KNOWLEDGE-BASED ARCHITECTURE FOR INTEGRATED CONDITION BASED MAINTENANCE OF ENGINEERING SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Abhinav Saxena

In partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy in Electrical and Computer Engineering

Georgia Institute of Technology

August, 2007

KNOWLEDGE-BASED ARCHITECTURE FOR INTEGRATED CONDITION BASED MAINTENANCE OF ENGINEERING SYSTEMS

Approved By

Dr. George Vachtsevanos
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Aldo A. Ferri
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ayanna Howard
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ashraf Saad*
School of Computing
Armstrong Atlantic State University
*Formerly with the *Georgia Institute of Technology*

Date Approved: May 04, 2007

To my parents

Madhu Saxena

And

Kailash Chandra Saxena

Acknowledgement

I extend my sincere gratitude and appreciation to many people who made this research possible. Special thanks are due to my advisor Dr. George Vachtsevanos for his continual guidance, willing advice and encouragement. I would also like to thank Dr. Magnus Egerstedt, Dr. Ashraf Saad, Dr. Aldo Ferri, and Dr. Ayanna Howard for being a member on my thesis committee, their wishful support, and constructive feedback whenever I needed it.

Many thanks go to the members of Intelligent Control Systems Lab who have been a continual support throughout this research. I would like to acknowledge Dr. Biqing Wu, Dr. Liang Tang, Dr. George Georgoulas and Dr. Bin Zhang for their guidance. Special thanks go to Marcos Orchard and Romano Patrick with whom I have been involved in technical discussions from time to time. I would also like to acknowledge other members of the lab who made it a wonderful experience in the lab.

I would also like to acknowledge with much appreciation the crucial roles played by Dr. Irtaza Barlas, Dr. J.L. Dorrity and Dr. Antonio Ginart from Impact Technologies Inc. and Mr. Gary O'Niel from Georgia Tech Research Institute (GTRI) for providing me research data and their technical inputs.

And last but not the least my friends deserve my gratitude and appreciation for they are responsible for an invaluable learning experience at Georgia Tech. Special thanks go to Manas Bajaj and Lalit Bohra their continual support and valuable friendship.

Table of Contents

ACKNOWLEDGEMENT IV

LIST OF TABLES VIII

LIST OF FIGURES IX

LIST OF ABBREVIATIONS XIII

SUMMARY XV

CHAPTER 1 CONDITION BASED MAINTENANCE OF ENGINEERING SYSTEMS 1

1.1 RESEARCH CONTEXT 2

1.1.1 The Problem 2

1.1.2 Research Proposition 4

1.2 SIGNIFICANCE OF THE RESEARCH 5

1.2.1 Integrating Knowledge into CBM: A Decision Support System 7

1.3 THESIS ORGANIZATION 10

CHAPTER 2 KNOWLEDGE-BASED INTEGRATED CBM ARCHITECTURE 14

2.1 CHAPTER OVERVIEW 15

2.2 CURRENT STATE-OF-THE-ART 15

2.3 KEY DRIVERS FOR CBM ENHANCEMENTS 17

2.4 AN APPROACH TO KNOWLEDGE INTEGRATION INTO CBM 19

2.4.1 Diagnostic Knowledge Management 21

2.4.2 Self-Evolving Maintenance Knowledgebase 23

2.4.3 A Decision Support System Based on Higher Level Reasoning 24

CHAPTER 3 KNOWLEDGE ENGINEERING 26

3.1 CHAPTER OVERVIEW 27

3.2 INTRODUCTION 27

3.3 KNOWLEDGE ACQUISITION 31

3.3.1 Diagnostic Data Collection 32

3.3.2 Experience Accumulation 34

3.3.3 A Systematic Approach to Knowledge Acquisition 35

3.4 KNOWLEDGE CODIFICATION 52
3.4.1 Process of Codification 53
3.4.2 Knowledge Model 55
3.4.3 Representation Language 57
3.5 CASE STUDIES 59
3.5.1 Case: Sludge Dewatering Centrifuges 60
3.6 CONCLUSIONS 67

CHAPTER 4 DYNAMIC CASE-BASED REASONING 68
4.1 CHAPTER OVERVIEW 69
4.2 WHY CBR AS KNOWLEDGE MODEL FOR CBM? 69
4.3 CASE BASED REASONING – AN ENABLING TECHNOLOGY 71
4.3.1 CBR Implementation Issues 74
4.3.2 The Issue of Uncertainty in CBR 77
4.3.3 CBR in Industrial Practice – State-of-the-Art 78
4.3.4 Shortcomings of the Conventional CBR 80
4.4 DYNAMIC CASE-BASED REASONING 81
4.4.1 The DCBR Lifecycle 81
4.4.2 Three Dynamic Components of DCBR 84
4.5 DCBR APPLICATION TO FLEET VEHICLES 87
4.5.1 The Dynamic Case 89
4.5.2 The Other Two Dynamic Components 93
4.6 EVALUATING DCBR PERFORMANCE 93
4.7 CONCLUSIONS 94

CHAPTER 5 INDUSTRIAL LANGUAGE PROCESSING 95
5.1 CHAPTER OVERVIEW 96
5.2 MOTIVATION 96
5.3 CURRENT STATE-OF-THE-ART 97
5.4 THE CONCEPT OF STANDARDIZED LANGUAGE 99
5.4.1 Standardized Languages in Industrial Domain 99
5.4.2 Description of Standardized Language for CBM 101
5.5 MEANING AMBIGUITY REDUCTION 103
5.6 INFORMATION EXTRACTION (IE) 104
5.6.1 Parsing Fault Symptoms 105
5.6.2 Data Sanitization 106
5.7 DATA REPRESENTATION 108
5.8 SIMILARITY CALCULATIONS 110

5.8.1 Similarity Concept for Triad Structure ... 111
5.9 CASE STUDIES ... 113
5.9.1 Automotive Maintenance ... 113
5.9.2 Sludge Dewatering Centrifuge ... 114
5.9.3 Discussion ... 117
5.10 EVALUATING ILP PERFORMANCE ... 117
5.11 CONCLUSIONS ... 119

CHAPTER 6 KNOWLEDGE MANAGEMENT SYSTEM ... 120
6.1 CHAPTER OVERVIEW ... 121
6.2 INTRODUCTION ... 121
6.3 ATTRIBUTES OF INTELLIGENCE IN A KNOWLEDGEBASE ... 122
6.3.1 Dynamic Structure ... 123
6.3.2 Self-Evaluation ... 127
6.3.3 Learning ... 128
6.3.4 Self-Maintenance ... 146
6.3.5 Autonomous Behavior ... 146
6.4 SIMULATION RESULTS AND DISCUSSION ... 147
6.5 THE KNOWLEDGE MANAGEMENT INTERFACE ... 153
6.6 EVALUATING KB PERFORMANCE ... 158
6.7 CONCLUSIONS ... 158

CHAPTER 7 CONCLUSIONS AND FUTURE WORK ... 159
7.1 CONCLUDING REMARKS ... 160
7.2 SUMMARY OF CONTRIBUTIONS ... 162
7.3 SUGGESTED FUTURE WORK ... 163
7.3.1 Guaranteeing Overall System Soundness ... 164
7.3.2 DCBR Extension to Prognosis ... 165
7.3.3 Improving ILP ... 165
7.3.4 Performance Metrics ... 166

APPENDIX A KNOWLEDGE, INFORMATION, AND DATA ... 167

APPENDIX B FMECA FOR MONOPROPELLANT PROPULSION SYSTEM (MPS) ... 169

APPENDIX C NOTIONS OF SIMILARITY ... 175

REFERENCES ... 179

List of Tables

Table 3.1 Different types of inference methods. 57

Table 3.2 FMECA chart for the separator unit... 65

Table 3.3 List of sensors on the separator unit... 66

Table 4.1 Types-of-cases classification. 75

Table 5.1 The standard language template for CBM of automotive and centrifuge cases. 102

Table 5.2 Examples to show standardized language translation of symptoms from automotive and centrifuge cases. 103

Table 5.3 Three conceptual *relations* capture most scenarios in industrial descriptions of failures.. 108

Table 5.4 Most commonly used similarity composition functions. 111

Table 5.5 Similarity calculation for triads-based semantic networks. 112

Table C.1 Generic similarity measures that can be composed to create custom similarity metrics... 177

List of Figures

Figure 1.1 A real world example to show significance of the research. 6

Figure 1.2 Transition from expert dependent legacy systems to knowledge based autonomous decision support systems for CBM. 9

Figure 1.3 Thesis Organization. 12

Figure 2.1 Paradigm shift in industrial maintenance. 18

Figure 2.2 CBM+: A maintenance centric approach to CBM adapted from [13]. 19

Figure 2.3 Knowledge integrated CBM+ architecture. 20

Figure 2.4 Key drivers and corresponding research goals: an overview. 21

Figure 2.5 System-of-systems hierarchy. 22

Figure 3.1 Knowledge Engineering: Task planning and execution adapted from [15]. .. 29

Figure 3.2 Main tasks involved in Knowledge Engineering (KE). 30

Figure 3.3 Sources of information for knowledge acquisition. 31

Figure 3.4 Types of data to be collected for building diagnostic knowledgebase. 32

Figure 3.5 Experience is accumulated as historical records and cause-effect associations. 34

Figure 3.6 A stepwise approach to FMECA study for CBM systems. 40

Figure 3.7 System analysis steps for FMECA study. 41

Figure 3.8 Hierarchical tree diagram to show structural decomposition of a system [19]. 42

Figure 3.9 Structural block diagram to show structural interrelationship between different components. 43

Figure 3.10 Functional block diagram adds input condition and sensor information along with the function of the component. 43

Figure 3.11 Functional block diagram of the system. 44

Figure 3.12 A generic FMECA worksheet with most relevant columns. (Adapted from various sources) 45

Figure 3.13 (a) A four category classification. (b) An alternative frequency of failure classification. 46

Figure 3.14 An example of severity classification [1]. 47

Figure 3.15 An example of detectability ranking method [19]. 49

Figure 3.16 Risk matrix to assess risk ranking. 50

Figure 3.17 Topography of knowledge types. 53

Figure 3.18 The system - sludge dewatering centrifuge. 60

Figure 3.19 Separator of the sludge dewatering centrifuge. 61

Figure 3.20 Structural decomposition of the separator. 62

Figure 3.21 Structural block diagram of the separator. 63

Figure 3.22 Functional block diagram of the separator. 63

Figure 4.1 The CBR Cycle, adapted from [27]. 72

Figure 4.2 Lifecycle of a DCBR system. 81

Figure 4.3 Creating and updating a dynamic case using a statistic vector. Descriptive features can be converted to numerical values using techniques like fuzzification. 83

Figure 4.4 High level schematic of Dynamic Case Based Reasoning highlighting its dynamic components. 86

Figure 4.5 Integrated reasoning architecture for fault diagnosis in industrial environments. 88

Figure 4.6 Generic dynamic case structure. Symptoms are stored as semantic networks. 90

Figure 4.7 Components of a dynamic case expressed as relations and weighted connections. 91

Figure 4.8 Example case from automobiles domain. 92

Figure 5.1 An illustration of resolving the paraphrase problem. 104

Figure 5.2 TreeTagger output. 106

Figure 5.3 A triad consists of two phrases (p_1 and p_2) and a relation (r). 109

Figure 5.4 A semantic network (type-II triad) consists of one or more type-I triads. ... 109

Figure 5.5 Semantic net pictorials of symptoms from automotive data. 114

Figure 5.6 Semantic networks for six different descriptions using the three relations described above. 115

Figure 5.7 Semantic networks for some descriptions from centrifuge data.................. 116

Figure 5.8 Semantic net pictorials of symptoms from centrifuge data. 116

Figure 6.1 A generalized basic unit of the knowledgebase connected with weighted associations and feedback rewards. 124

Figure 6.2 Pictorial representation of the dynamic structure of the knowledgebase. 126

Figure 6.3 State space representation of a diagnostic process. 129

Figure 6.4 Three modes of HITL integration in a Decision Support System. 135

Figure 6.5 A successful hypothesis and at least one successful diagnostic test............ 140

Figure 6.6 The unsuccessful hypothesis case... 141

Figure 6.7 A successful hypothesis but lower confidence due to poor quality data. 141

Figure 6.8 A flowchart showing the complete decision support process with learning. 143

Figure 6.9 Comparing Q-Learning performance with frequency based learning for symptom s_a. 149

Figure 6.10 Comparing Q-Learning performance with frequency based learning for symptom s_b 150

Figure 6.11 Comparing Q-Learning performance with frequency based learning in scenario 2. 152

Figure 6.12 Interface to enter structural model of the system. 154

Figure 6.13 Interface to add or modify an object into the knowledgebase. 155

Figure 6.14 SemanticNet visualization module. 157

Figure 6.15 Module to monitor weights between various entities of the knowledgebase. 157

Figure A.1 Schematic showing information hierarchy [15] and a parallel functional hierarchy [73] for a system-of-systems architecture.. 167

Figure B.1 Monopropellant Propulsion System (MPS) [75]. 170

Figure B.2 Structural decomposition of the MPS. 171

Figure B.3 Structural block diagram of the MPS.. 172

Figure B.4 Functional block diagram of the MPS. 173

Figure B.5 Failure probabilities associated with critical components of the MPS. 174

Figure C.1 Concept of utility is defined as a relevance between the new problem and solution of some old problem (case) [77]. 176

List of Abbreviations

AI: Artificial Intelligence

AR: Analogical Reasoning

CBM: Condition Based Maintenance

CBR: Case-Based Reasoning

DCBR: Dynamic Case-Based Reasoning

DKMS: Diagnostic Knowledge Management System

DSS: Decision Support System

FMEA: Failure Mode and Effect Analysis

FMECA: Failure Mode Effects and Criticality Analysis

HITL: Human-In-The-Loop

ILP: Industrial Language Processing

IR: Information Retrieval

KBS: Knowledge-Based System

KE: Knowledge Engineering

KMS: Knowledge Management System

LRU: Line Replaceable Unit

MDP: Markov Decision Process

ML: Machine Learning

MPS: Monopropellant Propulsion System

MTTR: Mean Time To Repair

MTBF: Mean Time Between Failures

NLP: Natural Language Processing

PHM: Prognostic Health Management

RL: Reinforcement Learning

RPN: Risk Priority Number

SoS: System-of-Systems

TCBR: Textual Case Based Reasoning

Summary

A paradigm shift is emerging in system reliability and maintainability. The military and industrial sectors are moving away from the traditional breakdown and scheduled maintenance to adopt concepts referred to as Condition Based Maintenance (CBM) and Prognostic Health Management (PHM). In addition to signal processing and subsequent diagnostic and prognostic algorithms these new technologies require storage of large volumes of both quantitative and qualitative information to carry out maintenance tasks effectively. From the volumes of data that can be obtained today, information extraction has been a challenging task and organizing this information, so that it can be considered useful knowledge, is yet another level of abstraction. This not only requires research and development in advanced technologies but also the means to store, organize and access this knowledge in a timely and efficient fashion. Knowledge-based expert systems have been recently shown to possess capabilities to manage vast amounts of knowledge, but an intelligent systems approach calls for attributes like self-evaluation (feedback), self-evolution (learning) and self-organization (maintenance) to build truly autonomous systems for CBM. Furthermore, an intelligent reasoner is required that can make judicious use of this knowledge and provide a substantial support in the decision making process.

This research presents an integrated knowledge-based approach to diagnostic reasoning for CBM of engineering systems. A two level diagnosis scheme has been conceptualized in which first a fault is hypothesized using the observational symptoms from the system and then a more specific diagnostic test is carried out using only the relevant sensor measurements to confirm the hypothesis. Utilizing the qualitative (textual) information obtained from these systems in combination with quantitative (sensory) information reduces the computational burden by carrying out a more informed testing. An Industrial Language Processing (ILP) technique has been developed for processing textual information from industrial systems. Compared to other automated methods that are computationally expensive, this technique manipulates standardized language messages by taking advantage of their semi-structured nature and domain limited vocabulary in a tractable manner.

A Dynamic Case-based reasoning (DCBR) framework provides a hybrid platform for diagnostic reasoning and an integration mechanism for the operational infrastructure of an autonomous Decision Support System (DSS) for CBM. This integration involves data gathering, information extraction procedures, and real-time reasoning frameworks to facilitate the strategies and maintenance of critical systems. As a step further towards autonomy, DCBR builds on a self-evolving knowledgebase that learns from its performance feedback and reorganizes itself to deal with non-stationary environments. A unique Human-in-the-Loop Learning (HITLL) approach has been adopted to incorporate human feedback in the traditional Reinforcement Learning (RL) algorithm.

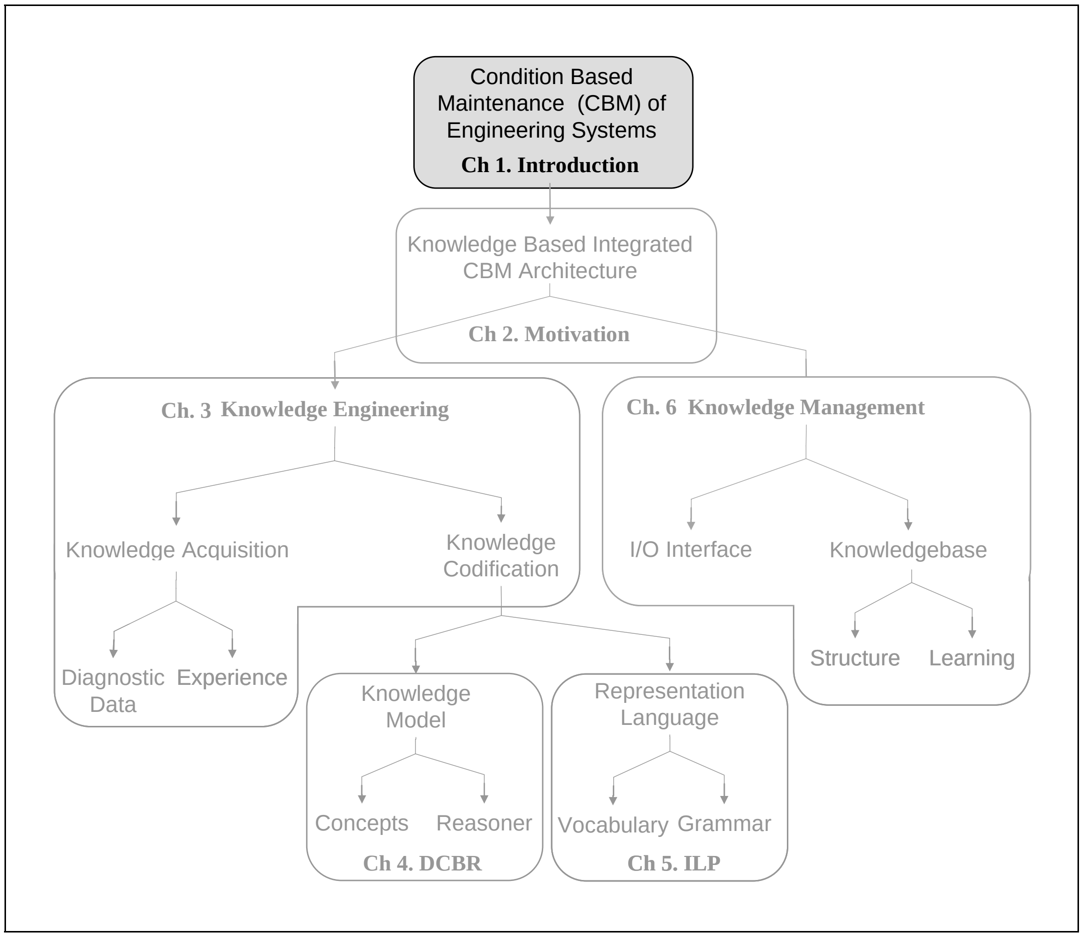
Main contributions of this research are:

1. Knowledge integrated CBM process model for automated diagnostic Decision Support System.
2. A self-evolving Knowledgebase (KB) that learns from its performance over time and a structured approach to acquire and modify knowledge to populate this KB.

3. A dynamic case-based reasoning platform for simultaneously utilizing qualitative and quantitative information to carry out diagnosis with less computational burden.
4. An Industrial Language Processing (ILP) technique to process industrial text, while retaining its domain specific information for effective diagnosis.

Chapter 1

Condition Based Maintenance of Engineering Systems



1.1 Research Context

Productivity is the key element of growth in demand for any complex dynamical system such as aircraft, power plant, or an automobile. Increased productivity can be achieved through increased availability, but all systems are subject to failure modes that tend to reduce uptime. Issues of reliability and maintainability have taken center stage over the past years and new paradigms are emerging in order to extend the useful lifetime of critical systems and make them available when needed. Condition-Based Maintenance (CBM) is the technology that strives to identify incipient faults before they become critical, which enables more accurate planning for preventive maintenance. This research aims at enhancing the capabilities of CBM through integration of knowledge-based techniques to automate the experience accumulation and reasoning tasks for maintenance operations.

1.1.1 The Problem

Currently there is a very limited acceptance of on-line or fully automated CBM systems in the industry, which may be attributed to several reasons. The maturity level within complex technical systems may be too low or various diagnostic techniques might exist at a very abstract level, ranging from the maintainer's experience to advanced detection algorithms. Further, in the absence of a unified architecture it becomes a very time-consuming and expensive investment to organize all the experts' knowledge in a coherent fashion before any positive effects are realized. CBM technologies so far have focused on individual components or subsystems of these complex systems and a lot of diagnostic knowledge has been developed in an ad hoc fashion. To be able to make better decisions, while keeping in mind the overall performance of a system, it is desirable to devise an integration mechanism for the operational infrastructure of a complete system. Diagnostic knowledge related to different components must be organized under a common framework so that it can be accessed whenever required and be reused for other components in appropriate situations. It is also desirable to have intelligent control algorithms that close the control loop, via active feedback, for improvement through adaptation and learning.

The need for an integrated system encompassing intelligent process operation stems from the growing complexity of current systems, as well as from the traditional expense, available computational resources, time constraints, and limited availability of human experts. The number of health-monitoring sensors has been increasing day by day, which consequentially requires a large amount of data processing. In several cases either a very simple processing is carried out or data analysis is not exhaustive. Similarly, often a lot of descriptive information is available from the operator observations or the maintenance logs. This information is text-based in most situations and is used by the experts for reference purposes only. Most of the current text processing techniques are very computationally expensive. Thus, given the limited processing capabilities and available time, a choice is usually made between the qualitative and quantitative information. In most cases only numerical data is processed and very few systems make use of textual information in automated ways.

1.1.2 Research Proposition

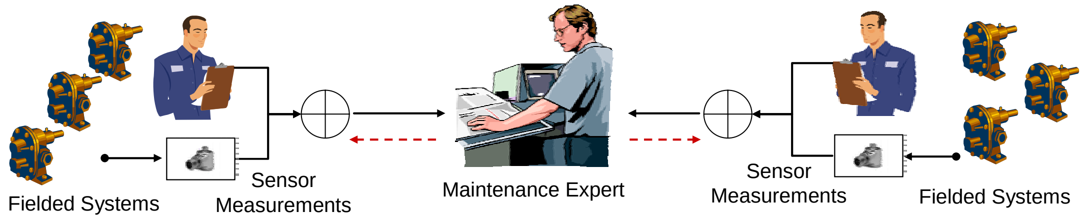
This research is motivated by the fact that textual information carries important pointers for fault diagnosis and helps in localizing the fault before explicit diagnostic tests need to be carried out. These descriptions help experts recall a similar looking situation from the past that was associated with a known fault and corresponding diagnostic tests. Thus, the maintainers do not consider an exhaustive set of diagnostic tests but just the most probable ones. Further, careful observation shows that industrial text in general is semi-structured and hence can be processed with relative ease and much less computational effort. Also, as the industry realizes the importance of standardized languages for communication and data interoperability, the use of simplified language is being promoted. This motivates an investigation in the direction of automatic text processing for semi-structured texts. Therefore, an attempt has been made to incorporate automatic text processing in addition to numerical data processing for improved diagnosis without increasing the computational intensity too much. This allows an integration of qualitative and quantitative information in two ways. First, the diagnostic process is carried out based on textual and numerical data on a common platform. Second, qualitative textual information is translated to numerical measures through methods like fuzzification techniques.

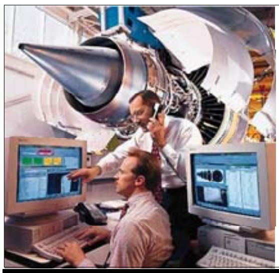
In addition to including textual data in the process, a significant focus is on developing an evolving knowledgebase that can store the useful experience knowledge and learn from new situations in an autonomous fashion. The unique aspect of learning involves integration of human-in-the-loop by incorporating human feedback reward to accelerate the learning activity. Supported by text processing capabilities in addition to numerical algorithms and an intelligent knowledgebase as its backbone, a reasoning system is the final piece that will result in a knowledge integrated decision support system for CBM.

1.2 Significance of the Research

This thesis is expected to set the framework for widely applicable software for the decision support operations of both military and commercial applications. Specifically, the maintainers of large-scale systems will benefit from an autonomous and self-evolving knowledgebase that incorporates information from both observational sources and physical measurements. This will provide not only a means to carry out a more informed testing to achieve better confidence, but also a reduced computational burden by avoiding unnecessary data processing from irrelevant sensors.

Potential applications include autonomous CBM for large-scale systems such as airframe subsystems, shipboard systems, gas turbines, HVAC, UAVs, automobiles, and other industrial systems. A particular interest in this research is from the Department-of-Defense (DoD). DoD's new programs like Joint Strike Fighter (JSF) and Advanced Amphibious Assault Vehicles (AAAV) plan for automated CBM/PHM technologies, while other programs seek tools that aid in the reasoning component of the redesign of current systems. Similarly, there is a great interest from the industry to increase their thrust toward after-market services. Aircraft companies like Pratt & Whitney and GE are pushing for real-time remote diagnostic systems for the CBM of their engines and gas turbines. As shown in Figure 1.1, consider the example of a gas turbine operator at a remote location any where in the world who monitors the machine and observes some symptoms of abnormal behaviors, e.g., vibrations, rising temperature, oil on floor, etc. and calls GE's Technical Response Center in Atlanta to linguistically describe the problem (no readings or measurements are available). GE experts need to infer from their knowledgebase what the problem could be and then recommend an appropriate action. Currently there is heavy involvement of *experts* in this troubleshooting process. Experts infer the symptoms and use their knowledgebase to come up with likely causes of the abnormal behavior. Based on their personal experience they suggest maintenance actions. There is a need to automate this process by combining text-based initial diagnosis and previous experience to propose appropriate repair actions.





"GE launched its Technical Response Center^{SM} in 1996 to speed up and improve engine maintenance. By monitoring engines in flight from the ground via satellite and other systems, **GE engineers** like Chris Henlein (seated) and Dave Brandel can diagnose potential engine problems, plan for maintenance or overhaul, and resolve problems more quickly than ever before. In addition to improving safety and reliability, real-time remote monitoring and diagnostics can lower operating costs for airlines by reducing delays and cancellations."

source: www.gepower.com

An Example

Figure 1.1 A real world example to show significance of the research.

For such capabilities it is important to develop a system that collects and interprets the information from remote locations and reasons without explicit human expertise as far as possible. Thus, the human experience must be systematically stored in a knowledgebase and a reasoning framework must be designed to carry out decision support for CBM autonomously.

1.2.1 Integrating Knowledge into CBM: A Decision Support System

During the lifetime of a machine, maintainers and experts seek and gather information that can help making decisions about everything from operation, support and maintenance to performance. The main goal of integrating knowledge into the CBM cycle is to automate this process of learning through computer systems and help decision makers by saving time and effort on remembering all experiences and retrieving relevant information. Even though AI strives to achieve complete autonomy and possibly replace human from the control loop, with the current state of the art this goal seems quite far fetched. Furthermore, while controlling mission critical systems a significant control is given to humans since in most of the cases convergence and optimality of AI systems can not be formally guaranteed. The only source of trust in these systems is through observation of their consistently good performance and improvement over time. In theory, after a significant learning over time, such systems can closely match human capabilities, given the learning model is fairly appropriate and all possible scenarios have taken place. Such assumptions are extremely optimistic and of little or no practical significance. Therefore, the best we can do in this scenario is to develop a Decision Support System (DSS) that aims to simplify human tasks by reducing the complexity of the problems or making the solution search faster and more relevant. These DSSs interact with humans and learn from their feedback to improve their performance. The human on the other hand interprets the DSS's recommendations and takes appropriate action. In an ideal case, learning should lead to positive improvement in DSS’s performance which in turn should result in user’s increased trust in DSS’s recommendations.

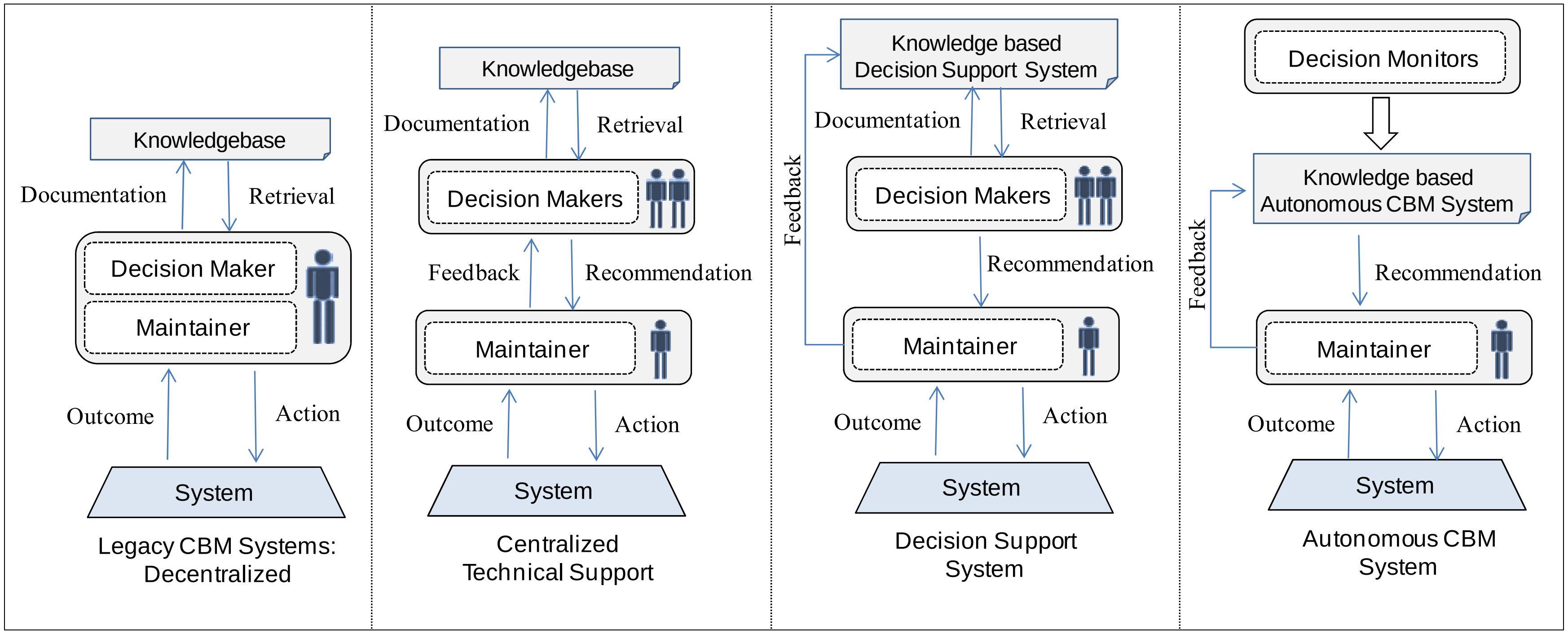


Figure 1.2 Transition from expert dependent legacy systems to knowledge based autonomous decision support systems for CBM.

1.3 Thesis Organization

This thesis addresses various concepts that are used in building a knowledgebase and subsequently using it to perform reasoning tasks. The overall organization and connections between various chapters has been shown in Figure 1.3. After a brief introduction to Knowledge-based systems, **Chapter 1** continues with highlighting their significance in the field of CBM. Various applications and current research programs have been cited that are relevant to the core theme of this thesis.

Chapter 2 outlines the scope of this thesis. There are many frontiers that can be achieved to build a truly autonomous and intelligent maintenance system. Improvements can be made at the lowest level of data processing in diagnostic algorithms, at a middle level of knowledge storage and retrieval algorithms, or at the highest level of reasoning. This thesis mainly deals with improvements at higher levels where a customized framework has been developed to provide specifics for knowledge-based approach for decision support systems as applicable to engineering systems. The conceptual development of the integrated CBM has been briefly outlined and contrasted with conventional CBM practices.

The two main aspects, while building a Knowledge-based system, are Knowledge Engineering and Knowledge Management. Knowledge Engineering, as introduced in **Chapter 3**, allows systematic acquisition and codification of knowledge into machine readable format. An approach specific to knowledge engineering for CBM knowledgebase has been identified and presented. This approach re-structures already existing knowledge representation formats and connects them for systematic acquisition using various tools commonly employed in the industry. Finally, two example cases have been presented to show the application of this approach for knowledge engineering in CBM domain.

After information has been collected and transformed into useful knowledge it must be codified using a suitable representation language and a knowledge model. The knowledge model assumes the task of carrying out reasoning based on current facts from the system and its own experience from previous situations. This thesis builds on a Dynamic Case-Based Reasoning (DCBR) approach to develop a knowledge model. In **Chapter 4**, the dynamic component has been introduced in multiple stages of the conventional Case-Base Reasoning (CBR) to build a DCBR architecture. This architecture has been further instantiated with an example for industrial systems that can be applied to fleet vehicles.

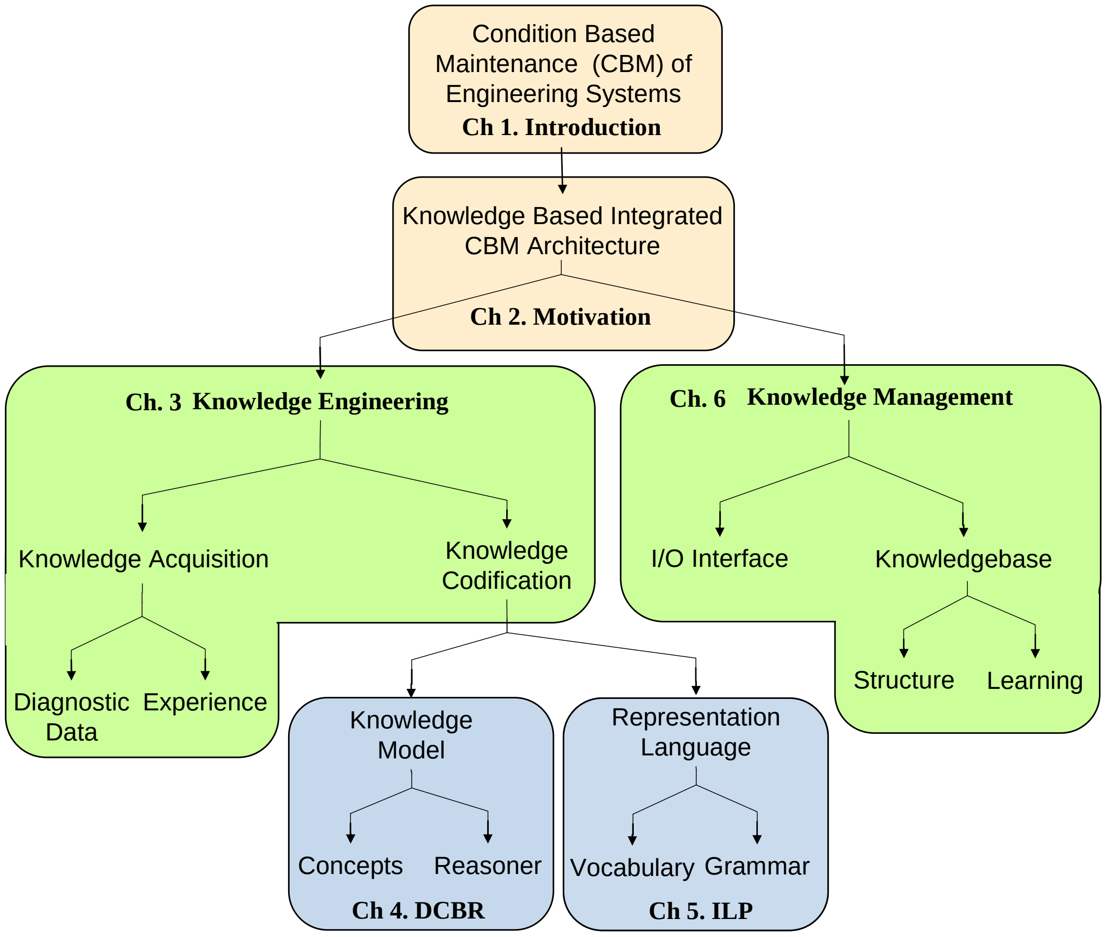


Figure 1.3 Thesis Organization.

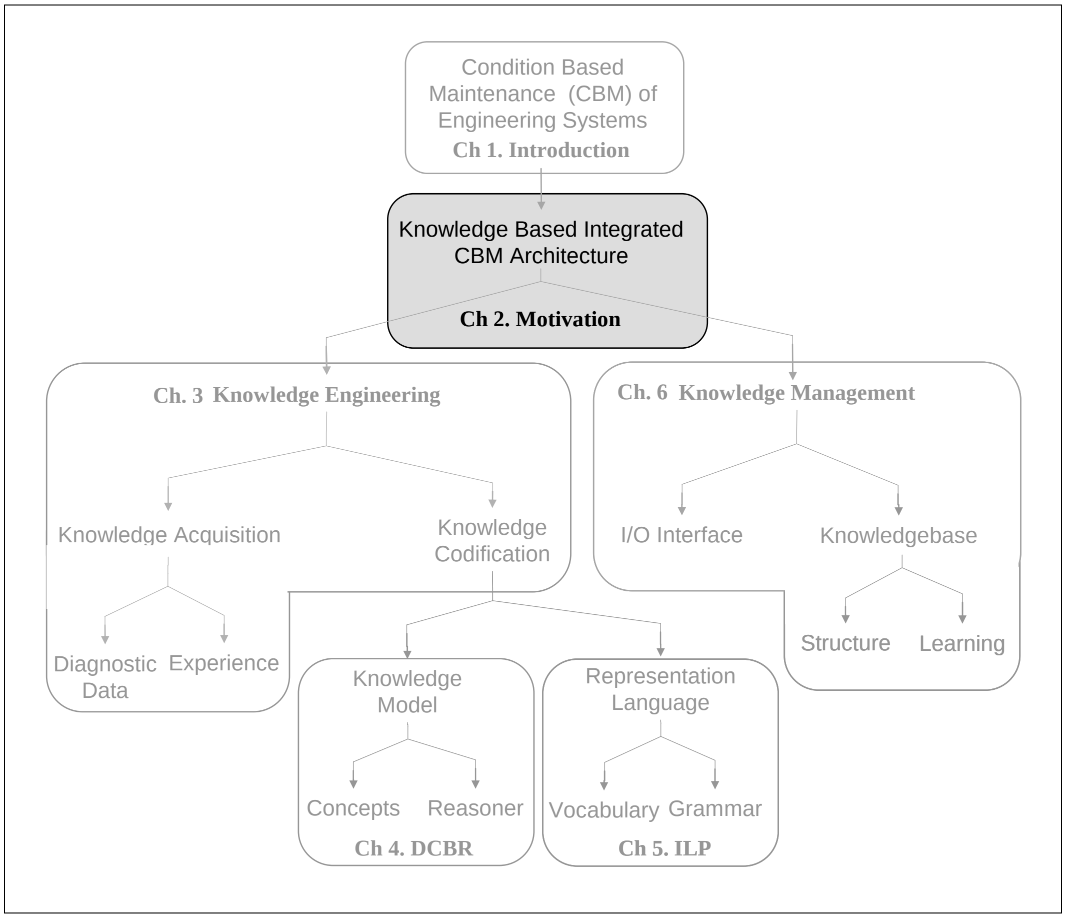
To carry out computations, as required by the knowledge model, a suitable knowledge representation language must be adopted. Knowledge representation for numerical information is relatively simpler to codify, however the codification of qualitative knowledge is not as straight forward. **Chapter 5** introduces a novel Industrial Language Processing (ILP) technique that can process simplified-english sentences into well defined knowledge structures for easier manipulation during reasoning tasks. The effectiveness of the approach has been shown with the help of results from processing dataset from automobile maintenance domain.

Moving from conceptual development to implementation details, this thesis dives into the issue of intelligent knowledgebase and knowledge management in **Chapter 6**. The core of this thesis lies in an evolving knowledgebase that exhibits various attributes of intelligence. These attributes have been shown with the help of various examples that build upon the data used for this research. These examples have been supported with corresponding simulation results that show the effectiveness of the overall system in carrying out decision support for maintenance tasks. Apart from the backbone of intelligent knowledgebase, a transparent and easy to use user interface is required for trustworthy decision support systems. Therefore, a knowledge management system has been developed, in Matlab environment, which allows users to store data, access data, and observe the learning activity in the knowledgebase.

Finally, **Chapter 7** concludes the thesis by highlighting various contributions of this research and discussing the scope of future work.

Chapter 2

Knowledge-based Integrated CBM Architecture



2.1 Chapter Overview

This chapter broadly outlines the scope of this thesis. First, it provides the motivation for enhancing CBM systems by highlighting the current practices and pointing out their shortcomings. Then, it delves into a conceptual development of *knowledge integrated CBM process* by defining its main components. This leads into the definition of objectives and goals of this thesis to define the overall scope. Improvements in CBM can be made at the lowest level of data processing and diagnostic algorithms, at the middle level of knowledge storage and retrieval algorithms, or at the highest level of reasoning. This thesis mainly deals with improvements at the higher level of reasoning for which a customized framework has been developed to provide specifics for a knowledge-based approach to decision support systems in an engineering environment.

Contribution: Conceptual integration of knowledge into the CBM process

2.2 Current State-of-the-Art

Until recently, the concept of CBM has been primarily fault diagnosis, which involves fault detection, identification, and isolation [1]. Several methods have been cited in the literature, which assume automation in fault detection through continuous system monitoring and sensor data analysis. The basic philosophy behind most of these systems is to compare the baseline data from continuously monitored run-time data and expect to observe significant differences, indicating the presence of an incipient failure. Some of the practical examples of such systems include Engine Monitoring Systems (EMS) in aircraft, Health Usage and Monitoring Systems (HUMS), and Rocket Engine Monitoring for Space Shuttle Main Engines (SSME) all use vibration data analysis for health monitoring as mentioned in a review paper by [2]. These systems have been rather limited in their scopes and have had a relatively narrow focus on a specific component or subsystem of the whole system. Further, there has been little communication for knowledge sharing and reuse for similar components employed in different locations. The inherent disadvantage of this approach is its limited coverage to preselected known failure modes only. Any new failure modes would require experts' attention to carry out full-scale analysis and develop corresponding diagnostic methods.

