_connected_to(coffee filter),with_connectivity_parameter([coffee])],

[coffee drip,is_a([regulator,for,[coffee]]),and_is_connected_to(coffee holder),with_connectivity_parameter([coffee])],

[coffee drip,is_a([regulator,for,[coffee]]),and_is_connected_to(water reservoir),with_connectivity_parameter([coffee,water])],

[coffee filter,is_a([filter,for,[grounds]]),and_is_connected_to(coffee drip),with_connectivity_parameter([coffee])],

[coffee filter,is_a([filter,for,[grounds]]),and_is_connected_to(coffee holder),with_connectivity_parameter([coffee])],

[coffee filter,is_a([filter,for,[grounds]]),and_is_connected_to(water reservoir),with_connectivity_parameter([coffee])],

[coffee holder,is_a([holder,for,[grounds]]),and_is_connected_to(coffee drip),with_connectivity_parameter([coffee])],

[coffee holder,is_a([holder,for,[grounds]]),and_is_connected_to(coffee filter),with_connectivity_parameter([coffee])],

[coffee holder,is_a([holder,for,[grounds]]),and_is_connected_to(water reservoir),with_connectivity_parameter([coffee])],

[coffee maker plug,is_a([regulator,for,[power]]),and_is_connected_to(coffee maker switch),
with_connectivity_parameter([coffee,heat,power,water])],

```

[coffee maker plug,is_a([regulator,for,[power]]),and_is_connected_to(coffee pot heating element),
with_connectivity_parameter([coffee,heat]]),

[coffee maker plug,is_a([regulator,for,[power]]),and_is_connected_to(water heating element),
with_connectivity_parameter([heat,water]]),

[coffee maker plug,is_a([regulator,for,[power]]),and_is_connected_to(water temperature heat control),
with_connectivity_parameter([heat,water]]),

[coffee maker switch,is_a([switch,for,[power]]),and_is_connected_to(coffee maker plug),
with_connectivity_parameter([coffee,heat,power,water]]),

[coffee maker switch,is_a([switch,for,[power]]),and_is_connected_to(coffee pot heating element),
with_connectivity_parameter([coffee,heat]]),

[coffee maker switch,is_a([switch,for,[power]]),and_is_connected_to(water heating element),
with_connectivity_parameter([heat,water]]),

[coffee maker switch,is_a([switch,for,[power]]),and_is_connected_to(water temperature heat control),
with_connectivity_parameter([heat,water]]),

[coffee pot,is_a([holder,for,[coffee]]),and_is_connected_to(coffee drip),with_connectivity_parameter([coffee]]),

[coffee pot,is_a([holder,for,[coffee]]),and_is_connected_to(coffee filter),with_connectivity_parameter([coffee]]),

[coffee pot,is_a([holder,for,[coffee]]),and_is_connected_to(coffee holder),with_connectivity_parameter([coffee]]),

[coffee pot,is_a([holder,for,[coffee]]),and_is_connected_to(coffee pot heating element),with_connectivity_parameter([coffee,heat]]),

[coffee pot,is_a([holder,for,[coffee]]),and_is_connected_to(water reservoir),with_connectivity_parameter([coffee]]),

[coffee pot heating element,is_a([heater,for,[coffee]]),and_is_connected_to(coffee maker plug),
with_connectivity_parameter([coffee,heat]]),

[coffee pot heating element,is_a([heater,for,[coffee]]),and_is_connected_to(coffee maker switch),
with_connectivity_parameter([coffee,heat]]),

[coffee pot heating element,is_a([heater,for,[coffee]]),and_is_connected_to(coffee pot),
with_connectivity_parameter([coffee,heat]]),

[water heating element,is_a([heater,for,[water]]),and_is_connected_to(coffee maker plug),
with_connectivity_parameter([heat,water]]),

[water heating element,is_a([heater,for,[water]]),and_is_connected_to(coffee maker switch),
with_connectivity_parameter([heat,water]]),

[water heating element,is_a([heater,for,[water]]),and_is_connected_to(water reservoir),
with_connectivity_parameter([heat,water]]),

[water heating element,is_a([heater,for,[water]]),and_is_connected_to(water temperature heat control),
with_connectivity_parameter([heat,water]]),

[water reservoir,is_a([holder,for,[water]]),and_is_connected_to(coffee drip),with_connectivity_parameter([coffee]]),

[water reservoir,is_a([holder,for,[water]]),and_is_connected_to(coffee filter),with_connectivity_parameter([coffee]]),

[water reservoir,is_a([holder,for,[water]]),and_is_connected_to(coffee holder),with_connectivity_parameter([coffee]]),

```
[water reservoir, is_a([holder, for, [water]]), and_is_connected_to(water heating element), with_connectivity_parameter([heat, water]))],
```

```
[water reservoir, is_a([holder, for, [water]]), and_is_connected_to(water temperature heat control),  
with_connectivity_parameter([heat, water]))],
```

```
[water temperature heat control, is_a([regulator, for, [water]]), and_is_connected_to(coffee maker plug),  
with_connectivity_parameter([heat, water]))],
```

```
[water temperature heat control, is_a([regulator, for, [water]]), and_is_connected_to(coffee maker switch),  
with_connectivity_parameter([heat, water]))],
```

```
[water temperature heat control, is_a([regulator, for, [water]]), and_is_connected_to(water heating element),  
with_connectivity_parameter([heat, water]))],
```

```
[water temperature heat control, is_a([regulator, for, [water]]), and_is_connected_to(water  
reservoir), with_connectivity_parameter([heat, water]))]
```

refined_text_parameters