A major shift in philosophy was observed with the introduction of expert systems in the CBM community. Systems like model-based diagnosis and rule-based diagnosis focused on combining health monitoring for various fault modes under one umbrella and provided a comprehensive unit. However, the basic premise revolved around data analysis only as far as industrial systems were concerned. Later, AI-based systems like Case-Based Reasoning (CBR), Model-Based Reasoning (MBR), and Probabilistic Belief Networks (PBN) were also used to encompass the attributes of learning and adaptation. This significantly improved the state of the art over previous systems. However, most of these systems use only the quantitative information available from the sensors to automate the diagnosis task, and almost no or very little use of the qualitative information is made [3-5]. On the other hand some of these systems only considered the qualitative textual information and ignored any sensor measurements [6, 7].

Thus little or no effort was made to use both the textual and numerical information at the same time. The non-availability of computationally affordable Natural Language

Processing (NLP) techniques and the difficulty in building a common platform to process heterogeneous data have hindered the use of such a hybrid system. The integration of tasks through an intelligent knowledge-based system architecture would lead to improved system availability and reliability by increasing interaction via information sharing and coordination for timely preventive maintenance [8].

2.3 Key Drivers for CBM Enhancements

A Paradigm Shift - A paradigm shift is emerging in system reliability and maintainability (Figure 2.1). The military and industrial sectors are moving away from the traditional "breakdown and scheduled maintenance" philosophy and adopting concepts referred to as Condition-Based Maintenance (CBM) and Prognostic Health Maintenance (PHM). The emphasis is on providing maintenance services to increase reliability and uptime rather than developing entirely new systems. This requires constant condition monitoring and assessment of system health before catastrophic failures actually occur.

DOD Vision for CBM+ - The Big Picture: Furthermore, a newer philosophy of CBM+ was rolled out by the Department of Defense (DoD) in which logistics information was integrated with CBM/PHM systems to carry out timely maintenance. It introduced concepts like capabilities for remote sensing and analysis, portable maintenance aids, and equipment health and usage systems installed on multiple platforms [9-11].

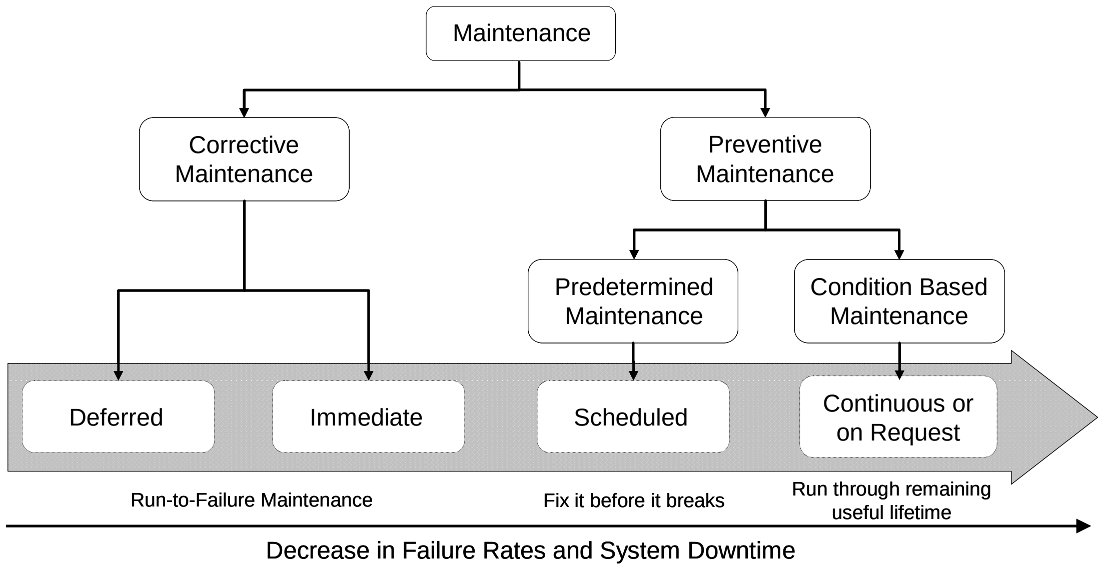

Maintenance
Corrective Maintenance
Preventive Maintenance
Predetermined Maintenance
Condition Based Maintenance
Deferred
Immediate
Scheduled
Continuous or on Request
Run-to-Failure Maintenance
Fix it before it breaks
Run through remaining useful lifetime
Decrease in Failure Rates and System Downtime

Figure 2.1 Paradigm shift in industrial maintenance.

Increased system complexity and the need for near real-time decision-making capabilities require extensive automation for maintenance tasks. Over the years a lot of analytical, operational, and structural knowledge has been generated in an ad hoc manner, which must be properly organized in a modular fashion so it can be shared and reused for similar components of a system. As shown in Figure 2.2, in maintenance operations several factors such as logistics, maintenance schedules, and Mean-Time-To-Repair (MTTR) should also be included in addition to actual data processing and decision-making algorithms for a successful and timely CBM [12]. Along with several data management techniques that have been developed in recent years, it is equally important that a robust integration mechanism be devised that integrates the tasks of data acquisition, information extraction, knowledge organization, and reasoning. The goal is to build an automatic diagnostic framework for a complete system by using the maximum available information without any duplication as far as possible.

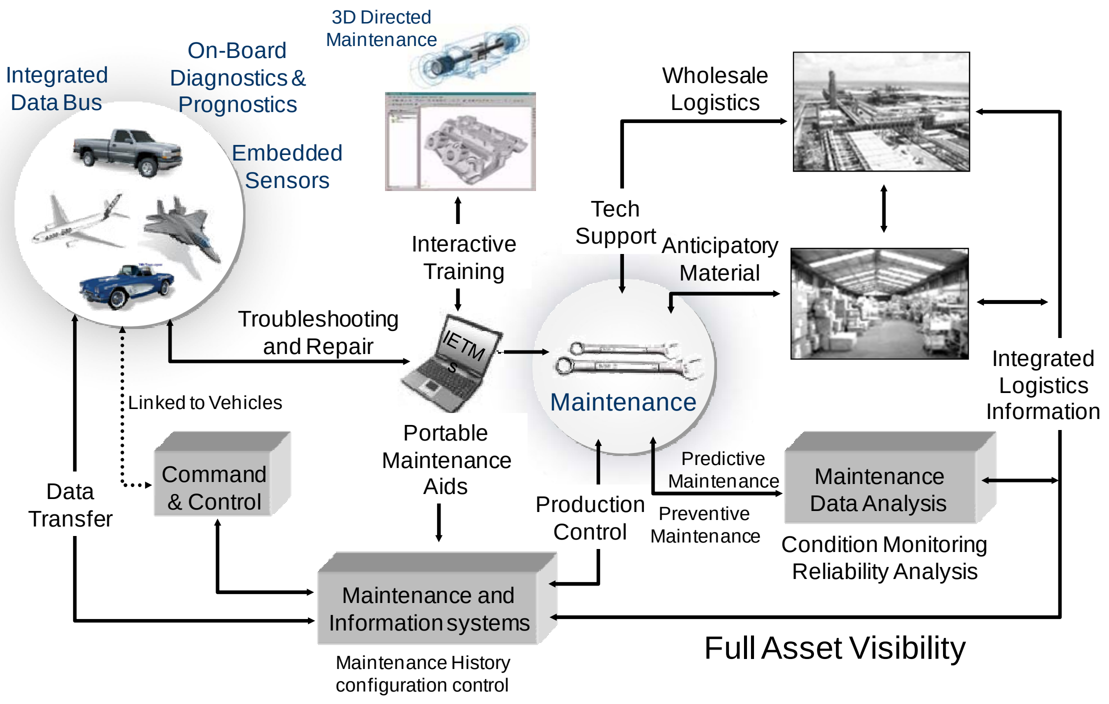


Figure 2.2 CBM+: A maintenance centric approach to CBM adapted from [13].

2.4 An Approach to Knowledge Integration into CBM

The main objective of this research is to develop a knowledge-integrated architecture for diagnostic reasoning and knowledge management for the CBM of engineering systems. Figure 2.3 shows the modified architecture of the CBM+ by integrating an intelligent knowledgebase into it. This knowledgebase supports a reasoning system that narrows the choices for the required diagnostic-data processing algorithms and helps in decision making based on past experiences stored in the knowledgebase. Another important aspect of this knowledge-based architecture is experience *accumulation* and *reuse* across multiple systems in a fleet. Various modules required to accomplish such a system are described next.

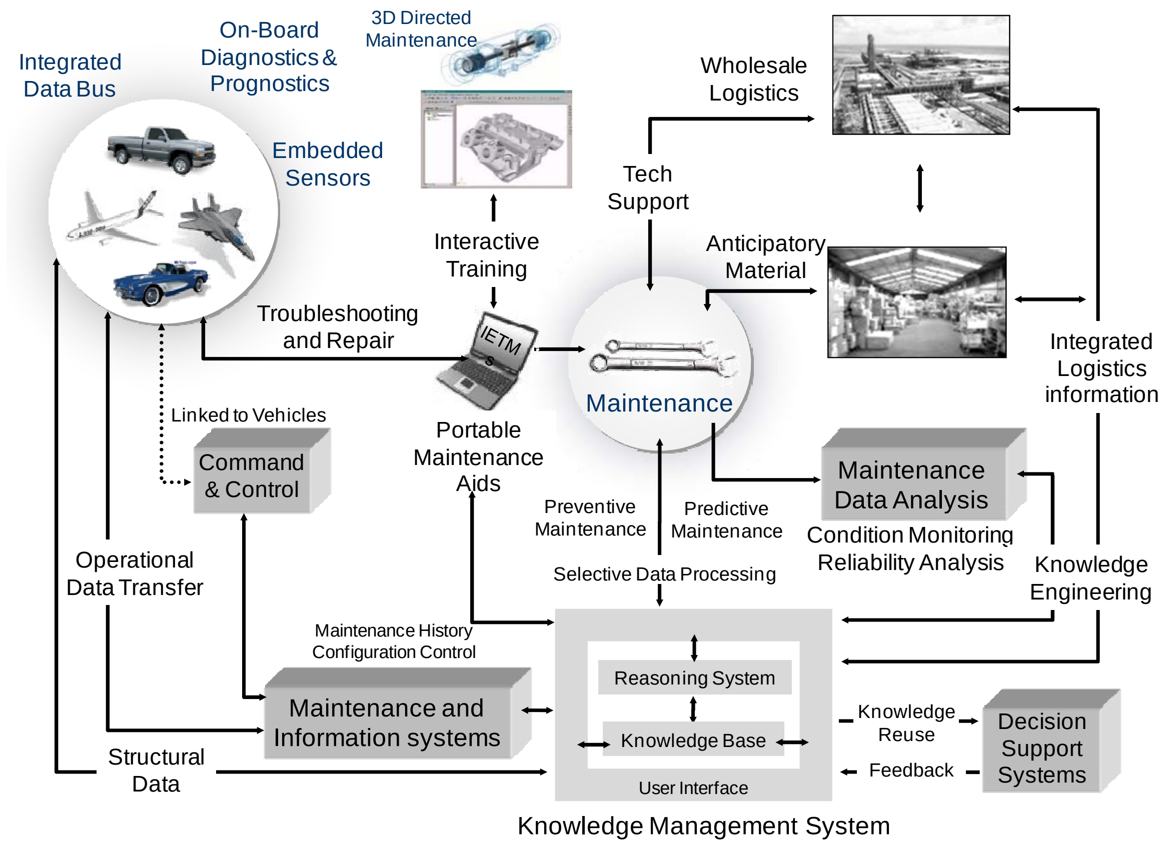


Figure 2.3 Knowledge integrated CBM+ architecture.

Figure 2.4 defines the research goals driven by the desired attributes in a knowledge-integrated CBM system. This research specifically promotes the use of experience-derived knowledge, accumulated over time, in an intelligent fashion that will help in automating the maintenance tasks as well as act as a decision support system by maintaining the corporate knowledge for later reuse. This system will have the capabilities to learn from feedback and adapt itself to changing environments. To accomplish these attributes, three key modules must be developed:

1) A knowledge management system for storing and accessing the knowledge
2) A knowledgebase with the attribute of intelligence, and

3) A reasoning system to assess the current system state and propose suitable solutions

As depicted in this figure, these three modules form the main pillars for the knowledge-based architecture. These modules are briefly discussed in the sequel.

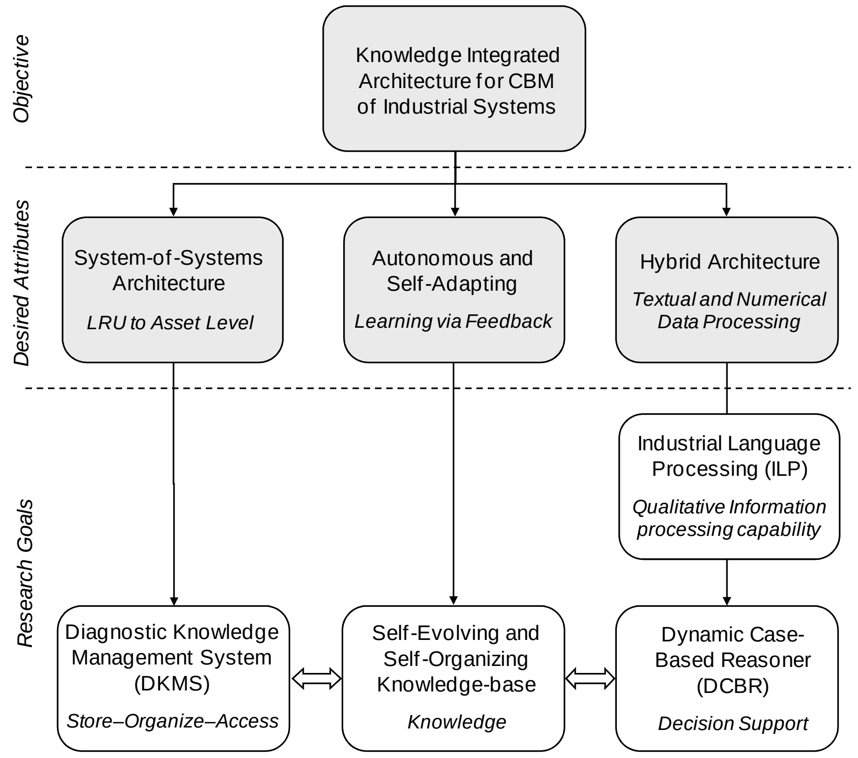

Objective
Desired Attributes
Research Goals
Knowledge Integrated Architecture for CBM of Industrial Systems
System-of-Systems Architecture
LRU to Asset Level
Autonomous and Self-Adapting
Learning via Feedback
Hybrid Architecture
Textual and Numerical Data Processing
Industrial Language Processing (ILP)
Qualitative Information processing capability
Diagnostic Knowledge Management System (DKMS)
Store–Organize–Access
Self-Evolving and Self-Organizing Knowledge-base
Knowledge
Dynamic Case-Based Reasoner (DCBR)
Decision Support

Figure 2.4 Key drivers and corresponding research goals: an overview.

2.4.1 Diagnostic Knowledge Management

Reusing existing systems in a dependable fashion without the need for extensive re-engineering is a key problem currently faced by industry. In response to this problem the corresponding desired attribute is a unified System-of-Systems (SoS) architecture for the diagnostic health management of a complex system. A system, as referred to herein, can consist of several subsystems that may themselves consist of other subsystems or components, as depicted in Figure 2.5.

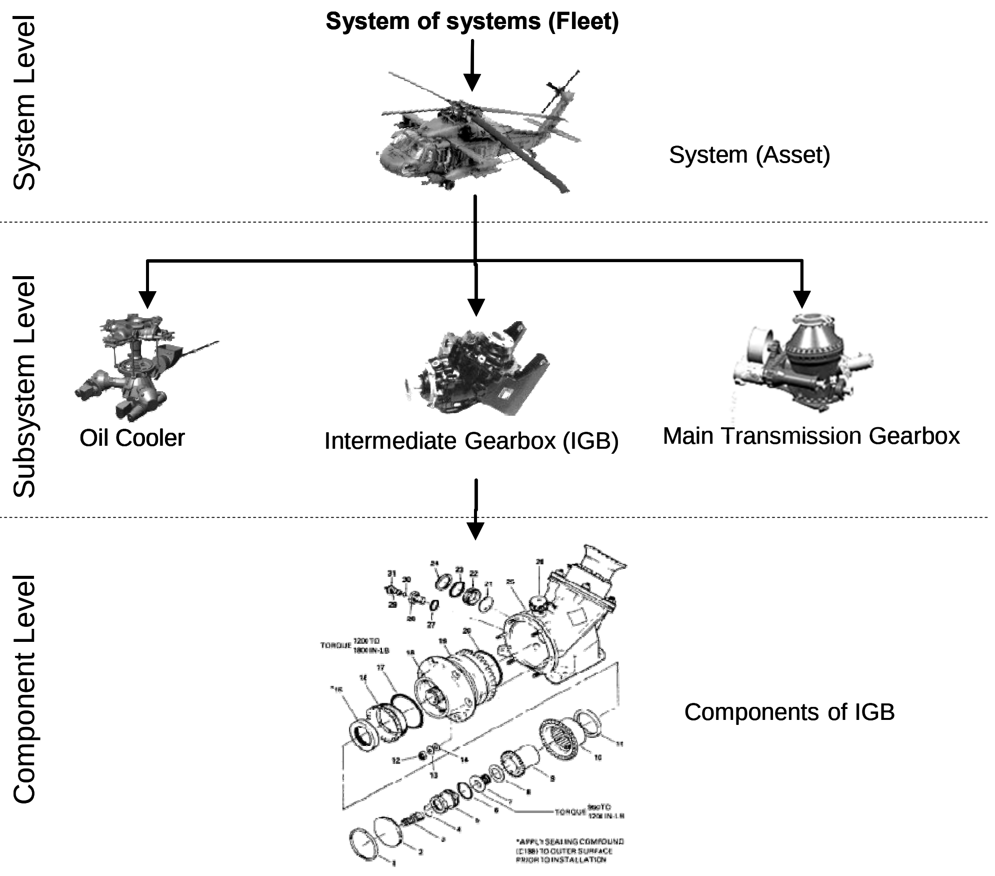


Figure 2.5 System-of-systems hierarchy.

The components of an SoS are generally existing systems, consisting of hardware and software, each potentially equipped with separate health management and maintenance techniques. Since these techniques are employed independently of individual subsystems, an SoS diagnostic architecture is not just a large complex distributed system, but rather one whose modules:

- fulfill valid purposes in their own right and continue to operate to fulfill those purposes if disassembled from the overall system, and
- are managed (at least in part) for their own purposes rather than the purposes of the whole.

However, well-coordinated information sharing and regular interaction among subsystems can improve system availability and reliability. This requires an integration of tasks through an intelligent system architecture for preventive maintenance.

This integration of knowledge from different subsystems imposes two requirements:

1) Since this knowledge cannot be abstracted to a fixed formal structure, there must be a hybrid knowledgebase that can accommodate knowledge in different forms.
2) A knowledge management system must exist that acts as an interface for knowledge transaction in the sharing and reuse of varied knowledge types,

Driven by these requirements, one of the goals of this research is to develop a Diagnostic Knowledge Management System (DKMS) that helps store, organize, and access the diagnostic techniques including signal processing, feature extraction, and fault classification methods. This module acts as an interface between various modules of the proposed architecture along with a user interface to provide the capability to examine, modify, and utilize this knowledge externally.

2.4.2 Self-Evolving Maintenance Knowledgebase

The next goal of this research is to develop a knowledgebase for CBM techniques that can accommodate knowledge in different forms (descriptions, data, algorithms, tests, models, etc.). A significant amount of structural, operational, and analytical knowledge has been developed over the years while these systems were built, tested, and maintained. Typically, components or subsystems of a large system are studied individually before relevant analytical techniques are developed. Even though these subsystems may be different, several constituent components share various structural and operational similarities. There is a need to organize this knowledge in such a manner that it can be easily shared and reused for similar components of a large system. Rather than re-developing this knowledge, minor adaptations should be able to save time and effort of the analysts. Thus an easily accessible knowledgebase containing these techniques must be created. Further, there must be a learning component in the knowledgebase that helps improve the performance over time. This process of self-evolution involves monitoring activities and their results to make internal adjustments for the next epoch [12].

Therefore, another goal of this research is to develop a self-evolving knowledgebase that learns from its diagnostic performance assessed through external feedback and reorganizes itself based on the temporal recency of the usage of its constituent knowledge capsules in order to adjust to the changing environment of the fielded systems.

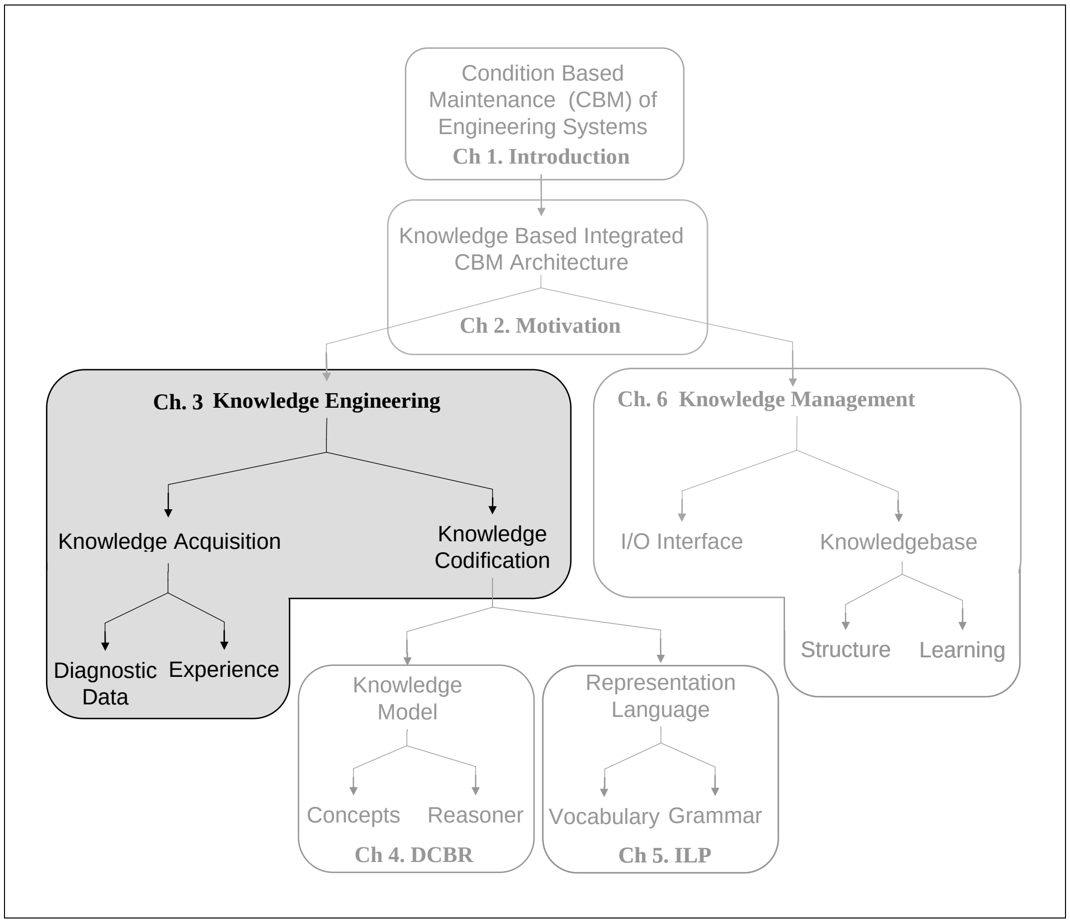
2.4.3 A Decision Support System Based on Higher Level Reasoning

The third goal of this research is to develop a higher-level reasoning paradigm to effectively utilize the knowledge contained in the knowledgebase. An alternative approach to computationally intensive classical modeling techniques is to reuse higher-level information acquired from previous experience for new but recurring situations. This approach automates the troubleshooting process to a large extent by providing a quicker decision support system using an extensive knowledgebase and reduced computations. Further, it must be realized that from the majority of these systems information is available in two different forms, qualitative (textual) and quantitative (numerical). In most cases only numerical information is used for diagnostic purposes even though qualitative observations can be very useful in localizing the fault. This is primarily because most text processing techniques from the AI domain are not computationally tractable. However, the semi-structured nature of industrial texts can help alleviate this problem.

Therefore, another goal of this research is to develop a method to utilize qualitative (textual) information available from the systems along with the conventional sensory (quantitative) measurements. This qualitative information facilitates a higher-level reasoning for localizing the faults, thereby shrinking the search space and consequently reducing the computational burden from exhaustively analyzing numerical data. The use of simplified language has been assumed in the industrial texts. Simplified language uses reduced grammar and a minimal number of words to express a situation [14]. The concept of simplified language is explained in detail in later sections. Finally, to carry out the reasoning tasks using knowledge from the knowledgebase and data from the system, a formal reasoning framework must be developed. Thus the corresponding goal is to develop a framework for diagnosis based on Case-Based Reasoning (CBR). This facilitates a hybrid reasoning system that can accommodate knowledge in multiple forms and that can perform higher-level reasoning. Conventional CBR has been expanded to DCBR that generates strategies for accessing relevant knowledge and to carry out diagnostic reasoning based on past experiences.

Chapter 3

Knowledge Engineering



3.1 Chapter Overview

A basic step to build a knowledgebase is to collect information that will constitute the knowledge. This chapter describes an introduction to knowledge engineering concepts that allow systematic acquisition and codification of knowledge into a machine readable format. Further, a knowledge engineering approach, specific to the CBM knowledgebase, has been identified and presented. This approach structures already existing knowledge representation formats and connects them for systematic acquisition using various tools commonly employed in the industry. Finally, an example case has been presented to show the application of this approach for knowledge engineering in the CBM domain.

Contribution: A structured approach to acquire and modify knowledge to populate the knowledgebase.

3.2 Introduction

Before a knowledgebase can be built, information must be gathered and modified in a structured manner so that it can be stored in a coherent format for easy storage and access. This task is formally accomplished through Knowledge Engineering (KE). KE is the technique of collecting, consolidating, structuring, and transforming relevant information into a computer-comprehensible format, to prepare the basic building material for intelligent systems: Expert Systems, Knowledge-based Decision Support Systems, Expert Database Systems, etc. Different systems generate data in different forms and the information is extracted in different ways depending on the task at hand. In other words, KE is an attempt to imitate the socio-cognitive process where knowledge is produced by human beings. It structures information according to our understanding of how human reasoning and logic work. Since 1980s, Knowledge Engineers have compiled a set of principles and guidelines based on various experiences they gained in developing knowledge-based Systems. Figure 3.1 shows various steps involved in KE. KE involves two main tasks namely *Knowledge Acquisition* and *Knowledge Codification*. In this thesis, we have further subdivided these tasks into specific activities suitable for CBM, as shown in Figure 3.2.

Codification: While building a knowledgebase, the most important factor to consider is whether the knowledge is *codifiable*, i.e. can it be codified in a manner so that there is no significant information loss on decoding it and the process cost is justifiable. A detailed discussion on *codifiability* is presented in Section 3.4.

Acquisition: Once the knowledge is declared codifiable, information must be acquired from the system. There are different forms of knowledge, and hence an appropriate approach is required for *acquisition*. There is no general methodology that can be used for all types of tasks. However, various categories have been defined and the task at hand should be first categorized into one or more of these categories and then the corresponding guidelines should be used with necessary adaptations. Similarly, there are different types of experts and expertise, and hence different ways of representing knowledge, which can aid acquisition, validation, and re-use of knowledge. Various methods have been devised that must be used to increase the efficiency of the acquisition process. In some cases the acquisition process can also be guided by the task objectives (goal-oriented acquisition).

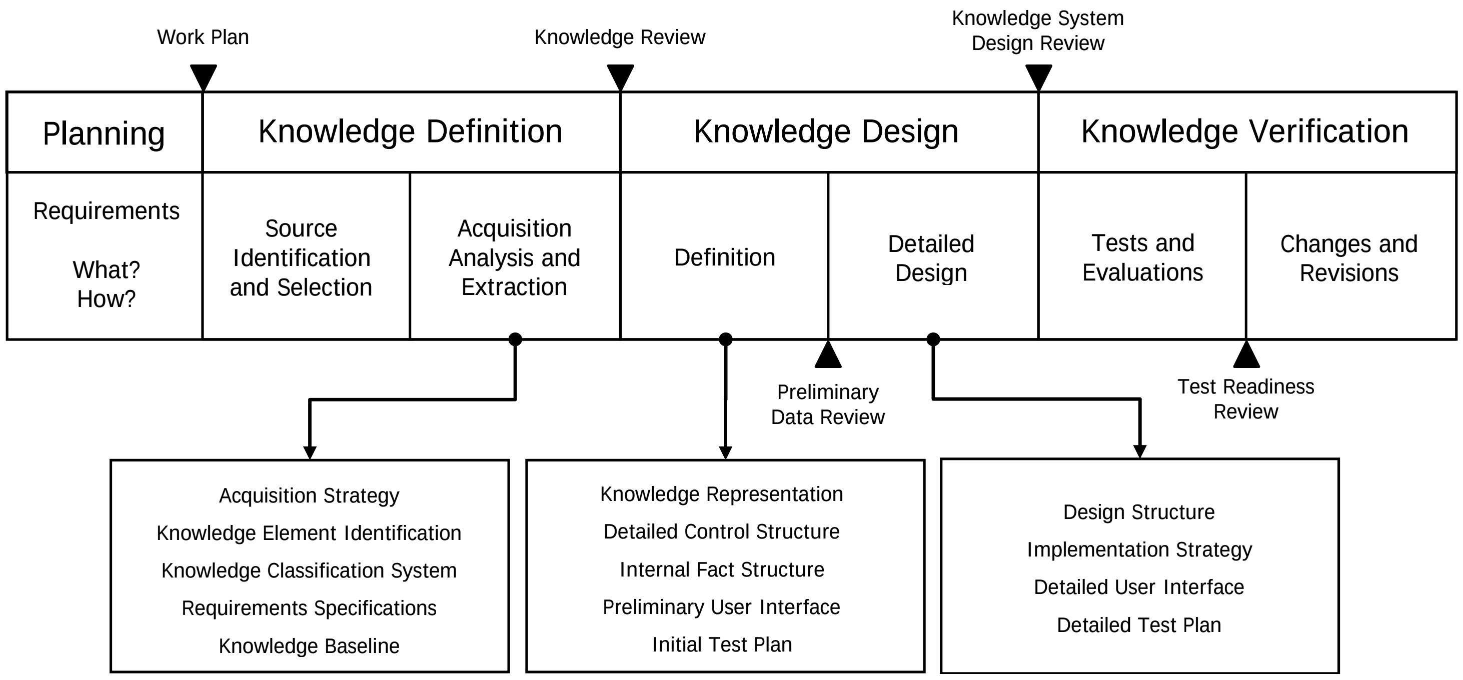


Figure 3.1 Knowledge Engineering: Task planning and execution adapted from [15].

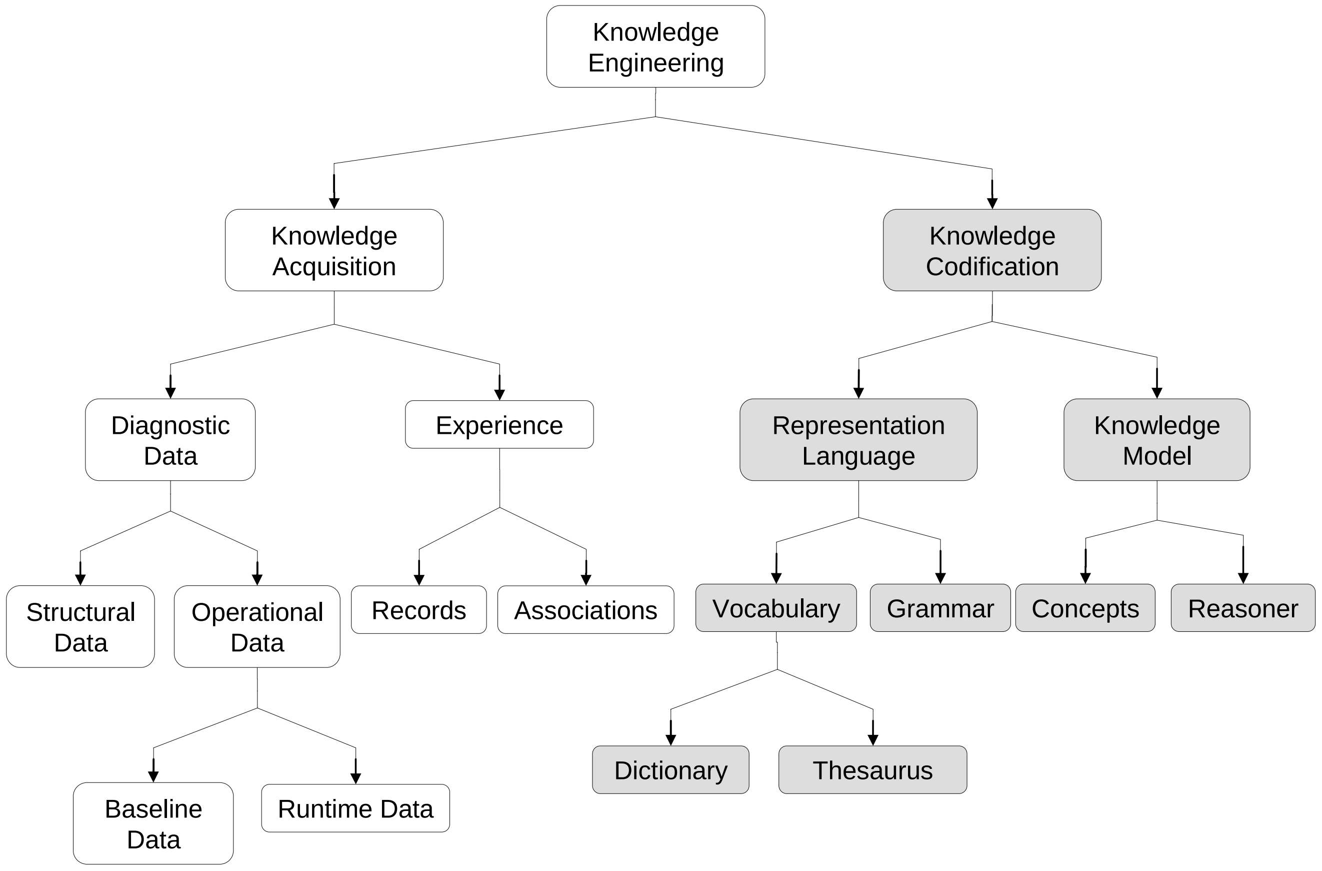


Figure 3.2 Main tasks involved in Knowledge Engineering (KE).

There are different levels of knowledge abstractions, namely noise, data, information, knowledge and wisdom. A brief discussion of these abstractions is provided in Appendix A. Keeping this hierarchy in mind, a knowledgebase should not be confused with a database. Unlike databases, a knowledgebase also contains attached semantics about how various data are related and provide useful information when brought together in different orders and combinations.

3.3 Knowledge Acquisition

For knowledge-based diagnosis of industrial systems two types of information must be gathered to create useful knowledge - *Diagnostic Data* and *Experience* (Figure 3.3).

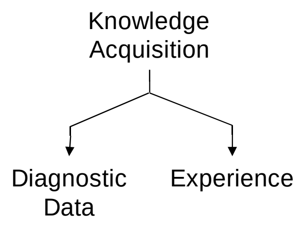


Figure 3.3 Sources of information for knowledge acquisition.

Useful knowledge refers to the abstraction of information structured in a way such that inferences about the health of a system can be drawn in a consistent manner. It should be possible to:

- structure, organize, and store information for easy and timely access,

- add more information as time passes by,

- extend the current knowledgebase to include more types of information, and
- reason about situations in a transparent and intuitive manner.

These two sources for knowledge acquisition have been further classified into their respective categories and are discussed next.

3.3.1 Diagnostic Data Collection

Data represent the facts about the system that can be gathered by examining it. Diagnostic data can be further divided into two types (see Figure 3.4):

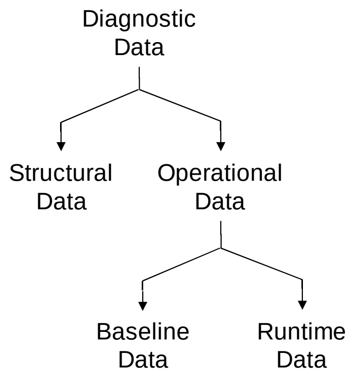


Figure 3.4 Types of data to be collected for building diagnostic knowledgebase.

Structural Data: Different component parts of the system are identified and their specific structural organization is stored as its *structural model*. Information about the location of each component and its relative proximity to other neighboring components is crucial in predicting how a local failure in one component may propagate though the entire system. Knowledge about the component interconnections is also important since the type of connectivity may sometimes restrict certain degrees of freedom of some component and hence affect its operational modes. Structural data mainly consist of the types of components, sensors, their locations, dependencies and interconnections, etc. Structural data are usually collected at the beginning while studying the system. In most cases, once collected, there are not many modifications to this dataset unless a part of the system changes. However, in the beginning, only a reduced dataset may be obtained from the system for quick deployment, and then additional data may be added to this database as time passes and more detailed information is desired.

Operational Data: Unlike structural data, operational data are more dynamic and are collected while the system is in operation. This data mainly consist of sensor measurements and operator observations. The characteristics of operational data may change over time due to changes in the environment or changes in the system itself (due to wear and tear in the system, or component replacement). For diagnostic purposes, operational data are further subdivided into two parts - *Baseline Operational Data* and *Runtime Operational Data* (see Figure 3.4). For a healthy system, both baseline and runtime operational data should be similar in a statistical sense. Some minor drifts may occur between the two due to changes in system characteristics over time. In such cases, fresh baseline data should be collected for recalibration. Any significant deviation between baseline and runtime data within unexpectedly low time spans indicates the possibility of a failure. With further analysis of runtime data, tasks of fault detection, isolation, and identification are carried out.

Operational data can also be analyzed to extract some higher level information about the system. For instance, statistical distribution of various faults, frequency of faults, and fault characteristics can be deduced from data collected over a long period of time. These data represent a collective summary of the behavior of a system or a fleet of similar systems over a long period of time.

3.3.2 Experience Accumulation

Experience represents the knowledge deduced from data/information collected over a period of time from one system or a fleet of similar systems. Experience can be accumulated in two forms - *records* of operational data and *cause-effect associations* explaining the behavior of the system (Figure 3.5).

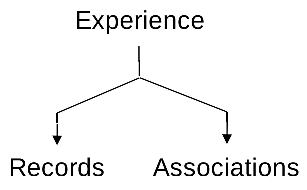


Figure 3.5 Experience is accumulated as historical records and cause-effect associations.

Eventually, experience tells about how to use data to perform reasoning tasks. This information is either gathered from the experts or it is learned over time. For instance, How to interpret a fault symptom and what diagnosis to perform? In computational terms, it can be viewed as connections between various data entries and their corresponding weights that offer support while computing a decision metric.

Experience accumulation includes methods for identifying, collecting, documenting, packaging, storing, generalizing, reusing, tailoring, and evaluating the experience [16]. An important attribute of the experience accumulation is its temporal dependence. First, the records should be prepared and accumulated over a long period of time before a statistically meaningful set of experiences can be constructed. Next, various data mining and machine learning techniques can be used to consolidate these records and extract useful knowledge from it. Therefore, learning is another attribute that must be included in experience accumulation. Case-based reasoning has been a useful tool for accomplishing the task of experience accumulation [17]. It has an added advantage of facilitating the reuse of this experience in an automated fashion. This thesis uses CBR as the learning backbone of the maintenance knowledgebase. A detailed discussion on CBR is included in Chapter 4.

3.3.3 A Systematic Approach to Knowledge Acquisition

For both kinds of knowledge acquisition a very methodical approach has been developed over the years, which systematically studies the system and identifies what information must be collected to carry out health maintenance tasks. Initially, Failure Modes and Effects Analysis (FMEA) was designed to improve the reliability of the system. FMEA is a methodology for analyzing potential reliability problems early in the development cycle where it is easier to take actions to overcome these issues, thereby enhancing reliability throughout design. FMEA is used to identify potential failure modes, determine their effect on the operation of the product, and identify actions to mitigate the failures. A crucial step in FMEA is anticipating what might go wrong with the system. Although, anticipating every failure mode is not possible, an effort must be made to formulate as extensive a list of potential failure modes as possible. Later, FMEA was enhanced by including *criticality* analysis to rank various failures in order of their frequency of occurrence and severity to prioritize the maintenance attention in the case of multiple simultaneous failures. The next section discusses this approach in detail with a systematic description of the steps that must be carried out to collect, organize, and acquire relevant information.

3.3.3.1 Failure Modes and Effects Criticality Analysis

FMECA is one of the earliest methods for failure analysis, developed by the US military in 1949. The main goal of a Failure Modes and Effects Criticality Analysis (FMECA) study is to relate failures to their root causes. Towards this goal, it addresses issues of identifying failure modes, their severity, frequency of occurrence, and testability. It also identifies fault symptoms that are suggestive of the system's behavior under fault conditions and the sensors required to monitor and track the system's fault symptomatic behaviors [1]. Ideally, FMECA must be performed during the initial and conceptual design phases of the system development to make sure that all possible failure modes have been considered and the corresponding failure mitigation strategies have been implemented. If successfully implemented, this would avoid costly re-engineering at a later date.

Advanced FMECA studies may recommend algorithms to extract optimum fault features or condition indicators, detect and isolate incipient failures and predict the remaining useful life of critical components [1]. FMECA studies aim to provide the designer with tools and procedures that will lead to a systematic and thorough framework for design. One can identify two approaches to FMECA:

Bottom up approach to FMECA: As discussed in Chapter 1, a systems approach leads to a very comprehensive coverage with respect to failure modes associated with different components in a system. Each component on the lowest level can be studied one-by-one and then the study is carried out at subsystem level and so on. This approach is exhaustive and hence the analysis is “complete” since all components are studied. This is suitable for cases where the system is already in operation and a post implementation analysis is required to improve its performance and reliability. FMECA for maintenance tasks lies in this category. However, instead of studying each and every component in a system, a more intelligent choice can be made by only studying those components that are more susceptible to failures and hence more critical. A detailed discussion is given in the next section.

Top down approach to FMECA: During an early design phase of a system, before a definite structure has been decided, a detailed function oriented study is carried out to ensure that all functions have been considered and appropriately included in the plan. There, it is important to know how a given system may fail in carrying out these functions. Functional failures with significant effects are considered with higher priority in the analysis and hence this study is more selective. The top down analysis will not necessarily be complete as less important areas are left out from the analysis. Alternatively, the top-down approach may be used on an existing system to focus only on the problem areas.

In the published literature there are several categories of FMECA studies that can be employed at various stages of an industrial system. A broad classification of different FMECAs is:

- System - focuses on system level functions.
- Design - focuses on components and subsystems individually.
- Process - focuses on manufacturing and assembly processes.
- Service - focuses on service functions like maintenance operations.
- Software - focuses on software functions for automation processes.

3.3.3.2 FMECA for CBM

CBM lies under the *Service* category of FMECA, where systems are already in operation and adequate information must be acquired to carry out effective maintenance tasks. For CBM, FMECA generates the template for diagnostic algorithms. The FMECA framework may be integrated into existing Supervisory Control and Data Acquisition (SCADA) or other appropriate data management and control centers to provide the operator with a convenient access to information regarding failure events and their root causes [1]. FMECA studies require the contribution of domain experts, reliability engineers, monitoring and instrumentation specialists as well as input from designers charged with the responsibility to develop a diagnostic and prognostic reasoner. Enabling technologies for FMECA design begin with simple spreadsheet type tables accompanied by explanation modules to more sophisticated tools such as rule-based expert systems, decision trees, and Petri nets, among others. Some of the main benefits of FMECA are:

- Early identification of potential failure modes to employ preventive actions.
- Identifies product/process deficiencies and prioritizes corrective actions.
- Captures engineering/organization knowledge
    - Documents risk and actions taken to reduce risk
- Provides focus for improved testing and development.
- Minimizes late changes and associated costs.
- Improves product/process reliability and quality.
- Increases customer satisfaction and confidence in automated maintenance.

FMECA is an enhancement of the FMEA methodology in which a criticality analysis is performed. Criticality analysis involves assigning a *frequency* to each failure mode and a *severity* to each failure effect. Criticality is a function of the severity of the effect and the frequency with which it is expected to occur. The purpose of this analysis is to rank each potential failure mode identified in the FMEA study according to the combined influence of severity classification and its probability of occurrence. Severity categorizes the failure mode according to its ultimate consequence. Figure 3.6 shows a stepwise approach to FMECA for CBM.

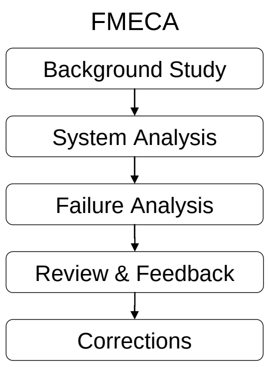


Figure 3.6 A stepwise approach to FMECA study for CBM systems.

STEP 0 - Background Study: Before starting the actual FMECA analysis, the background study identifies two important issues. First, it defines the system to be analyzed. The system definition includes:

a) *Scope of analysis*: decide on what parts and components of the system will be included in the study.

b) *Functionality*: decide which system functions will be included in the analysis. List of selected functions will depend on mission objectives for the system.

c) *Operational modes*: decide which operational modes will be included. Different operational modes have different priorities depending on mission requirements and environmental conditions.

Second, it identifies the sources for information collection. Information and data exist in variety of forms. All documents including system engineering drawings, schematics, specifications, component lists, functional descriptions, design documents, reliability data, maintenance manuals, maintenance logs, etc. should be collected. In addition, interviews should be conducted with operations and maintenance personnel, process experts, designers, and component manufacturers to acquire as much information as possible. Interaction with component manufacturers can often provide valuable information based on feedback they get from a variety of customers who use their products. Once all this information is collected, the next step is to study the system.

STEP 1 - System Analysis: the system level analysis can be carried out at two levels – *Structural* and *Functional*. Given a system, a *system model* is first created as a pictorial representation, which shows the interconnections between the physical components. Such models are usually available as schematic descriptions or block diagrams of the system from the technical documentations like maintenance manuals. Using this structural model, structural decomposition of the system is carried out to identify all critical components of the system. Then a list of priority components is prepared. Next a *structural body diagram* is created using this list of priority components followed by a *functional block diagram* or *functional model* of the system. This approach is depicted in Figure 3.7.

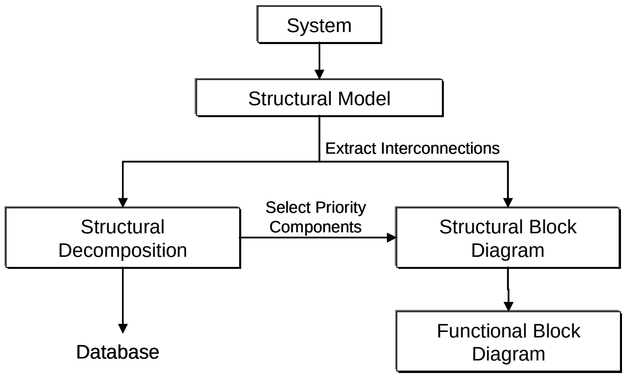


Figure 3.7 System analysis steps for FMECA study.

First, a *structural analysis* is carried out. The system is divided into smaller units or sections that are usually considered based on functional descriptions. The level of detail depends on the objectives of the study. The system is divided into subsystems and components to create a hierarchical tree diagram to show the structural decomposition of the system. This yields an exhaustive list of components from which more critical components must be identified and selected for further analysis (Figure 3.8). Information about the location of each component and its relative proximity to other neighboring components is crucial in predicting how a local failure in one component may propagate through the entire system [18]. This analysis starts with identifying structural links between various components of the system by creating a structural block diagram that includes, preferably; only the selected components (see Figure 3.9).

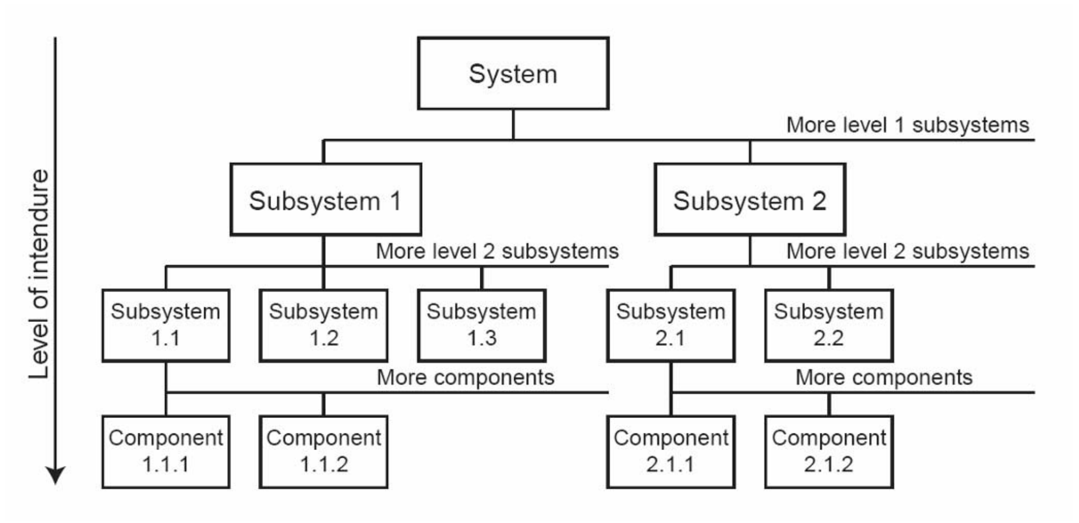


Figure 3.8 Hierarchical tree diagram to show structural decomposition of a system [19].

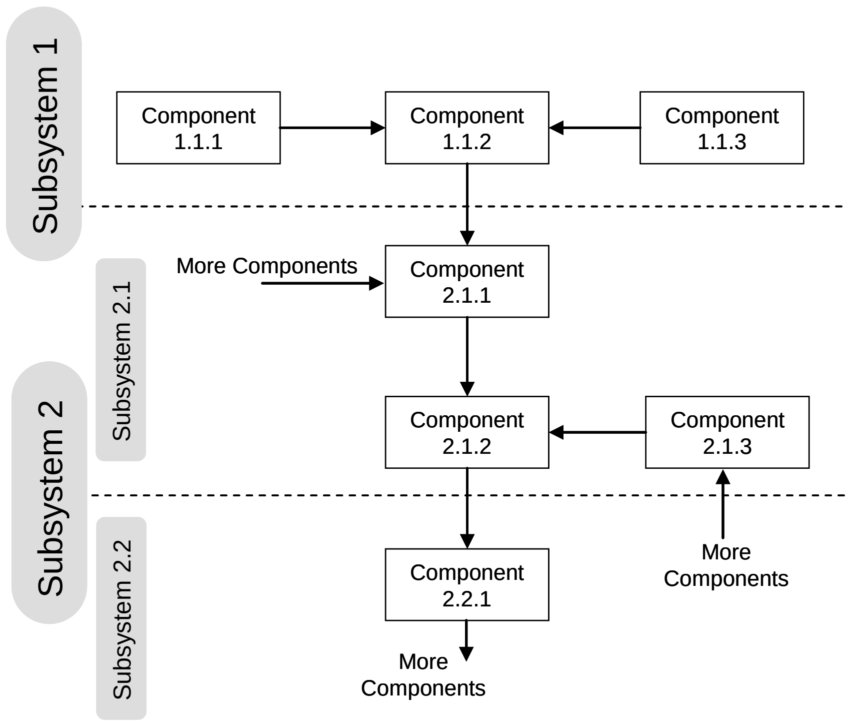


Figure 3.9 Structural block diagram to show structural interrelationship between different components.

Next, a *functional study* may be conducted to add finer details to the analysis. The functional model of the system is constructed by traversing the partially connected graph represented by the structural block diagram and including the corresponding function for each component from the database (Figure 3.10).

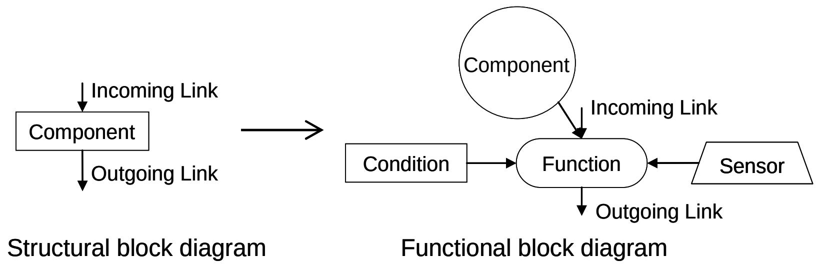


Figure 3.10 Functional block diagram adds input condition and sensor information along with the function of the component.

This model allows explaining the exhibited behavior of the system in terms of the functions performed by each component. Any anomaly in the system response can be reasoned about and expressed in terms of faulty operational mode(s) of one or more components. In addition to the structural information about the system, this analysis also generates a list of all sensors employed near the critical components. This information is particularly helpful in selecting the sensors when numerical data analysis must be carried out to confirm the presence of a fault. For illustration, the functional model for the above structural model can be derived by adding extra details to its structural block diagram as shown in Figure 3.11.

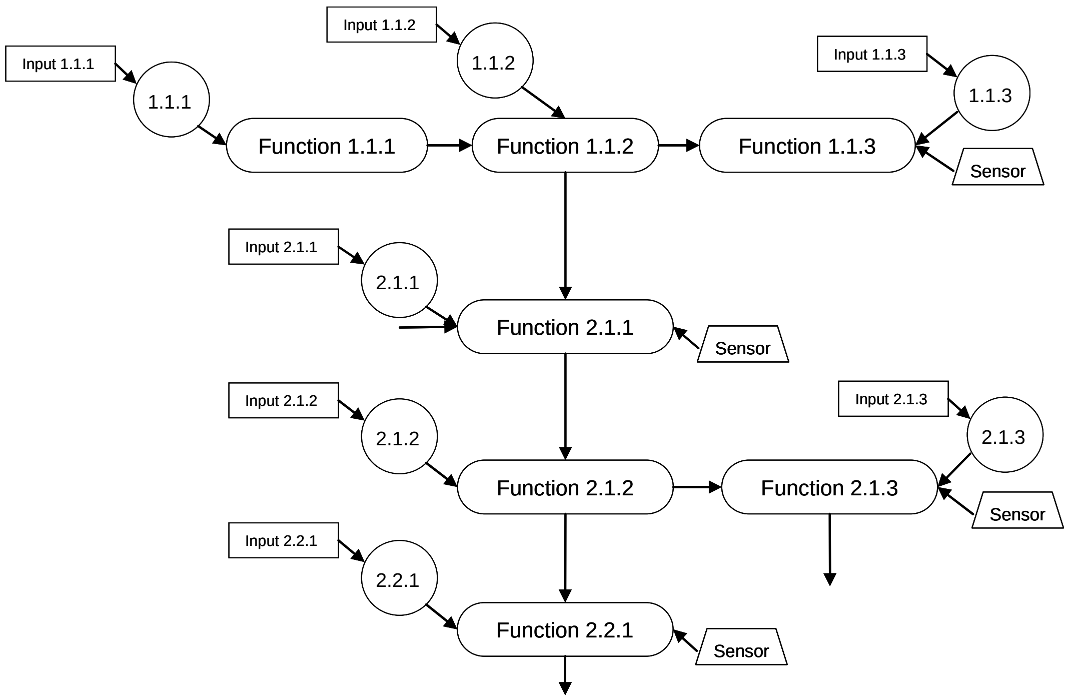
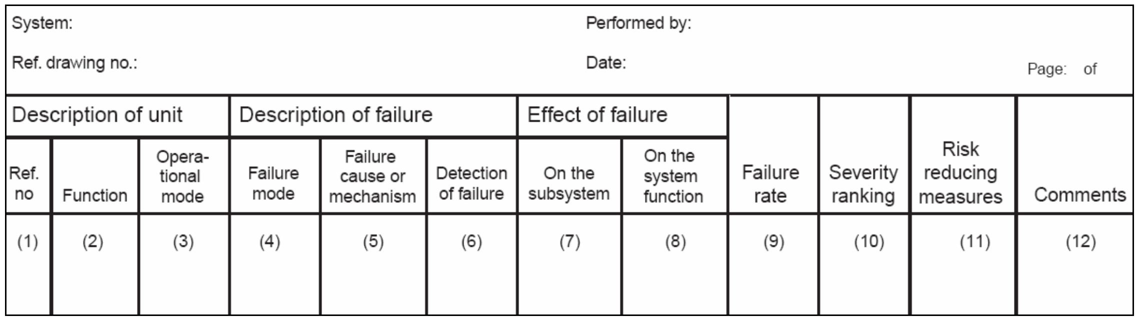


Figure 3.11 Functional block diagram of the system.

This analysis could grow very complex given the increased complexity of industrial systems. Therefore, it is advisable to carry out system analysis at as high level in the system hierarchy as possible. Later, if a finer resolution is desired for some particular subsystem or component in particular, they can be further expanded to include lower levels. This top down approach saves on effort and money compared to a complete analysis, which may not be even required.

STEP 2 - Failure Analysis: The next step is the core of FMECA study and identifies the critical failure modes that a system can be subjected to. It is extremely crucial to understand the physics of failure mechanisms for a good CBM/PHM system design. However, in several cases, a statistical account of frequency of occurrence from historical data provides valuable information for reasoning through expert systems. A systematic collection of data and algorithms for consistent inferences is extremely important in designing automated maintenance systems. The most common approach followed for such tasks is to prepare FMECA worksheets that consist of several columns describing the necessary details about each failure (Figure 3.12).



System: Performed by:

Ref. drawing no.: Date: Page: of

Description of unit			Description of failure			Effect of failure					
Ref. no	Function	Opera-tional mode	Failure mode	Failure cause or mechanism	Detection of failure	On the subsystem	On the system function	Failure rate	Severity ranking	Risk reducing measures	Comments
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)

Figure 3.12 A generic FMECA worksheet with most relevant columns. (Adapted from various sources)

These worksheets are a valuable resource and must be, therefore, carefully prepared from the point of view of building a knowledgebase. Apart from the description columns, the core columns that represent the FMECA philosophy are *failure rate* and *severity ranking*. In some cases additional columns, like *detectability*, and *replaceability* are added to get more specific information.

Frequency - For frequency of occurrence different classifications can be used. Figure 3.13 shows two such possible classifications.

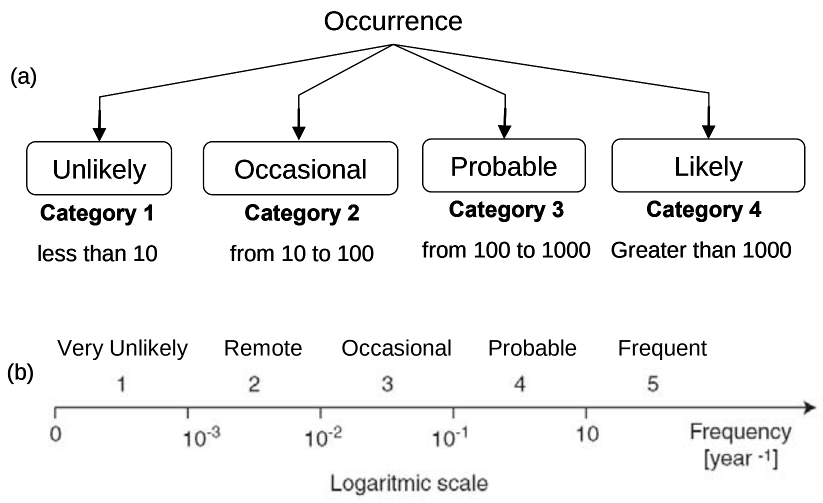


Figure 3.13 (a) A four category classification. (b) An alternative frequency of failure classification.

First, a four category classification distinguished on the basis of Mean Time Between Failures (MTBF) ranges has been shown. As an example, for a particular failure mode, based on a MTBF of 10,000 hours, the four categories may encompass the ranges shown in Figure 3.13(a). The probability of a fault occurrence may be based on a classification category number from 1 to 4 (or possibly more divisions) with 1 being the lowest probability to occur. Separation of the four classes is determined on a log power scale. The classification number is derived based on failure occurrence for the particular event standardized to a specific time period and broken down into *likely*, *probable*, *occasional*, and *unlikely*. In another example, as shown in Figure 3.13(b), a five category classification shows frequency of failure per year.

Severity – The next column is *severity* that categorizes a failure mode according to its ultimate consequence. The severity of a failure mode is assigned based on its worst possible consequence considered on the overall system. Several factors can be included to assess severity, but the most important ones are appropriate system operation, fault propagation, quality delivery, and operation safety. There can be different ways to classify severity as appropriate, for example a possible class breakdown may be as shown in Figure 3.14.

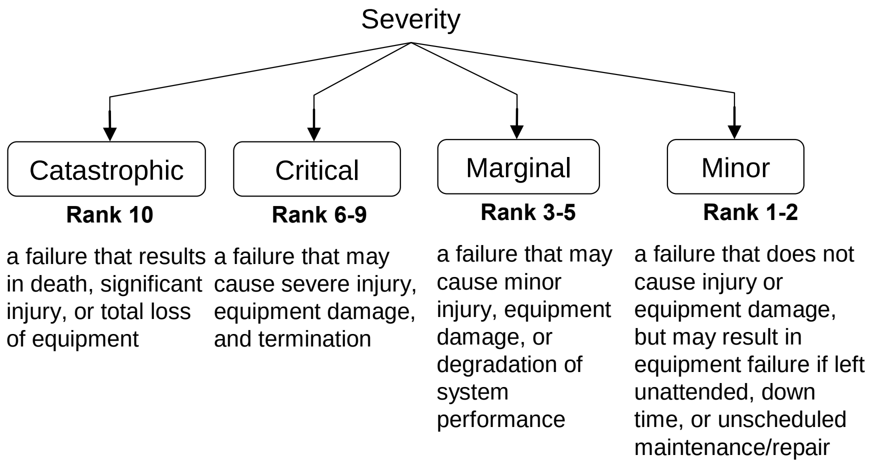
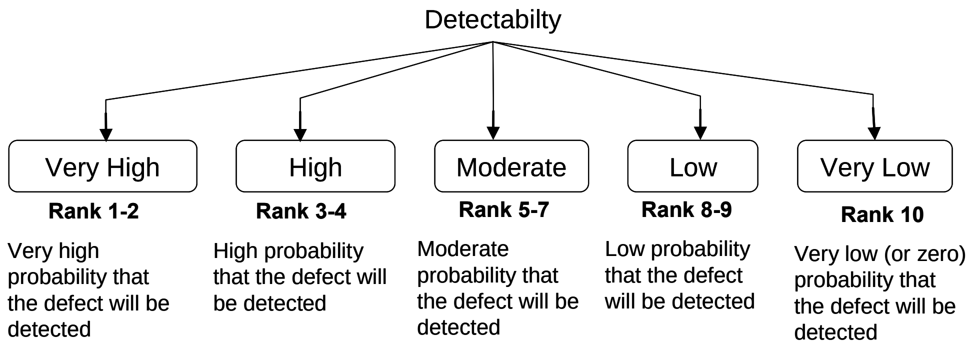

Severity
Catastrophic
Critical
Marginal
Minor
Rank 10
Rank 6-9
Rank 3-5
Rank 1-2
a failure that results in death, significant injury, or total loss of equipment
a failure that may cause severe injury, equipment damage, and termination
a failure that may cause minor injury, equipment damage, or degradation of system performance
a failure that does not cause injury or equipment damage, but may result in equipment failure if left unattended, down time, or unscheduled maintenance/repair

Figure 3.14 An example of severity classification [1].

Detectability - *detectability* or *testability* represents the ranking of failures based on the likelihood that the failure will be detected using the current configuration of the CBM system. Those failure modes that can not be observed are excluded from the candidate failure mode set. If the excluded failure mode is critical, additional sensing capabilities may be required otherwise no immediate action is needed. Assessing detectability may not be easy, as failures with low detectability may not show up until failure postmortem is carried out, and then a detailed analysis must be carried out to rank the detectability with the existing infrastructure. For instance, this parameter may consider two factors. One is related to 'how detectable the symptoms that a failure produces are'. Common symptoms are noise, vibration, or some other specific operational behavior, that can be associated to this type of the problem. The second factor is related to the time window available before the breakdown actually occurs. An easily detected failure with a long warning period should be given a higher ranking and an undetected failure or a detected one with imminent failure should be ranked low on detectability scale. The main implications of detectability include conditions with multiple simultaneous failures or failures under high noise environments where a more critical failure may go undetected. Better signal preprocessing and de-noising techniques are employed to improve the detectability of the failures. Once such rankings are generated they can be included in the criticality analysis by preferential weighting methods. For instance, a critical failure with a low detectability may be given a higher priority than a non-critical failure with high detectability. Based on the situation at hand and available detectability assessment capability a suitable ranking method may be coined as illustrated in Figure 3.15.



Detectabilty
Very High
High
Moderate
Low
Very Low
Rank 1-2
Rank 3-4
Rank 5-7
Rank 8-9
Rank 10
Very high probability that the defect will be detected
High probability that the defect will be detected
Moderate probability that the defect will be detected
Low probability that the defect will be detected
Very low (or zero) probability that the defect will be detected

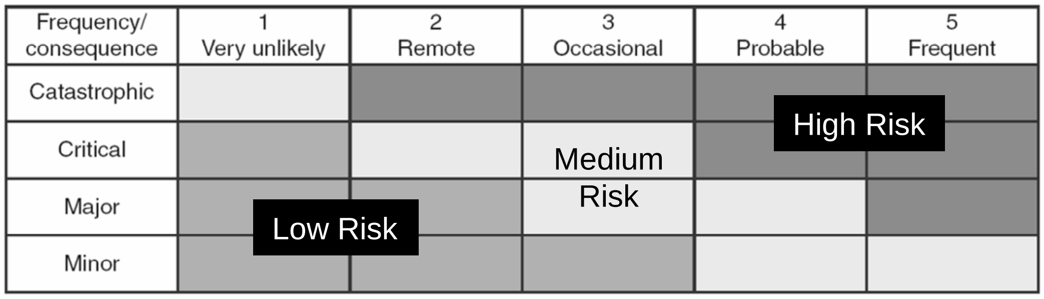
Figure 3.15 An example of detectability ranking method [19].

Similarly, latter columns in the FMECA worksheets can include possible actions to correct faults, and other remarks or information not yet covered in other columns.

STEP 3 – Risk Ranking and Review: once the system analysis is complete, overall metrics are computed to assess the overall risk before assigning priorities to different failure modes. Two of the most common methods used for this purpose are:

- Risk Matrix, and
- Risk Priority Number (RPN).

Risk matrix is plotted between two factors, namely the frequency of occurrence and the severity of the failure, as shown in Figure 3.16. More sophisticated measures or metrics can be designed to include more factors. One of the most commonly used measures is *Risk Priority Number* (RPN). There is no fixed definition of RPN; rather it is defined based on expert opinions. The simplest of RPN is a product of the ranks of the severity (s), Frequency of occurrence (o), and detectability (d). This definition of RPN requires all rankings to be done on a comparable scale, e.g. 1-10. Lower RPN indicates lower risk from that fault. Other definitions of RPN can be used as suggested by the experts, e.g. assigning different weights to different parameters.



Frequency/ consequence	1 Very unlikely	2 Remote	3 Occasional	4 Probable	5 Frequent
Catastrophic	░	▓	▓	▓	▓
Critical	▒	░	░	▓	▓
Major	▒	▒	░	░	▓
Minor	▒	▒	▒	░	░

High Risk

Medium Risk

Low Risk

Figure 3.16 Risk matrix to assess risk ranking.

After assessing the RPN, a review team should decide if the system is acceptable and how it can be improved to reduce the risk, by either improving the parameters, like detectability using additional sensors, and/or better detection algorithms, or by making the system more robust by correcting the major sources of these failures thereby reducing the frequency or the severity of the faults.

STEP 4 – Feedback and Corrections: Once the FMECA study has been conducted, the knowledge-based system can be built and deployed for use in the field. It is extremely important to keep monitoring the performance of the deployed system and measure any inefficiencies and inaccuracies during its operation. These deficiencies may arise due to two main reasons:

- Some critical failure modes went unnoticed or were not considered as critical enough while FMECA was being conducted. Such failures must be included in the study whenever discovered.

- Change in the external environment or the system itself results in new failure modes that should be accounted for and included in the FMECA study.

Therefore, a continuous feedback is required to identify any such deficiencies that should be corrected as soon as possible. In general an FMECA document is a running document that should be continuously updated.

FMECA is a very structured and reliable method for evaluating systems and at the same time it is very easy to understand and learn. It breaks down the complex systems into smaller modules and makes the analysis much simpler by varying the focus on different modules in order of their criticality. However, the whole process can still be very time consuming and tedious and therefore, it is important to make use of any existing information available from the manufacturers, operators or the domain experts. Also this study is not particularly suitable for simultaneous multiple failures and more specialized techniques need to be used to extend the basic FMECA methodology.

After acquiring the required information, the next important step is to codify this information in a form that can be accessed and reused for reasoning tasks. The next section discusses codification of knowledge.

3.4 Knowledge Codification

A very important concept in implementing an automated reasoning system is the Codifiability of the knowledge. Codifiability is the ability to make tacit knowledge explicit using formal written documents [20]. Knowledge codification involves turning knowledge, or parts of it, into messages that can be processed as information [21]. Thereafter, the codified knowledge exists in the form of codes or messages expressed in symbols. For these symbolic representations to be useful in a general sense, both the representational rules (grammar) and the notation (vocabulary) must be stable, and to some extent, standardized. A consensus must be established between the codifiers and the interpreters of the codified knowledge, regarding its meaning, without ambiguities. Further, the codified knowledge must be easier to distribute, store and recall, since these activities are all valuable, and an efficient coding of the knowledge should lower the costs of all of them. The codification should be done in a form/structure that eventually builds a knowledgebase to support decision making.

In theory, all knowledge may be codifiable but in practice it may not be possible to do so (Figure 3.17). Codifying knowledge involves effort in terms of labor, time and material, and hence there is a cost associated with doing so. If this cost is not justified, when compared to the benefits drawn out of it, is considered prohibitive for the codification process. Thus, whether a piece of knowledge is codified will depend on the relative costs and benefits of doing so at each level of the process. Of course, the evaluation of the relative costs and benefits can change over time and with circumstances. On the other hand, not all knowledge is codifiable. *Tacit knowledge* (e.g., human expertise) is identified and converted to a form that can be represented explicitly such that it can be shared and transferred without ambiguity. This explicit knowledge is organized, categorized, indexed, and accessed for automated use through expert systems.

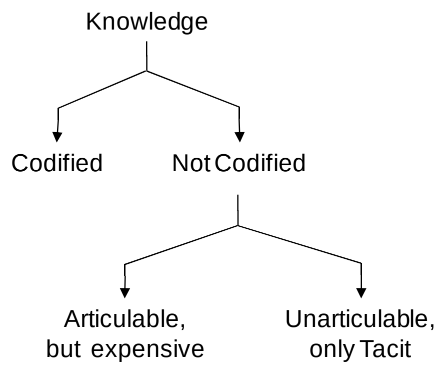


Figure 3.17 Topography of knowledge types.

There are several codification tools that are usually employed for a systematic knowledge codification. Some of the common tools include knowledge maps, decision tables, decision trees, frames, production rules, case based reasoning, and knowledge-based agents. Whereas, these tools use different forms, the main goal remains the same, and the choice of such tools depends on the situation and ease of molding the knowledge into a form suitable to any particular tool.

3.4.1 Process of Codification

To effectively practice knowledge engineering, a knowledge engineer requires knowledge in two main areas to carry out codification: *Knowledge Representation* and *Knowledge Modeling* [22, 23]. The main task of codification is to create messages, in a transferable form, that represent the tacit knowledge about the operation of the system and explain its behavior. But in order to create messages, a basic infrastructure of a suitable *representational language* is required in which these messages can be “written” and “read”. However, a language that is suitable in a particular context presupposes a *model* of the phenomenon. Therefore, in creating a knowledge-based expert system, first a model of the process should be created. This model is not the structural or operational model of the system; instead it is the model of the task at hand that needs to be performed by the knowledge-based system. Thus, for CBM systems it entails a detailed description of how a failure manifests itself in the system and in what form it becomes observable, e.g., through sensor measurements or in the form of peculiar symptoms that can be observed by the operators. In general there are two main aspects of the codification process:

- creating a model of the knowledge to be codified, and
- creating a representational coding language to express the model.

Once these aspects have been resolved the final task is to convert the knowledge into the coding language, which may not be trivial in all cases and must be handled systematically. The codification process can not be considered a simple transfer or translation operation as this aspect of creation of models and languages greatly influences the whole process. This process of creation defines a transformation at a fundamental level to describe how the knowledge is organized. Depending on the accuracy and fidelity of this transformation the codified knowledgebase may not entirely cover the entirety of the tacit knowledge that exists in the form of experts’ experience. Therefore, constant reviews and revisions need to be carried out by experts before an acceptable codification scheme can be adopted.

3.4.2 Knowledge Model

Knowledge Model refers to the framework that must be adopted to describe how the information collected from a system can be used to explain the system behavior and reason in unknown complex situations. Knowledge modeling is often considered as the first step in developing Knowledge-Based Systems (KBS). The aim of this process is to understand the types of data structures and relationships within which knowledge can be held, and reasoned with. The automation of usage and creation of knowledge can be considered as imitation of expertise in solving a specific class of problems. The choice of a correct knowledge model is very important for an effective knowledge-based reasoning system. Although there is no stable generic recipe to come up with the most appropriate knowledge model, some guidelines can be helpful. A knowledge model can be considered as a specialized case of specification requirement that satisfies all observed characteristics of a system phenomenon. A knowledge model mainly requires two components – *concepts* and a *reasoning methodology*.

Concept: A concept is an abstraction, typically associated with a corresponding representation in a description language, which denotes all members of a category, interactions, phenomena, and relationships between them. Concepts are derived by grouping multiple objects by virtue of their similarities and omitting the differences between them. In order to define a concept, a description language must be conceived. A knowledge specification is constructed first using a semi-formal language. As will be discussed later, defining a specification language itself assumes a knowledge model and hence a final formal language can not be specified from the very beginning. Once a crude model is specified in this semi-formal language, the language can be refined further before the knowledge model can be improved. This is an iterative procedure which results in refined concepts at a suitable level of abstraction [24].

Concepts can be represented in various forms (data-structures) that have been developed by artificial intelligence researchers e.g., semantic networks, frames, cases, decision diagrams, logic diagrams, etc., just to name a few. The choice of a suitable data-structure depends on the model of 'how experts reason in similar situations'.

Reasoner: A reasoner is the *inference engine* that makes use of data-structures describing the concepts, to manipulate them while performing reasoning tasks and suggest solutions to a problem. It makes inferences by deciding which rules are satisfied by facts or objects, prioritizes the satisfied rules, and executes the rules in order of decreasing priority [15]. As shown in **Table 3.1**, there are several different kinds of inferences and a choice of "which one to use" should be made depending on the application and format of the data available.

Table 3.1 Different types of inference methods.

Inference Type	Method
Deduction	Conclusions follow from their premises using logical reasoning
Induction	Reasoning from a specific case to general case
Intuition	Can not be explained, however possibly by unconsciously recognizing an underlying pattern
Heuristics	Rule of thumb based on experience
Generate and Test	Trial and error method, often used for quick search
Abduction	Reasoning back from a true conclusion to premises that may have caused that conclusion
Default	In absence of any specific knowledge, some default knowledge is assumed
Auto-epistemic	Self-knowledge
Non-monotonic	Previous conclusions may prove incorrect when more information is available
Analogy	Reasoning based on similarities to another situation

3.4.3 Representation Language

The primary aim of a knowledgebase is to store knowledge so that programs can process it and achieve the verisimilitude of human intelligence. AI researchers have borrowed representation theories from cognitive science to create representation languages. There are representation techniques such as frames, rules, and semantic networks which have originated from theories of human information processing and each of them makes use of different representation languages. Since knowledge is used to achieve intelligent behavior, the fundamental goal of knowledge representation is to represent knowledge in a manner that facilitates *inferencing,* i.e., drawing conclusions from knowledge. In order to manipulate and carry out inferencing, a representation language must be expressed (in written or spoken form) and should be allowed to be manipulated in a predefined consistent manner. A representation language mainly consists of two components – *vocabulary* and *grammar*. Vocabulary is a complete set of words that must be used to describe "any" situation related to the system. It consists of *nouns* that represent the names of various parts and/or processes in the system and *verbs* describing actions or activities taking place in the system. A complete list of such nouns and verbs generates a *dictionary*. A dictionary is a reference document that must be used while codifying and de-codifying messages in order to reduce ambiguity and loss of data during communication. It is desired to keep the vocabulary as small as possible but at the same time as complete as possible. Therefore, suitable abstractions of these words can be extracted and grouped under a common name to generate a *thesaurus*. A thesaurus contains synonym terms to establish relationship between texts with different words but similar meanings. Thus a dictionary would only contain these abstractions along with a pointer to the thesaurus. If a particular word is not available in the dictionary, the thesaurus is referenced and the corresponding abstraction is used for codification. Using these abstractions leads to some loss of information in the codification process but at the same time saves a significant amount of processing time in building and searching a relatively large vocabulary. Thus a balance should be established in deciding the level of abstraction and the resolution of the codification.

In addition to the vocabulary, a set of rules is needed to manipulate and interpret the combination of words before a meaningful and unambiguous context can be understood from it. Such a set of rules generates grammar for the language. Grammar restricts the manner in which the vocabulary can be used to impart an unambiguous contextual meaning to a *sentence* consisting of words in a particular order. Formal grammars are codifications of usage that are developed by observation. Just like a dictionary, grammar is also needed while codifying as well as de-codifying knowledge.

The concepts of knowledge acquisition discussed above will be illustrated with the help of two case studies. The emphasis here is on acquiring knowledge through FMECA study.

3.5 Case Studies

In order to illustrate and validate the theory presented in this thesis, different case studies will be used. These cases have been taken from different engineering domains to show the applicability of the generic approach. Data were partially available in most of these cases and hence these cases have been suitably selected to show different concepts wherever they are applicable.

For the illustration purposes, the knowledge acquisition concepts discussed in this chapter are demonstrated via two selected cases:

- CBM of Sludge Dewatering Centrifuges

- Monopropellant Propulsion System (included in Appendix B)

The knowledge acquisition approach is illustrated employing these cases in the following discussion.

3.5.1 Case: Sludge Dewatering Centrifuges

As described earlier, FMECA studies were conducted to acquire relevant data from the centrifuge system. The various steps of analysis are presented next.

The System: *Sludge Dewatering Centrifuge*: the system can be divided into two main parts – *Separator* and *Auxiliary systems*. (Figure 3.18)

STEP 0 - Background Study

a) *Scope of Analysis*: for the purpose of this study only the primary drive in the separator was chosen to be analyzed because the most critical and frequent failures were in the drive bearing, which is a part of the primary drive.

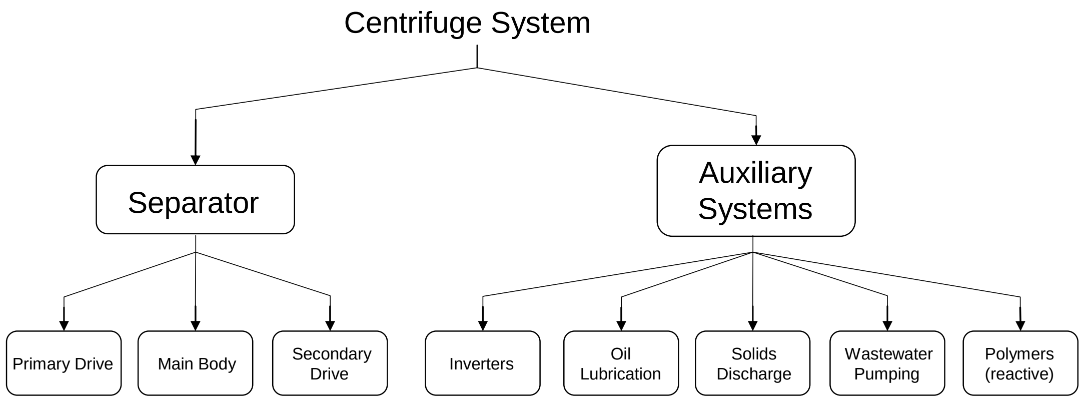


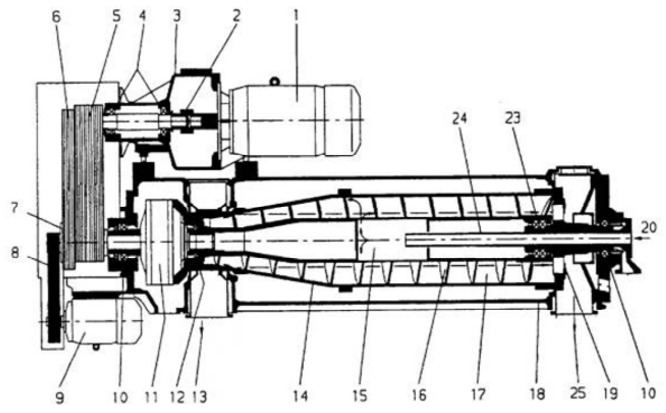
Figure 3.18 The system - sludge dewatering centrifuge.

b) *Functionality*: all components of the primary drive were included in the study.

c) *Operational modes*: the ‘normal operational’ mode was chosen to analyze the system.

STEP 1 - System Analysis

First, the separator unit (Figure 3.19) was divided into its constituent subsystems and components.



1	Drive Motor	13	Solids discharge
2	Clutch	14	Bowl
3	Clutch Housing	15	Distributor
4	Drive bearing	16	Scroll
5	Bowl drive	17	Separation chamber
6	Scroll drive	18	Housing
7	Secondary gear	19	Regulating ring
8	Variable speed drive	20	Feed
9	Secondary motor	23	Scroll bearing
10	Bowl bearing	24	Feed tube
11	Primary gear	25	Discharge clarified liquid
12	Scroll Bearing		

Figure 3.19 Separator of the sludge dewatering centrifuge.

Various parts were labeled with numbers and a structural decomposition tree was built (Figure 3.20).

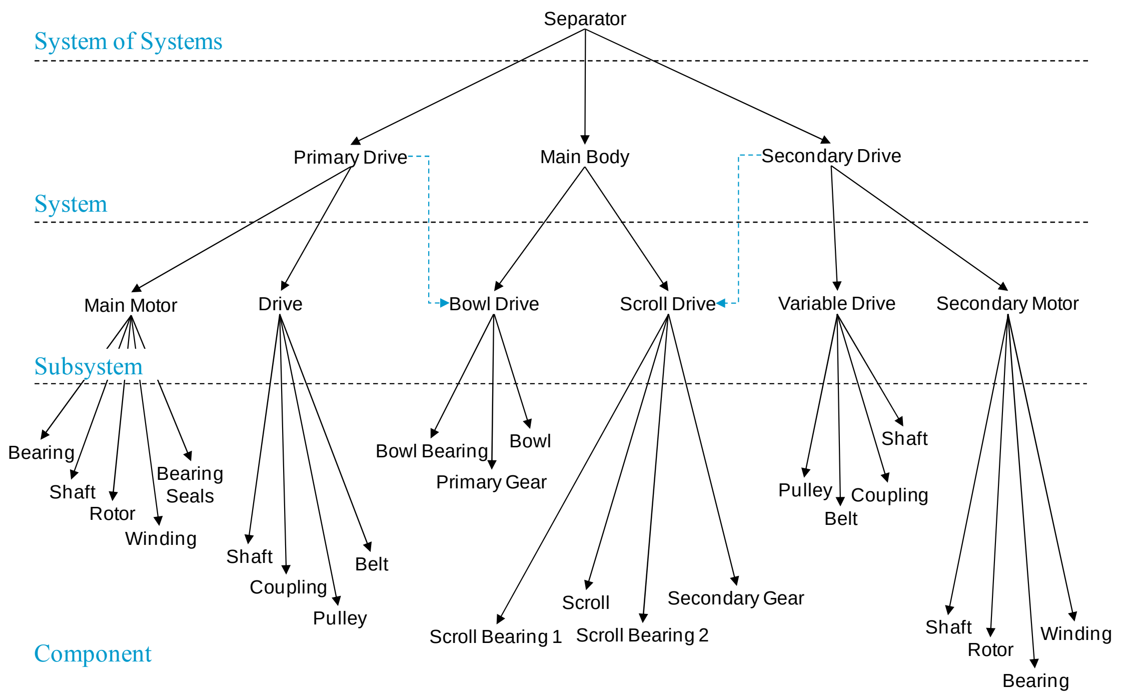


Figure 3.20 Structural decomposition of the separator.