```
[[coffee drip, has_parameters([coffee])],  
[coffee filter, has_parameters([coffee])],  
[coffee holder, has_parameters([coffee])],  
[coffee maker plug, has_parameters([coffee, heat, power, water])],  
[coffee maker switch, has_parameters([coffee, heat, power, water])],  
[coffee pot, has_parameters([coffee, heat])],  
[coffee pot heating element, has_parameters([coffee, heat])],  
[water heating element, has_parameters([heat, water])],  
[water reservoir, has_parameters([coffee, heat, water])],  
[water temperature heat control, has_parameters([heat, water])]]
```

device_component_behaviour_models

```
[[[function(coffee drip, regulates(coffee+)),  
input(coffee drip, coffee+),  
output(coffee drip, coffee+),  
regulator(coffee drip, coffee+),  
behaviour(for(coffee drip),  
behaviour_is_proportional([increase_in(coffee+), increases(coffee+), decrease_in(coffee+), decreases(coffee+)])),  
[[coffee drip, [[gap, extracted, coffee, input, [coffee+], output, [coffee+]]]]]],
```

```
[function(coffee filter, filters(coffee+)),  
input(coffee filter, filter([grounds]), fluid(water+)),  
output(coffee filter, fluid(coffee+)),  
regulator(coffee filter, water+),  
behaviour(for(coffee filter),  
behaviour_is_proportional([increase_in(water+), increases(coffee+), decrease_in(water+), decreases(coffee+)])),  
[[coffee filter, [[gap, extracted, coffee, input, [water+], output, [coffee+]]]]],
```

```
[function(coffee holder, vessel_for(water+)),  
input(coffee holder, is(water+)),  
output(coffee holder, is(coffee+)),  
regulator(coffee holder, none),  
behaviour(for(coffee holder),  
behaviour_is_proportional([increase_in(water+), increases(coffee+), decrease_in(water+), decreases(coffee+)])),  
[[coffee holder, [[gap, extracted, coffee, input, [water+], output, [coffee+]]]]],
```

```

(function(coffee maker plug,regulates(power+)),
input(coffee maker plug,power+),
output(coffee maker plug,power+),
regulator(coffee maker plug,power+),
behaviour(for(coffee maker plug),
behaviour_is_proportional([increase_in(power+),increases(power+),decrease_in(power+),decreases(power+)])),
[[coffee maker plug,[[gap,extracted,coffee,input,[power+],output,[power+]],gap,extracted,heat,input,[power+],output,[power+]],
[gap,extracted,power,input,[power+],output,[power+]],gap,extracted,water,input,[power+],output,[power+]]]]].

```

```

(function(coffee maker switch,has_states(on/off)),
input(coffee maker switch,input_states(on/off)),
output(coffee maker switch,output_states(power/no power)),
regulator(coffee maker switch,none),
behaviour(for(coffee maker switch),
behaves_according_to_state(on/off,implies_respective(power/no power))),
[[coffee maker switch,[[gap,extracted,coffee,input,[on/off],output,[power/no power]],
[gap,extracted,heat,input,[on/off],output,[power/no power]],gap,extracted,power,input,[on/off],output,[power/no power]],
[gap,extracted,water,input,[on/off],output,[power/no power]]]]].

```

```

(function(coffee pot,vessel_for(coffee)),
input(coffee pot,is(coffee)),
output(coffee pot,is(coffee)),
regulator(coffee pot,none),
behaviour(for(coffee pot),behaviour_is_proportional([increase_in(coffee),increases(coffee),decrease_in(coffee),decreases(coffee)])),
[[coffee pot,[[gap,extracted,heat,input,[coffee],output,[coffee]]]]].

```

```

(function(coffee pot heating element,heat([coffee])),
input(coffee pot heating element,is(power+)),
output(coffee pot heating element,is(heat+)),
regulator(coffee pot heating element,power+),
behaviour(for(coffee pot heating element),
behaviour_is_proportional([increase_in(power+),increases(heat+),decrease_in(power+),decreases(heat+)])),
[[coffee pot heating element,[[gap,extracted,coffee,input,[power+],output,[heat+]],gap,extracted,heat,input,[power+],output,[heat+]]]]].

```

```

(function(water heating element,heat([water])),
input(water heating element,is(power+)),
output(water heating element,is(heat+)),
regulator(water heating element,power+),
behaviour(for(water heating element),
behaviour_is_proportional([increase_in(power+),increases(heat+),decrease_in(power+),decreases(heat+)])),
[[water heating element [[gap,extracted,heat,input,[power+],output,[heat+]],gap,extracted,water,input,[power+],output,[heat+]]]]].

```

```

(function(water reservoir,vessel_for(water)),
input(water reservoir,is(water)),
output(water reservoir,is(water)),
regulator(water reservoir,none),
behaviour(for(water reservoir),
behaviour_is_proportional([increase_in(water),increases(water),decrease_in(water),decreases(water)])),
[[water reservoir,[[gap,extracted,coffee,input,[water],output,[water]],gap,extracted,heat,input,[water],output,[water]]]]].

```

```

(function(water temperature heat control,regulates(heat+)),
input(water temperature heat control,heat+),
output(water temperature heat control,water+),
regulator(water temperature heat control,heat+),
behaviour(for(water temperature heat control),
behaviour_is_proportional([increase_in(heat+),increases(water+),decrease_in(heat+),decreases(water+)])),
[[water temperature heat control,[[gap,extracted,heat,input,[heat+],output,[water+]],gap,extracted,water,input,[heat+],output,[water+]]]]].

```

yes
| ?-

END

2 5-0 5-9 5

FIN