The depth of the structural decomposition tree depends on the level of detail with which the analysis needs to be carried out. In this example we divided the system into four levels as shown in the figure. As mentioned earlier, only the primary drive was considered for analysis. A list of relevant components was compiled and a structural block diagram was prepared only with these components (Figure 3.21). Now each component included in the hierarchy is labeled with a hierarchical label number that identifies its parent system. This nomenclature helps in organizing the components in the knowledgebase.

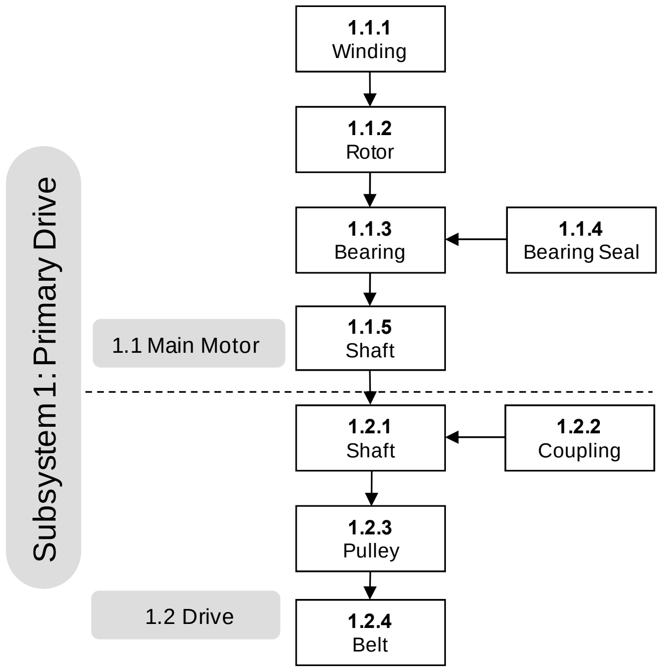
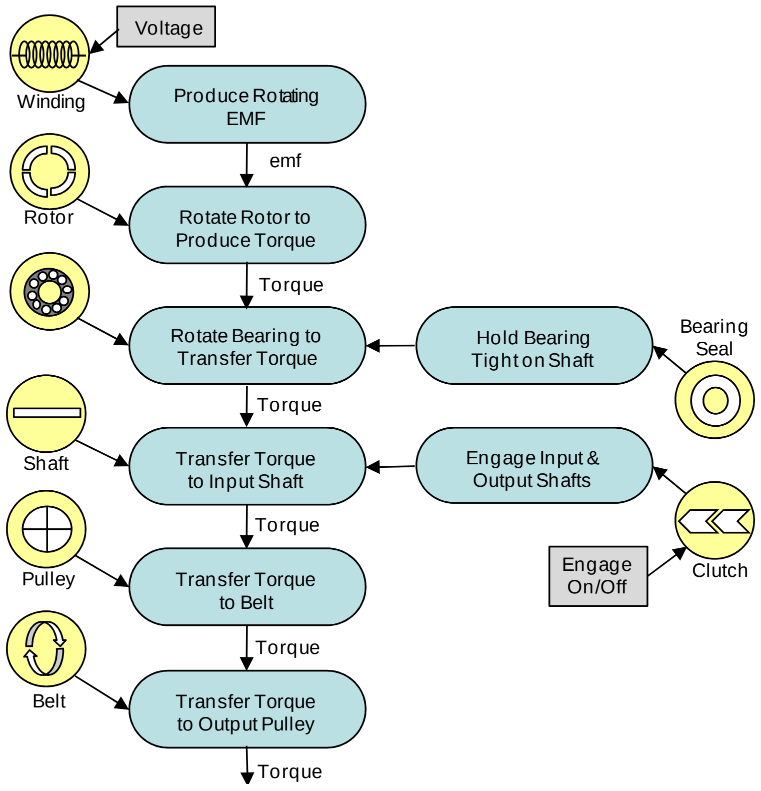


Figure 3.21 Structural block diagram of the separator.



Voltage
Winding
Produce Rotating EMF
emf
Rotor
Rotate Rotor to Produce Torque
Torque
Rotate Bearing to Transfer Torque
Hold Bearing Tight on Shaft
Bearing Seal
Torque
Shaft
Transfer Torque to Input Shaft
Engage Input & Output Shafts
Clutch
Engage On/Off
Torque
Pulley
Transfer Torque to Belt
Torque
Belt
Transfer Torque to Output Pulley
Torque

Figure 3.22 Functional block diagram of the separator.

STEP 2 - Failure Analysis:

Table 3.2 shows various failure modes associated with different components of the separator unit of the centrifuge. Data regarding various failure modes and their frequency, etc., were obtained through experts' interviews and maintenance logs and reports available from the facility. These failure modes were rated on a scale of 1-4 for frequency, severity, testability, and replaceability. In this case, these ratings were obtained as subjective evaluations of the operators in the facility. Furthermore, the system was divided into four distinct monitoring zones namely - main body, main motor, secondary motor, pulleys, and belts. Out of several sensors installed on the system, 10 relevant sensors were chosen to collect data and were mapped onto the separator unit as shown in Figure 3-25. A list of these sensors is given in Table 3.3.

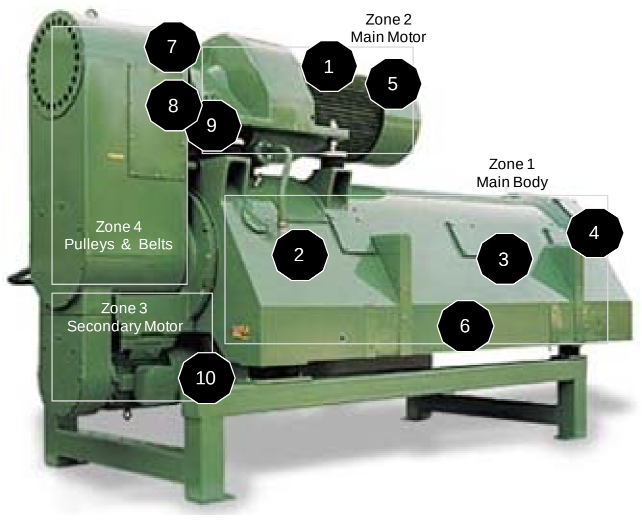


Figure 3.25 Sensor locations on the separator unit.
(Courtesy: Intelligent Automation Systems Inc., Atlanta)

Table 3.2 FMECA chart for the separator unit (Courtesy: Intelligent Automation Systems Inc., Atlanta).

System	Sub System	Parts	Failure Mode	Primary Cause	Symptoms	Evaluation					Recommended Action
						F	*S*	*T*	*R*	*I*	
Main Drive	Main Motor	Fan	Axle loose	Vibration and improper mounted	Noise and vibration	1	3	3	2	9	Replacement
			Blocked	External Obstruction	High current						
		Drive shaft and rotor	Wear Bend	Misalignment	Noise and vibration	1	4	2	2	16	Replace the motor
		Bearing	Blistering	Overheat Lack of lubrication Wear and tear Induce current	Noise and vibration High temperature	2	4	3	3	32	Replacement
			Crack								
			Wear								
		Winding	Loss of insulation Short circuit Mechanical contact	Overheat High dived humidity	High current High temperature	1	3	2	2	9	Rewinding
	Drive	Bearing	Blistering Cracks Rae defects	Overheat Lack of lubrication Wear and tear Induce current Misalignment	Noise and vibration	4	4	3	2	64	Replacement
		Seal	Crack Ware	Overheat Wear and tear High pressure	Noise Oil Leaked	4	4	3	4	32	Replacement
		Drive shaft	Bend	Wear and tear Misalignment	Noise and vibration	1	4	2	2	16	Replace the Motor
		Pulley	Bend Unbalance	Misalignment Wear and tear	Vibrations	1	3	2	1	24	Replacement
		Belt	Crack Break	Wear and tear	Noise and vibration	2	1	2	4	2	Regularly inspect and clean

Table 3.3 List of sensors on the separator unit. (Courtesy: Intelligent Automation Systems Inc., Atlanta)

Sensor No.	Type	Acquisition Speed	Description	Band-Width	CC	Notes	Location
1	Current	T=100 s 0.01Hz	Current measurement (Ammeter) (Main Motor)	1kHz	Digital	PLC	Zone 2
2	Temperature	T=100 s 0.01Hz	Temp. Measurement Temp. Feeler (PT100)	0.1Hhz	Digital	PLC	Zone 1
3	Temperature	T=100 s 0.01Hz	Temp. Measurement Temp. Feeler (PT100)	0.1Hhz	Digital	PLC	Zone 1
4	Flow	T=100 s 0.01Hz	Throughput measurement	0.1Hhz	Digital	PLC	Zone 1
5	Torque	T=100 s 0.01Hz	Computed	0.1Hhz		PLC	Zone 2
6	Vibration	T = 004s 5 KHz	Vibration monitor Accelerometer - Main Body	0.1 Hhz			Zone 1
7	Temperature	T=100 s 0.01Hz	External RTC Added (Main motor)	0.1Hhz	Analog 0-20 mA	Added to PLC	Zone 4
8	Temperature	T=100 s 0.01Hz	External RTC Added (Drive bearing housing)	0.1Hhz	Analog 0-20 mA	Added to PLC	Zone 4
9	Vibration	T = 001s 10 KHz	Accelerometer – Drive Bearing Housing	0.1 Hhz			Zone 2
10	Vibration	T = 001s 10 KHz	Accelerometer – Secondary Motor	0.1 Hhz			Zone 10

Once these data have been collected, the knowledgebase can be populated.

STEP 3 – Risk Ranking and Review:

To assign priorities to various failure modes, a composite *risk index* (I) was designed shown in equation (3.1). In this case the risk ranking considered the two the most critical factors namely the frequency (F) and the severity (S) of the failure modes. Other similar measures can also be designed if desired.

$$\mathbf{I = F{*}S^2.} \tag{3.1}$$

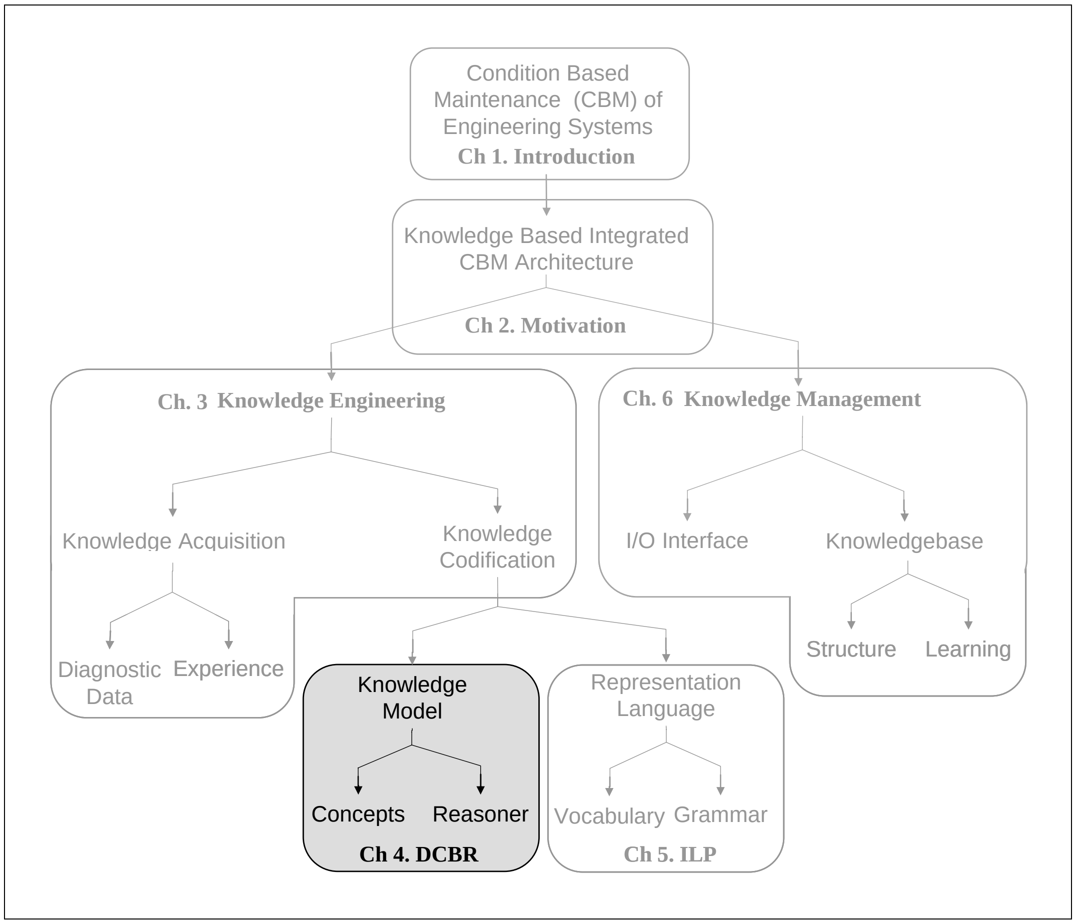
The final step of feedback and correction is not very relevant from the knowledge acquisition point of view and hence the corresponding discussion is not included here.

3.6 Conclusions

This chapter presented the knowledge engineering aspect of building a knowledgebase for the CBM systems. Building on the theory presented in this chapter, we have shown how common practices in the industry can be combined in a structured manner to gather data for the knowledgebase. Finally, an example case has been presented showing the application of such an approach.

Chapter 4

Dynamic Case-Based Reasoning



4.1 Chapter Overview

The next step after information acquisition and its transformation into useful knowledge is to codify it using a suitable representation language and a knowledge model. As discussed in Chapter 3, a knowledge model assumes the task of carrying out reasoning based on current facts from the system and its own experience from previous situations. This thesis builds on a Dynamic Case-Based Reasoning (DCBR) approach to accomplish the knowledge model. The Chapter starts with a brief description of conventional Case-Based Reasoning (CBR). Key components of CBR relevant to knowledge-based CBM have been discussed where further improvements are desired. Further, the DCBR philosophy has been described by introducing the dynamic components at multiple stages of the conventional CBR. This process model has been further instantiated with an example for industrial systems that can be applied to fleet vehicles.

Contribution: A Dynamic Case-Based Reasoning framework for CBM knowledge model.

4.2 Why CBR as Knowledge Model for CBM?

Tracking the central theme of this thesis to integrate knowledge into the CBM process, a knowledgebase must be built to store relevant maintenance experience. This knowledgebase should contain not only the structural and operational data from the system but also the experience in the form of associations between different problem situations and their corresponding remedies. These associations must be learned and modified over time as more experience is accumulated. Further, there must be a provision to appropriately use this knowledge in an automated manner. As previously discussed in Chapter 2, for this knowledge integration, a knowledge model is required that poses the following two main requirements:

Reasoner: To make use of knowledge contained in the knowledgebase an inference engine is required, which, based on its inference methods/rules, suggests a solution in a given situation. Several reasoning mechanisms have been used in the literature, e.g., logic based reasoning, model based reasoning, probabilistic or Bayesian reasoning, etc. The choice of a reasoning method is largely governed by the problem domain and the kind of data available from the system. In the CBM domain, the data are available in both, qualitative and quantitative forms and previous experience is very important in carrying out effective maintenance tasks. Keeping this in mind this research makes use of Case-Based Reasoning (CBR) as the main enabling technology to build and use the maintenance knowledgebase.

Concepts: From the coding point of view, the form and structure of the data must be defined. This mainly depends on the reasoner and the software design. Therefore concepts define the data structures suitable for a reasoning mechanism; these concepts represent meaningful knowledge capsules relevant to the problem domain. For maintenance tasks, CBR allows one to define concepts in the form of cases that represent past experiences. The concept of conventional cases has been extended to dynamic cases to incorporate the hybrid nature of case contents.

A detailed discussion on CBR and its current usage in the industry has is in the following sections.

4.3 Case Based Reasoning – An Enabling Technology

Mechanical systems in industrial environments are very complex and extremely difficult to model. Moreover, the increasing demand for efficiency and continuous uptime requires a quick and robust solution to commonly occurring problems in such environments. In most cases, first, it is next to impossible to establish precise and accurate models for these systems and even if they can be modeled, it is prohibitively expensive to solve them in real time to produce a solution. An alternative approach for this task is *not* to solve these problems every time from scratch but to recall the solutions from past experience instead, if such experience exists, of course. This approach is based on the premise that *problems recur* in nature. Thus, if in the past a successful solution was generated for a similar problem at least once, it is not required to explicitly search for a solution over the entire search space again. The previous solution is likely to be in the proximity of the required solution for the current problem and hence the search space is considerably pruned. Therefore, the solutions can be derived by drawing an analogy to a previous situation and this forms the basis of Analogical Reasoning (AR).

Analogical Reasoning is an AI technique which emulates the process of human reasoning by remembering [25]. It tries to recall the already known situations which are similar to the situation at hand and draws analogies to come up with an acceptable solution. A wide spectrum of AR problems can be defined where, at one end the analogies are drawn within the *same domain* and, at the other, analogies are drawn between *different domains*. The main difference between the two reasoning extremes is the *mapping* or *matching* of components in the two systems that play similar roles even if their domains are not the same. The hard problem in AR is to decide *what to transfer* and *what not to transfer* between the two domains in terms of knowledge. This is an active area of research and no concrete solutions are yet available. However, by constraining within a single domain or very closely related domains the problem can be considerably relaxed. This relaxed form of AR is Case-Based Reasoning (CBR), which solves problems by adapting successful solutions that were used for similar problems in the past [26]. The history of CBR can be traced back to about the year 1977. However, due to technical limitations of limited computing power it was only a topic of academic interest until the late 1980s and early 1990s before it became popular in industrial applications.

The classical and most widely adopted model of CBR was described as a cyclical process comprising the *four Rs* (Retrieve, Reuse, Revise, and Retain) [27] as shown in Figure 4.1.

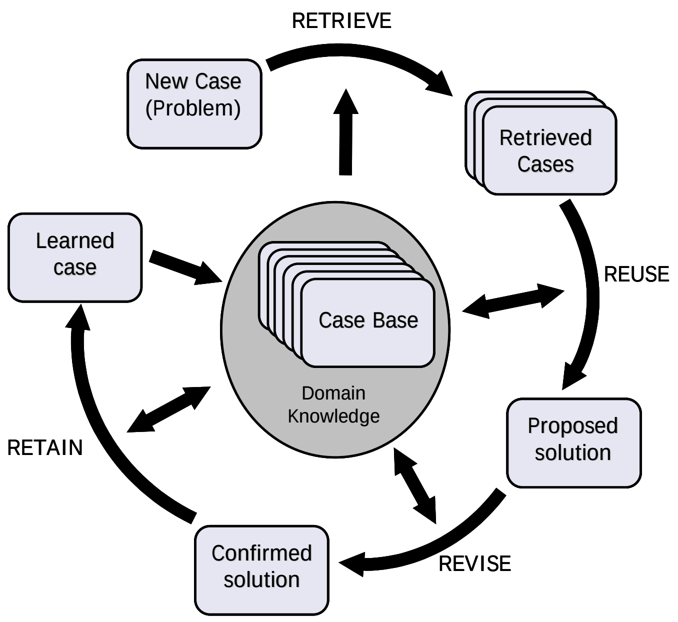


Figure 4.1 The CBR Cycle, adapted from [27].

These four fundamental steps are:

- RETRIEVE the most similar case(s),
- REUSE the case(s) to attempt to solve the problem,
- REVISE the proposed solution if necessary, and
- RETAIN the new solution as part of a new case.

A new problem (*query*) is matched against cases in the existing case-base and one or more similar cases are *retrieved*. A solution suggested by the matching cases is then *reused* and tested for success. Unless the retrieved case is a close match, the solution needs to be *revised* producing a new case that may be *retained* for later reference.

Currently, this cycle rarely occurs without human intervention and most CBR systems are mainly used as case retrieval and reuse systems [28]. Case revision (i.e., adaptation) is often carried out manually by the managers of the case base. AI systems are not yet completely autonomous with current state-of-the-art; however, they have proven to be very successful as *Decision Support Systems* (DSS). Several applications are discussed in the following sections where appropriate machine learning techniques for data mining are employed by CBR to propose and approximate solution and make it much simpler for the users to come up with a good solution.

4.3.1 CBR Implementation Issues

CBR requires a specific structure in which the domain knowledge should be codified. However, the format and content of this structure can vary according to the problem domain. A brief description of how these processes can be carried out is included below.

4.3.1.1 Case Representation

A case is a contextualized piece of knowledge representing an experience. It contains the past lesson, i.e., the content of the case and the context in which the lesson can be used [29]. Typically a case comprises:

- the *problem* that describes the state of the system when the case occurred,
- the *solution* which states the derived solution to that problem, and/or
- the *outcome* which describes the state of the system after the case occurred.

In other cases, additional case components have been suggested to further enhance the CBR structure. A notion of *maintainable* cases has been suggested in [30] for which he includes an administrative part in the case called *administrativa*. Irrespective of what other case components may be defined for improved capabilities, the basic philosophy revolves mainly around the *problem* and the *solution* parts.

In its simplest form, cases can be represented as vector of *attribute-value* pairs characterizing the problem and solution respectively. Further, based on the nature of case contents, cases can be classified in different forms as summarized in Table 4.1.

Table 4.1 Types-of-cases classification.

Classification Criteria	Classification	Definition	Ref.
Distinction between Problem and solution parts of the case	Rule-based cases	It is predetermined at the design time which attributes are part of the Problem and which are a part of the solution.	[31]
	Constraint-based cases	It is determined only at run time which attributes are part of the Problem and which are a part of the solution	
Homogeneity of cases	Homogeneous cases	The set of attributes in a case remains fixed. For example, real estate domain	[28]
	Heterogeneous cases	It is difficult to know full set of attributes. New attributes may be learnt with time. Therefore cases in a case base may contain some non-common attributes. E.g., patient data in a hospital	
Source of cases	Episodic cases	Cases are records of events. E.g., equipment fault logs, patient files etc.	[28]
	Prototypical cases	Cases are designed by experts as examples of events. E.g. symptoms of failures acquired using field tests	
Structure, coverage and interpretations	Simple cases	Have fixed structure, cover specific situations clearly and interpreted in a well defined way while ready for reuse.	[32]
	Complex cases	Defined by complex layouts, topologies or structures difficult to define with attribute-value pairs, may need several partial cases to cover a situation and may be interpreted in different way for different situations.	
Time period over which cases carry information	Discrete time cases	Cases represented by a snapshot of events. All values are recorded at a particular time instant to measure the state of the system	[33]
	Continuous cases	Cases are represented by continuous time events or a series of discrete events. It is important where current state depends on past temporal states	
Evolution of case contents	Static cases	Case contents remain fixed over a period of time. With significant difference in attribute values a new case is instantiated.	[34-36]
	Dynamic cases	Case contents keep changing or evolving as new experience accumulates. i.e., a case contains statistical account of values rather than fixed values.	
Type of case contents	Pure cases	Case attributes are all of similar type, e.g., numerical, textual, graphical, etc.	[35]
	Hybrid cases	Case attributes can be of mixed types	

Cases can be represented in a variety of forms using the full range of AI representational formalisms including frames, objects, predicates, semantic nets and rules. The choice of a particular formalism is largely governed by the case content. There is a lack of consensus within the CBR community as to exactly what information should be in a case [28]. However, [29] suggests two pragmatic measures that can be taken into account while deciding what should be represented in cases: the *functionality* and the *ease of acquisition* of the information represented in the case. Practically, there is a trade off between how much information should be included in the cases and memory requirements, and a balance should be established based on the application domain and requirements.

4.3.1.2 Similarity Metrics for Case Comparisons

Under the core CBR assumption, that similar problems have similar solutions, the usefulness of stored cases is evaluated by comparing their respective problem situation to the problem description in the *query* case (*target* case). The query case is not just an informal description of the problem at hand, but a formalized version complying with a case representation language, already specified for the problem domain at hand. In most situations the query cases are expected to be incomplete and vague in the beginning. The type of similarity metric used may be different depending on the nature of various case attributes. In some cases, where the attribute's structure is more complex and requires special considerations, several types of similarity notions may be combined to form a composite similarity metric. In most situations, cases are represented as attribute vectors: $a = [a_1, a_2, \ldots a_k]$. In these situations, first a *local similarity* is computed between each component and then composed into a *global similarity* by taking a weighted sum of all components. In addition to these notions for similarity, other notions of *utility* and *acceptance* have been introduced for case comparisons. A detailed discussion on similarity metrics is included in Chapter 5.

4.3.2 The Issue of Uncertainty in CBR

Traditional information systems simply convert data into information and rarely address uncertainty because their inputs and outputs are not expected to be flexible and all the flexibility is expected from humans. However, systems especially the ones that manipulate knowledge explicitly are expected to be flexible as they try to emulate human level intelligence.

[37, 38] define three broad categories of uncertainty:

- *Incompleteness* occurs when there are missing values for elements.
- *Imprecision* occurs when a value of an element is given but not with necessary precision.
- *Uncertainty* occurs when a given statement might be wrong.

Uncertainty arises due to the fuzziness or randomness in the inputs which then propagates to the decision making. Attempts to minimize this uncertainty are made in [38].

A four-container concept was given by [31] to define the sources of uncertainty in knowledge and information in CBR systems. These four knowledge containers are:

- *Vocabulary*: it includes the definition and descriptions of elements, entities, attributes, strategies, conditions etc., which generates imprecision depending on the level of abstraction chosen to represent the case.

- *Similarity Measure*: it includes indexing, similarity functions, aggregation of similarity functions, and selection. Several methods to learn similarity measures, feature weights, and indexing are also included in this container. [39] discuss how distance measure as a notion of similarity leads to uncertainty in CBR systems.

- *Case-Base*: it contains the organizational structure of cases (flat, hierarchical or networks), case-base maintenance methods, case contents (complete and incomplete cases), etc. An insufficient case base that does not cover all prospective problems is yet another source of uncertainty. Methods to learn new cases also add to the uncertainty.

- *Solution Transformation*: it includes the methods used for the reuse and revise steps in CBR. One source of uncertainty may arise when there are more than one candidate solutions and a method for solution composition is required. A new problem that is more specific than the stored cases leads to imprecision. And an insufficiently designed adaptation leads to ignorance.

4.3.3 CBR in Industrial Practice – State-of-the-Art

In most situations health information is obtained from monitoring sensors and automatically generated activity logs. But most of the systems use only the quantitative information available from sensors to automate the diagnosis task and almost none or very little use of the qualitative information is made. For instance, a CBR system for fault diagnosis in industrial robots has been described in [40] using a case base consisting of acoustic signals acting as fault signatures. But this approach largely depends on signal processing of the data and can be aptly applied only where good sensors and features have been identified for fairly accurate fault detection and identification. Another attempt focused on diagnosis of electronic ballast on the airplanes for developing an aircraft maintenance system [3]. Several numerical features were included and genetic algorithms were employed to assign weights to those features for similarity calculations. But any available qualitative information was again ignored. GE has been using CBR for monitoring and diagnostics in a variety of systems including medical equipment (ELSI), locomotives (ICARUS), and heavy-duty gas turbines [4, 5]. But none of these systems have yet considered both numerical and textual data together. A diagnostic system for aircraft fleet maintenance was developed using failure and warning messages generated by on-board aircraft diagnostic routines [7]. This approach considered only formatted text messages for which a trigram-matching technique was utilized to retrieve similar cases. This approach works well only if the text structure is fixed and word ordering does not dictate the meaning of the phrase. Similarly, a partial matching technique, based on key words and constraint-nets, was used to retrieve similar cases from the case-base [6]. This process also focused only on textual information and no numerical analysis was carried out.

4.3.4 Shortcomings of the Conventional CBR

Case-based reasoning (CBR) is an approach to problem solving based in retrieval and adaptation of cases, or episodic descriptions of problems and their associated solutions [17, 27, 29]. As discussed earlier CBR cycle typically involves four *REs* namely *Retrieve, Reuse, Revise* and *Retain* as described in [41]. CBR is a knowledge representation and manipulation technique that recalls solutions to past problems that are likely candidates for solutions to new problems. It is not uncommon for several cases to be retrieved at any one time. The retrieved cases must be ranked in order of their relevance to the current case. Six "filters" have been used for case ranking: goal-directedness, salient features, specificity, frequency of recall, recency, and ease of adaptation [29]. In practice, both the implementation and inclusion of each of these steps and the knowledge representation used for cases varies widely. For example, case retrieval has been implemented using nearest neighbor algorithms, decision trees, or connectionist associative memories; case representations range from free-text documents to data base records to semantic networks [17]. However, in almost all cases the content of a case is predetermined and remains fixed while that case resides in the active case library. Moreover, since CBR has been applied mostly in specialized domains, the adaptation algorithms are more or less fixed and any adjustments due to changes in the environment are carried out through scheduled case-base maintenance operations. In fact adaptation in most commercial decision support systems is bypassed by suggesting an approximate solution obtained from the closest matching case without any modifications.

4.4 Dynamic Case-Based Reasoning

A dynamic case-based reasoning system strives to improve on the above shortcomings of the simple CBR systems. Several variations in the structure of CBR have been implemented, based on specific requirements, as the use of CBR is being explored for more applications. In [36] authors describe a system where time-tagged indexes and dynamic composite features make the CBR system dynamic. In another approach cases are extracted and expanded dynamically based on context and the facts specified in advance [42]. Although the basic steps in DCBR cycles essentially remain the same, they can be internally modified to incorporate a dynamic component to increase autonomy of these systems.

4.4.1 The DCBR Lifecycle

Lifecycle of a DCBR system can be divided into three major phases, shown in Figure 4.2.

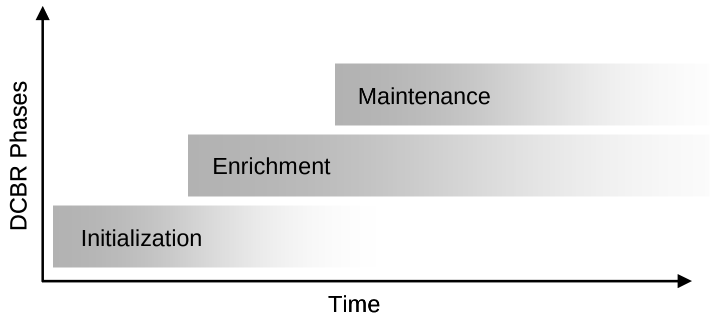


Figure 4.2 Lifecycle of a DCBR system.

Initialization Phase: This phase includes designing a general representation of the information, choosing appropriate features for indexing and a mode of interaction between the application and the actual system. This phase includes initial installation of the DCBR system using a "seed" case base to provide a baseline for the application. This initial case base is sparsely populated with cases that have been encountered most frequently and constitutes the prior knowledge available to the DCBR system. It may not cover the entire range of problems because they have either not been encountered so far or have been forgotten from the past. This knowledge is made available through current experts of the system or maintenance logs.

Enrichment Phase: The initial case base is used and refined through the validation and storage steps performed concurrently throughout the user base. This involves populating the case library with newer cases as they are encountered. This is an ongoing process throughout the life of a DCBR system, and remains relatively more active until a valid case base covering a large portion of the application area is complete. This phase tends to fill the gaps, between various possible scenarios, that were created during initialization. Every time a problem case is presented (assuming a fault has been detected), it can either be grouped with one of the old cases based on its similarity or considered a new case. If a new case is detected, the case library is simply updated by including this case. But if it matches an existing case with some predefined degree of confidence, the statistic-vector of the corresponding representative case is updated. This approach mainly works if the case contents are either numerical in nature or can be otherwise transformed into numerical entities. All similar cases are grouped and, instead of storing multiple cases, a statistical distribution of each is kept in a dynamic case. A dynamic case thus represents an n-dimensional cluster where n is the number of different components in the case. A new dynamic case is generated when including a new case increases the variance of the cluster above a preset limit. Using statistical inferencing techniques, like hypothesis testing, this statistic-vector can be used for:

- *Fault Detection*: to decide if the fault is present or not, and
- *Fault Identification* and Isolation: to decide which fault(s) is (are) present.

This may indicate the presence of several faults and hence more than one matching cases (some with a different degree of severity, while others with different fault characteristics). Cases can be ranked according to the preferences like the ones enunciated by [29]. Figure 4.3 provides a simple illustration of a dynamic case with a statistic vector.

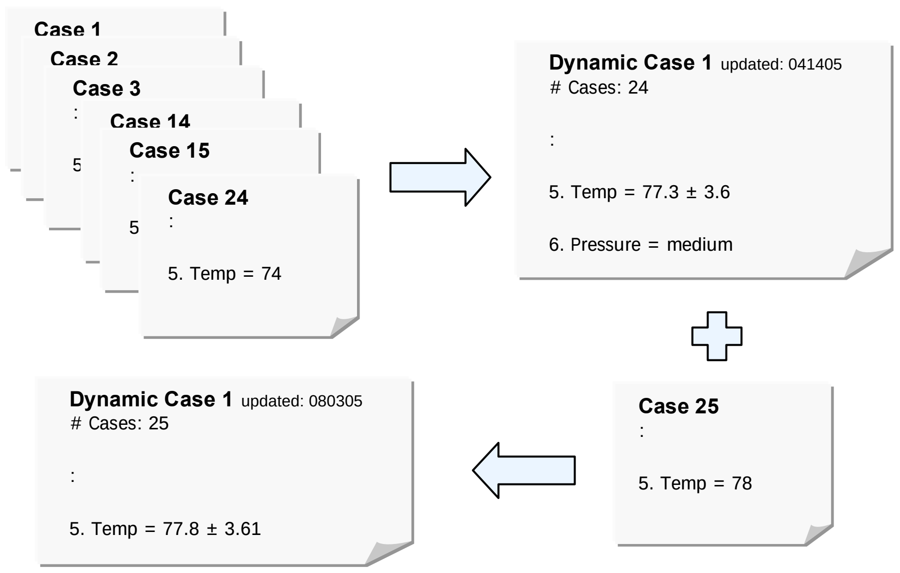


Figure 4.3 Creating and updating a dynamic case using a statistic vector. Descriptive features can be converted to numerical values using techniques like fuzzification.

The advantage of such a technique is that the similarity metric is not calculated based on one representative case, which may or may not be the best representative for the corresponding problem, instead it is based on a distribution of all similar cases encountered in the past without explicitly storing all of them. Hence a suitable statistic vector replaces the description of case, making it “dynamic”.

Case Base Maintenance Phase: This is a relatively less frequent process but continues throughout the life of the system to cope with the non-stationarity of the environment. This phase is expected to commence after the active enrichment phase is completed. Based on past successes and failures, a performance assessment can be carried out for various parameters that are being monitored. Weights can be assigned to cases and their constituent components to express their relevance in solving a particular fault. Some sensors can be tagged as redundant and others as more important or the need for alternative sensors or sensor placement can be inferred. Once the required changes have been incorporated, the case structure is likely to change and the old case structure can be modified such that the useful part of the old cases can still be used and the redundant part is discarded. This can also include time stamping and archiving of very old cases which have not been used for a long time and have less chance of appearing again because either the system has changed or they are covered by some newer cases.

4.4.2 Three Dynamic Components of DCBR

Figure 4.4 illustrates a high level schematic of DCBR that has been extended over the classical CBR cycle adopted in [27]. DCBR is “dynamic” by three means as compared to the conventional CBR.

Dynamic Case Library: As in conventional CBR the case library is dynamic by virtue of its evolution for almost as long as newer cases are encountered. Beyond that, the case library is not just a static collection of old cases but an entire knowledgebase that continuously learns from external feedback. Attributes of self-evaluation and self-organization directly define a dynamic case library.

Dynamic Case: The cases are dynamic as they continuously evolve over time. As more similar cases are encountered they are grouped based on their spatial locations and compressed into a dynamic case by creating the statistic-vector illustrated earlier in Section 4.4.1. Thus a case is a statistical representation of a group of closely matching situations rather than one situation per case.

Dynamic Reasoning: The reasoning is dynamically performed based on what part of the information (case component) included in the cases is given more importance for making inferences and hence producing an appropriate solution scheme. Several adaptation schemes and similarity metrics can be dynamically chosen based on what was more successful in the past for similar situations. A specific example for diagnosis of fleet vehicles is described in the next section.

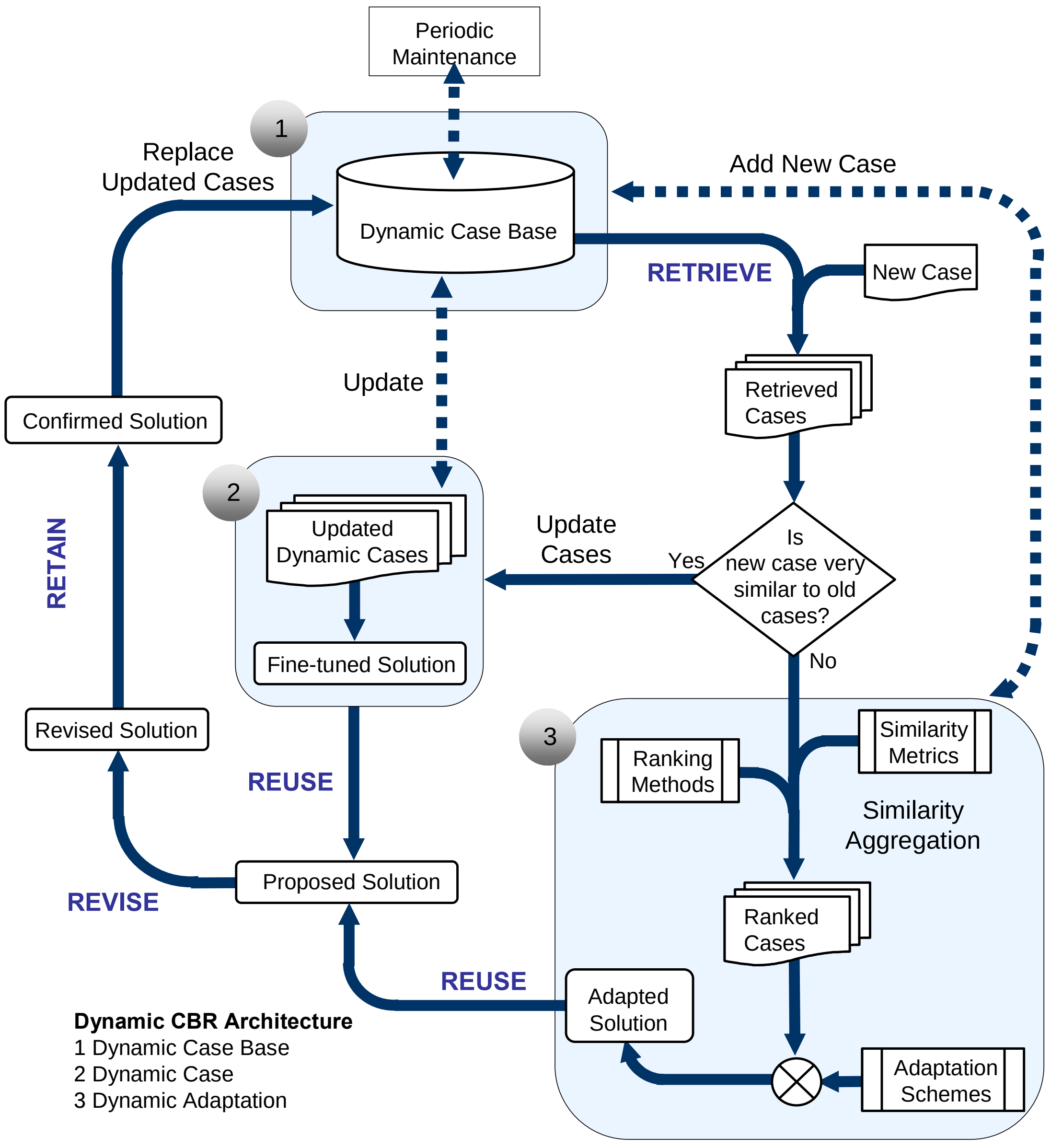


Figure 4.4 High level schematic of Dynamic Case Based Reasoning highlighting its dynamic components.

4.5 DCBR Application to Fleet Vehicles

An integrated architecture has been developed for CBM of fleet vehicles using the above mentioned DCBR concepts [34]. DCBR offers a good promise for diagnostic and decision support systems by emulating the human reasoning process one step further. It narrows down the problem search space by dividing the diagnosis tasks into smaller steps. In most cases, industry uses a two-step procedure. First, the operators suspect the problem by observing unusual symptoms or the error logs from the system. Maintenance experts try to come up with possible explanations for those symptoms. Then using known analytical techniques relevant diagnostic tests are run to confirm the problem. After a problem has been diagnosed correctly experts use their experience to plan and execute the repair task. Our diagnostic system follows a similar approach. It refines an asynchronous stream of symptom and repair actions along with sensor information into a compound case structure to organize the relevant information into the case memory, as shown in the schematic in Figure 4.5.

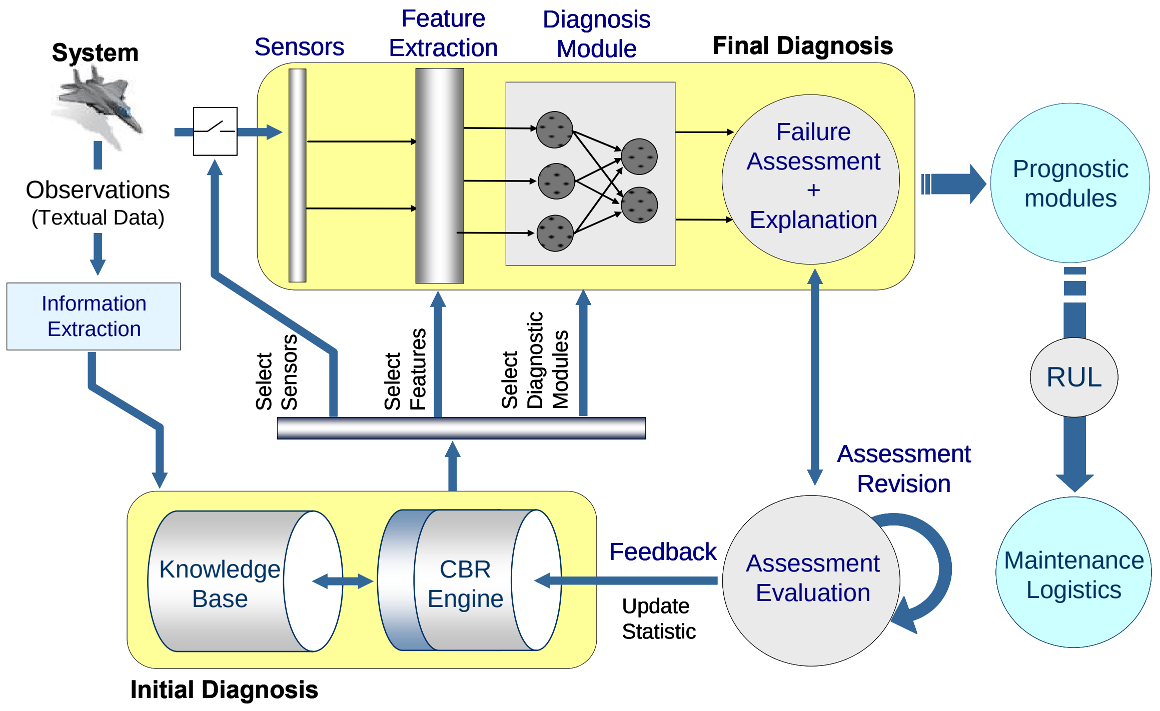


Figure 4.5 Integrated reasoning architecture for fault diagnosis in industrial environments.

Qualitative information (textual descriptions) is used as the initial query. These descriptions are converted into semantic networks (see Chapter 5), which preserve the meaning of the text and at the same time convert the text into a defined structure for easier analysis [43]. The case-base is searched based on these semantic networks and relevant hypotheses are generated. These hypotheses are ranked based on past experience and the most probable hypothesis is tested first by automatically activating the relevant data acquisition and diagnostic modules. If the hypothesis is confirmed to be true its solution is suggested for the current situation and its success rate is updated. Otherwise the next probable hypothesis is tested and the corresponding success rates are updated. The procedure is repeated until a useful solution is obtained or a new case is generated and stored in the case base. For new cases approximate solutions are suggested based on closest matching cases. They are further revised based on feedbacks until a satisfactory solution has been found.

4.5.1 The Dynamic Case

A list of all the sensors employed in the system and their target components is usually available from the manufacturers or the operators and is acquired under the process of knowledge acquisition. Similarly a list of diagnostic features from the sensor data to diagnose known fault modes can be compiled. Relative importance of features and their underlying philosophy can also be included in such a list. The relative importance of features may not always be available, especially in the case of new situations. In such situations these weights are learnt using the corresponding success/failure rates. Knowledge of this kind can be added to the list as it is generated in due course. All this information can be organized into a dynamic case structure as shown in Figure 4.6. This case structure is dynamic by virtue of two attributes:

- Case components are loaded step by step as the hypotheses are generated. Therefore, there is no fixed case structure with a fixed set of attribute-value pairs.

- Case components can have multiple values ranked with the corresponding weights unlike one-to-one attribute value pairs in conventional cases.

Case Structure					
.ID	Case_ID				
.Component	Component_Name				
.Location	Component_Location				
.Symptom	**S_ID**	**Symptom**	**Weight**	**Sementic_net**	**Hypotheses**
	1	s1	w_{s1}	SemNet1	h_1, h_2
	2	s2	w_{s2}	SemNet2	h_1
	3	s3	w_{s3}	SemNet3	h_2
.Hypothesis	**H_ID**	**Hypothesis**	**Weight**	**Diagnosis**	
	1	h_1	w_{h1}	d_1	
	2	h_2	w_{h2}	d_2, d_3	
.Diagnosis	**D_ID**	**(Sensor,Feature) pairs**	**Weight**	**Solution**	
	1	d_1:{(S_1,F_1),(S_1,F_2)}	w_{d1}	r_1	
	2	d_2:{(S_1,F_2),(S_1,F_3)}	w_{d2}	r_2	
	3	d_3:{(S_5,F_1)}	w_{d3}	r_3	
.Repair	**R_ID**	**Repair**	**Weight**		
	1	r_1	w_{r1}		
	2	r_2	w_{r2}		
	3	r_3	w_{r3}		
.Version	**Last_Update**	**Case_Quality**	**Success**	**Failure**	**Condition**
	mm:dd:yy	W	nS	nF	C1,C2,C3

Figure 4.6 Generic dynamic case structure. Symptoms are stored as semantic networks.

Figure 4.7 shows a graphical version of the dynamic case. The static component consists of relational entities that are used to retrieve matching cases. The dynamic component of the case is connection *weights* that keep changing with time. These weights help accumulate evidence for incoming node from the outgoing nodes. These cases are *hybrid* cases as they can contain different types of components such as semantic nets, numerical data, text data, and Meta data such as ranking weights. Primary case indexes are CaseID, Component and Component Location.

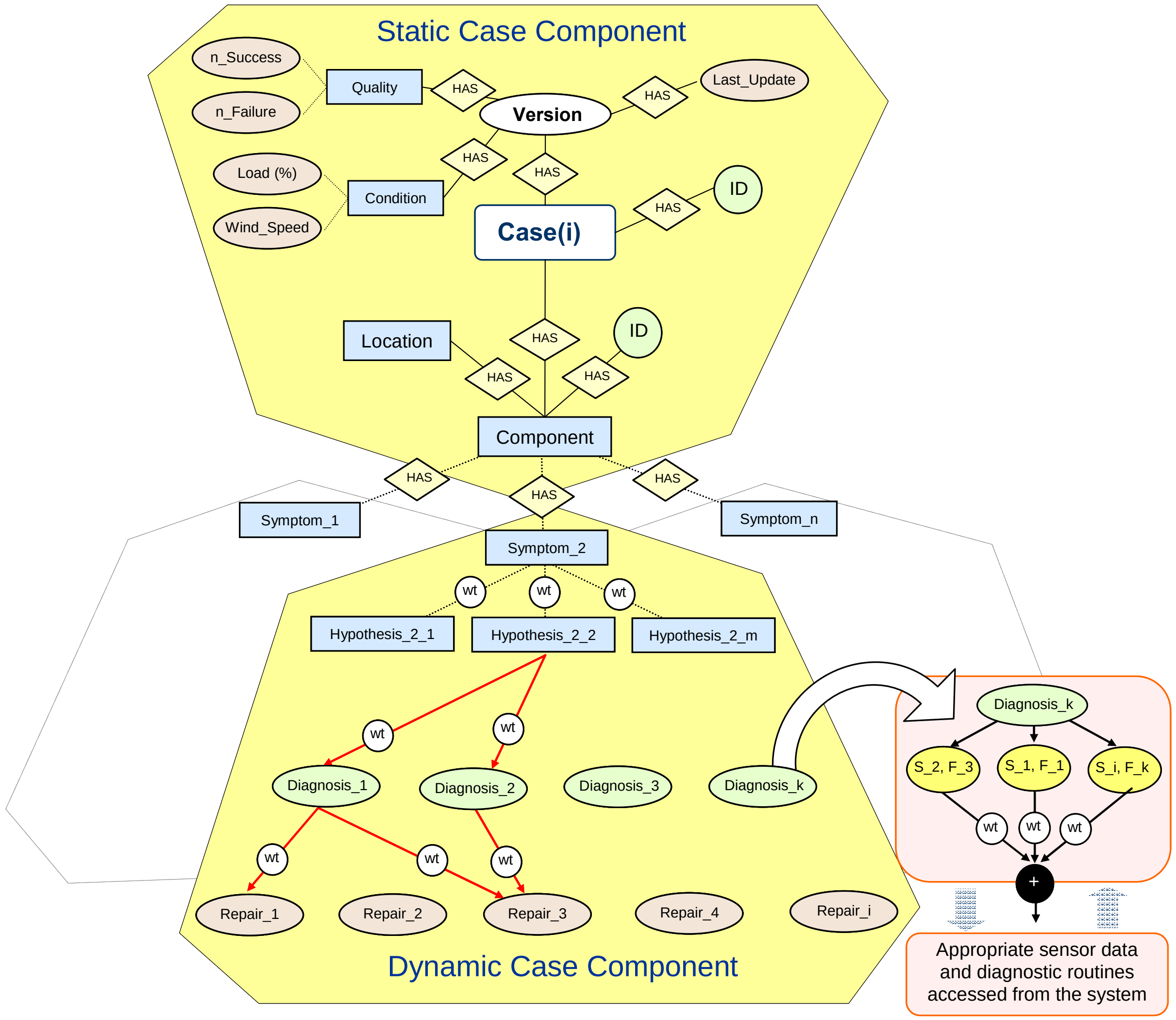


Figure 4.7 Components of a dynamic case expressed as relations and weighted connections.

.ID	B10				
.Component	Bearing				
.Location	Main_Transmission				
.Symptom	**S_ID**	**Symptom**	**Wt**	**Sementic_net**	**Hypotheses**
	1	Noisy in neutral with engine running	0.57	SemNet1	h_1, h_2, h_3
	2	Vibration	0.43	SemNet2	h_1, h_2, h_4
.Hypothesis	**H_ID**	**Hypothesis**	**Wt**	**Diagnosis**	
	1	Primary Gear worn	0.65	d_1	
	2	Primary Bearing worn	0.15	d_2, d_4	
	3	Clutch Release Bearing worn	0.10	d_3, d_4	
	4	Lack of oil	0.10	d_5	
.Diagnosis	**D_ID**	**(Sensor,Feature) pairs**	**Wt**	**Solution**	
	1	d_1:{(S_3,F_1),(S_3,F_4)}	0.75	r_1, r_3	
	2	d_2:{(S_1,F_{15}),(S_1,F_8)}	0.80	r_2, r_3	
	3	d_3:{(S_2,F_{15}),(S_2,F_8)}	0.80	r_3	
	4	d_3:{(S_1,F_{13}),(S_4,F_{13})}	0.6	r_3	
.Repair	**R_ID**	**Repair**	**Wt**		
	1	Change Primary Gear	0.25		
	2	Change Primary Bearing	0.25		
	3	Replenish oil	0.25		
	4	Change Clutch Release Bearing	0.25		
.Version	**Last_Update**	**Case_Quality**	**Succ**	**Fail**	**Conditions**
	03:16:05	0.8	8	2	Full Load Windy

Figure 4.8 Example case from automobiles domain.

Figure 4.8 shows an example case from the dataset being used in this study. Based on the query-semantic-network, relevant cases matching these primary indexes are retrieved and a set of possible hypotheses is generated. Since each symptom may be associated with multiple hypotheses, a list of hypotheses is generated at the initial diagnosis step. Two approaches can be taken at this point to prioritize hypothesis testing. Diagnosis can either be performed based on a set-covering algorithm to find a solution that explains most of the hypotheses [44] or different hypotheses can be tested one by one in order of decreasing success rates. The important point to note here is that, although a generic case structure is used, most of the actual contents are dynamically loaded depending on the situation at hand.

4.5.2 The Other Two Dynamic Components

The other two dynamic components suggested in the DCBR scheme are:

- *Dynamic Case Base*: Dynamic case base has been conceptualized as an intelligent knowledgebase with five attributes of intelligence. A detailed discussion on this knowledgebase is presented in Chapter 6. This knowledgebase keeps evolving with time and hence whenever it is referenced as a case library, results also change with time making the whole reasoning process dynamic.

- *Dynamic Adaptation*: Similarity metrics, similarity ranking methods and adaptation schemes all can be chosen from the knowledgebase dynamically. To limit the scope of this thesis, this issue has not been addressed in detail but some ideas for future work are proposed in Chapter 7. However, a suitable similarity metric has been developed for the semantic net matching for the textual data and a detailed discussion is included in Chapter 5.

4.6 Evaluating DCBR Performance

In the DCBR framework presented in this thesis, the matching activity mainly takes place at the triad level. Once the matching triads are retrieved, only those hypotheses are fetched that are connected to these triads through significant weights. Therefore, the issue of performance, in retrieving relevant cases, arises from the triad level. We covered this issue in the ILP performance evaluation section in the next chapter.

The DCBR framework searches for solutions in a reduced search space. Unlike conventional CBR, where search is carried out by matching several case attributes, DCBR performs a contextual matching by matching the symptoms (triads) directly. Therefore, the DCBR performance can be measured by keeping two factors in mind; *total solution search time* and *the quality of the solutions generated*.

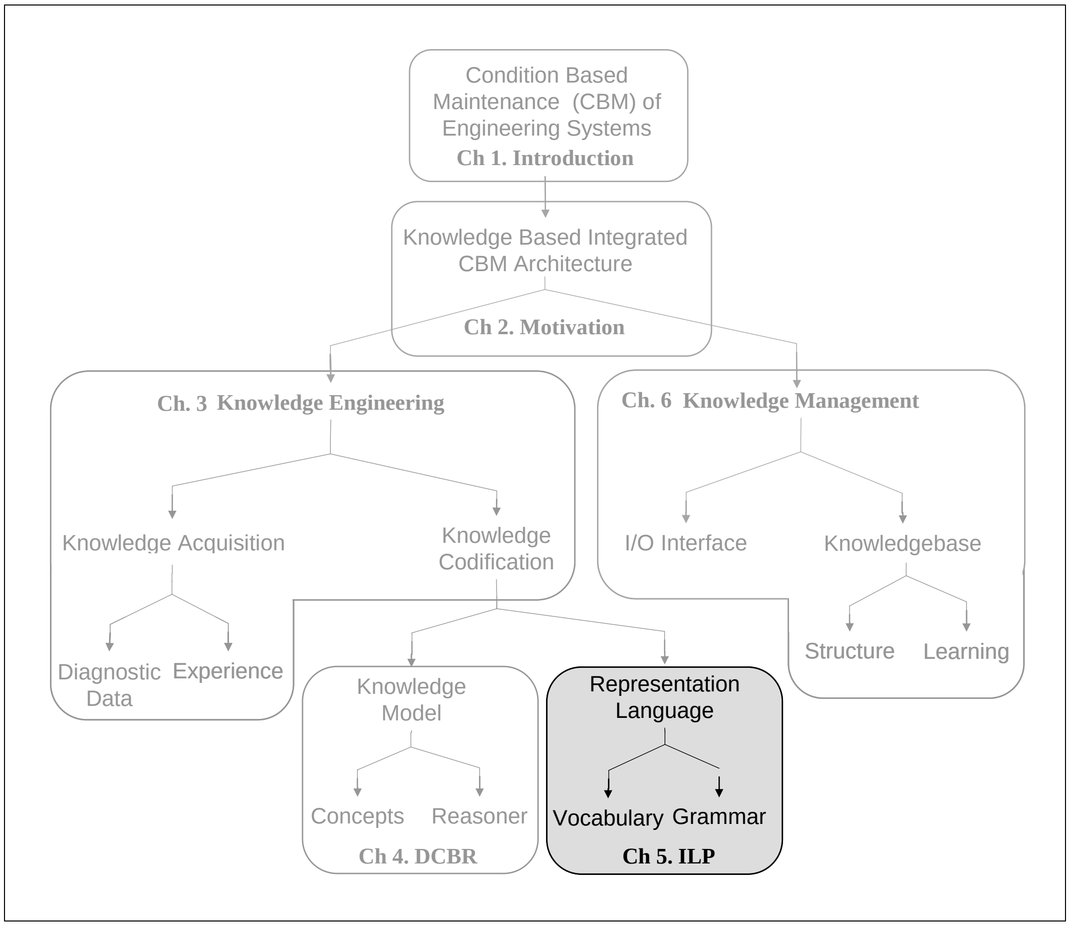
If a conventional CBR system is available for comparison, the time needed in searching for a “useful” solution can be compared with the corresponding time taken by a DCBR system. In the first case, the total time will comprise of time taken in matching the attributes against all cases present in the case base, retrieving the relevant cases and ranking them to suggest a course of action. For DCBR, the total time comprises of time taken to match and retrieve relevant triads and prepare a list of connected hypotheses. Further, the effectiveness of the solution can be measured by the success rate of the suggested solutions. It should consider two factors; one, the success rate of the solution suggested as the top candidate and two, the overall success if a successful solution is found at all after trying various options one by one in the order of preference.

4.7 Conclusions

In this chapter we have developed the concept of dynamic case-based reasoning. Pointing out shortcomings of the conventional CBR systems, three possible improvements have been suggested to build a DCBR based process model. With the help of an example from the automotive maintenance domain some of these concepts have been instantiated for better understanding. Lastly, few notions for the performance evaluation of the DCBR system have been suggested.

Chapter 5

Industrial Language Processing



5.1 Chapter Overview

This chapter discusses one of the major contributions of this thesis, namely the Industrial Language Processing (ILP) technique. First, other relevant techniques have been discussed with their limitations to motivate the development of ILP. Then, the ILP technique has been explained in detail followed by the corresponding similarity calculation methods for matching and retrieval purposes. The chapter concludes with some results on the effectiveness of this technique on the automotive data and the centrifuge data followed by an analytical discussion on the results.

Contribution: Industrial Language Processing technique to integrate textual information with numerical information in the DCBR framework and maintenance knowledgebase.

5.2 Motivation

A large amount of machine condition information is obtained in terms of operator observations expressed as textual descriptions, which are rarely used in automation of the health maintenance process. These symptoms carry important information about the system, which may not always be evident from sensory measurements. An important aspect of the proposed architecture is to take advantage of the information embedded in descriptive data while making diagnostic decisions. In most cases operator observations of fault symptoms are the first indicators of the system malfunction. These observations are conveyed or documented as fault descriptions and observed symptoms. To automate the decision making based on these symptoms, the maintenance system should be capable of processing these descriptions in a coherent and unambiguous fashion. At the same time it is desirable to keep the computation time to a minimum without losing much information that is embedded in these descriptions. This requires a text processing methodology that is capable of carrying out these tasks.

This Chapter presents an Industrial Language Processing methodology that can be adapted for any industrial environment if the following two assumptions are satisfied:

1) Use of pre-defined standardized language to express these observations.

2) Availability of expert evaluation during the initial phase of rule generation.

Keeping these requirements in mind, this methodology can be adapted to meet a desired level of accuracy and computational overhead. The processed data is then stored in the maintenance knowledgebase for later reuse.

5.3 Current State-of-the-Art

Several Information Retrieval (IR) methods have been used for textual data, which primarily depend on the statistical account of occurrence of words and do not consider the semantic relationships between them. This results in several problems like meaning ambiguity and paraphrase expressions (different expressions for the same meaning). An alternative approach of *n*-gram matching [45] has been used to retrieve relevant documents, but this approach also does not permit any integration of additional knowledge like domain specific thesauri or glossaries. The use of Textual CBR (TCBR) has been proposed in [46] to explicitly allow the integration of semantic knowledge using some Natural Language Processing (NLP) and to establish an indexing vocabulary.

Careful analysis of the domain is carried out to devise similarity measures that extend beyond statistical term weighting. NLP techniques like parts-of-speech tagging are used to tag the words in the texts and extract the basic linguistic structures. TCBR is typically built for specific domains to address the ambiguity problem. In another attempt, a feature vector is used to index text documents and two approaches have been proposed to reduce its size; feature selection with boosting and feature generalization with association rules [41]. Feature selection helps in identifying discriminatory features while the feature generalization captures semantic relationships. But this method still does not express a semantic relationship explicitly. Use of graphs-based methods for TCBR has been described in [47]. Graphs offer several advantages over conventional feature vector based methods. They can create rich representations of descriptions. The structure and word order can be retained to capture relationships between two elements and any number of elements can be added or deleted at will. So far, the TCBR has been mainly considered as a tool for an independent domain of books, web documents, reports, documentations and manuals, etc. Elaborate methods have been developed which try to accommodate the complex structure and enormity of the language vocabulary and grammar [48].

In this research we identified that with the increased usage of standardized languages in industrial domains, one does not need such extensive text processing methods and a simpler technique customized to the maintenance process would be more appropriate instead. This led to the development of the ILP technique, the details of which are included in the following sections.

5.4 The Concept of Standardized Language

Unlike traditional practices, the use of standardized or controlled language is being promoted in industrial environments for improved efficiency, accuracy, and data interoperability [41, 49]. The objective of a standardized language is to improve the consistency, readability, translatability, and retrievability of information [50]. Considering the industry initiative to standardize communication protocols and use of a simplified language, an explicit advantage can be derived that can help reduce the complexity in processing the textual data. A structured syntax and a fixed domain vocabulary reduce the task of NLP significantly and offer several advantages over using non-standardized language in industrial environments. This not only helps reduce communication errors by avoiding ambiguities but also simplifies electronic textual data management and technology transfer between manufacturers, users, and maintainers.

5.4.1 Standardized Languages in Industrial Domain

Using a domain-limited vocabulary and a well-defined documentation format makes technical language globally interpretable and reduces multicultural linguistic barriers. For instance, formal communication within the aviation maintenance domain is defined and regulated [14]. A hierarchy of written correspondence is defined in the Federal Aviation Regulations (FARs), which includes airworthiness directives (ADs), notices to airmen (NOTAMs), maintenance manuals, work cards, and other types of information, that are routinely passed among manufacturers, regulators, and maintenance organizations. The international aviation maintenance community has adopted a restricted and highly structured subset of the English language, namely ATA-100 and AECMA Simplified

English, to improve communication [49]. For instance, AECMA Simplified English limits the length of instructional sentences to no more than 20 words. It forbids the omission of articles in noun phrases, and requires that sequential steps be expressed in separate sentences.

Similar to the aviation industry, the importance of standardized technical documentation is gaining importance in the manufacturing and automotive industries. Efforts are being made to enhance the ability to update support documents during the life cycle of a system as it is maintained, modified, or resold, to form a valuable archive of knowledge concerning safe and reliable operation of the system. By now, hundreds of companies have turned to standardized languages as a means of improving readability or facilitating translation to other languages. Originally the Caterpillar Tractor Company (USA) created the Caterpillar Fundamental English (CFE) in the 1960s. Perhaps the best known recent controlled language is AECMA Simplified English [51], which is unique in that it has been adopted by the entire aerospace industry. The standard was developed to facilitate the use of maintenance manuals by non-native speakers of English. Aerospace manufacturers are required to write aircraft maintenance documentation in Simplified English. Some other well-known standardized languages are Smart's Plain English Program (PEP), White's International Language for Serving and Maintenance (ILSAM), Perkins Approved Clear English (PACE), and COGRAM. (See [52], which refers to some of these systems). Many standardized languages are considered proprietary by the companies that developed them. However, these languages are now converging to a common set through various standards. For example, ASD-STE100 (ASD Simplified

Technical English) has been prepared in accordance with Aerospace and Defense Industries Association of Europe (ASD) [53].

5.4.2 Description of Standardized Language for CBM

As mentioned in the preceding section, several industrial standardized languages have been developed. Different industries have adopted different standards based on their specific requirements of task and usage domain. For instance, standards required for maintenance in the aviation industry may not be the same as standards required for process automation in an automobile assembly plant. Therefore, these standards define translation rules at an abstract level and the exact instantiation of these rules are conveniently left to the specific usage. Keeping these rules in mind, a specific format of symptom descriptions was adopted for this research. The same format was applied to both automobile and centrifuge cases to obtain reasonable results as described next.

The Standard Language Template: After studying a large number of fault descriptions, four main constituent phrases were identified in a typical symptom. These phrases are connected by *propositions* or connecting words. For the purpose of standardization these phrases were ordered in a specific manner as shown in Table 5.1.

For a fault symptom to be descriptive enough it should contain at least the *component* and the *condition* phrases. The other two phrases are optional because they either add more detail or help in localizing the fault if there is ambiguity between several similar components in different locations. For this research we manually translated the original symptoms into this standard template. In actual industrial practice, owing to the large size of datasets, automatic translators are used that parse natural language sentences and re-order them in the template format.

Table 5.1 The standard language template for CBM of automotive and centrifuge cases.

Constituent Phrase	**Component**		**Location**		**Condition**		**Operating Mode**
Required	Y		N		Y		N
Example 1	engine			does	no start	when	gear is neutral
Example 2	lubricant	in	transmission	is	leaking		

Since this translation was carried out manually, parsing was still required in the next step of Information Retrieval; otherwise the terms in these descriptions can be tagged at the same time to avoid parsing at a later stage. Some of the results of these translations are shown in Table 5.2.

It can be noticed that in many cases a standardized language sentence may not be grammatically correct but it is able to convey the same meaning, as in the original sentence, without any ambiguity.

Table 5.2 Examples to show standardized language translation of symptoms from automotive and centrifuge cases.

Original Symptom	Symptom in Standardized Language
Examples from automotive case: component - transmission	
Transmission will not downshift (kickdown) with accelerator fully depressed	Transmission | | does | no downshift | when | accelerator fully depressed
Transmission noisy in neutral with engine running	Transmission | | is | noisy in neutral | when | engine is running
Transmission does not drive in forward or reverse gear	Transmission | | has | no drive | if | gear is forward Transmission | | has | no drive | if | gear is reverse
Transmission lubricant leaked	Lubricant | in | transmission | is | leaking | |
Transmission slipped	Transmission | | slips | |
Problems in gear selection	Transmission | | has | problems in gear selection | |
Examples from centrifuge case	
Oil leaking from shaft seal ring behind main bearing on drive end	Shaft seal ring | behind | main bearing on drive end | has | oil leak | |
Movement in small flat belt pulley on rotating assembly	Small flat belt pulley | on | rotating assembly | moves | |
Liquid side bowl has high pitch whine	Bowl | on | liquid side | has | high pitch whine | |
Loud noise from main motor drive	Drive | on | main motor | has | loud noise | |

5.5 Meaning Ambiguity Reduction

As mentioned earlier, natural language processing suffers from two inherent problems, namely the meaning ambiguity and paraphrase expressions (different expressions for the same meaning). Use of a domain-limited vocabulary reduces these problems to a large extent. Use of textual case-based reasoning (TCBR) has been suggested to address these problems since it is typically built for specific domains [46]. Explicit use of a thesaurus can replace all synonyms to a preferred word before matching two expressions (Figure 5.1). This reduces the problems due to paraphrase expressions.

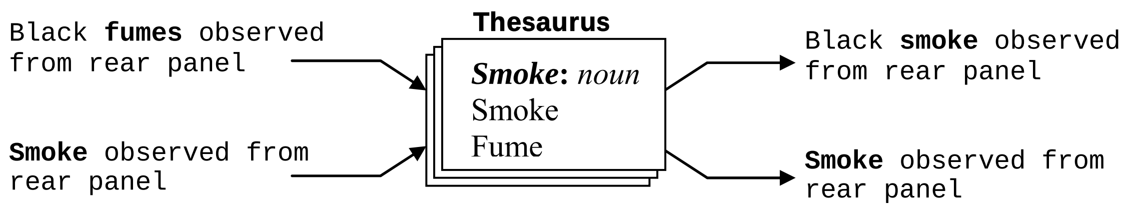

Black **fumes** observed from rear panel
Smoke observed from rear panel
Thesaurus
Smoke: *noun*
Smoke
Fume
Black **smoke** observed from rear panel
Smoke observed from rear panel

Figure 5.1 An illustration of resolving the paraphrase problem.

Further to resolve the ambiguity issue, a reduced domain vocabulary can be established to fix the meaning of ambiguous words. For example, in mechanical domain, *bearing* is a component, i.e., a noun and not the present participle of the verb *to bear*. Therefore, while words like *bearing* in the industrial text are not expected to be used for their other meanings, they will be interpreted only for their fixed domain specific meanings even if they are used otherwise.

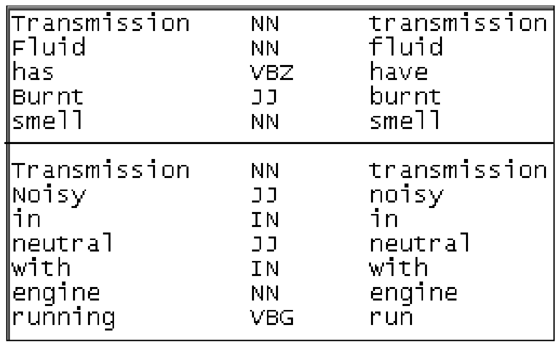
5.6 Information Extraction (IE)

To process a natural language sentence, the tokens of the language must be first isolated and identified. For NLP, lexical processing operates at the single word level and involves identifying words and determining their grammatical classes or parts-of-speech before a higher level language analysis can take place [54]. A shallow-NLP technique, which tags each word with its probable class such as a noun, verb, etc., and identifies the corresponding word stem, has been suggested in [46]. This method is both efficient and robust as compared to other complex NLP techniques. Our IE technique follows a similar approach and is presented next.

5.6.1 Parsing Fault Symptoms

A PC based demo version of TreeTagger tool [55], developed at the Institute of Natural Language Processing (IMS) at Stuttgart University, has been used. As compared to other commercial programs, this version had some limitations that were, nevertheless, not considered critical for the goal of this research. This version did not allow modifications in the associated dictionary that could have been useful in reshaping this dictionary to suit industrial domain and reduce ambiguity. For instance, words like *gear* and *bearing* can be annotated with ‘noun’ referring to mechanical components only, thereby removing the annotation ‘verb’ altogether. Furthermore, we could not add new words to include domain specific technical vocabulary. The complete version also provides the probabilities associated with a word, if there are more than one possibility for the corresponding tag. This capability was also not available in the demo version, which could have been helpful in ambiguity reduction. However, an explicit effort to achieve these tasks was out of the scope of this thesis.

The output of the TreeTagger program is three columns, the first one containing the original word as it appears in the sentence, the second one contains the tag abbreviation (e.g., NNP for proper noun, VB for verb, etc.), and the last column contains the stem of the word (e.g., 'run' for 'running' and 'be' for 'have'). Figure 5.2 shows a snapshot of the output file from TreeTagger.



```
Transmission     NN     transmission
Fluid            NN     fluid
has              VBZ    have
Burnt            JJ     burnt
smell            NN     smell

Transmission     NN     transmission
Noisy            JJ     noisy
in               IN     in
neutral          JJ     neutral
with             IN     with
engine           NN     engine
running          VBG    run
```

Figure 5.2 TreeTagger output.

5.6.2 Data Sanitization

After all words are tagged, a set of syntactic rules are employed to extract relationships between different words in the sentence and discard redundant words. In all, three types of rules were employed.

Rule Set I - The first set of rules is more of a preprocessing step that combines multiple words, if they together define one object or a single situation. For instance, two consecutive nouns are combined because they, most likely, describe a single object.

E.g., Transmission fluid has burnt smell → Transmission_fluid has burnt smell

Here, the words *transmission* and *fluid* are combined to represent transmission fluid. This helps in distinguishing between different situations involving *transmission*, *transmission fluid* and *break fluid* which are different contexts involving the same words. However, this method does not entirely disregard the partial similarity between such situations. This partial similarity does provide some information about the location of the fault and may be useful in providing an alternative hypothesis if the best matching hypothesis has been confirmed negative. Negations like *no* or *not* are combined with the corresponding verbs.

E.g., Engine does not start → Engine does not_start

This takes care of negation, which is extremely important for a CBR system [56]. For instance, here the word *start* would not perfectly match to *not_start* appearing in a different situation.

Rule Set II - The second set of rules links different words based on their parts-of-speech category and word order.

E.g., Transmission will not downshift with accelerator fully depressed.

In this sentence, at the first level downshift should be associated with transmission and depressed with accelerator. And these two pairs should be further linked with each other at the next level in the manner (transmission will not downshift) – when – (accelerator is fully depressed).

Rule Set III - The third set of rules involves sentences containing conjunctions (and, or, /). It creates multiple associations of an object to accommodate all descriptions connected via these conjunctions.

E.g., Horn inoperative or unsatisfactory in operation → (Horn is inoperative), (Horn is unsatisfactory in operation)

Here, the same object horn is associated to two different conditions and a match to any of these two is likely to involve similar or related diagnosis. Although in some standardized languages use of conjunctions is not allowed, this rule was still added to suit the available data set.

5.7 Data Representation

After the information extraction process, it is important to represent this information in a suitable format without losing the meaning of the text, which allows easy and fast similarity calculation. Semantic networks were selected for textual data representation in this research. This is a graph-based technique that organizes all the words in a sentence in a tree like structure which has some special properties to retain word order or the meaning of the text. The syntactic rules, described above, divide a sentence into smaller segments that independently define a relevant concept. It was found that three basic relations could be defined that explained most of the relationships between the kinds of concepts commonly occurring in this type of data. These relations are shown in Table 5.3.

Table 5.3 Three conceptual *relations* capture most scenarios in industrial descriptions of failures.

A--IN--B → A `in_condition` B *or* A `when` B
Here A is typically a noun or a *triad* and B is mostly a *triad*
e.g. `Transmission fluid has burnt smell` translates into
(`Transmission_fluid`) **--IN--** (`smell` --IS-- `burn`)

A--IS--B → A `has_property` B *or* A `is_type` B
Here A is typically a noun or a *triad* and B is an adjective
e.g. `Transmission fluid has burnt smell` translates into
(`Transmission_fluid`) --IN-- (`smell` **--IS--** `burn`)

A--AT--B → A `in_state` B *or* A `exhibits_state` B
Here A is typically a noun or a *triad* and B is a verb
e.g. `Transmission slips` translates into
(`Transmission`) **--AT--** (`slip`)

These concepts can be represented by a data structure called a *triad*. A triad τ is a three-tuple consisting of two phrases p_1, p_2 and a relationship r between them (Figure 5.3).

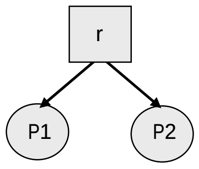


Figure 5.3 A triad consists of two phrases (p_1 and p_2) and a relation (r).

A phrase can be a noun, adjective, verb or a triad itself. The set of relations, R, is a finite set of three relations as described above. These triads can be combined to create the corresponding semantic networks (Figure 5.4).

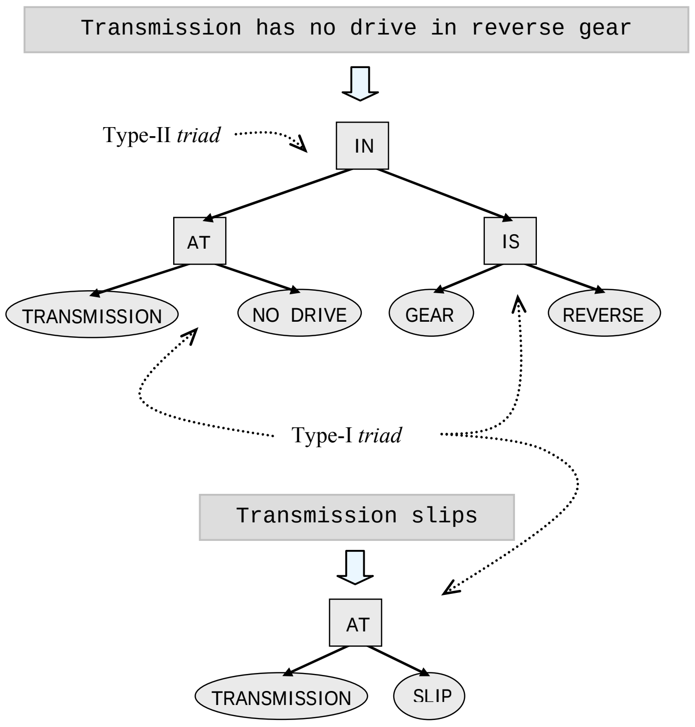


Figure 5.4 A semantic network (type-II triad) consists of one or more type-I triads.

Two types of triads were defined. Both phrases in a Type-I triads are single words and do not involve triads. Type-II triads on the other hand may consist of one or more Type-I triads. This makes the semantic networks a binary tree with words occurring only at the leaves and the rest of the nodes as relations. To improve uniformity across different usage of the same words, only word stems are used in forming the triads.

5.8 Similarity Calculations

Similarity calculation plays a key role in defining the quality of retrieval of matching cases and hence the quality of solution suggested by the expert system. In the following section some general notions of similarity applicable in the CBR domain have been presented followed by the similarity measure defined for the triad structure.

Traditionally, similarity metrics have been described based on the objective notions of similarity and distance. Definition of a similarity metric depends on the structure of cases and in many cases multiple metrics may be defined to cover different aspects. A list of commonly used similarity metrics has been included in Appendix C. This list can be used as a general template to generate customized metrics. Once appropriate similarity metrics have been chosen, they must be composed to form a composite similarity measure. A composite similarity measure can be defined as:

$$\begin{aligned} sim(a_1,a_2) &= sim([a_{11},a_{12},...,a_{1n}],[a_{21},a_{22},...,a_{2n}]) \\ &= \phi\, sim(sim_1(a_{11},a_{22})),\ldots,sim_n(a_{1n},a_{2n})) \end{aligned} \tag{5.1}$$

where, the function $\phi : \Re^n \rightarrow \Re$ combines feature similarity functions $sim_i : W_i \times W_i \rightarrow \Re$ as shown above.

The most common composition function is the weighted average of feature similarities. This average can be algebraic or geometric as shown in Table 5.4.

Table 5.4 Most commonly used similarity composition functions.

Expression	**Similarity Function**
$sim = \frac{\sum_{i=1}^{n} g_i \cdot sim_i}{n}$	Algebraic weighted composite similarity
$sim = \sqrt{\frac{\sum_{i=1}^{n} g_i \cdot sim_i^2}{n}}$	Geometric weighted composite similarity

5.8.1 Similarity Concept for Triad Structure

For the purpose of retrieving the matching cases a similarity metric needs to be established between the *query* case (or *target* case) and the source cases in the case-base. There are three matching strategies: nearest neighbor, inductive or knowledge guided, and a combination of them [57]. In the nearest neighbor approach, a case is selected based on the degree of match for every feature of the input case with the features of the retrieved case. This can be tedious for large case bases. On the other hand, only a few key features are assessed in the inductive approach. Therefore, a hybrid of the nearest neighbor and inductive approaches is both precise and fast [6]. This research uses semantic networks to retrieve the cases from the case-base. Similarity assessment is done based on matching the triads constituting these semantic networks. First the lowest level of triads is matched.

Example: the input semantic net: (Transmission—IS—Noisy)—IN—(Gear—IS—Neutral) consists of three triads:

τ_1: (Transmission—IS—Noisy)

τ_2: (Gear—IS—Neutral)

τ_3: (τ_1—IN— τ_2)

All semantic nets that contain similar type-I triads (e.g., τ_1 or τ_2 in this case) or the same terminal words will be retrieved first. Similarity of triads is computed based on how closely the three constituents match.

With the semantic net (Transmission—AT—No_Drive)—IN—(Gear—IS—Neutral) the above query semantic net will be matched in a manner as shown in Table 5.5.

Table 5.5 Similarity calculation for triads-based semantic networks.

τ	INPUT Query	RETRIEVED Case	SimVal
τ_1	(Transmission—IS—Noisy)	(Transmission—AT—No_Drive)	(1*0.5 + 0*0.2 + 0*0.3) = 0.5
τ_2	(Gear—IS—Neutral)	(Gear—IS—Neutral)	(1*0.5 + 1*0.2 + 1*0.3) = 1
τ_3	(Transmission—IS—Noisy)—IN—(Gear—IS—Neutral)	(Transmission—AT—No_Drive)—IN—(Gear—IS—Neutral)	(0.5*0.4 + 1*0.2 + 1*0.4) = 0.8

For triads with links AT or IS the weights associated with p_1, p_2 and r are 0.5, 0.3, 0.2 respectively acknowledging the fact that component (p_1) matching is more important than its exact condition as far as localizing the fault is concerned. Fault identification may require more accurate condition matching but here the emphasis is on fault localization and identification, is left to further diagnosis using dedicated diagnostic algorithms. For triads containing links IN the weights associated with p_1, p_2 and r are 0.4, 0.4, and 0.2 respectively as both p_1 and p_2 convey equally important information here. These weights can be assigned through expert experience or learned from the data. Several approaches have been implemented for feature-weighting algorithms that can be used to learn these weights [4, 5].

Even though similarity calculation is a subgraph isomorphism problem in graph theory, that is NP-complete [58], a smaller size of semantic networks from standardized language makes this problem insignificant to a large extent. The effectiveness of this approach has been shown in the next section with the help of a dataset acquired from the automotive troubleshooting domain.

5.9 Case Studies

5.9.1 Automotive Maintenance

For the purpose of evaluation of this technique, a simple data set was acquired from an automotive troubleshooting database, which lists several symptoms and their possible diagnoses and repairs [59]. Very short descriptions have been listed using common vehicle terminology as a car mechanic would use. As mentioned earlier, these descriptions were manually translated to standardized language. For example, all sentences were converted to active voice. Use of conjunctions and determiners was maximally reduced. Very long and complex sentences were broken into smaller ones.

Since a domain-limited customized vocabulary could not be incorporated in TreeTagger, the use of ambiguous words was reduced. Our data size being fairly small these modifications were carried out manually. But it is expected that with the promotion of usage of standardized language this step may not be required and in the absence of use of standardized language customized translators may be used. Some typical descriptions were processed by our programs. The pictorials of these semantic networks are shown in Figure 5.5 and the corresponding semantic networks are included in Figure 5.6.

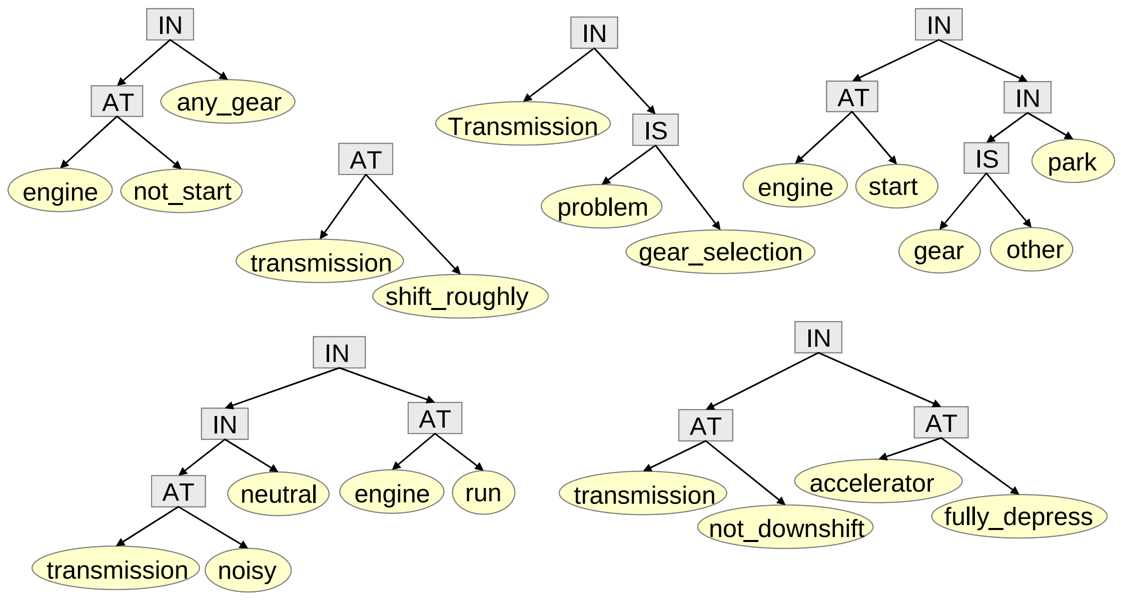


Figure 5.5 Semantic net pictorials of symptoms from automotive data.

5.9.2 Sludge Dewatering Centrifuge

The rule set developed using the automotive data was also applied to the centrifuge data and reasonable results were obtained as shown in Figure 5.7 and Figure 5.8.

engine does not start in any gear

```
sNet(1) = ((engine -<AT>- not_start) -<IN>- any_gear)
```

engine started in gear other than Park

```
sNet(2)  = (((engine -<AT>- start) -<IN>- ((gear -<IS>- other) -<IN>- park)
```

transmission shifted roughly

```
sNet(3)  = (transmission -<AT>- shift_roughly)
```

transmission has problems in gear selection

```
sNet(4)  = (transmission -<IN>-(problem -<IN>- gear_selection))
```

transmission does not downshift when accelerator is fully depressed

```
sNet(5)  = ((transmission -<AT>- not_downshift) -<IN>- (accelerator -<AT>- fully_depress))
```

transmission is noisy in neutral with engine running

```
sNet(6) = (((transmission -<IS>- noisy) -<IN>- neutral) -<IN>- (engine -<AT>- run))
```

Figure 5.6 Semantic networks for six different descriptions using the three relations described above.

shaft sealing ring behind main bearing on drive end has oil leak

```
sNet(1) = ((seal_ring -<IN>- (bearing -<IS>- main)) -<IN>- drive_end)
-<IN>- (oil -<AT>- leak)
```

small flat belt pulley on rotating assembly has movement

```
sNet(2) = ((flat_belt_pulley -<IN>- movement)-<IN>-(assembly -<AT>-
rotate))
```

decanter has oil leak

```
sNet(3) = (decanter -<IN>- (oil -<AT>- leak))
```

bowl on liquidside has high pitch whine

```
sNet(4) = (liquidside_bowl -<IN>- (whine_pitch -<IS>- high))
```

main motor drive has loud noise

```
sNet(5) = ((motor_drive -<IS>- main) -<IN>- (noise -<IS>- loud))
```

Figure 5.7 Semantic networks for some descriptions from centrifuge data.

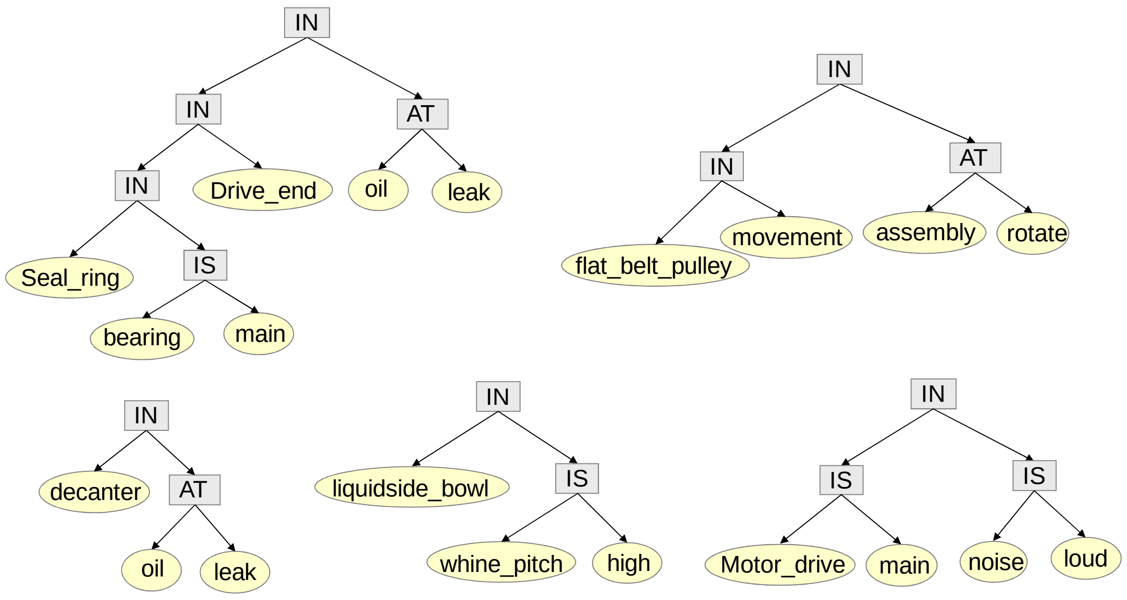


Figure 5.8 Semantic net pictorials of symptoms from centrifuge data.

5.9.3 Discussion

As shown above, the new ILP technique can process short text descriptions quite effectively as long as these descriptions are in a standardized language. For this research we used a set of 15-20 rules for information extraction. This resulted in a reasonable performance in the sense that most of the symptoms were processed satisfactorily when evaluated manually. It was realized that combining the *noun* words may not always be the best idea as they result in new *nouns* and a perfect matching will not be achieved when the original *nouns* are encountered. This issue can be resolved by introducing another relation to combine such words into a triad. However, to limit the scope of ILP development this was not carried out. Moreover, in more than 90% of the cases the current form of ILP satisfied most requirements.

5.10 Evaluating ILP Performance

Usually, the main issue related to any text processing methodology is the trade-off between its computational time and fidelity of information extraction. A high fidelity information extraction (IE) algorithm increases the computational burden. Therefore, the level of detail must be decided in advance, which in turn determines the number of IE rules that are applied. More rules can handle a wider variety of sentence structures but at the same time increase the processing time whenever a symptom arrives. In our case, however, we take advantage of two unique characteristics that occur in the problem domain and make these issues less critical. One, the symptoms are expressed in standardized language that makes them short and fairly structured. Thus exponential complexity arising due to graph structures does not worsen things to a great extent when compared to linear search techniques. Second, most of the processing, in our application, does not require real-time solution generation and it is acceptable to wait a little longer as long as a correct solution is generated. Therefore, a greater emphasis is on correctness (relevance) of the solution rather than speeds provided they do not take longer than acceptable limits. To evaluate correctness of the generated solutions, the two most widely used performance criteria are *precision* and *recall*. Traditionally, these metrics are defined as follows:

$$P = \frac{\#\, relevant\;\; candidates\;\; retrieved}{\#\;\; total\;\; candidates\;\; retrieved} \tag{5.2}$$

$$R = \frac{\#\, relevant\;\; candidates\;\; retrieved}{\#\;\; relevant\;\; candidates\;\; in\; memory} \tag{5.3}$$

The *recall* metric assumes prior knowledge about the number of relevant candidates in the memory. This needs a subjective opinion of experts to determine and is often not available. Furthermore, in the CBM domain the *recall* metric must be slightly modified owing to different requirements. In this scenario, it is not required to retrieve several relevant candidates but just one suffices if it leads to a correct solution. Thus, in the modified definition, *recall* is equal to 1 if at least one relevant candidate is retrieved for a symptom and equal to 0 otherwise. And then to assess the performance over a set of symptoms the corresponding recall values can be averaged from all symptoms.

The number of candidates, that are retrieved, actually depends on the preset threshold for similarity values. This threshold must be set with experience. In the initial phase more options must be retrieved as the system has not yet completely learned the correct weights and is still in the exploration phase. Later this threshold can be stricter to retrieve only the best candidates and improve the *precision*. In a typical precision-recall curve, it is observed that improvement in one results in degradation in another. In CBM domain, as argued earlier, a high precision is preferred over high recall. A combined metric such as *F-measure* can be used using weighted averages as shown below:

$$F_\alpha = \frac{(1+\alpha)\times precision \times recall}{(\alpha \times precision) + recall}; \alpha \in \Re^+ \tag{5.4}$$

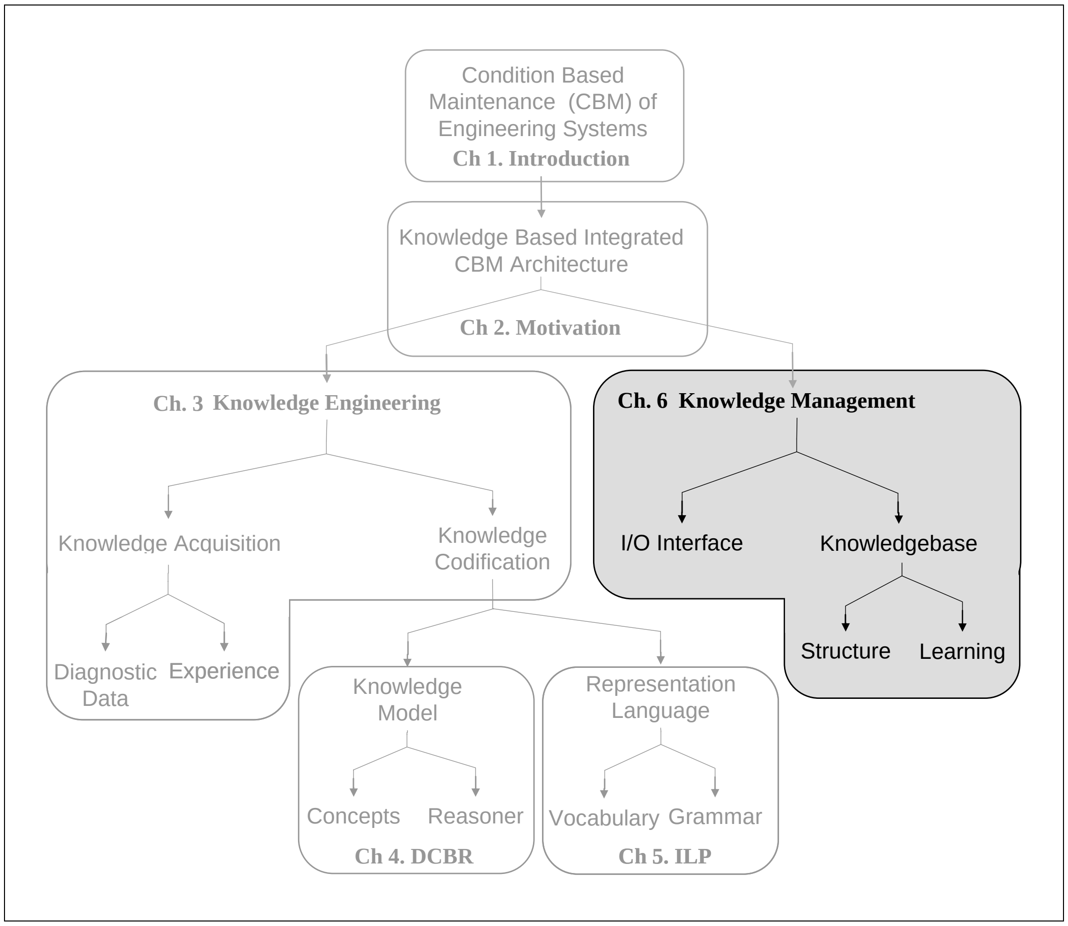
F-measure is harmonic mean of recall and precision metrics. These weights can be biased towards any of these two metrics by modifying the parameter *α*. A commonly used metric for CBM type applications is $F_{0.5}$ measure that weighs precision twice as much as recall.

5.11 Conclusions

With the help of two examples we have shown that short and semi-structured technical textual descriptions can be abstracted using three simple relations that form the basis for semantic networks. These structures not only provide means of representing textual descriptions in a structured manner but also preserve the semantic meaning of the sentence. These semantic networks are stored in the knowledgebase. For each symptom linked to a hypothesis, its corresponding triads are also linked to that hypothesis. Therefore, once a query symptom is presented, hypotheses linked to its constituent triads are retrieved and help in the initial phase of diagnosis. They form a part of cases in the DCBR system. This helps in retrieving short text based cases to generate an initial hypothesis thereby considerably reducing the search space for further diagnosis.

Chapter 6

Knowledge Management System



6.1 Chapter Overview

In this chapter we discuss the attributes of an intelligent and evolving knowledgebase. Maintaining the core theme of this thesis to integrate knowledge with CBM, it is important to describe a maintenance knowledgebase and how its integration offers to enhance CBM. Although five different attributes have been enumerated, the emphasis has been on the learning capability of the knowledgebase. With the help of some examples it has been shown how the learning algorithm can accommodate the changes in the environment. Apart from the backbone of intelligent knowledgebase, a transparent and easy to use user interface is required for trustworthy decision support systems. Users should be able to visually access the processing carried out by the system. Therefore, a knowledge management system has been developed in the Matlab environment, which allows users to enter data, access data, and observe data manipulation and learning that take place in the knowledgebase over time.

Contribution: A knowledge management system with an evolving knowledgebase as backbone for consolidating CBM knowledge.

6.2 Introduction

The key goal of this research is to develop a knowledgebase for CBM techniques that can accommodate knowledge in different forms (descriptions, data, algorithms, tests, models, etc.). A lot of structural, operational, and analytical knowledge has been developed over the years while these systems were built, tested, and maintained. Typically, components or subsystems of a large system are studied individually before relevant analytical techniques are developed. Even though these subsystems may be different, several constituting components share various structural and operational similarities. There is a need to organize this knowledge in such a manner that it can be easily shared and reused for similar components of a large system. Rather than re-developing this knowledge, minor adaptations should be able to save time and effort of the analysts. Thus, an easily accessible knowledgebase of these techniques must be created. A knowledgebase is a special kind of database for knowledge management. It facilitates computerized collection, organization and retrieval of useful information or knowledge.

To achieve an increased level of autonomy it is equally important for a knowledgebase to show attributes of intelligence. It should be able to adapt according to changing environments and provide suitable solutions. As compared to a database, a knowledgebase also has a capability to improve the search results using the attached semantics to the data it contains. No formal definition of intelligent knowledgebase yet exists in the literature and different researchers have argued in favor of different attributes. In the next section we will discuss five attributes that we think are the most important ones to create an evolving knowledgebase. Some ideas regarding how these can be incorporated into a CBM knowledgebase have also been presented.

6.3 Attributes of Intelligence in a Knowledgebase

Attributes of intelligence have been debated in the AI domain for a long time where concepts like autonomy, intelligent agents, reactivity, temporal continuity and goal directedness take a center stage [60, 61]. With the presence of intelligence, a knowledgebase is expected to suggest solutions to given problems based on user feedback, and is capable of learning from experience. With the attribute of *learning* it has an inherent characteristic of human beings to adapt its behavior to changing environments and improve its performance [62]. In response to a US Navy's solicitation for Small Business Innovative Research (SBIR) on self-evolving maintenance knowledgebase, the authors proposed four attributes of intelligence, namely; adaptation and learning, self-organization, conflict detection and resolution, and fault-tolerance in a system called Case-based Temporal Reasoner (CaTeR) [63]. In another application a CBR database was created to build a self-evolving maintenance and operations reasoner (SEMOR) [64]. The concept of self evolution in itself has been argued as an attribute of intelligence, which possibly includes self-evaluation, learning, self-organization and autonomous behavior [12]. Systems that can monitor their own activities and results to improve their performance are considered self-evolving and generate a dynamic knowledgebase that track certain activities to achieve their design goals [65].

From all these references it is clear that there are certain common characteristics that must be exhibited by a system to show intelligence. For the knowledgebase in this research we have mainly considered five attributes that will fulfill the overall goal of a more informed and automated decision support for CBM of engineering systems. These attributes have been outlined in the following discussion.

6.3.1 Dynamic Structure

The knowledgebase is characterized by semantics attached to the stored data about how various data are related and provide useful information when brought together in different orders and combinations. In this research we connect various data entities with weighted associations as depicted in Figure 6.1. Further, a feedback reward is propagated backwards to reassess these weights.

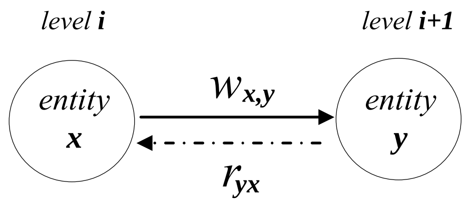


Figure 6.1 A generalized basic unit of the knowledgebase connected with weighted associations and feedback rewards.

At any point of time more entity levels or more entities within a level can be added. The initialization of weights varies depending on the type of association between the entity levels. There are two categories of associations between the various entities.

1) **Dynamic Associations (α)**: the degree of association and appropriateness must be learned in an evolving environment. Weight initialization may be random or uniform across all connections. Weights must be learnt from data through some learning algorithms to emulate experience. Feedback is crucial for this learning. For instance, associations between triads and hypotheses must be learnt from data. A triad can be a part of various symptoms and hence may be connected to multiple hypotheses through different weights. Thus, an association between a triad and a hypothesis can be a consequence of several symptoms pointing to the same hypothesis. In some cases weights may be available from experts' experience and may only be modified or fine-tuned over a period of time. For example, associations between hypotheses and diagnostic tests must be initially

assigned using an expert's experience. These weights may change with time as data suggests.

2) **Static Associations (β)**: degree of association is deterministic and is available in advance. Weights are determined through offline analysis during experimentation and testing phase and once identified; they are assigned to the system. For example, evidences from multiple pairs of sensors and features can be fused through weighted averages. These weights are determined during the initial analysis when the diagnostic algorithms are being developed. There is no dynamic learning involved as such; however, these weights may be manually modified when needed.

In this research we have used various entities like: triads, hypotheses, diagnoses and repairs. They can be pictorially depicted as in Figure 6.2.

Symptoms are received as external input. These symptoms may be weighted (ω) by the user to specify which symptom is more prominent or which symptom looks dangerous. Each symptom is decomposed into its constituent triads. Each triad is connected (α-association) to a set of hypotheses. All these hypotheses are collected and ranked based on the total degree of support from the triads, which is calculated using symptom weights ω and association weights α. For the highest ranked hypothesis a list of diagnostic tests is collected. The chosen diagnostics is performed using a weighted combination of sensor-feature pairs. If a diagnosis is successful, the corresponding repair action is suggested.

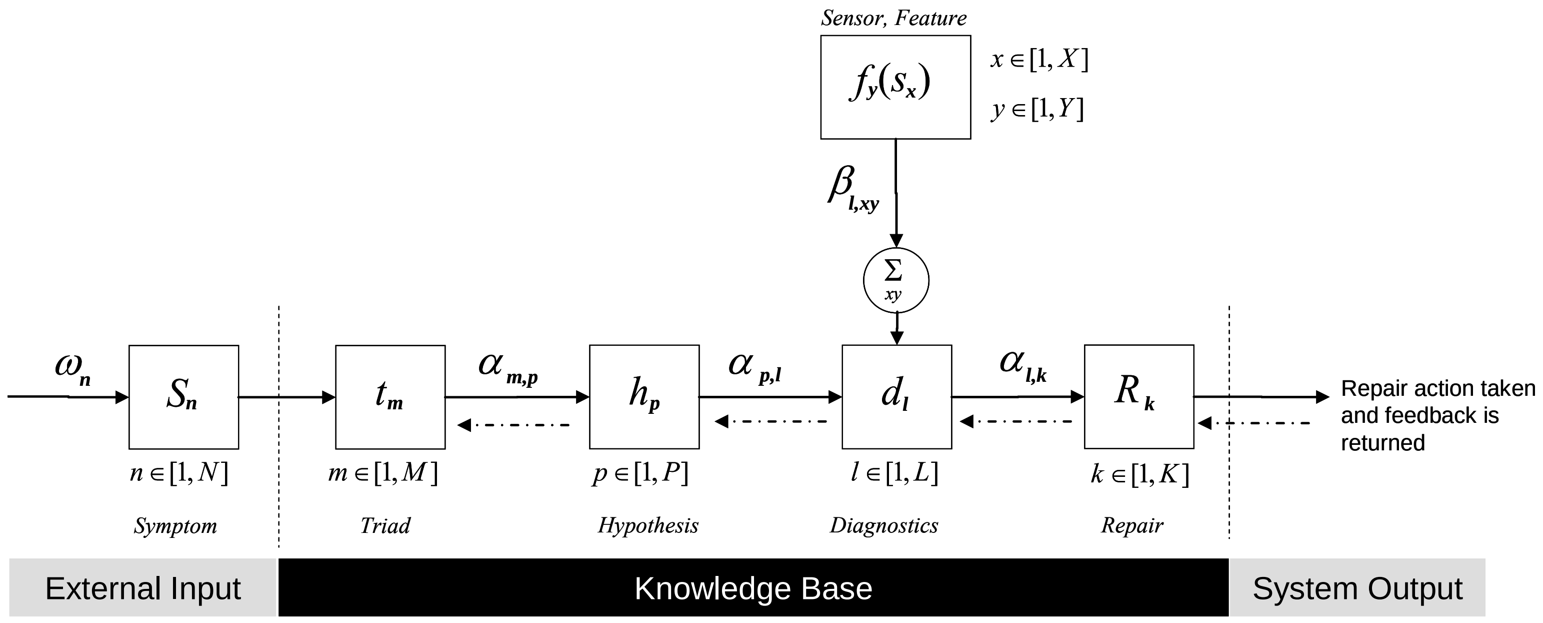


Figure 6.2 Pictorial representation of the dynamic structure of the knowledgebase.

In this manner the knowledgebase is capable of containing both structural and operational data. The knowledge is preserved in the form of associations between these data entities

6.3.2 Self-Evaluation

Self-evaluation is an intermediate task that the knowledgebase performs to support learning and self-maintenance activities on a continuous basis. Based on the feedback from the next level, the knowledgebase should keep track of its successes and failures at each level. This account of performance can be utilized in two ways.

1) **Short Term Evaluation**: At each level, if a decision results in a failure, it is desirable to evaluate if the failure occurred due to a wrong decision at the current level or due to a wrong decision at a previous level. Consequently, the weights must be modified at the corresponding erroneous level, and not always at the current level. Therefore, in case of a failure, the system waits until all options have been exhausted before propagating a penalty to the previous level. If a successful option is found at the current level then a penalty is issued for the unsuccessful options at the current level itself. In this manner a self-evaluative process determines the correct location of wrong decision making.

2) **Long Term Evaluation**: If system realizes at any point of time that a particular entity always results in a negative outcome and hence its weight has dropped below certain level, then that association is a likely candidate to be removed from the knowledgebase. This action will be performed whenever the self-maintenance operation is scheduled next. For instance, a group of sensor-feature pairs may no

longer be effective and the diagnostic result always returns a low-confidence outcome, then either these pairs can be removed from the knowledgebase or a flag be thrown for the experts to re-evaluate the corresponding test.

6.3.3 Learning

Learning is the most critical attribute of intelligence and an evolving knowledgebase. The system should learn from experience about how the environment behaves and should create a model of this environment. The next time a query arrives it should use this model to predict the most probable state of the environment. Further, if the environment changes, system should be able to recognize the changing behavior and adapt to new behavior of the environment. In the literature three broad approaches to learning can be found depending on the task at hand. These approaches are briefly mentioned below.

Supervised Learning: Under supervised learning the agent is provided with a *target* or a purely *instructive* feedback, i.e., the environment tells the learner about what exactly its output should be. The agent then compares its response with the target and adjusts its internal memory in such a way that it produces a more appropriate response the next time it receives the same input. The instructive feedback in this case is independent of the action taken by the agent and always tells about the correct action it should have taken. Thus, supervised learning is learning through several examples provided by an external knowledgeable supervisor.

Unsupervised Learning: This is the other extreme for learning where the agent does not receive any feedback from the environment. The agent instead has to abstract the input information into clusters or categories or by using a reduced set of dimensions so that when a familiar situation is encountered, an output is generated based on that category of situations and is likely to cover slightly different problems as well. Unsupervised learning is based on similarities and differences in the input patterns.

Reinforcement Learning: This lies somewhere in the middle of the supervised and unsupervised learning techniques, however it is closer to supervised learning. In this technique the agent receives an *evaluative* feedback about the appropriateness of its response. Purely evaluative feedback indicates how good the action taken is, but not whether it is the best or the worst action possible. The evaluative feedback completely depends on which action was taken, unlike instructive feedback [66].

6.3.3.1 Learning in the Knowledgebase

The Model: The initial diagnosis component of our knowledge-based CBM architecture can be modeled as a finite Markov Decision Process (MDP) (Figure 6.3) i.e.

- There is a finite set of distinct symptoms indicating the failure in the system. This constitutes the *failure-unknown* state.

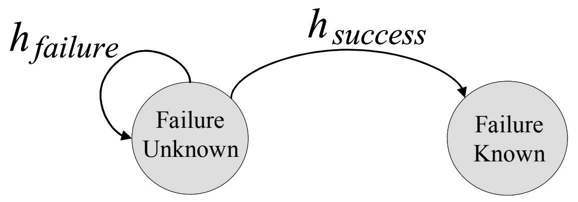


Figure 6.3 State space representation of a diagnostic process.

- There is a finite set of hypotheses that can be proposed to explain these symptoms in the initial phase. These hypotheses are tightly coupled to their respective diagnostic tests or actions that the knowledgebase suggests to perform to confirm a hypothesis in the final diagnosis phase. Therefore, there is a finite set of choices (actions) that can be taken while at failure-unknown state.

- If by choosing a hypothesis and carrying out the corresponding diagnostic tests a fault is clearly identified, the system moves to *failure-known* state. If the test fails to identify the fault, system comes back to failure-unknown state.

The one step dynamics of this finite MDP are completely defined by the set of states and hypotheses together with the probability of the next state, given the current state and a chosen hypothesis. These transition probabilities are given as:

$$P_{ss'}^{h} = pr\{s_{t+1} = s' \mid s_t = s, h_t = h\} \tag{6.1}$$

Similarly, given a state *s* and hypothesis *h* together with the next state s_{t+1}, the expected value of next reward (success or failure) can be expressed as:

$$R_{ss'}^{h} = E\{r_{t+1} \mid s_t = s, h_t = h, s_{t+1} = s'\} \tag{6.2}$$

Here, the goal is to learn these probabilities from the environment (data) such that the expected value of rewards is maximized over a long period of time.

The Learning: A finite MDP presents a natural choice for a reinforcement learning approach. In reinforcement learning, an *action-value* is associated with each action. These action-values are modified every time the corresponding action is taken and a feedback is received in the form of reward from the environment. These action-values can be considered as weights, like in other machine learning algorithms, that can be used to make a decision about the action selection the subsequent time. The action-selection policies can vary depending on the nature of the problem. A brief discussion on action-selection policies follows in the latter paragraphs.

There are three main algorithms for reinforcement learning namely, Monte Carlo (MC) methods, Dynamic Programming (DP) and Temporal-Difference (TD) learning, each having their own strengths and weaknesses. DP methods are well developed mathematically but require a complete and accurate model of the environment. MC methods are conceptually simple and don't require a model but are not suited for step-by-step incremental computation. TD methods do not require a model and are completely incremental, but are more complex to analyze. In particular, a Temporal Difference (TD) method for reinforcement learning is most suitable for these kinds of problems [66].

***Q*-Learning**: The most popular TD algorithm for reinforcement learning is the *Q-learning* algorithm [67]. In this algorithm the learned action-value function, *Q*, directly approximates the optimal value, *Q**, irrespective of the action selection policy. Detailed proofs of optimality and convergence of *Q*-learning have been developed in they literature [66]. In brief, one step *Q*-Learning is defined by

$$Q^{new}(s_t, h_t) \leftarrow (1-\alpha)Q^{old}(s_t, h_t) + \alpha[r(s_t, h_t) + \gamma \max_h Q(s_t, h_t)] \tag{6.3}$$

where for any failure hypothesis h_t assumed in the state s_t
Q : is the learned action - value function
r : is the reward recieved after carrying out diagnostic test correspond ing to hypothesis h_t in state s_t
α : is the learning rate parameter s.t. $0 \le \alpha \le 1$ and,
γ : is the discount parameter

For this research, Q values are used as learnt preference weights to choose from various options. Reward r is obtained from the maintainer in predefined quantized values as discussed in the next section. This forms an important step in integrating human-in-the loop. The discount parameter γ is a measure of how much attention we pay to possible rewards we might get in the future. Since there are only two states in our scenario and there are no actions associated with *failure known* state there is no expected reward from that state. Therefore, the discount parameter has little or no significance in this case. The learning rate parameter α is however very crucial in determining the time this system takes in adapting to changing environments. A large value of α leads to oscillations in the Q-values due to random nature of failure occurrences as it gives a higher weight to current event over the past history. A smaller value, on the other hand, leads to slow learning by expressing more trust in past experience over current events. Therefore, a balance must be established for desirable performance.

Hypothesis-Selection Policy: This is the scheme adopted to make a decision about which hypothesis to select among the various available options. The main goal is to maximize the total rewards accumulated over a long term. The most common approach is the greedy policy $\pi^*(s)$ where the hypothesis with the highest action-value is always selected:

$$\pi^*(s) = \arg\max_{h_t} Q(s_t, h_t) \tag{6.4}$$

This is also called as *exploitation* approach where current knowledge is exploited to make a decision. The other approach is that of *exploration* where one of the non-greedy hypotheses is chosen. This allows exploring other options that may fare better than the current best one. A simple approach is to behave greedily most of the time, but, every once in a while with a small probability ε, choose a hypothesis randomly, uniformly and, independently of its action-value. These methods are called *near-greedy* or *ε-greedy* selection policies. In the initial phase of learning it is desirable to include an exploratory approach and then switch to exploitation later on.

For a decision support system however, it may never be desirable to choose a completely irrelevant hypothesis. A simple ε-greedy approach chooses uniformly from all hypotheses ranging from the worst appearing ones to next-to-best options. To take care of this drawback, a *softmax* selection procedure is employed, in which selection probabilities are a graded function of action-values and hence a better option has a higher probability of being chosen. There can be several grading functions but the most commonly used softmax method uses Gibbs or Boltzmann distribution [66] where the probability of selecting a hypothesis h_t out of all H possible hypotheses is given by

$$P(h_t) = \frac{e^{Q_t(h_t)/\tau}}{\sum_{b=1}^{H} e^{Q_t(b)/\tau}} \tag{6.5}$$

Where, τ is a positive parameter called temperature. A high temperature forces nearly equal probabilities for all hypotheses, and a lower temperature causes more gradation based on action-value $Q_t(h)$.

6.3.3.2 Human-in-the-Loop Learning (HITLL)

Machine Learning (ML) is one of the key tools in developing autonomous intelligent systems. While designing such learning environments it is critical to collect, codify and present an exhaustive set of experiential knowledge to the system during the training phase, which poses several challenges. First, it is almost next to impossible to hard-code all information even if we assume that such knowledge is available in the form of human experience and can be somehow articulated and codified. Furthermore, other practical problems, such as long training time requirements and non-scaling state representations, make it even more difficult for real-life implementations. In such environments, introduction of human teachers in the learning loop can make this task relatively simpler and tractable. Several Reinforcement Learning (RL) applications have been developed with an interactive human teacher specifically in the field of mobile robotics [68-70]. While RL is not traditionally designed for interactive supervisory input from a human teacher, several works in both robot and software agents have adapted it for human input by letting a human trainer control the reward signal [69, 71]. They show that human-given reward is compatible with the traditional reward signal and can significantly accelerate the learning activity. Therefore, we adopt a Human-in-the-loop-learning (HITLL) framework where not only the experts are kept in the loop for continuous performance monitoring, but also the system learning activity is accelerated.

6.3.3.3 Integrating Human in CBM Knowledgebase

In the framework adopted in this thesis the human element is integrated into the system in three modes. In each mode the specific role played by the human is different yet important to achieve the over all goal of a successful Decision Support System (DSS). These three modes are described as follows (Figure 1):

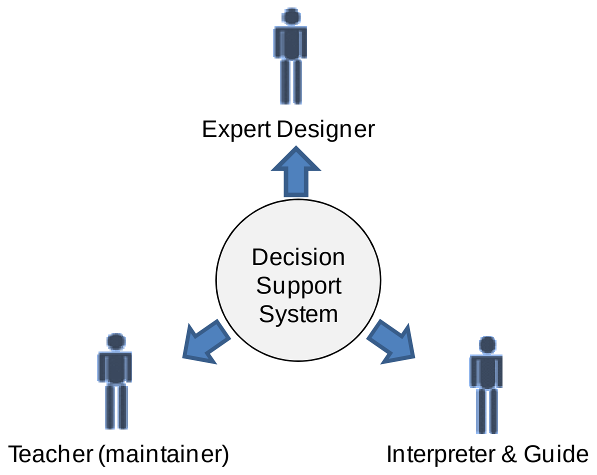


Figure 6.4 Three modes of HITL integration in a Decision Support System.

1) **Expert Designer**: Typically while implementing a system, a model and set of features are developed, trained on some training data and evaluated on some development data. Then the model and features are augmented/changed, retrained and retested on the development data. These iterations are carried out by expert designers of the system until they are satisfied with the developmental results. This is "human in the loop" because in each iteration we're using our human knowledge to add some additional features that will hopefully correct for errors on the development data. In building a CBM knowledgebase development of various diagnostic algorithms like feature calculation, classifier implementation, and preference weight assignments for information fusion falls under this

category of human involvement. In most cases this is offline analysis carried out in the design phase.

2) **Interpreter and Guide**: As discussed earlier, the main goal of knowledge integration into the CBM cycle is to build a DSS. Through its attributes of self-evolution the knowledge based system is expected to learn experts' experience to the extent that knowledge can be successfully articulated and codified. To address the rest of the situations, where this knowledge can not be possibly imparted to the system, presence of a human expert is deemed necessary to avoid any incorrect decisions as far as possible. Therefore, human plays an important role as the interpreter of the suggestions generated by the DSS. It helps the decision maker by narrowing down the choices based on past scenarios, but the actual decision making still stays at the discretion of the user of the DSS. Another role that can be possibly played by human in this capacity is to guide the learning activity through *anticipatory guidance rewards* [69]. This activity is similar to populating the knowledgebase during initialization phase when different preference weights are hardcoded based on expert opinions. The similar process can be formally carried out in the real-time when experts know the correct solution to the current problem and guide the DSS to the corresponding suggestion through anticipatory guidance rewards.

3) **Teacher (maintainer)**: The third role played by a human is that of a teacher that provides feedback on system's performance and guides the reinforcement learning. This task is performed by maintainers of the system who actually act upon the suggestions from the DSS and evaluate if the action taken was

successful or not. A challenging problem in such Human-in-the-loop (HITL) systems is that the evaluation of the system performance is inherently subjective or is based on judgments made by maintenance experts with criteria formed from their experience, which they are unable (or unwilling) to articulate. In this thesis we have devised a way in which the user's feedback is quantized in a standard manner and is interpreted by the system in a consistent manner over a period of time. The user is given a predetermined set of choices to express feedback, which are interpreted based on a reward structure designed in advance. Two kinds of reward structures are suggested. Some rewards are simply +1 or -1 representing success or failure of an action. These rewards are *sparse* and are fairly simple to formulate. In other cases a *dense* reward structure is used where the reward is proportional to the quality of action performed [72]. We illustrate these concepts in the following sections.

In many cases these distinct roles may be played by the same pool of people at different stages of a DSS lifecycle even though respective integration mechanisms are fairly different.

6.3.3.4 Learning Process

For the initial diagnosis scenario, consisting of symptoms and hypotheses, there are only two states as shown in Figure 6.3. Having received some symptoms, the system knows that there is an anomaly in the system, which could be a failure in good probability. But the system has not yet identified the failure. By proposing the most probable hypothesis for the failure it will try to maximize the probability of success in isolating and identifying the correct fault mode. The complete learning process can be described in the following steps and is shown in Figure 6.8.

- At any discrete time step t the system receives a failure symptom s_t. It decomposes the symptom into its constituent triads to generate a list of possible hypotheses. These hypotheses are ranked based on the weights associated with them.

- One of the hypotheses is chosen according to the action-selection policy employed. This hypothesis (h_t) is assumed to be the main cause for the observed symptoms and is selected to execute the diagnostic tests associated with it.

- The human feedback is obtained in the form of the usefulness of the actions performed through a reinforcement $r(s_t, h_t)$ and the next state s_{t+1}. The next state could be the failure-unknown state or failure-known state depending on the success of the choice made. The reinforcement is formalized as a number, larger for beneficial and smaller for detrimental choices, respectively.

Algorithm: Q-Learning with HITL Feedback Rewards:
s_t = current state, s_{t+1} =next state, h_t = current selection of hypothesis, r = feedback human reward

```
1:  while learning do
2:      $h_t$ = select hypothesis according to policy $\pi^*(s)$ from options weighted by $Q[s,h]$
3:      carry out diagnostic tests corresponding to $h_t$ and transition to $s_{t+1}$
        (allow small delay to receive human reward)
4:      obtain reward $r$
5:      update $Q$-values according to:
        $Q^{new}(s_t,h_t) \leftarrow (1-\alpha)Q^{old}(s_t,h_t) + \alpha[r(s_t,h_t) + \gamma \max_h Q(s_{t+1},h)]$
6:  end while
```

Reward Structure: As depicted in Figure 6.8, the learning activity takes place at three locations, i.e. weights between triads-hypotheses, hypotheses-diagnoses and, diagnoses-repairs. The nature of all three instances is different and hence the reward structure is different as well.

Rewards for triad-hypothesis (*t-h*) associations: Initially, during the pure learning phase each constituent triad of a symptom will have an equal association with the corresponding hypothesis. Later, if any of these triads appears as part of another symptom explained by a different hypothesis, its association with the new hypothesis will grow and with the previous hypothesis will diminish. Thus, over a period of time associations between symptoms and hypotheses will transform into associations between triads and hypotheses. This will help in suggesting solutions where an entirely new symptom is observed, which consists of already existing triads in the knowledgebase. For this situation a simple *sparse reward* structure is suitable where a reward of +1 is awarded to a successful hypothesis and -1 for a failed hypothesis.

Rewards for hypothesis-diagnosis (*h-d*) associations: In this case not only the success and failure of a diagnostic tests matters, but also its effectiveness. For instance, if a diagnostic test confirms a fault or no-fault condition with a higher confidence than another test it must receive a higher reward. This method implements a *dense reward* structure, where the reward is a function of the confidence measure calculated in identifying a failure.

To accomplish this, a two step procedure was adopted.

Step 1: A preset threshold (T_0) for the desired confidence level is chosen. Each diagnostic test is tested to achieve T_0. A test that declares a no-fault condition is considered a failed test and included in the failed tests list. If, at least one test declares a fault condition with the preset confidence level the hypothesis is confirmed as true and is rewarded with a positive reward. At the same time all the failed tests are updated with a negative reward and the successful test is updated with a positive reward (Figure 6.5). Otherwise, if all tests fail to cross the desired threshold step 2 is applied.

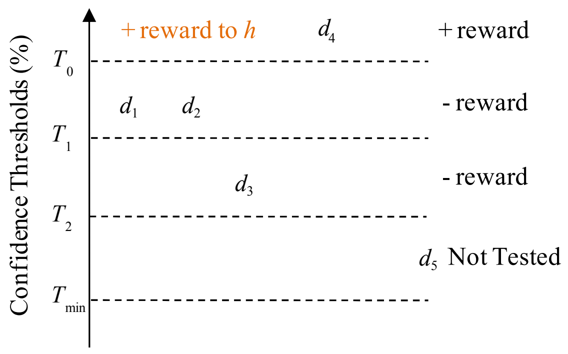


Figure 6.5 A successful hypothesis and at least one successful diagnostic test.

Step 2: There can be two reasons why all tests failed to confirm the presence of the fault.

1) An incorrect hypothesis was chosen at the previous level.

2) The data is too noisy to achieve the desired confidence level in fault detection.

In the first case, a negative reward is sent to the incorrect hypothesis without updating the weights of the diagnostic tests (Figure 6.6).

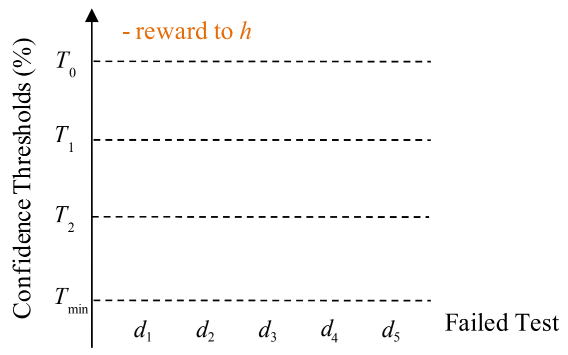


Figure 6.6 The unsuccessful hypothesis case.

However in the second case, another chance must be given to the hypothesis. Therefore a preset minimum allowable confidence threshold (T_{min}) is chosen and all tests are carried out against it. If none of the tests still cross the threshold the hypothesis is declared false and updated with a negative reward. If one or more tests clear the threshold T_{min}, they are further ranked based on the maximum threshold levels they can clear. The rewards to these successful candidates are proportional to the confidence levels they achieve. Thus a better test gets a higher reward. And the corresponding hypothesis is updated with a positive reward (Figure 6.7).

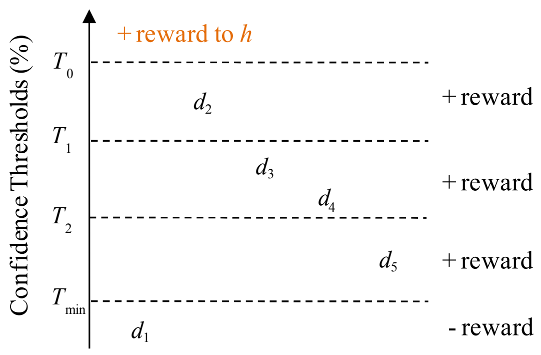
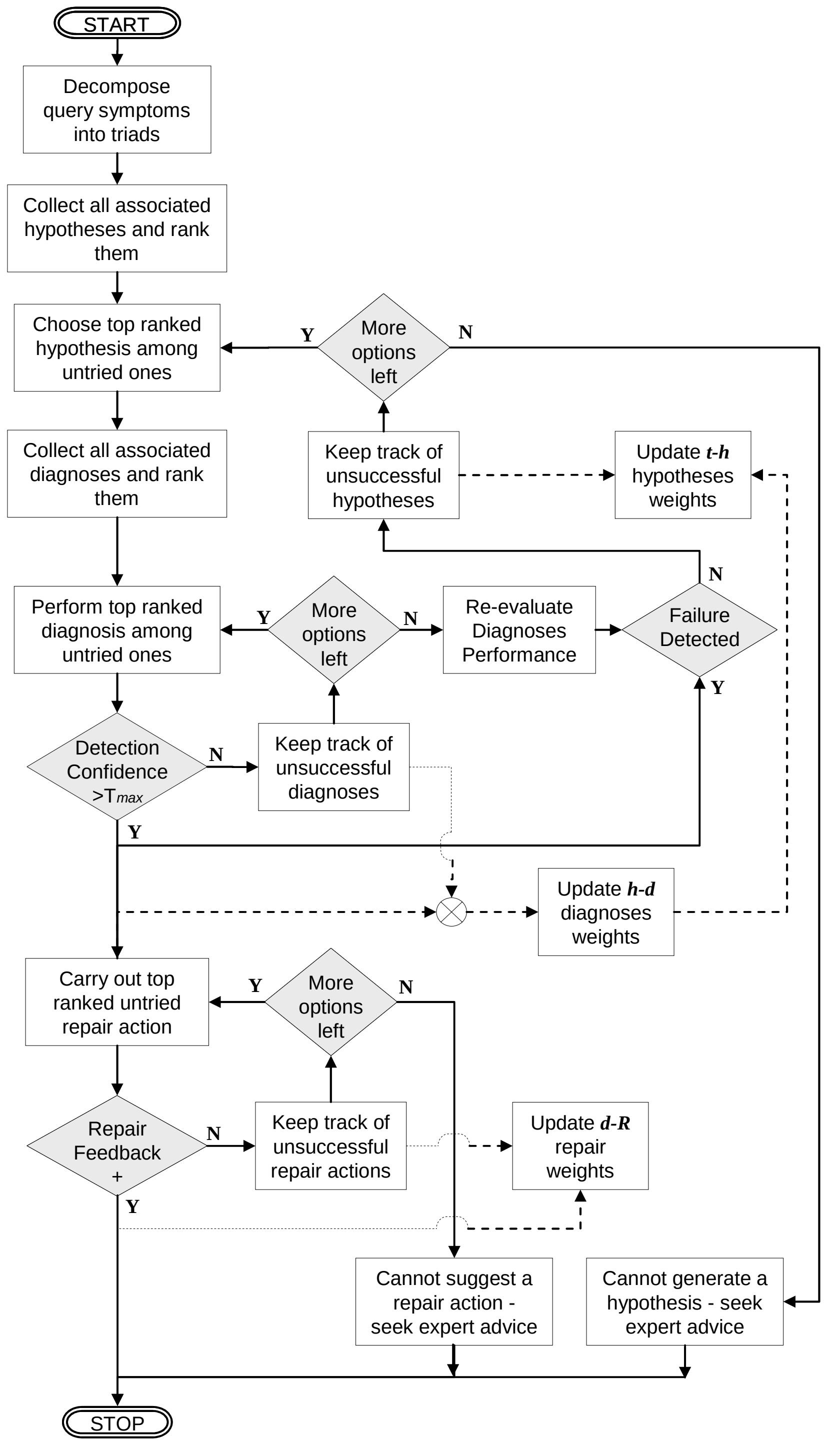


Figure 6.7 A successful hypothesis but lower confidence due to poor quality data.

In the example illustrated below, the rewards are proportional to the success of each test, i.e., for the successful tests the rewards structure follows an order $r_2 > r_3 = r_4 > r_5$.

Rewards for diagnosis-repair (*d*-*R*) associations: Once a diagnostic test confirms the presence or absence of a suspected fault a reward is issued accordingly to update the weights for the corresponding hypothesis and the test. If a hypothesis is successful, it is assumed that the fault has been identified. Now, the weights on repair actions are completely independent of the success at the previous level. If a repair action fails to fix the diagnosed problem, its outcome does not propagate back to the hypothesis or diagnosis level. Hence, only the weights on repair actions are affected. The rewards structure can be either +1 or -1 for success and failure, respectively, or if the operator feedback can be obtained about the effectiveness of the repair action, a reward can be a number directly proportional to its degree of effectiveness. If none of the available repair actions are effective then an expert's advice must be taken because the system has not seen such a situation before and hence can not help in decision making in this case. Again, it must be noted that this is a decision support system and therefore the presence of a human expert is expected to make decisions in situations where DSS fails to provide any conclusive suggestion.


START
Decompose query symptoms into triads
Collect all associated hypotheses and rank them
Choose top ranked hypothesis among untried ones
Y
More options left
N
Collect all associated diagnoses and rank them
Keep track of unsuccessful hypotheses
Update *t-h* hypotheses weights
N
Perform top ranked diagnosis among untried ones
Y
More options left
N
Re-evaluate Diagnoses Performance
Failure Detected
Y
Detection Confidence >T*max*
N
Keep track of unsuccessful diagnoses
Y
Update *h-d* diagnoses weights
Carry out top ranked untried repair action
Y
More options left
N
Repair Feedback +
N
Keep track of unsuccessful repair actions
Update *d-R* repair weights
Y
Cannot suggest a repair action - seek expert advice
Cannot generate a hypothesis - seek expert advice
STOP

The diagnostic process starts when a query symptom is received as a user input.

The user has a choice to assign priority to various symptoms if multiple symptoms are received. These priority weights further percolate to modify the ranking of all the hypotheses associated with the constituent triads.

The weights are not updated until all options have been exhausted. If all options have been exhausted with negative outcomes then most likely the choice made at the previous level was incorrect and a negative reward is propagated back.

To classify a diagnostic test as success or failure, a confidence measure is calculated for the diagnostic outcome. If this confidence level is below a preset threshold the test is considered ineffective.

If all options are classified as ineffective, re-evaluation is carried out with lower thresholds. If failure is now detected, each diagnosis is updated with a weight proportional to its maximum confidence level.

If all options are still ineffective, a negative reward is propagated to decrease the weight of chosen hypothesis.

If a **fault** is detected with desired confidence level a positive reward is issued for both, the current diagnosis and the hypothesis.

If **no_fault** is detected with the desired confidence level then a positive reward is issued to the diagnosis but a negative reward is issued to the hypothesis.

In situations where a fault has been identified but none of the repair actions seem suitable, experts' advice must be taken

In situations where none of the hypotheses suggested by the knowledge-base explains the symptoms correctly, expert's advice must be taken.

Figure 6.8 A flowchart showing the complete decision support process with learning.

6.3.3.5 Learning Scenarios

In general, several scenarios about how the environment behaves can occur in practice. For this research two main scenarios were considered to show a proof of concept about how learning can be adapted to different scenarios involving HITL. These scenarios are briefly described below.

Scenario 1

In the first scenario, it is assumed that a symptom can be explained by several hypotheses (fault modes). However, in the current environment the relative distribution of occurrence of these fault modes is different. These faults occur in a random order with a probability reflecting their relative frequencies. The learning activity should result in a higher weight for a fault mode (hypothesis) that occurs more frequently than less frequent fault modes. Later as the environment changes this distribution may also change and the learning activity should reflect this change. An important consideration in judging the effectiveness of a learning scheme is the response time in which these changes are reflected in the decision making process while making a selection for the most probable hypothesis.

Example: Unusual vibrations occurring in the centrifuge result from main motor bearing defect in 75% of cases, faulty primary gear in 20% of cases, and other drive faults in the rest 5% of cases. Thus, whenever unusual vibrations are observed the system should check for main motor bearing defects with a higher priority than carrying out diagnostic tests for faulty gear defects or other drive defects.

Scenario 2

In the second scenario, it is assumed that although a symptom can be explained by different fault modes that are equally frequent, the changes in system behavior are reflected with a different degree of severity from this symptom. Hence, a particular hypothesis has varying degrees of support from different symptoms. A symptom with a higher degree of support for a particular fault mode should have a higher weight on its association with that hypothesis. Now, if at some point of time a new failure mode occurs in the system which has an even stronger characteristic of unusual vibration, the learning scheme should reflect this change in the weights associated with different hypotheses.

Example: An unusual vibration in centrifuge is a stronger characteristic of a main motor bearing defect than a primary gear defect. But a shrieking sound is a stronger characteristic of primary gear defect than a bearing defect. Both defects are equally likely to occur statistically. Thus the vibration symptom should have a stronger association with bearing defect than the gear defect. Later, if a shaft misalignment problem results in more prominent unusual vibrations then the decision support system should suggest solutions accordingly.

Other scenarios may also occur and the *Q*-learning scheme can easily be adapted in a manner suitable to those scenarios. Some simulation results exhibiting learning in both scenarios have been included in the results section.

6.3.4 Self-Maintenance

Self-maintenance refers to reorganization of the knowledgebase. Each data entity is labeled with a time stamp indicating the time of its most recent use. At the time of scheduled self-maintenance all the entities that have not been used in a long time (limit preset by the user) may be shifted to a passive memory. At any point of time, the list of options is generated only from the active memory contents. Once all options have been exhausted, more options may be looked up from the passive memory. In case more options are found in the passive memory they may be tried and if successful can be brought back to the active memory before a negative reward is propagated to the previous level. Similarly the entities whose weights drop below a minimum threshold may also be shifted to the passive memory. They may be recalled and tried only if all other options have been exhausted.

The passive memory can be configured in First-In-First-Out (FIFO) manner such that once it gets full the oldest entries are flushed out first to make room for new entrants. In this manner the knowledgebase always contains the most relevant data and does not grow in an uncontrolled manner.

6.3.5 Autonomous Behavior

All the activities described above are carried out in an autonomous fashion. In some cases, where expert assistance may be required, a flag is thrown to indicate such condition. Mainly, if activities like self-evaluation, learning and self-maintenance can be carried out in an autonomous fashion the system can be regarded as autonomous.

However, it is important to have access to various modules to manually monitor the activities and evaluate the performance. The knowledge management system, described in the next sections, is a transparent means to access the knowledgebase for expert's inspection and modification.

6.4 Simulation Results and Discussion

Two different experimental simulations were designed to show learning in the different scenarios discussed above.

Scenario 1

Initially there are two symptoms s_a and s_b each with its respective set of hypotheses Ha and H_b, where $H_a = \{h_1, h_2, h_3\}$ and $H_b = \{h_4, h_5\}$ with their respective initial probabilities of occurrence $p_a = [0.05, 0.9, 0.05]$ and $p_b = [0.3, 0.7]$. After about 400 episodes the environment changes and the change is reflected in the changed probability distribution $p_a' = [0.75, 0.2, 0.05]$ and $p_b' = [0.5, 0.5]$. The following simulation results show that these changes are effectively reflected in the learning algorithm. The response to this changes by Q-learning has been compared with the simple frequency based learning scheme. To compare the results with frequency based weights, all Q-values have been normalized to a number between 0 and 1. Both symptoms are independent of each other and occur randomly with equal probability. The results have been averaged over 20 experiments. The reward structure here is +1 if a correct hypothesis is chosen and -1 if an incorrect hypothesis is chosen.

Results from scenario 1 have been compiled in Figure 6.9 and Figure 6.10. The first row in each figure shows the *Q*-learning performance and the second row shows frequency based learning. The first column plots in each figure represent simulations for a stationary environment where no significant change was observed as time passed by. The second row shows when the probability distribution of different faults changed drastically after 400th episode.

As can be seen from these figures, for stationary environments frequency based method exactly replicates the probability of occurrence. However, for non-stationary case the learning of new probabilities is extremely slow, whereas *Q*-learning quickly adapts to the new environment. This rate of adaptation can be changed by changing the learning rate parameter α.

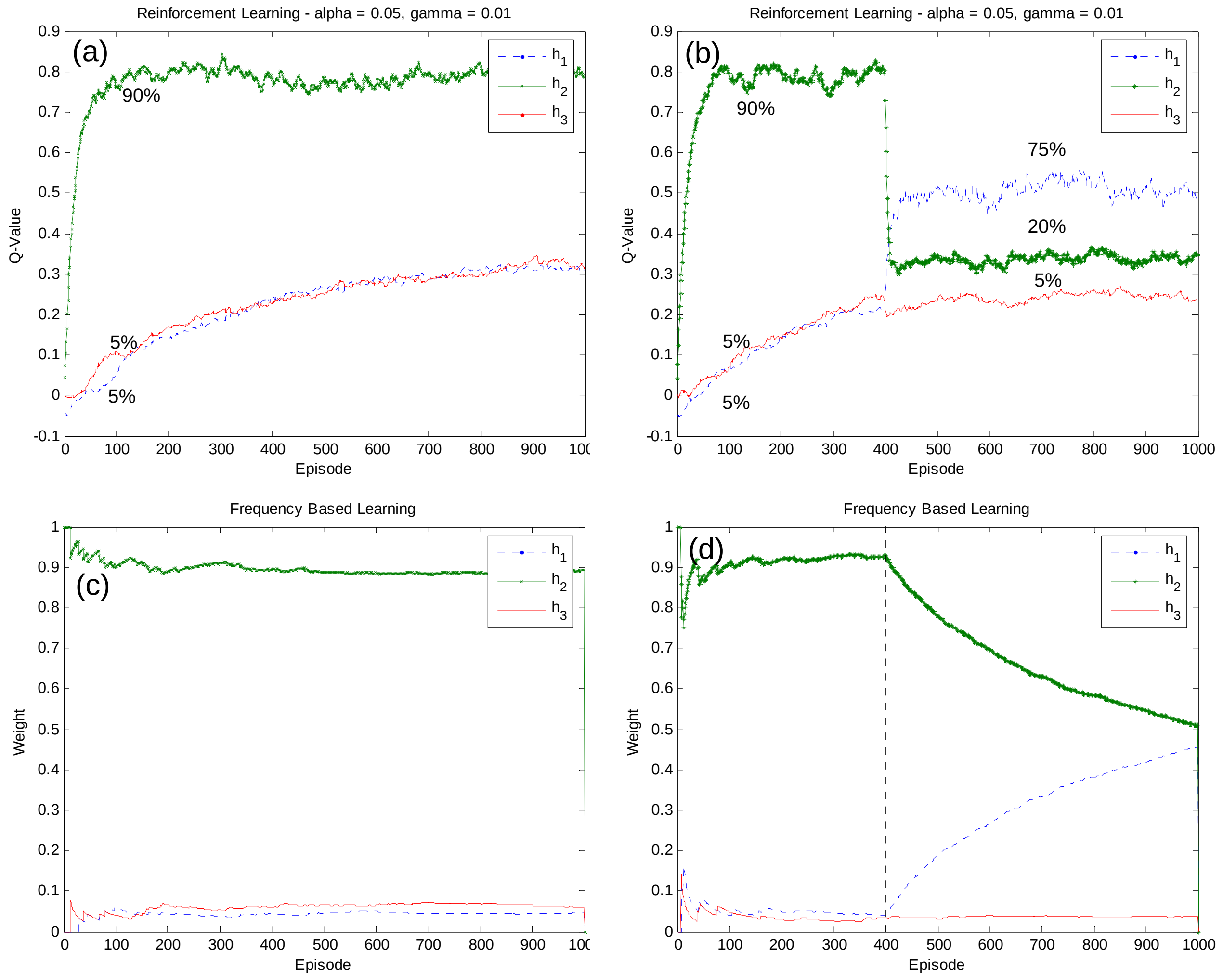


Figure 6.9 Comparing *Q*-Learning performance with frequency based learning for symptom s_a.

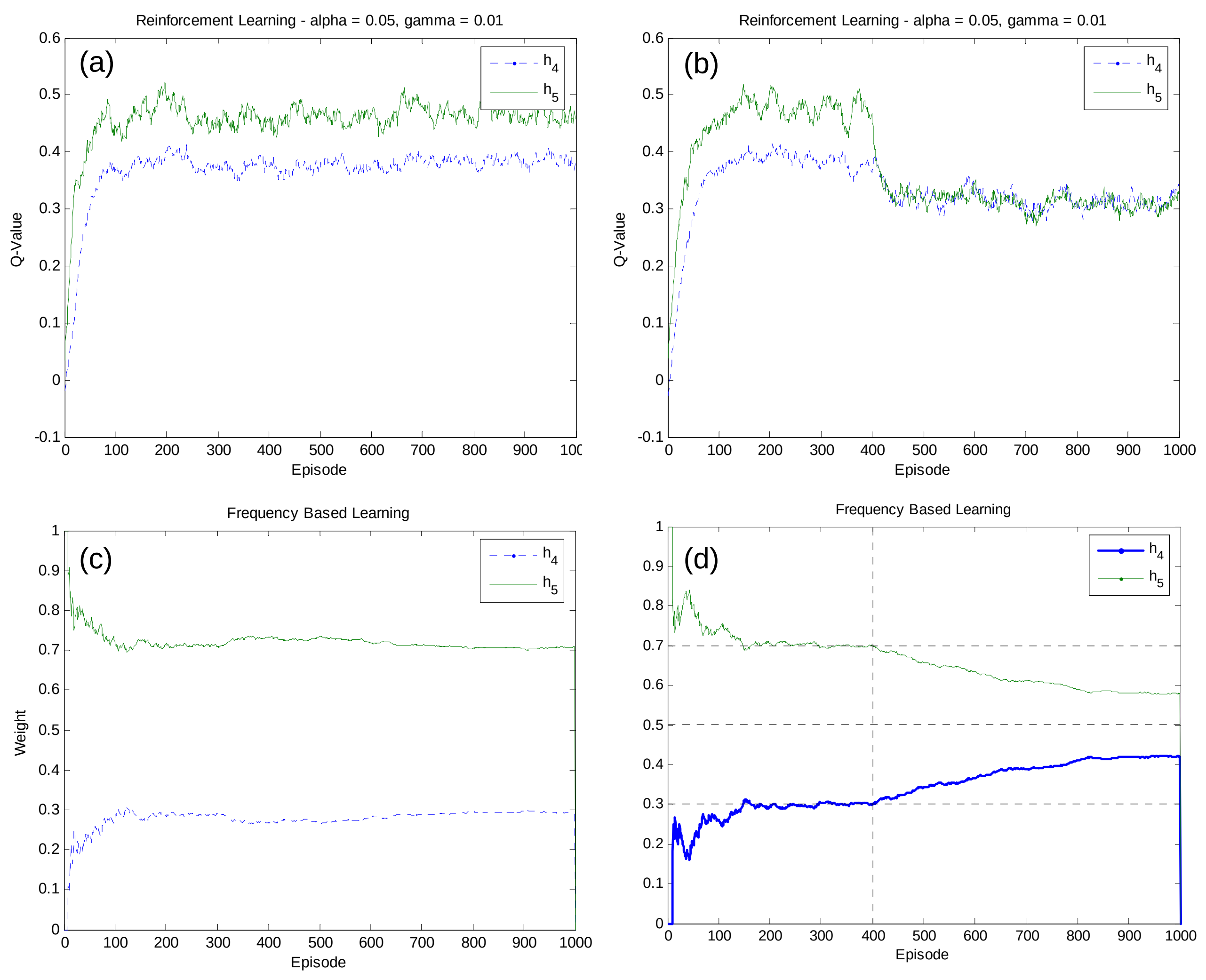


Figure 6.10 Comparing *Q*-Learning performance with frequency based learning for symptom s_b

Scenario 2

A symptom s is explained by a set of hypotheses $H = \{h_1, h_2\}$. Both h_1, h_2 have equal probabilities of occurrence but h_1 is detected with a maximum confidence of 0.80 than h_5, which is detected with a maximum confidence of only 0.75. This means that s is a stronger characteristic of h_1. After about 400 episodes a new hypothesis (h_3) is added to the system for which s is a very strong symptom and results in detection confidence of 0.95. The learning of Q-values reflects these changes as shown in the following results. The results have been averaged over 20 simulations, and each of the three failure modes were simulated to occur with an equal probability.

A similar trend in performance is observed in scenario 2 as well. Since all three faults are equally likely their weights are close to each other (i.e. converge to 0.5 for two and 0.33 for three hypotheses). However, the hypothesis that is more convincingly explained by this symptom has a slightly higher weight in Q-learning case. For the frequency based method this information about sensitivity is not included at all. Further, Figure 6.11 shows that as soon as a better faring diagnostic test is included in the knowledgebase its Q-value improves sharply and very quickly it becomes the favorite test to be carried out. In the frequency based method, first, the learning is extremely slow and next it will never learn the preferences. Frequency based methods are extremely simple and intuitive but they are not capable of handling non-stationary environments.

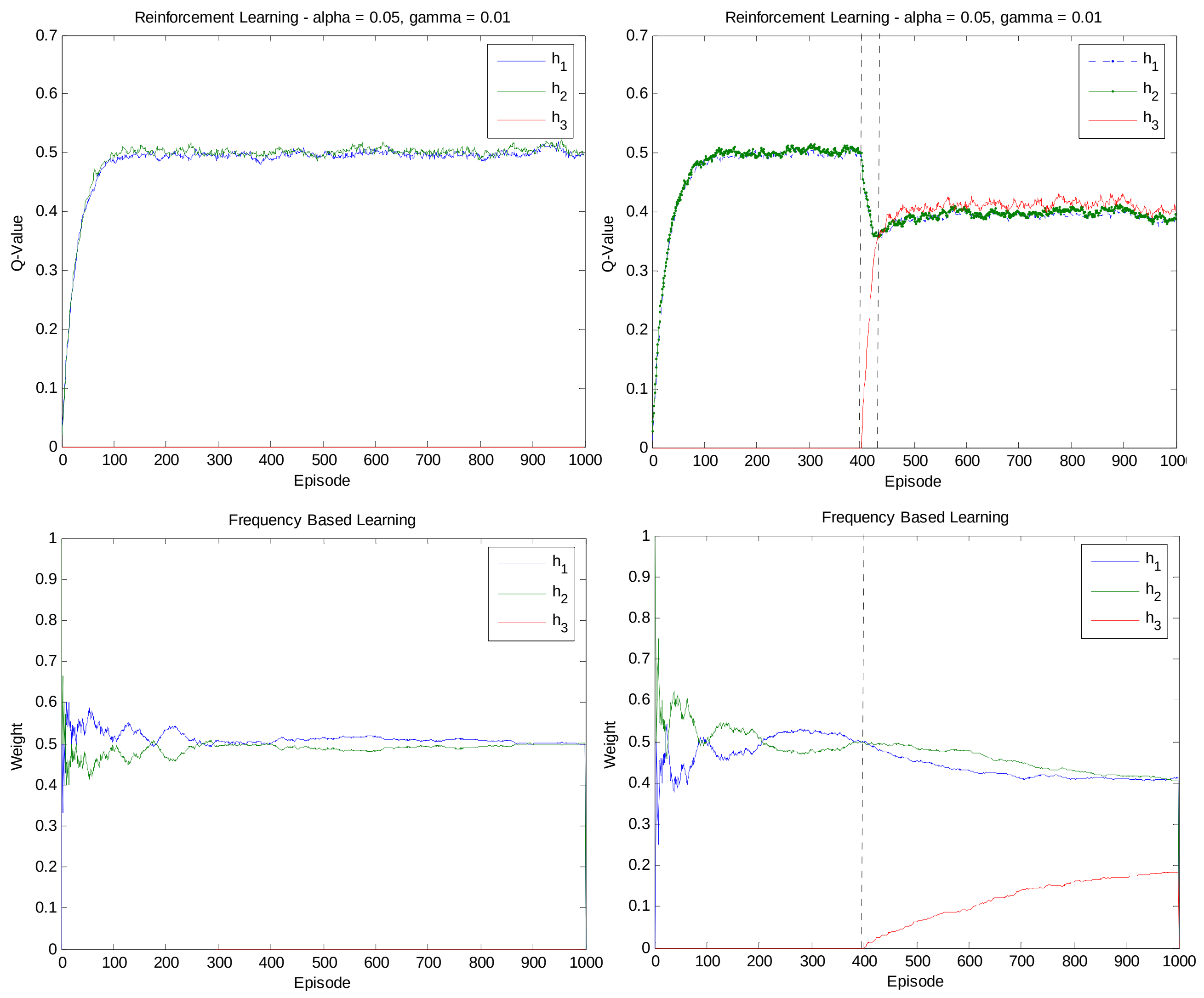


Figure 6.11 Comparing *Q*-Learning performance with frequency based learning in scenario 2.

6.5 The Knowledge Management Interface

As much as it is important to design and structure a knowledgebase it is equally important to provide means to store and access the data into the knowledgebase. At the same time it is desirable to have an interface that allows experts to peek into the reasoning process at different stages for monitoring from time to time of the workings of an autonomous system. Another aspect of such an interface is to let experts carry out any modifications in the knowledge at anytime they consider appropriate.

In this research few basic interface modules have been designed to visually show the dynamics of the knowledgebase. These modules have been briefly described and shown below.

Data Entry Module

First, a structural model of the system must be entered into the knowledgebase. This allows defining the system-of-systems hierarchy and also provides the list of all components in the system (see Figure 6.12).

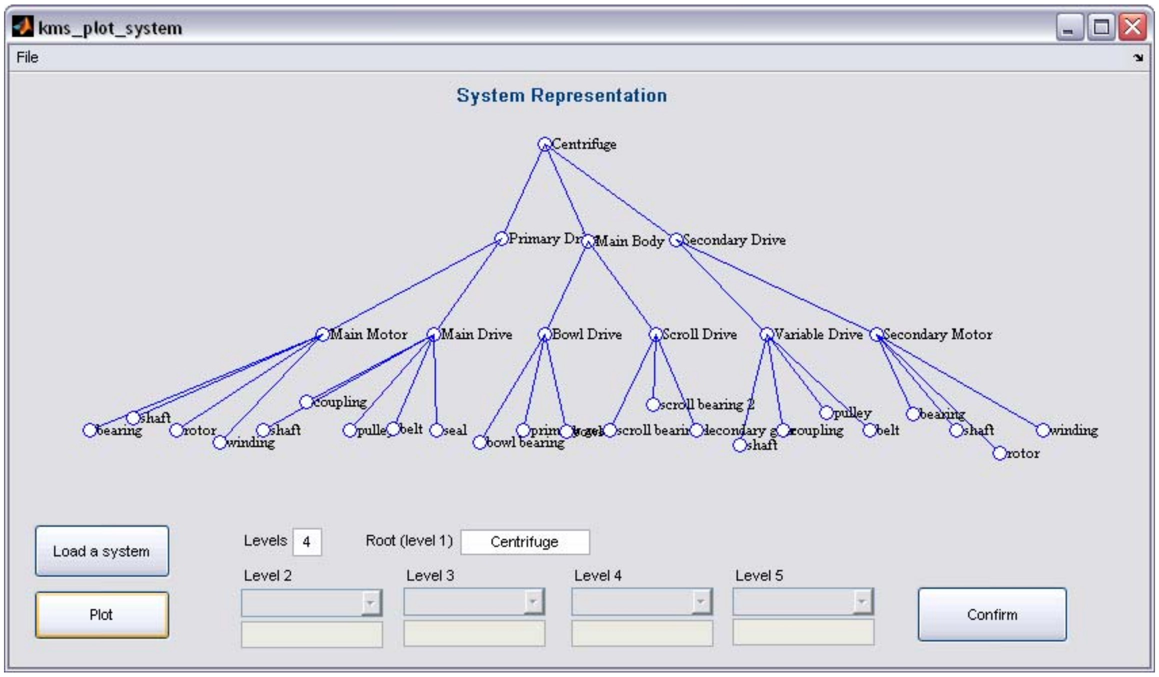
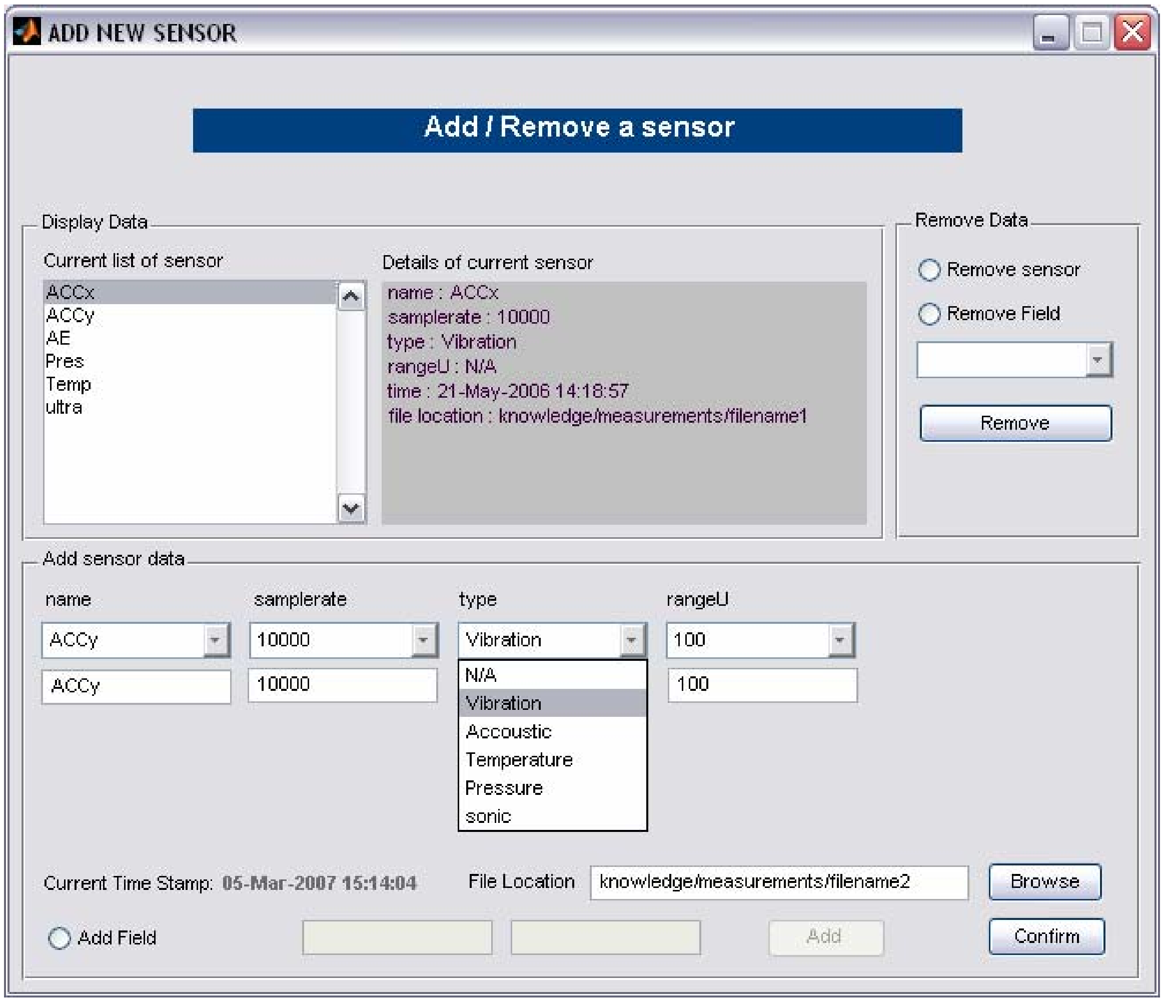


Figure 6.12 Interface to enter structural model of the system.

It also provides the connections between various components and their respective subsystems that are often important in localizing the faults. Once the system has been entered, it can be stored in the knowledgebase and accessed later by loading it into the active workspace.

The next step is to specify the attributes associated with different kinds of objects. For instance, information regarding various components, sensors and their specifications, feature calculating algorithms and their parameters, etc. must be added to the knowledgebase. At the same time the interface should allow removing or modifying any object whenever required. This was accomplished though a user interactive interface for adding a knowledge element as shown in Figure 6.13.



name : ACCx
samplerate : 10000
type : Vibration
rangeU : N/A
time : 21-May-2006 14:18:57
file location : knowledge/measurements/filename1

name: ACCy, ACCy
samplerate: 10000, 10000
type: Vibration
rangeU: 100, 100

Current Time Stamp: 05-Mar-2007 15:14:04
File Location: knowledge/measurements/filename2

Figure 6.13 Interface to add or modify an object into the knowledgebase.

The main attributes of this module are:

- One can add an object, e.g. a new sensor may be added to the knowledgebase.
- More fields may be added to an object, e.g. another field rangeL, specifying the lower range of sensor sensitivity may be added.
- At any instant all details associated with the selected object are displayed.

- These attributes include a time stamp associated with each object, which gets updated whenever that object is accessed last.

- Also included is location information where the data corresponding to this object may be obtained from. E.g. in our case we use sensor data files stored in the database.

- Finally, any object or any of its attributes may be deleted at any time. Whenever deleting an object, a confirmation dialogue pops up to avoid deletion by mistake.

A similar interface is available for other types of objects as mentioned earlier.

Visualization Module

Visualization module mainly consists of two interfaces. One of these interfaces shows a query symptom into its semantic network form (Figure 6.14). This allows users to monitor the performance of the ILP algorithm. At anytime the semantic nets created by the ILP algorithm may be compared with what experts would manually draw. This helps in identifying trouble spots whenever an unknown structure is encountered in symptoms. Further, a transcript of the ILP sequence of steps is printed on the command prompt to guide the debugging process in an easy manner.

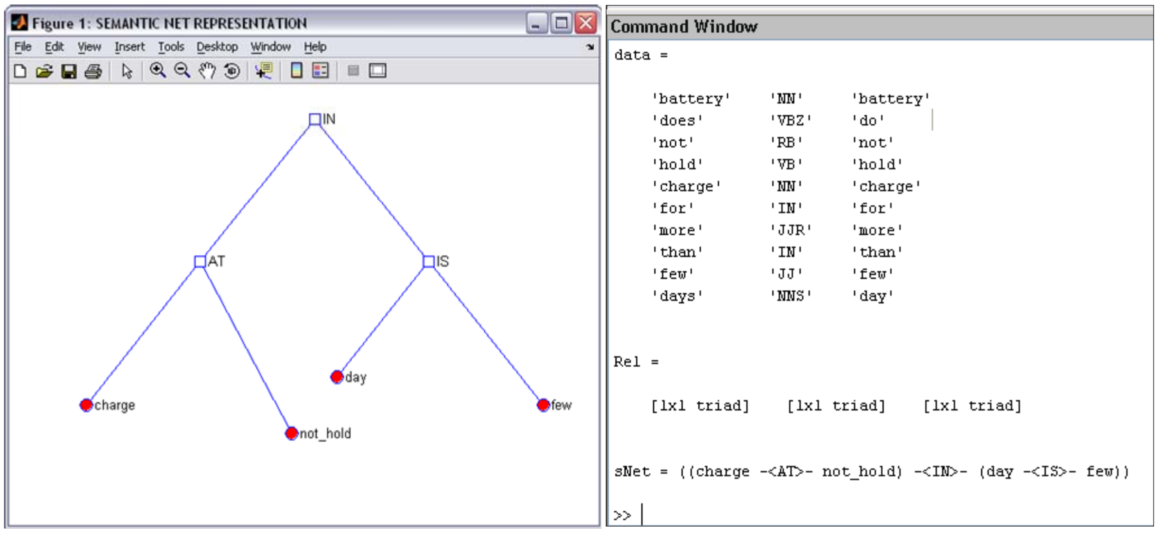
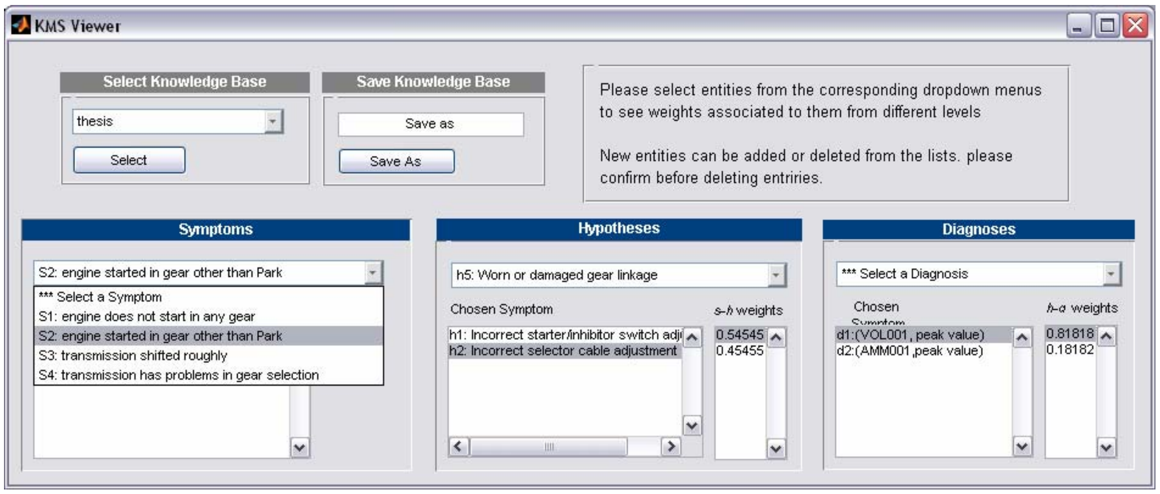


Figure 1: SEMANTIC NET REPRESENTATION
File Edit View Insert Tools Desktop Window Help
IN
AT
IS
charge
not_hold
day
few
Command Window

```
data =

    'battery'    'NN'     'battery'
    'does'       'VBZ'    'do'
    'not'        'RB'     'not'
    'hold'       'VB'     'hold'
    'charge'     'NN'     'charge'
    'for'        'IN'     'for'
    'more'       'JJR'    'more'
    'than'       'IN'     'than'
    'few'        'JJ'     'few'
    'days'       'NNS'    'day'


Rel =

    [1x1 triad]    [1x1 triad]    [1x1 triad]


sNet = ((charge -<AT>- not_hold) -<IN>- (day -<IS>- few))

>>
```

Figure 6.14 SemanticNet visualization module.

The next module shows the learning behavior of the knowledgebase by showing the association weights between various entities at any point of time. Since this application is episodic in nature, after each episode the changes in the weights between various entities can be monitored.



thesis

Save as

Please select entities from the corresponding dropdown menus to see weights associated to them from different levels

New entities can be added or deleted from the lists. please confirm before deleting entriries.

S2: engine started in gear other than Park
*** Select a Symptom
S1: engine does not start in any gear
S2: engine started in gear other than Park
S3: transmission shifted roughly
S4: transmission has problems in gear selection

h5: Worn or damaged gear linkage

Chosen Symptom	s-h weights
h1: Incorrect starter/inhibitor switch adj	0.54545
h2: Incorrect selector cable adjustment	0.45455

*** Select a Diagnosis

Chosen Symptom	h-d weights
d1:(VOL001, peak value)	0.81818
d2:(AMM001 ,peak value)	0.18182

Figure 6.15 Module to monitor weights between various entities of the knowledgebase.

6.6 Evaluating KB Performance

The knowledgebase structure developed in this thesis should be evaluated for performance issues arising from attributes like learning, self-maintenance, and self-evaluation. The learning rate parameter α should be adjusted such that learning in new environments is quick but at the same time any differences arising due to noise should not affect the outcome of the system.

To improve and optimize KB performance, a temporal recency threshold is set beyond which the entries are moved to passive memory. This threshold should be carefully determined taking into account the total time to fetch entries from the passive memory and their relative frequency of being fetched from the passive memory.

6.7 Conclusions

In this chapter we have developed a framework for intelligent knowledgebase for integration with CBM architecture for an autonomous DSS. With the help of several examples, various attributes of intelligence were described that bring this process model closer to autonomy. Further, a particular emphasis was put on learning mechanism and a Q-learning scheme, incorporating human-in-the-loop experience, was adopted and was shown to be effective in two different simulation scenarios. Last but not the least a knowledge management interface was described with the help of few modules developed during this research.

Chapter 7

Conclusions and Future Work

7.1 Concluding Remarks

This thesis addresses a key issue of knowledge integration in the development of CBM/PHM technologies. In addition to signal processing and subsequent diagnostic and prognostic algorithms these new technologies require storage of large volumes of both quantitative and qualitative information to carry out maintenance tasks effectively. From the volumes of data that can be obtained today, information extraction has been a challenging task and organizing this information, so that it can be considered useful knowledge, is yet another level of abstraction. A knowledge integrated approach provides means to store, organize, and access this knowledge in a timely and efficient fashion.

We presented a two-level diagnosis scheme in which first a fault is hypothesized using the observed symptoms from the system and then a more specific diagnostic test is carried out to confirm the hypothesis. This results in a pin-pointed diagnostic data processing with reduced computational overhead. An Industrial Language Processing (ILP) technique has been developed for processing textual information from industrial systems. Compared to other automated methods that are computationally expensive, this technique manipulates standardized language messages by taking advantage of their semi-structured nature and domain limited vocabulary.

The key assumption in this architecture is the use of standardized language in the industrial environments. Although this has been a preferred practice in the aviation industry, other industries are yet to embrace this concept completely. However, there is significant evidence that efforts are being made in this direction at the organizational level. Although, ideally a common standard specific to an industry would be preferred, a competitive attitude and high overhead is currently resulting into an individualistic effort from different organizations. However, the approach presented in this thesis is quite generic and can be easily adapted to specific cases.

Further, an intelligent reasoner is required that can make judicious use of this knowledge and provide a substantial support in the decision making process. A Dynamic Case-based reasoning (DCBR) framework has been used as a hybrid platform for diagnostic reasoning and an integration mechanism for the operational infrastructure of an autonomous CBM system. This integration involves data gathering, information extraction procedures, real-time reasoning frameworks and decision-support systems to facilitate the strategies and maintenance of critical systems. A structured approach to data acquisition and information extraction is outlined that makes use of already existing various industrial practices. This knowledge is stored in a self-evolving knowledgebase that learns from its performance feedback and reorganizes itself to deal with non-stationary environments. Attributes of learning, feedback, self-evaluation, self-maintenance and autonomous behavior are discussed and instantiated with the help of some example scenarios. We have used Q-learning using external human feedback as the main learning algorithm for experience accumulation. However, other learning methods may be adopted depending on the structure of the knowledgebase and the problem scenarios.

Despite logical arguments regarding the effectiveness of the two-level diagnosis approach and the evolving knowledgebase, a theoretical evaluation is required to clearly show the advantages and disadvantages of the suggested architecture. For this purpose it is obviously desired to use commonly used performance measures in these domains. However, due to some fundamental differences in the nature of the problem, owing to the application domain and the absence of adequate data to compare or benchmark with other techniques, customized methods were devised to carry out these evaluations.

These differences arise form the fact that in most cases one requires an experts' subjective assessment in defining the quality of the suggested solution while evaluating performance measures like accuracy and precision. Further, these systems are episodic in nature and any performance evaluation requires external feedback available at the end of the episode. Moreover, several episodes over a long period of time are needed to make any conclusions, unlike continuous time algorithms. Therefore, we have modified some of the conventionally used performance measures and defined new ones wherever appropriate.

In concluding this research we note that integrating knowledge into the CBM architecture is a significant step towards achieving an autonomous DSS in the maintenance process. The learning in such DSS, however, involves human-in-the-loop to receive feedback from the domain experts and learn from their experiences. Adopting this approach in the industrial environments not only the processes can be improved but also the corporate knowledge can be retained as an electronic expert for later reuse.

7.2 Summary of Contributions

Main contributions of this research are:

- Knowledge integrated CBM process model for automated diagnostic Decision Support System.
    - A Knowledge Management System (KMS) for easy storage, access and manipulation of knowledge for later reuse.
- Self-evolving Knowledgebase (KB) that learns from its performance over time and a structured approach to acquire and modify knowledge to populate this KB.
    - A *Q*-learning algorithm with human-in-the-loop feedback to learn from the experience of domain experts.
- Dynamic case-based reasoning platform for simultaneously utilizing textual and numerical information to carry out diagnosis with less computational burden.
    - Two level diagnostic framework
- Industrial Language Processing (ILP) technique to process industrial text, while retaining its domain specific information for effective diagnosis.
    - Information extraction technique for standardized language sentences.
    - Similarity evaluation methods for semantic networks in ILP for matching and retrieval purposes.

7.3 Suggested Future Work

In this thesis we have presented a Decision Support System for CBM. This system integrates various modules in a unique fashion to overcome practical problems of computational tractability and interfacing between qualitative data, quantitative data, and human feedback. In absence of real industrial data these modules were independently developed and tested in an isolated manner. From a systems point of view, performance of the complete system should be assessed as a whole. This performance assessment was not addressed in this research. However, a platform has been set for such experiments and analyses. Here we suggest some important research directions that should be further pursued to guarantee desired performance.

7.3.1 Guaranteeing Overall System Soundness

HITL based learning systems invariably encounter time delays while they wait for human feedback. Such time delays should be modeled in the dynamics of the system to guarantee reasonable performance. In some cases feedback reward may not be used, immediately after it is available, until the correct location of the decision making node is identified. In case of failures, only that specific node should be penalized where the incorrect decision was made and in case of success, all nodes must be rewarded where correct decisions were made. Furthermore, different steps in the diagnostic process take different processing times and therefore provisions should be made to incorporate time delays arising from such situations. Such study was not carried out in this research and therefore, it is desired to carry out a formal study on how to model such time delays to guarantee the best performance.

This thesis presents a closed loop system, where incorporating the external feedback forms a crucial part of the learning activity. This in turn raises the convergence and stability issues. Design parameters like *learning rate* and *time delays* should be carefully chosen to guarantee convergence and stability even in highly random and noisy environment.

7.3.2 DCBR Extension to Prognosis

The DCBR framework described in this thesis can be expanded for prognosis task with little efforts. The most straight forward expansion is based on the fact that already existing prognostic algorithms can be integrated in a similar fashion as diagnostic algorithms to get activated once the fault has been localized. Fault is first identified by diagnostic routines followed by activating the corresponding prognostic routines.

The case structure itself can also be used as a higher level prognostic platform. A time history of the situations can be included as a part of the case. Such a history can be implemented as *Traces. Traces* not only keep track of current state of the system but also the evolution of the state in the recent past. Similarly, the time-tagged indexes as described in [34-36] can be used for generating trends. These trends can be used to make subsequent prognosis.

7.3.3 Improving ILP

As described in Chapter 5, for improved performance more IE rules can be included in the rules list. These rules must be generated once a larger set of standardized sentences is available and more abstract rules covering a number of sentences can be identified. Similarly, more relations may be defined in addition to the three relations used in this thesis. Further, a specialized domain limited vocabulary can be incorporated by using customized dictionaries and thesauri. All these enhancements are purely subjective and can be made as the need is felt.

7.3.4 Performance Metrics

We have suggested some generic performance metrics for individual modules in respective chapters. Depending on the specific application custom performance metrics can be designed to ensure that the system meets desired specifications. These metrics should be used to further refine the respective modules. Similarly, some performance metrics should also be designed to evaluate the over all system. Further research can be carried out on generating confidence bounds around the solutions generated by the DSS. Such confidence bounds should incorporate the uncertainties arising due to environmental noise, incorrect or incomplete data, unknown failure modes, simultaneous multiple failures, variability in human perception of the situation, etc.

Appendix A

Knowledge, Information, and Data

The term information is generically used to refer to all manner of descriptions or representations from raw signals on the one hand to knowledge and understanding on the other. It is important to recognize that information can be categorized in five different classes based on the usefulness that can be derived from it. One must understand the differences between these classes because they are of different value in decision making and maintenance.

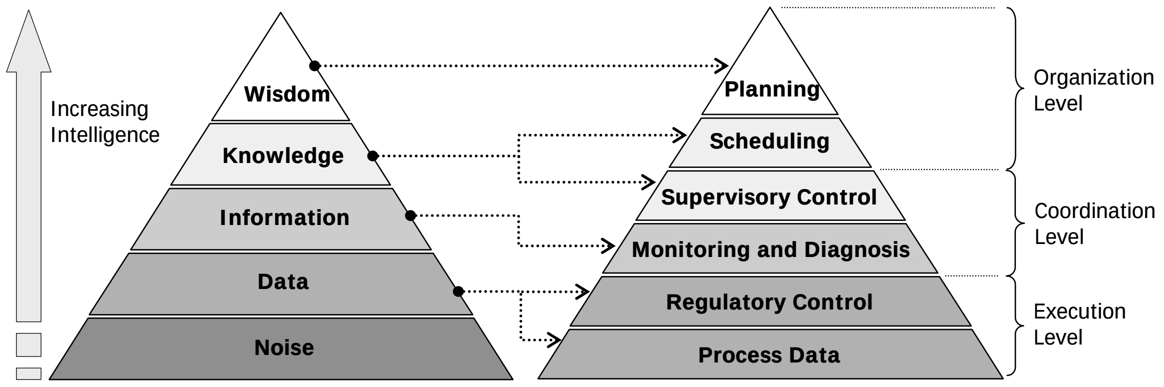


Figure A.1 Schematic showing information hierarchy [15] and a parallel functional hierarchy [73] for a system-of-systems architecture.

As shown in Figure A.1, the information can be categorized in a hierarchical fashion, which at different stages of the hierarchy supports decisions at various operational levels. *Noise*, which lies at the bottom of the information hierarchy, mainly consists of any unwanted data and no useful information is expected out of it. *Data* [15, 23] are symbols that represent information for processing purposes, based on implicit or explicit interpretation rules. In general, data lacks semantics. *Information* is data with formal and explicit semantics. Information can be communicated between two or more partners [15]. Semantics is a key aspect of information because the partners need to have a unique and unambiguous understanding of every piece of information. *Knowledge* [23] extends beyond the notion of information by also including relationships between pieces of information. In an engineering context, knowledge includes taxonomies, rules, and constraints and is also considered as value-added information for decision making. *Wisdom* [12, 15, 74] is the development of "grasp" of the overall situation with the ability to predict and project in the given domain. The ability to make use of knowledge and exhibit wisdom has been mainly attributed to human beings.

Appendix B

FMECA for Monopropellant Propulsion System (MPS)

The System: Monopropellant propulsion system (MPS): consists of Electrical and Mechanical modules.

The MPS uses hydrogen peroxide (H_2O_2) that passes over a catalyst and decomposes into oxygen, water, and heat, creating an expanding gas that produces the required thrust. The system consists of a reservoir tank of inert gas that feeds through an isolation valve IV1 to a pressure regulator RG. The pressure regulator senses the pressure downstream and opens or closes a valve to maintain the pressure at a given set point. Separating the inert gas from the propellant is a bladder that collapses as the propellant is depleted. The propellant is forced through a feed line to the thruster isolation valve IV2 and then to the thrust chamber isolation Valve IV3. For the thruster to fire, the system must first be armed, by opening the IV1 and IV2. After the system is armed, a command opens the IV3 and allows H_2O_2 to enter the thrust chamber. As the propellant passes over the catalyst, it decomposes producing oxygen, water vapor and heat. The mixture of hot expanding gases is allowed to escape through the thruster nozzle, which in turn creates the thrust. The relief valves RV1-4 are available to dump inert gas/propellant overboard should an overpressure condition occur in any corresponding part of the system.

For this study only the mechanical module was considered.

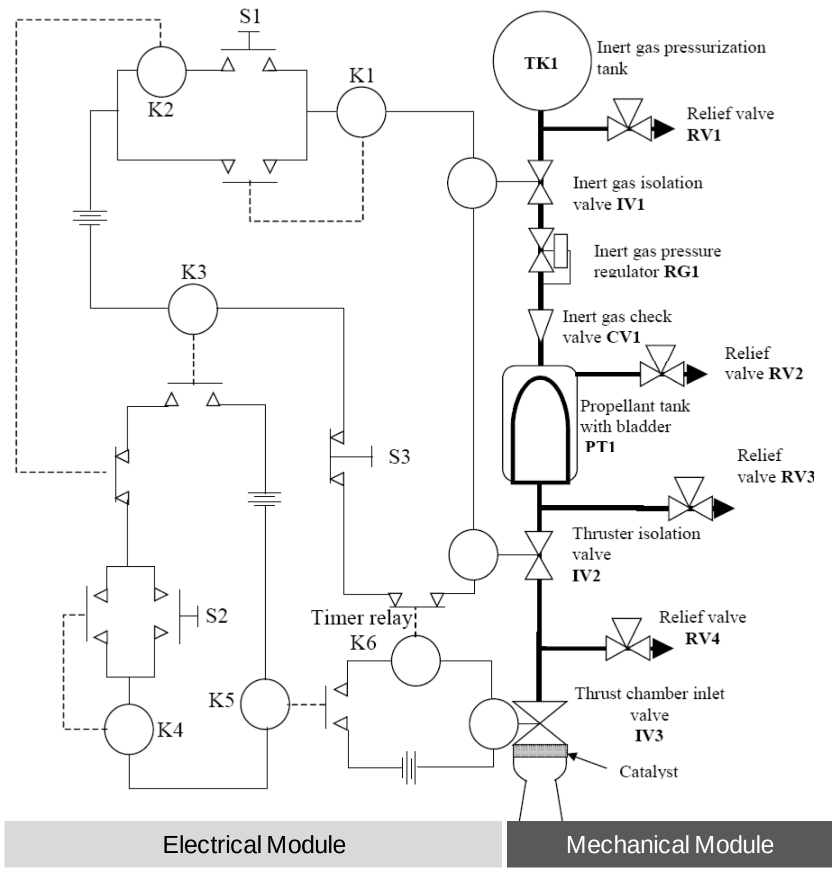

S1
K2
K1
TK1
Inert gas pressurization tank
Relief valve RV1
Inert gas isolation valve IV1
Inert gas pressure regulator RG1
K3
Inert gas check valve CV1
Relief valve RV2
Propellant tank with bladder PT1
S3
Relief valve RV3
Thruster isolation valve IV2
S2
Timer relay K6
Relief valve RV4
K5
K4
Thrust chamber inlet valve IV3
Catalyst
Electrical Module
Mechanical Module

Figure B.1 Monopropellant Propulsion System (MPS) [75].

STEP 0 - Background Study

Scope of analysis: for this study only mechanical module of the MPS was chosen.

Functionality: all components of the mechanical module were included in the study.

Operational modes: the ‘thrust’ operational mode was chosen to analyze the system.

STEP 1 - System Analysis

Figure B.2 shows the structural decomposition of the complete MPS. The selected components were used to prepare the structural block diagram (Figure B.3).

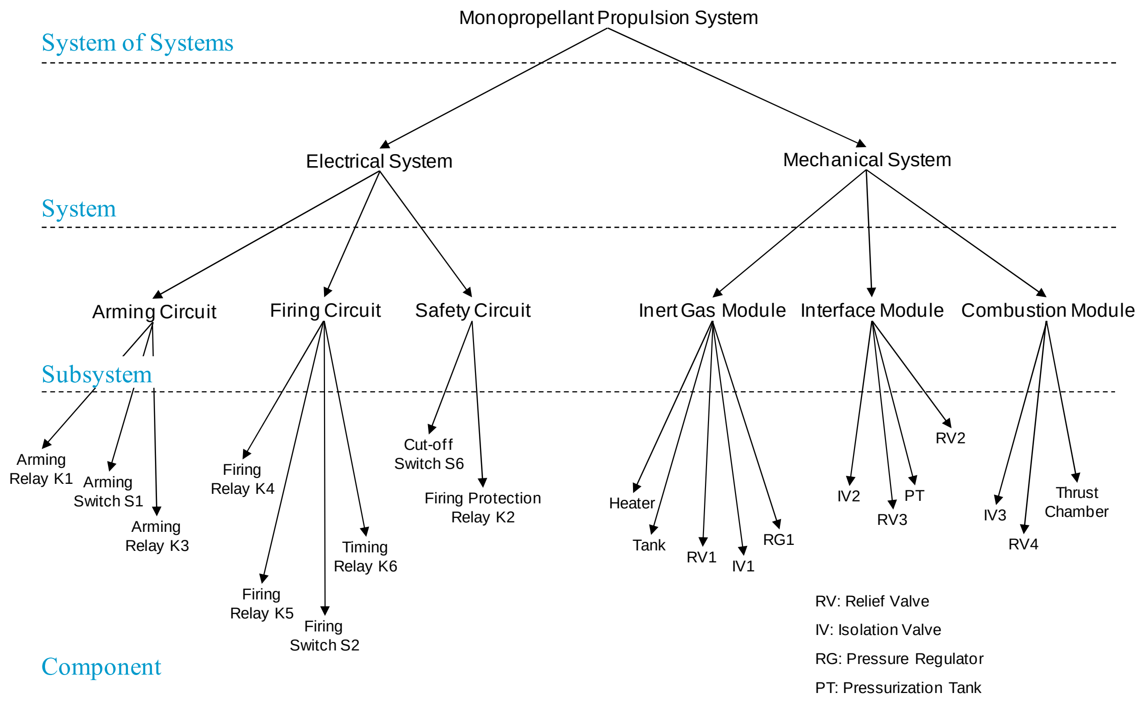


Figure B.2 Structural decomposition of the MPS.

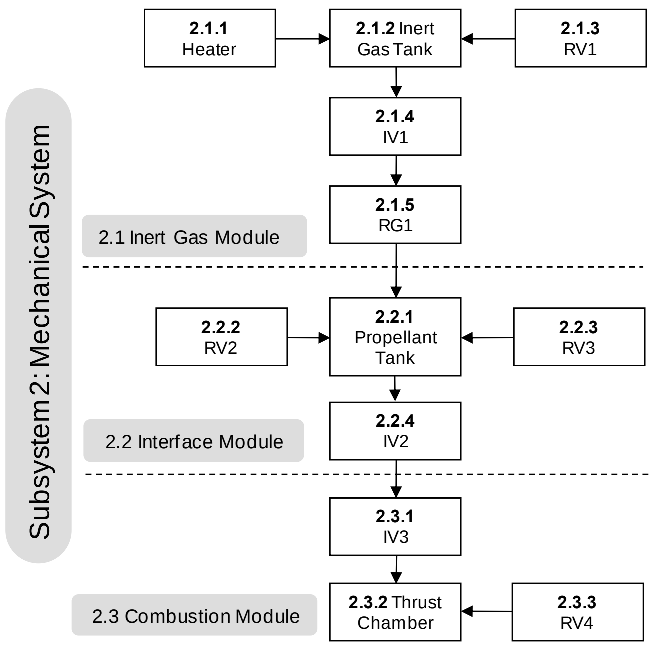


Figure B.3 Structural block diagram of the MPS.

This structural block diagram was further transformed into the functional block diagram as shown in Figure B.4. This functional block diagrams not only provides the structural information about the system but also the sensor locations and the functionality of various components. This is crucial information for populating the knowledgebase.

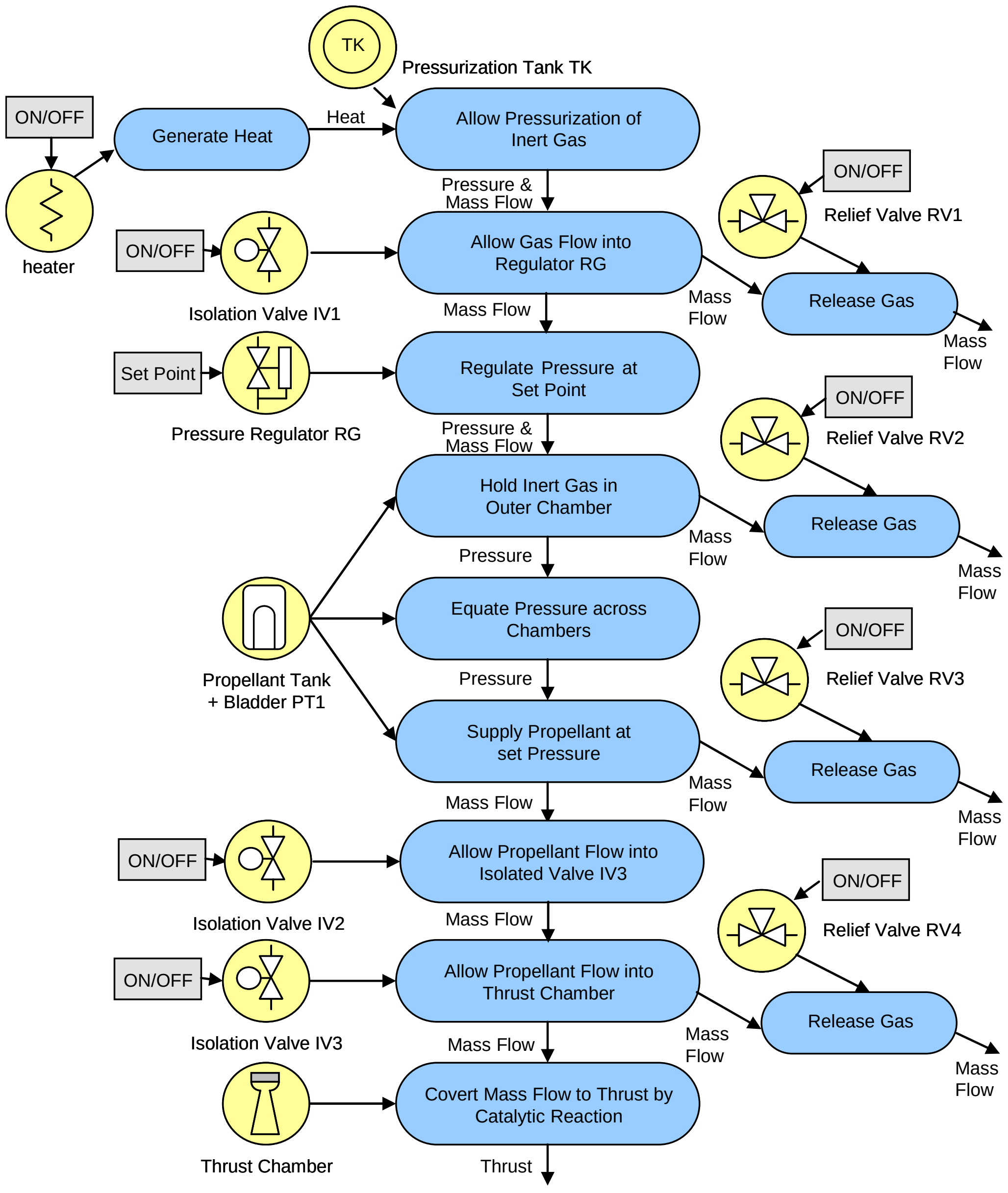


Figure B.4 Functional block diagram of the MPS.

STEP 2 - Failure Analysis:

The information regarding failures in this system was very limited. Only the failure probabilities associated with some failures were available, as shown in Figure B.5. Two more columns were added to rank these failures based on severity and frequency. This example shows that exhaustive data is not always available. But partial data is good enough to populate the knowledgebase at the beginning. More information can be added whenever it becomes available.

Component	Failure mode	Failure probability	Frequency (1-4)	Severity (1-4)
Heater	Stuck_ON	3×10^{-6}	1	2
Heater	Stuck_OFF	3×10^{-5}	2	2
Regulator	Regulator Failure 1	2×10^{-4}	3	4
Regulator	Regulator failure 2	1×10^{-5}	1	2
Isolation Valves	Stuck_ON	2×10^{-3}	4	3
Relief Valves	Stuck_OFF	5×10^{-3}	4	4

Figure B.5 Failure probabilities associated with critical components of the MPS.

STEP 3 – Risk Ranking:

A simple risk ranking metric can be defined as $I = F.S^2$. However, more combinations of these parameters can be employed.

In this example we have shown how a generic FMECA approach can be used to acquire information for knowledge acquisition for the maintenance knowledgebase.

Appendix C

Notions of Similarity

The concept of similarity is represents the notion of inexact matching and can be considered as the dual of distance concept. In general the basic similarity metric can be expressed as $sim(a_1,a_2)$, where the following properties are satisfied.

$(i) \quad 0 \le sim(a_1,a_2) \le 1 \Rightarrow Normalization$
$(ii) \quad sim(a_1,a_1) = 1 \Rightarrow Each\ entity\ is\ neighbor\ of\ itself$
$(iii) \quad sim(a_1,a_2) = sim(a_2,a_1) \Rightarrow Similarity$
$(iv) \quad d(a_1,a_1) \le d(a_1,a_1) + d(a_1,a_1) \Rightarrow Triangle\ inequality\ in\ terms\ of\ dis\tan ce\ measures$

Generally distance $d(a_1,a_2)$ is inversely proportional to similarity $sim(a_1,a_2)$, for example as shown below

$$sim(a_1,a_2) = \frac{1}{1 + dist(a_1,a_2)} \tag{7.1}$$

However, in real life systems such objective notions do not suffice and a fair amount of flexibility is required while dealing with uncertainties arising due to discontinuity and non-linearity in the data. A concept of *acceptance* has been introduced in [76] that tries to quantify the subjective notion of usability of a source case. Therefore, even if the traditional similarity metrics suggest close similarities, a source case may not be as useful depending on the context.

Further, a notion of *utility* has been introduced as an extension to the traditional similarity metrics in [77]. In contrast to a similarity measure, where the distance is calculated between the query problem and the problem part of a case, utility-measure directly assesses relevance between the query problem and the solution part of the past cases (see Figure C.1). This approach aims at improving the effectiveness of problem solving however, stands on a crucial assumption that the knowledge about the utility of cases for particular problem solutions is known. While case knowledge is often already available in the form of existing or easily collectable data-sets, the knowledge about the utility of cases for new problem situations is usually not available in such an explicit form. It must be acquired by consulting a domain expert who possesses implicit knowledge about the underlying utility function. Further, the acquired knowledge has to be formalized into similarity representation structures. This knowledge engineering task is a difficult and time consuming procedure, rendering this approach limited to academic purposes only.

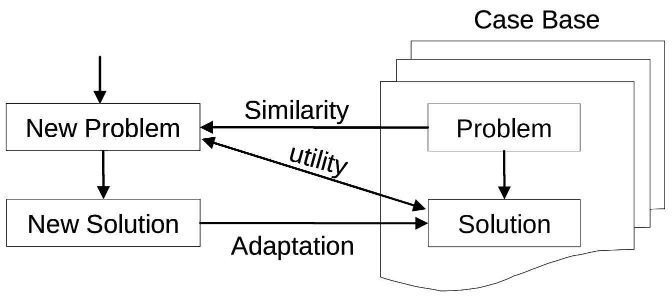


Figure C.1 Concept of utility is defined as a relevance between the new problem and solution of some old problem (case) [77].

Another alternative has been suggested in [78] where the concept of utility need not be established explicitly but is inherent in top-down approach of learning similarity function.

In this approach, exact utility need not be known but a relative assessment is considered between various candidate cases. This knowledge is not considered complete for assessing similarity but is used only to improve the regular bottom-up approach of similarity calculation.

The most common approach taken is to use simple similarity functions to express closeness on various dimensions and then compose them into one compound metric through weighted averages. Table C.1 lists a number of generic similarity functions that can be used in different situations and later combined into a composite similarity measure as explained in Chapter 5.

Table C.1 Generic similarity measures that can be composed to create custom similarity metrics.

Similarity Function	Expression
Equality Test	$sim(a_1,a_2)=\begin{cases}1 & if\ a_1=a_2\\ 0 & otherwise\end{cases}$
Symmetric ordinal similarity	$sim(a_1,a_2)=1-\frac{\lvert pos(a_1,list)-pos(a_2,list)\rvert}{\lvert list\rvert-1}$
Symmetric set similarity	$sim(a_1,a_2)=1-\frac{\lvert a_1\cap a_2\rvert}{\lvert a_1\cup a_2\rvert}$
Symmetric similarity of numbers with lower bound	$sim(a_1,a_2)=1-\frac{\lvert a_1-a_2\rvert}{\max(a_1,a_2)-lowerbound}$
Symmetric similarity of numbers with upper bound	$sim(a_1,a_2)=1-\frac{\lvert a_1-a_2\rvert}{upperbound-\min(a_1,a_2)}$
Symmetric similarity of numbers in an interval	$sim(a_1,a_2)=1-\frac{\lvert a_1-a_2\rvert}{upperbound-lowerbound}$
Asymmetric ordinal similarity	$\Delta pos=1-\frac{\lvert pos(a_1,list)-pos(a_2,list)\rvert}{\lvert list\rvert-1}$ $=1-\frac{\lvert p_1-p_2\rvert}{\lvert list\rvert-1}$ $sim(a_1,a_2)=\begin{cases}\Delta pos^{1/3} & if\ p_1>p_2\\ 1-\Delta pos^{1/3} & otherwise\end{cases}$

Similarity Function	**Expression**
Asymmetric set similarity	$sim(a_1, a_2) = \frac{\lvert a_1 \cap a_2 \rvert}{\lvert a_1 \rvert}$
Asymmetric similarity of numbers with lower bound	$\Delta a = \frac{\lvert a_1 - a_2 \rvert}{\max(a_1, a_2) - lowerbound}$ $sim(a_1, a_2) = \begin{cases} \Delta a^{1/3} & if\ a_1 > a_2 \\ 1 - \Delta a^{1/3} & otherwise \end{cases}$
Asymmetric similarity of numbers with upper bound	$\Delta a = \frac{\lvert a_1 - a_2 \rvert}{upperbound - \min(a_1, a_2)}$ $sim(a_1, a_2) = \begin{cases} \Delta a^{1/3} & if\ a_1 > a_2 \\ 1 - \Delta a^{1/3} & otherwise \end{cases}$
Asymmetric similarity of numbers in an interval	$\Delta a = \frac{\lvert a_1 - a_2 \rvert}{upperbound - lowerbound}$ $sim(a_1, a_2) = \begin{cases} \Delta a^{1/3} & if\ a_1 > a_2 \\ 1 - \Delta a^{1/3} & otherwise \end{cases}$
Algebraic average similarity	$sim = \frac{\sum_{i=1}^{n} sim_i}{n}$
Geometric average similarity	$sim = \sqrt{\frac{\sum_{i=1}^{n} sim_i^2}{n}}$

References

[1] Vachtsevanos, G., F.L. Lewis, M. Roemer, A. Hess, and B. Wu (2006), *Intelligent Fault Diagnosis and Prognosis for Engineering Systems*: Wiley (September 29, 2006). (Pages 456).

[2] Tumer, I.Y. and A. Bajwa (1999), *A Survey of Aircraft Engine Health Monitoring Systems*. in 35th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit. Los Angeles, CA: AIAA-99-2528.

[3] Chiu, C., N.-H. Chiu, and C.-I. Hsu (2004), *Intelligent Aircraft Maintenance Support System Using Genetic Algorithms and Case-Based Reasoning.* International Journal of Advanced Manufacturing Technology. **24**(5-6): p. 440-446.

[4] Devaney, M. and B. Cheetham (2005), *Case-Based Reasoning for Gas Turbine Diagnostics*. in 18th International FLAIRS Conference Clearwater, FL.

[5] Varma, A. and N. Roddy (1999), *ICARUS: Design and Deployment of a Case-Based Reasoning System for Locomotive Diagnostics.* Engineering Applications of Artificial Intelligence. **12**: p. 681-690.

[6] Lee, S.G. and Y.C. Ng (2006), *Case-Based Reasoning for On-Line Product Fault Diagnosis.* International Journal of Advanced Manufacturing Technology. **27**(7-8): p. 833-840.

[7] Lehane, M., F. Dubé, M. Halasz, R. Orchard, R. Wylie, and M. Zaluski (1998), *Integrated Diagnostic System (IDS) for Aircraft Fleet Maintenance*. in AAAI '98 Workshop: Case-based Reasoning Integrations. Madison, Wisconsin, USA. p. 91-95.

[8] Power, Y. and P.A. Bahri (2005), *Integration Techniques in Intelligent Operational Management : A Review.* Knowledge-Based Systems. **18**: p. 89-97.

[9] Gaines, J. and P.J. Sisa (2005), *Machinery Health Monitoring – Sense & Respond Logistics*. Maintenance Technology January 28, 2007 [cited; Available from: http://www.mt-online.com/articles/1205equipmentreliability.cfm.

[10] DoD (2003), *Condition Based Maintenance Plus (CBM+)*. [cited 2007 Jan 28]; Fact Sheet - US Air Force]. Available from: http://www.af.mil/shared/media/document/AFD-060831-040.pdf.

[11] DoD (2006), *U.S. Army CBM+ Roadmap, Revised Draft 3*, DCS Logistics,G-4, Headquarters, Department of the Army.

[12] Barlas, I., A. Ginart, and J.L. Dorrity (2005), *Self-Evolution in Knowledge Bases*. in IEEE Autotestcon. Orlando, FL. p. 325-331.

[13] O'Niel, G. (2004), *CBM+*, Georgia Tech Research Institute: Atlanta.

[14] Drury, C.G. and J. Ma (2003), *Language Errors in Aviation Maintenance.* Year 1 Interim Report for Federal Aviation Administration, State University of New York Buffalo

[15] Giarratano, J. and G. Riley (1998), *Expert Systems*. 3rd ed: PWS Publishing Company. (Pages 856).

[16] Nick, M. and K.-D. Althoff (2001), *Engineering Experience Based Maintenance Knowledge.* IESE-Report,018.01/E, Fraunhofer IESE

[17] Allen, P.A. (1994), *Case-Based Reasoning: Business Applications, Knowledge Engineering Systems.* Communications of the ACM. **37**(3): p. 40-42.

[18] Saha, B. and G. Vachtsevanos (2006), *A Novel Model-Based Reasoning Approach to System Fault Diagnosis*. in 10th WSEAS International Conference on SYSTEMS.

[19] Rausand, M. and A. Høyland (2004), *System Reliability Theory: Models, Statistical Methods, and Applications*. 2nd ed: Wiley-Interscience; 2 edition. (Pages 664).

[20] Balconi, M. (2002), *Tacitness, Codification of Technological Knowledge and the Organisation of Industry.* Research Policy. **31**(3): p. 357-379.

[21] Cowan, R. (2001), *Expert Systems: Aspects of and Limitations to the Codifiability of Knowledge.* Research Policy. **30**: p. 1355 - 1372.

[22] Motta, E. (1999), *Reusable Components for Knowledge Modeling*, in Frontiers in Artificial Intelligence and Applications, IOS Press.

[23] Wilson, D. and P. Schenck (1994), *Information Modeling: The EXPRESS Way*: Oxford University Press.

[24] Schreiber, A.T., J.M. Akkermans, A.A. Anjewierden, R. De Hoog, W. Van De Velde, and B.J. Wielinga (1998), *Engineering of Knowledge: The CommonKADS Methodology*. version 0.5 ed: University of Amsterdam.

[25] Gentner, D. and A.B. Markman (1997), *Reasoning and Learning by Analogy: Introduction.* American Psychologist, **52**: p. 45-56.

[26] Schank , R.C. and C.K. Reisbeck (1989), ed. *Inside Case-Based Reasoning*. 1st ed., Lawrence Erlbaum Associates: Hillsdale, New Jersey.

[27] Aamodt, A. and E. Plaza (1994), *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches.* AI- Communications **7**(1): p. 39-59.

[28] Watson, I. and F. Marir (2004), *Case-Based Reasoning: A Review*. in 7th European Conference, ECCBR. Berlin Heidelberg: Springer-Verlag. p. 361-374.

[29] Kolodner, J.L. (1993), *Case-Based Reasoning*, San Mateo, CA: Morgan Kaufmann Publishers, Inc (Pages 612).

[30] Roth-Berghofer, T.R. (2003), *Knowledge Maintenance of Case-Based Reasoning Systems - The SIAM Methodology*, University of Kaiserslautern, Germany. p. 240.

[31] Richter, M.M. (1997), *Generalized Planning and Information Retrieval.* Artificial Intelligence - Knowledge-Based Systems, University of Kaiserslautern

[32] Gebhardt, F., A. Voss, W. Grather, and B. Schmdt-Belz (1997), *Reasoning With Complex Cases*. The Kluwer International Series In Engineering and Computer Science, Boston: Kluwer Academic Publishers. (Pages 250).

[33] Miquel, S.-M., C. Ulises, M. Montse, C. Joaquim, R.-R. Ignasi, M.-A. Hector, and R. Francesco (2005), *An Approach for Temporal Case-Based Reasoning: Episode-Based Reasoning*. in ICCBR : International Conference on Case-Based Reasoning Chicago, IL: Springer, Berlin p. 465-476.

[34] Saxena, A., B. Wu, and G. Vachtsevanos (2005), *Integrated Diagnosis and Prognosis Architecture for Fleet Vehicles Using Dynamic Case Based Reasoning*. in IEEE Autotestcon. Orlando, FL. p. 96-104.

[35] Saxena, A., B. Wu, and G. Vachtsevanos (2006), *A Hybrid Reasoning Architecture for Fleet Vehicle Maintenance.* IEEE Instrumentation and Measurement Magazine, **9**(4): p. 29-36.

[36] Xia, Q. and M. Rao (1999), *Dynamic Case-Based Reasoning for Process Operation Support Systems.* Engineering Applications of Artificial Intelligence. **12**(3): p. 343-361.

[37] Bonissone, P. and R. Tong (1985), *Editorial: Reasoning with Uncertainty in Expert Systems.* International Journal of Man-Machine Studies. **22**: p. 241-250.

[38] Weber, R. (2006), *Fuzzy Set Theory and Uncertainty in Case-Based Reasoning.* International Journal of Engineering Intelligent Systems. **14**(3): p. 121-136.

[39] Burkhard, H.-d. and M.M. Richter (2001), ed. *On the Notion of Similarity in Case Based Reasoning and Fuzzy Logic*. Springer-Verlag: London.

[40] Olsson, E., P. Funk, and M. Bengtsson (2004), *Fault Diagnosis of Industrial Robots Using Acoustic Signals and Case-Based Reasoning*. in European Conference on Case-Based Reasoning ECCBR: Berlin Heidelberg Springer-Verlag. p. 686-701.

[41] Nirmalie, W., I. Koychev, and S. Massie (2004), *Feature Selection and Generalization for Retrieval of Textual Cases*. 1st ed. Advances in Case-Based Reasoning - LNAI, ed. P. Funk, C. Gonzalez, and A. Pedro. Vol. 3155, Berlin Heidelberg: Springer-Verlag. (Pages 822).

[42] Mostek, T.A., K.D. Forbus, and C. Meverden (2000), *Dynamic Case Creation and Expansion for Analogical Reasoning*. in Seventeenth National Conference on Artificial Intelligence (AAAI). p. 323-329.

[43] Winston, P.H. (1993), *Artificial Intelligence*. 3rd ed: Addison-Wesley Publishing Co. (Pages 691).

[44] Stefik, M. (1995), *Introduction to Knowledge Systems*. 1st ed, San Francisco, CA: Morgan Kaufmann Publishers (Pages 896).

[45] Manning, C.D. and H. Schütze (1999), *Foundations of Statistical Natural Language Processing*. 1st ed: MIT Press. (Pages 620).

[46] Lenz, M., A. Hubner, and M. Kinze (1998), *Textual CBR*, in Case-Based Reasoning Technology: From Foundations to Applications, M. Lenz, et al., Editors, Springer Verlag.

[47] Cunningham, C.M., R. Weber, J.M. Proctor, C. Fowler, and M. Murphy (2004), ed. *Investigating Graphs in Textual Case-Based Reasoning*. 1st ed. Advances in Case-Based Reasoning - LNAI, ed. P. Funk, C. González, and P. A. Vol. 3155, Springer-Verlag: Berlin Heidelberg.

[48] Schenkar, A., M. Last, H. Bunke, and A. Kandel (2003), *Clustering of Web Documents Using a Graph Model*, in Web Document Analysis: Challenges and Opportunities, A. Antonacopoulos and J. Hu, Editors. p. 1-16.

[49] Greenough, R. *e-Smart – Electronic Support of Manufacturing Technology* Work Package Report, Cranfield University, UK

[50] Wojcik, R.H. and J.E. Hoard (1996), *Survey of the State of the Art in Human Language Technology*, in *Controlled Languages in Industry*, R.A. Cole, et al., Editors.

[51] Unwalla, M. (2004), *AECMA Simplified English*. The AECMA SE Guide [cited 02/02/2007]; Available from: http://www.simplifiedenglish-aecma.org/Simplified_English.htm.

[52] Adriaens, m.G. and D. Schreuers (1992), *From COGRAM to ALCOGRAM: Toward a controlled English grammar checker*. in 14th International Conference on Computational Linguistics. Nantes, France. p. 595-601. .

[53] ASD-Stan (2006), *ASD-STE100 Simplified Technical English*, Retrieved 05/10/07.from: http://www.asd-stan.org/

[54] Smeaton, A.F. (1992), *Progress in the Application of Natural Language Processing to Information Retrieval Tasks.* The Computer Journal(3).

[55] Schmidt, H. (1994), *Probabilistic Part-of-Speech Tagging Using Decision Trees*. in International Conference on New Methods in Language Processing. Manchester UK. p. 44-49.

[56] Ashley, K. (1999), *Progress in Text-Based Case-Based Reasoning*. in 3rd International Conference on Case-Based-Reasoning. Seeon, Germany.

[57] Liu, L.F., Z.Q. Mi, Z. Zhang, and B.Z. Liu (1998), *Research on Case Organization and Retrieval of Case and Rule Based Reasoning Approaches for Electric Power Engineering Design*. in International Conference in Power Systems Technology. Beijing, China p. 1082-1085.

[58] Garey, M.R. and D.S. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, New York: W.H. Freeman and Company. (Pages 340).

[59] Peugeot (2005), *Vehicle Fault Finding*. [cited February, 2007]; Available from: http://peugeot.mainspot.net/fault_find/index.shtml.

[60] Wooldridge, M. and R.N. Jennings (1995), *Agent Theories and Architectures, and Languages: A Survey*, in Intelligent Agents, M. Wooldridge and R.N. Jennings, Editors, Springer-Verlag: Berlin. p. 1-22.

[61] Maes, P. (1995), *Artificial Life Meets Entertainment: Life Like Autonomous Agents.* Communications of the ACM. **38**(11): p. 108-114.

[62] Konar, A. (2000), ed. *Artificial Intelligence and Soft computing: Behavioral and Cognitive Modeling of the Human Brain*. CRC Press.

[63] Barlas, I. (2004), *Case-Based Temporal Reasoner.* Self-Evolving Maintenance Knowledge Bases: Navy SBIR FY2004.1,Navy SBIR FY2004.1, Intelligent Automation Systems,Inc.

[64] Tang, L., G.J. Kacprzynski, J.R. Bock, and M. Begin (2006), *An Intelligent Agent-Based Self-evolving Maintenance and Operations Reasoning System*. in IEEE Aerospace Conference. p. 12.

[65] Sugiyama, S. (1994), *Self Evolving Knowledge Base*. in IEEE Systems, Man and, Cybernetics. San Diego, CA, USA. p. 1978-1983.

[66] Sutton, R.S. and A.G. Barto (1998), *Reinforcement Learning: An Introduction* Cambridge, Massachusetts: The MIT Press. (Pages 322).

[67] Watkins, C.C.H. and P. Dayan (1992), *Q-Learning.* Machine Learning. **8**: p. 279-292.

[68] Isbell, C., C. Shelton, M. Kearns, S. Singh, and P. Stone (2001), *Cobot: A Social Reinforcement Learning Agent*. in 5th International Conference on Autonomous Agents.

[69] Thomaz, A.L. and C. Breazeal. (2006), *Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance*. in Proceedings of the 21st National Conference on Artificial Intelligence (AAAI).

[70] Kaplan, F., P.-Y. Oudeyer, E. Kubinyi, and A. Miklosi (2002), *Robotic Clicker Training,.* Robotics and Autonomous Systems. **38**(3-4): p. 197-206.

[71] Thomaz, A.L., G. Hoffman, and C. Breazeal (2006), *Reinforcement Learning with Human Teachers: Understanding How People Want to Teach Robots*. in Proceedings of the 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN). Univ. of Hertfordshire, Hatfield, UK. p. 352-357.

[72] Smart, W. and L. Kaelbling (2002), *Effective Reinforcement Learning for Mobile Robots*. in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '02). p. 3404 - 3410.

[73] Reklaitis, R.G.V. and L.B. Koppel (1995), *Role and Prospects for Intelligent Systems in Integrated Process Operations*. in International Conference on Intelligent Systems in Process Engineering. p. 71-84.

[74] Ackoff, R.L. (1989), *From Data to Wisdom.* Journal of Applied Systems Analysis. **16**.

[75] Vesely, W., M. Stamatelatos, J. Dugan, J. Fragola, J. Minarick III, and R. J. (2002), *Fault Tree Handbook with Aerospace Applications.* version 1.1, NASA Office of Safety and Mission Assurance

[76] Burkhard, H.-D. (1998), *Extending Some Concepts of CBR - Foundations of Case Retrieval Nets*, in Case-Based Reasoning Technology, From Foundations to Applications, M. Lenz, Editor, Springer-Verlag: London, UK. p. 17-50.

[77] Bergmann, R., M.M. Richter, S. Schmitt, A. Stahl, and I. Vollrath (2001), *Utility-Oriented Matching: A New Research Direction for Case-Based Reasoning*. in German Workshop on Case Based Reasoning, GWCBR. Baden-Baden Germany: Shaker-Verlag. p. 264-274.

[78] Stahl, A. (2002), *Defining Similarity Measures: Top-Down vs. Bottom-Up*. in 6th European Conference on Case-Based Reasoning (ECCBR): Springer. p. 406-420.